



USER MANUAL
GRID RESOURCE MANAGEMENT

WP3.2

| | |
|--------------------------|--|
| Document Filename: | CG3.2-v1.2-UAB-GridResourceManagementUserManual.doc |
| Work package: | WP3.2 |
| Partner(s): | UAB, CSIC, DATAMAT |
| Lead Partner: | UAB |
| Config ID: | CG3.2-v1.2-UAB-GridResourceManagementUserManual |
| Document classification: | PUBLIC |

Document Log

| Version | Date | Summary of changes | Author |
|---------|------------|---|------------------|
| 1.0 | 16/11/2004 | First version | Enol Fernández |
| 1.1 | 03/12/2004 | Added DAG commands | Enol Fernández |
| 1.2 | 09/12/2004 | Added information about monitoring plugin | Álvaro Fernández |
| | 26/01/2005 | Verified by the QE | Robert Pajak |

CONTENTS

| | |
|--|-----------|
| COPYRIGHT NOTICE | 4 |
| 1. INTRODUCTION..... | 5 |
| 1.1. ABBREVIATIONS AND ACRONYMS | 5 |
| 1.2. REFERENCES AND SOURCE CODE | 6 |
| 2. PRODUCT USAGE | 7 |
| 2.1. RUNNING THE PRODUCT | 7 |
| 2.1.1. <i>Operating Requirements</i> | 7 |
| 2.1.2. <i>Step-by-Step User Setup</i> | 7 |
| 2.2. BASIC OPERATION | 9 |
| 2.3. ADVANCED FEATURES..... | 11 |
| 2.3.1. <i>Multiple CEs selection</i> | 11 |
| 2.3.2. <i>Interactive MPICH-P4 and MPICH-G2 Jobs support</i> | 17 |
| 2.3.3. <i>Monitoring Plug-in Support</i> | 22 |
| 2.3.4. <i>DAG support</i> | 28 |
| 3. INTERFACE REFERENCE GUIDE..... | 38 |
| 3.1. EDG-JOB-SUBMIT | 38 |
| 3.2. EDG-JOB-GET-OUTPUT | 51 |
| 3.3. EDG-JOB-LIST-MATCH | 53 |
| 3.4. EDG-JOB-CANCEL | 57 |
| 3.5. EDG-JOB-STATUS | 60 |
| 3.6. EDG-JOB-GET-LOGGING-INFO..... | 65 |
| 3.7. EDG-JOB-ATTACH..... | 69 |
| 3.8. EDG-JOB-GET-CHKPT..... | 72 |
| 4. CONTACT INFORMATION AND CREDITS..... | 75 |
| 5. THE EDG LICENSE AGREEMENT | 76 |

COPYRIGHT NOTICE

Copyright (c) 2004 by Álvaro Fernández Casani, Enol Fernández del Castillo, Antonio Hervás Vilchez, Elisa Heymann Pignolo, Anna Morajko and Miquel Angel Senar on behalf of the EU CrossGrid. All rights reserved.

This research is partly funded by the European Commission IST-2001-32243 Project “CrossGrid”.

Use of this product is subject to the terms and licenses stated in the EDG license agreement. Please refer to Section 6 for details.

This software uses code from the following products:

EDG Workload Management. Copyright (c) 2002 CERN and INFN on behalf of the EU DataGrid.

1. INTRODUCTION

Resource management in a Grid has to deal with a heterogeneous multi-site computing environment that in general exhibits different hardware architectures, loss of centralized control, and as a result, inevitable differences in policies. Additionally, due to the distributed nature of the Grid environment, computers, networks and storage devices can fail in various ways. The resource management system that we are developing in the CrossGrid is targeted to a kind of applications that have received very little attention up to now. Most existing systems have focussed on the execution of sequential jobs, the Grid being a large multi-site environment where the jobs run in a batch-like way.

CrossGrid jobs are computationally intensive applications that are mostly written with the MPI library. In some MPI jobs, once the job has been submitted to the Grid and has started its execution on remote resources, the user may want to steer its execution in an interactive way. This is required to analyze intermediate results produced by the application and to react according to them. For instance, in the case where a simulation is not converging, the user may kill the current job and submit a new simulation with a different set of input parameters.

Other CrossGrid applications may consist of inter-dependent jobs, where information or tasks are passed from one job to another for action, according to a set of rules. Such applications are known as workflows and consist of a collection of jobs that need to be executed in a partial order determined by control and data dependencies. Normally, a user should submit to a Grid system manually, job by job, following the rules of dependencies that appear between jobs. The manual tracking of the application workflow may be very ineffective, time consuming and may produce many errors in application execution.

1.1. ABBREVIATIONS AND ACRONYMS

| | |
|------------|---|
| API | Application Program Interface |
| CE | Computing Element |
| ClassAdd | Classified Advertisements |
| Condor-G | Condor Component that interfaces with globus |
| CPU | Central Processing Unit |
| CVS | Concurrent Versioning System |
| DataGrid | The EU DataGrid Project IST-2000-25182 |
| EDG | European Datagrid abbreviation |
| Globus | Grid middleware |
| II | Information Index |
| JDL | Job Definition Language |
| JSS/ JSS's | Job Submission Services |
| LB | Logging&Bookkeeping |
| LCG/LCG-1 | LHC Computing Grid (version 1) |
| MPICH | Implementation of the Message Passing Interface library |
| PBS | Portable Batch Scheduler |
| RB | Resource Broker |
| RPMS | Red Hat Package Management (packages) |
| RS | Resource Selector |
| SE | Storage Element |

| | |
|-----|----------------------------|
| UI | User Interface |
| WMS | Workload Management System |

1.2. REFERENCES AND SOURCE CODE

Current development of CrossGrid middleware is based on the DataGrid WP1 WMS Software, with the corresponding modifications to add the new functionality required by the CrossGrid applications.

The edge release currently used is EDG 2.1.15, and CrossGrid specific release is currently version 2.5. We name the rpm packages provided with the juxtaposed number version, which makes our current release number **2.1.15.2.5** (lcg-2 compliant).

Source code can be found at CrossGrid CVS repository, under crossgrid/crossgrid/wp3/wp3_2-scheduling directory. Please refer to installation guide for building instructions.

References:

- [1] DATAGRID WP1 – WMS Software administrator and User Guide (DataGrid-01-TEN-0118-1_1.doc)download at <http://server11.infn.it/workload-grid/documents.html>
- [2] The EU-Crossgrid Approach for Grid Application Scheduling (Elisa Heymann, Álvaro Fernandez, Miquel A. Senar, Jose Salt). To be published at post-proceedings in the Springer Lecture Notes in Computer Science. Presented at Across Grids Conference and available at <http://alpha.ific.uv.es/~alferca/SchedulingEuCrossgridApproach.doc>
- [3] Crossgrid Deliverable D3.5 ([CG3.0-D3.5-v1.2-PSNC010-Proto2Status](http://www.crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf))download <http://www.crossgrid.org/Deliverables/M24pdf/CG3.0-D3.5-v1.2-PSNC010-Proto2Status.pdf>
- [4] Work Package 3.2 Installation Guide (CG3.2-D3.3-v2.1-CSIC-installationguide.doc) download at http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp3/wp3_2-scheduling/docs/CG3.2-v2.2-InstallationGuide.pdf
- [5] Data grid: Definition of the architecture, technical plan and evaluation criteria for the resource coallocation framework and mechanisms for parallel job partitioning. WP1: Workload Management. DataGrid-01-D1.4-0127-1_0. Deliverable DataGrid-D1.4. 13.09.2002.
- [6] <http://infnforge.cnaf.infn.it/cgi-bin/cvsweb.cgi/workload/>
- [7] <http://www.cs.wisc.edu/condor/dagman/>
- [8] Job Description Language HowTo (DataGrid-01-TEN-0102-02) download at http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2.pdf
- [9] Crossgrid WP3.2 Monitoring Tools Integration. Internal document. http://savannah.fzk.de/distribution/crossgrid/crossgrid/wp3/wp3_2-scheduling/docs/CG3.2-v1.2-MonitoringPlugin.pdf

2. PRODUCT USAGE

This guide focuses on the User Interface (UI), which is the component that allows users to access the functionality offered by the Workload Management System.

The user interaction with the system is assured by means of a JDL and a command-driven user interface providing commands to perform a certain set of basic operations. Main operations made possible by the UI are:

- Submit a job for execution on a remote Computing Element, also encompassing:
 - automatic resource discovery and selection
 - staging of the application sandbox (input sandbox)
- Find the list of resources suitable to run a specific job
- Cancel one or more submitted jobs
- Retrieve the output files of a completed job (output sandbox)
- Retrieve and display bookkeeping information about submitted jobs
- Retrieve and display logging information about submitted jobs.
- Retrieve checkpoint states of a submitted checkpointable job.
- Start a local listener for an interactive job.

The User Interface depends on two other Workload Management System components:

- the Network Server that provides support for the job control functionality
- the Logging and Bookkeeping Service that provides support for the job monitoring functionality.

2.1. RUNNING THE PRODUCT

2.1.1. Operating Requirements

The User Interface is installed as a separate machine using an LCFG server. This machine has no special requirements and should not be installed manually. For installation instructions refer to Installation guide [4].

2.1.2. Step-by-Step User Setup

There is no special setup needed after installation, the command are ready to use. In special cases where sites are not installed by an LCFG server, configuration of the is accomplished through the file `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf`

This file is installed by the LCFGng object `edg-lcfg-cliconfig`, so if the installation is not performed using LCFGng, after having installed the UI rpm (`edg-wl-ui-cli-X.Y.Z-K_gcc3_2_2.i486.rpm`) that creates the file:

```
$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf.template
```

you must copy it in:

```
$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf
```

and update the content of the latter file opportunely.

The `edg_wl_ui_cmd_var.conf` file is a classad containing the following fields:

- **requirements** this is an expression representing the default value for the requirements expression in the JDL job description. This parameter is mandatory. The value of this parameter is assigned by the UI to the *requirements* attribute in the JDL if not specified by the user. If the user has instead provided an expression for the *requirements* attribute in the JDL, the one specified in the configuration file is added (in AND) to the existing one. E.g. if in the `edg_wl_ui_cmd_var.conf` configuration file there is:

```
requirements = other.GlueCEStateStatus == "Production";
```

and in the JDL file the user has specified:

```
requirements = other.GlueCEInfoLRMSType == "PBS";
```

then the job description that is passed to the NS contains

```
requirements = (other.GlueCEInfoLRMSType == "PBS") &&  
(other.GlueCEStateStatus == "Production");
```

Obviously the setting TRUE for the *requirements* in the configuration file does not have any impact on the evaluation of job requirements as it would result in:

```
requirements = (other.GlueCEInfoLRMSType == "PBS") && TRUE;
```

- **rank** this is an expression representing the default value for the rank expression in the JDL job description. The value of this parameter is assigned by the UI to the *rank* attribute in the JDL if not specified by the user. This parameter is mandatory.
- **RetryCount** this is an integer representing the default value for the number of submission retries for a job upon failure due to some grid component (i.e. not to the job itself). The value of this parameter is assigned by the UI to the *RetryCount* attribute in the JDL if not specified by the user.
- **DefaultVo** this is a string representing the name of the virtual organisation to be taken as the user's VO (*VirtualOrganisation* attribute in the JDL) if not specified by the user neither directly in the job description nor through the `--vo` option.
- **ErrorStorage** this is a string representing the path of the directory where the UI creates log files. This directory is not created by the UI, so It has to be an already existing directory. Default for this parameter is `/tmp`.
- **OutputStorage** this is a string defining the path of the directory where the job *OutputSandbox* files are stored if not specified by the user through commands options. This directory is not created by the UI, so It has to be an already existing directory. Default for this parameter is `/tmp`.
- **ListenerStorage** this is a string defining the path of the directory where are created the pipes where the `edg_grid_console_shadow` process saves the job standard streams for interactive jobs. Default for this parameter is `/tmp`.
- **LoggingDestination** this is a string defining the address (`<host>:[<port>]`) of the logging service (`edg-wl-logd` logging daemon) to be targeted when logging events. The UI first check the environment for the `EDG_WL_LOG_DESTINATION` variable and only if this is not set, the value of the `LoggingDestination` parameter is taken into account.
- **LoggingTimeout** this is an integer representing the timeout in seconds for asynchronous logging function called by the UI when logging events to the LB. Recommended value for UI that are non-local to the logging service (`edg-wl-logd` logging daemon) is not less than 30 seconds.

- **LoggingSyncTimeout** this is an integer representing the timeout in seconds for synchronous logging function called by the UI when logging events to the LB. Recommended value is not less than 30 seconds.
- **DefaultStatusLevel** this is an integer defining the default level of verbosity for the *edg-job-status* command. Possible values are 0, 1 and 2. 0 is the default and means minimum verbosity. Default for this parameter is 0.
- **DefaultLogInfoLevel** this is an integer defining the default level of verbosity for the *edg-job-get-logging-info* command. Possible values are 0,1 and 2. 0 is the default and means minimum verbosity. Default for this parameter is 0.
- **NSLoggerLevel** this is an integer defining the quantity of information logged by the NS client. Possible values range from 0 to 6. 0 is the defaults and means that no information is logged. Default for this parameter is 0.

Hereafter is provided an example of the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` configuration file.

```
[
  requirements = other.GlueCEStateStatus == "Production" ;
  rank = - other.GlueCEStateEstimatedResponseTime ;
  RetryCount = 3 ;
  ErrorStorage= "/var/tmp" ;
  OutputStorage="/tmp";
  ListenerStorage = "/tmp"
  LoggingTimeout = 30 ;
  LoggingSyncTimeout = 45 ;
  LoggingDestination = "rb01.lip.pt:9002" ;
  DefaultStatusLevel = 1 ;
  DefaultLogInfoLevel = 0;
  NSLoggerLevel = 2;
  DefaultVo = "cms";
]
```

The files:

`$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_err.conf`

and

`$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_help.conf`

contain respectively the error codes and error messages returned by the UI and the text describing the commands usage.

2.2. BASIC OPERATION

The first example shows how to list the testbed resources that match the job requirements without actually submitting the job.

```
$ cat > test1.jdl <<EOF
Executable           = "myscript.pl";
VirtualOrganisation = "cg";
StdOutput            = "test1.out";
StdError             = "test1.err";
InputSandbox         = {"test1.pl"};
OutputSandbox        = {"test1.out", "test1.err"};
EOF

$ edg-job-list-match test1.jdl
```

The example shows first a simple JDL specification to execute a Perl script named *myscript.pl*. The output is written in a file named *test1.out* while the errors are sent to *test1.err*. The input sandbox contains the files that must be transferred to the remote site prior to execution while the output sandbox contains the files that must be transferred back at the end. In this case the Perl script is transferred to the remote site before the execution and the output and error files are transferred back in the end.

To actually submit the job the command *edg-job-submit* is used. If successful in transferring the job to the RB the command will print a job identifier in the form of an URL.

```
$ edg-job-submit test1.jdl
```

The job identifier is required to obtain information about the job, retrieve the job output, cancel the job or remove the output sandbox files produced.

```
$ edg-job-status \
https://rb01.lip.pt:9000/IRYtzJTrTPlboJamGtdV6A
```

A typical *edg-job-status* is shown below.

```
*****
BOOKKEEPING INFORMATION:

Printing status info for the Job :
https://rb01.lip.pt:9000/IRYtzJTrTPlboJamGtdV6A
Current Status: Running
Status Reason: Job successfully submitted to Globus
Destination: ce02.lip.pt:2119/jobmanager-pbs-short
reached on: Mon Nov 10 00:19:44 2003
*****
```

When the job finishes and reaches the **DONE** state the output sandbox files can be transferred from the RB to the UI using the command *edg-job-get-output*. The files are usually transferred to a directory under */tmp*.

```
$ edg-job-get-output \
https://rb01.lip.pt:9000/IRYtzJTrTPlboJamGtdV6A
```

To cancel a submitted job the user must issue the command *edg-job-cancel* with the job id as argument.

```
$ edg-job-cancel \  
https://rb01.lip.pt:9000/IRYtzJTrTPlboJamGtdV6A
```

2.3. ADVANCED FEATURES

2.3.1. Multiple CEs selection

One of the new features is the selection of multiple CEs for submitting an mpi job to the selected resources.

1.- What we need first is a JDL file defining the mpi job that we want to submit to the resource broker. This jdl file shall contain the specifications and requirements of the job, and also the new fields that we have established to correctly define the mpi jobs.

These new fields are:

| | |
|----------------|---|
| JobType | Field that defines that is an mpi job. Possible values are: |
| | mpich - defines a mpich-p4 job |
| | mpich-g2 - defines a mpich-g2 job |
| | normal - (default) common sequential job |

NodeNumber Field that defines the required number of cpus to execute the mpi job

For testing purposes we show one jdl file, for example:

```
VirtualOrganisation = "cg";  
Executable = "mpi_app";  
JobType = "mpich-g2";  
NodeNumber = 10;  
Arguments = "-n";  
StdOutput = "std.out";  
StdError = "std.err";  
Requirements = other.GlueCEInfoLRMSType=="pbs";  
Rank = other.GlueHostBenchmarkSI00;  
InputSandbox = {"mpi_app"};  
OutputSandbox= {"std.out", "std.err"};
```

We can see that in this description we are looking for groups of CEs whose queue type is PBS and that have at least 10 free CPUs counting all the involved CEs in the group, to run our mpi job.

2.-Next we will submit the jdl file to the modified RB that supports this new syntax in the jdl file. The command to get the available CEs would be, as usual:

edg-job-list-match file.jdl

```

*****
                                GROUPS OF CE IDs LIST
The following groups of CE(s) matching your job requirements have been found:
                                *Groups with 1 CEs*                                *TotalCPUs* *FreeCPUs*
[Rank=650]
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-infinite                10                10
[Rank=650]
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-long                   10                10
[Rank=650]
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-short                 10                10
[Rank=630]
cluster.ui.sav.sk:2119/jobmanager-pbs-workq                    16                16
[Rank=400]
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-infinite               58                57
[Rank=400]
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-long                  58                57
[Rank=400]
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-short                 58                57

                                *Groups with 2 CEs*                                *TotalCPUs* *FreeCPUs*
[Rank=440 TotalCPUs=12 FreeCPUs=12]
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite                4                 4
ce100.fzk.de:2119/jobmanager-pbs-long                           8                 8
[Rank=498 TotalCPUs=10 FreeCPUs=10]
ce01.lip.pt:2119/jobmanager-pbs-infinite                        2                 2
ce100.fzk.de:2119/jobmanager-pbs-long                           8                 8
[Rank=433.6 TotalCPUs=10 FreeCPUs=10]
ce100.fzk.de:2119/jobmanager-pbs-long                           8                 8
cg01.ific.uv.es:2119/jobmanager-pbs-infinite                    2                 2

                                *Groups with 4 CEs*                                *TotalCPUs* *FreeCPUs*
[Rank=435.6 TotalCPUs=10 FreeCPUs=10]
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite                4                 4
ce01.lip.pt:2119/jobmanager-pbs-infinite                        2                 2
cg01.ific.uv.es:2119/jobmanager-pbs-infinite                    2                 2
cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite                 2                 2
[Rank=500 TotalCPUs=12 FreeCPUs=10]
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite                4                 4
ce01.lip.pt:2119/jobmanager-pbs-infinite                        2                 2
cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite                 2                 2
cms.fuw.edu.pl:2119/jobmanager-pbs-infinite                     4                 2
*****

```

The obtained command output contains a list of resources or groups of resources where to execute the parallel job. Such list is composed of:

- a) Groups of elements that contain only 1 CE, so the job could be submitted to just one CE or cluster. This is the best desirable situation. We have 7 groups of 1 CE, being the first best resource is:

```
*Groups with 1 CEs*                *TotalCPUs* *FreeCPUs*

[Rank=650]
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-infinite    10    10
```

The CE **ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-infinite** has all 10 free CPUs and a global rank (based on the SI00 of that CE) of 650. This resource will be the first selected by the Application Scheduler.

As it can be seen, the next resources that would be selected would be:

```
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-long
ce001.grid.ucy.ac.cy:2119/jobmanager-pbs-short
cluster.ui.sav.sk:2119/jobmanager-pbs-workq
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-infinite
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-long
zeus24.cyf-kr.edu.pl:2119/jobmanager-pbs-short
```

- b) After single CEs, groups of CEs that fulfill the requirements are formed. In this case we find 11 groups with 2 CEs suitable for executing our job. The best one, according with the rank is:

```
*Groups with 2 CEs*                *TotalCPUs* *FreeCPUs*

[Rank=650 TotalCPUs=10 FreeCPUs=10]
ce01.lip.pt:2119/jobmanager-pbs-infinite            2    2
xgrid.icm.edu.pl:2119/jobmanager-pbs-infinite       8    8
```

This group is the one with more computed Rank from the groups with 2 CEs. It has also its 10 needed cpus free (from a total of 10) and a computed rank of 650. It is again calculated with the weighted rank of every CE of the group (**ce01.lip.pt:2119/jobmanager-pbs-infinite** with a rank of 650, and **xgrid.icm.edu.pl:2119/jobmanager-pbs-infinite** also with rank 650 makes the weighted rank of the same 650, no matter how many cpus are contributing with).

We can see another meaningful example in the group:

```
[Rank=440 TotalCPUs=12 FreeCPUs=12]
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite    4    4
ce100.fzk.de:2119/jobmanager-pbs-long               8    8
```

This group formed by 2 CEs from **tcd** and **fzk** have a total number of 12 CPUS, from where all of them are free (without running jobs). This group fulfils the requirements, having also at least the cpus wanted.

The computed rank of 440 is not the average of the ranks of the components (400 and 460 that would be 430) but the weighted rank calculated considering the number of free cpus of each component.

c) As we can see there are no possible groups with 3 CEs that group the required number of cpus that we are asking for, and that are not taken into account in the previous groups (with 1, or 2 CEs). However we can find 4 groups with 4 CEs each one, and this is the biggest that we can from with the CEs in the testbed and the situation in the moment that we are asking. The best group would be:

| *Groups with 4 CEs* | *TotalCPUs* | *FreeCPUs* |
|--|-------------|------------|
| [Rank=500 TotalCPUs=12 FreeCPUs=10] | | |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite | 4 | 4 |
| ce01.lip.pt:2119/jobmanager-pbs-infinite | 2 | 2 |
| cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite | 2 | 2 |
| cms.fuw.edu.pl:2119/jobmanager-pbs-infinite | 4 | 2 |

This group if formed by CEs from **tcd**, **lip**, **uoa** and **fuw**, collecting 10 free cpus from a total of 12. The original Ranks from every site are (not shown in the logs):

| | |
|--|------------------------|
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite | 400 (contributes 4/10) |
| ce01.lip.pt:2119/jobmanager-pbs-infinite | 650 (contributes 2/10) |
| cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite | 400 (contributes 2/10) |
| cms.fuw.edu.pl:2119/jobmanager-pbs-infinite | 650 (contributes 2/10) |

Giving the weighted computed rank of 500.

For a more detailed explanation of how the process of resource selection is carried, document [2] can be consulted.

Finally, the command used to submit such file is:

```
edg-job-submit file.jdl
```

The output of the command will give the following results:

Connecting to host aow5grid.uab.es, port 7772

Logging to host aow5grid.uab.es, 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://aow5grid.uab.es:9000/jR0hjTzOlyFkRkpP_i1R8Q

This output indicates that the job has been sent to the RB and now the user must check its status using the `edg-job-status` command. The job will pass through three different states: Waiting, Running and Done. If there is any kind of error during the execution of the job and the application launcher is not able to get the checkins from all the subjobs, the job will be aborted and the `edg-job-status` command will show an output like this:

BOOKKEEPING INFORMATION:

Printing status info for the Job :

<https://aorbgrid.uab.es:9000/PQBDcaLSUaDCAEJooBtWag>

Current Status: Aborted

Status Reason: Could not receive all checkins from subjobs

Destination: ce.grid.cesga.es:2119/jobmanager-pbs-infinite

reached on: Thu Mar 25 12:12:36 2004

When the job has finished, the user can get the output using the command `edg-job-get-output`, this command output is depicted next:

Retrieving files from host aow5grid.uab.es

JOB GET OUTPUT OUTCOME

Output sandbox files for the job:

- https://aow5grid.uab.es:9000/jR0hjTzOlyFkRkpP_i1R8Q

have been successfully retrieved and stored in the directory:

`/tmp/jobOutput/jR0hjTzOlyFkRkpP_i1R8Q`

In the directory specified by the `edg-job-get-output` can be found the output and error files of each subjob of the application

2.3.2. Interactive MPICH-P4 and MPICH-G2 Jobs support

Another new feature is the support of interactive `mpich-p4` and `mpich-g2` jobs. The source code of interactive programs for the `mpich-p4` and `mpich-g2` testing can be downloaded from cvs at:

- http://savannah.fzk.de/cgi-bin/viewcvs.cgi/CrossGrid/CrossGrid/wp3/wp3_2-scheduling/etc/tests/interactive-tests/

1. First of all, we need to define this feature in the job descriptor file, to this purpose we will use the `JobType` field. In order to do this we will write one of the next attributes:

| | | |
|----------------|-------------------------------|--|
| JobType | { "interactive", "mpich" } | - Defines an interactive <code>mpich-p4</code> job |
| | { "interactive", "mpich-g2" } | - Defines an interactive <code>mpich-g2</code> job |

We can see that this field uses the same attributes than the interactive jobs and `mpich-p4/mpich-g2` jobs, the difference resides that now we use them simultaneously.

Note#1: In the JDL file, we must specify both the fields which define an `mpich-p4/g2` job and the ones dealing with interactivity. e.g.: `ListenerPort` for an interactive Job and `NodeNumber` for an `mpich` job.

Note#2: As in interactive jobs we must not define `OutputSandbox`, `StdOutput` and `StdError` attributes.

For testing this new feature we show one jdl file, for example:

```
VirtualOrganisation = "cg";
Type                = "Job";
Executable          = "interactive_mpich-g2_app";
JobType             = { "interactive", "mpich-g2" };
NodeNumber          = 10;
ListenerPort        = 24100;
Arguments           = "-n";
FuzzyRank           = true;
InputSandBox        = {"interactive_mpich-g2_app"};
Requirements        = - other.GlueCEStateEstimatedResponseTime;
Rank                = other.GlueCEStateStatus == "Production";
```

2. Next, we submit the jdl file using the modified UI to the modified RB that supports this new feature.

edg-job-submit file.jdl

The output of the command will give the following results:

```
Selected Virtual Organisation name (from JDL): cg
Connecting to host aorbgrid.uab.es, port 7772
Logging to host aorbgrid.uab.es, 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://aorbgrid.uab.es:9000/IAYUQS7E6J4aySd3bjImVQ

---

The Interactive Session Listener has been successfully launched
with the following parameters:

Host:                                aouigrid.uab.es
Port:                                24501

*****

*****
Interactive Job console started for
https://aorbgrid.uab.es:9000/IAYUQS7E6J4aySd3bjImVQ
Please press ^C to exit from the session
*****
```

This output indicates that the job has been sent to the RB and now the user must wait for the beginning of the job execution.

When the job is running, if it is an interactive mpich-g2 job, the output should be similar to the following:


```

Job Status: Aborted (Some subjobs got an error while in the
CondorG queue)
*****
Interactive Session was aborted.
Removing Listener and input/output streams...
Done
Press <enter> to go to prompt
*****

```

However, if the user cancelled the job, by pressing ctrl+c, appears the next message in the console:

```

*****
Interactive Session ended by user.
Removing Listener and input/output streams...
Done
Press <enter> to go to prompt
*****

```

If the job was cancelled, by the user, the job must be removed using the command *edg-job-cancel*, but if the job was not executing in the remote host still, it can be restored with *the edg-job-attach* command:

```

edg-job-attach https://aorbgrid.uab.es:9000/ta84GWSCp0qnuTH3g2N8yQ

*****
JOB ATTACHED:
The Interactive Session Listener has been successfully launched
with the following parameters:
---
Host:                aouigrid.uab.es
Port:                24501
*****

*****
Interactive Job console started for
    https://aorbgrid.uab.es:9000/ta84GWSCp0qnuTH3g2N8yQ
Please press ^C to exit from the session
*****

```

From then on, the job will continue its execution exactly as if the *edg-job-submit* command had been used.

2.3.3. Monitoring Plug-in Support

We have developed an interface for integrating the monitoring tools developed in the Crossgrid project to be used by the final user. The basic ideas for the APIs and the working mode can be found in the document [9].

Currently this API has been implemented by the Postprocessing tool and has been integrated in the latest release of our packages.

For installing the support for this, the related rpm packages that are modified are described in the installation guide [4].

For the final user, these monitoring tools can be tested using the provided functions in the Rank sections of the JDL of their jobs. The final specification of these functions, and the related parameter and metrics used, will be provided by the particular tool plugin.

Currently we only could use the information provided by the Information System, for example the default ranking of resources is made in terms of the Response Time (the less response time, the first ranking positions):

```
rank = - other.GlueCEStateEstimatedResponseTime;
```

Many other values and complex expressions can be crafted to design expression for ranking with the information provided by the II.

- With the new functionality for integrating the monitoring tools, and using the mechanism provided by the specification of the JDL we can include the calls to the monitoring tools in the “Rank” section. so for example if we have a monitoring tools that gives a new value like the “available bogomips” we could have a rank expression like this:

```
rank = other. GlueCEStateFreeCPUs *  
ppt_getClusterParameter(“max”, “idle_bogomips”, other.GlueCeUniqueId, 0)
```

This is an expression that ranks the Ces according to its free CPUs multiplied by the value “idle_bogomips” given by the postprocessing monitoring plugin

- With the use of matchmaking and ranking of groups (for mpich-g2 jobs, for example), more complex functionality is possible. If for example we have available information about the bandwidth of a set of resources we could rank with these kinds of expressions:

```
rank = monitoring_api(“bandwith”, other.groupGlueCEUniqueId );
```

This expression will give first the groups of Ces with more aggregated bandwidth.

We have introduced a new value to refer to the group of Ces in which the rank is evaluated, and is **groupGlueCEUniqueId**. This value refers to a list of Ces, and can be used in user functions of the monitoring tools that give values depending of the group of Ces that is being evaluated like is the case of the bandwidth for example.

The ranking procedure is slightly different for groups of CEs, and its internals can be seen also in the related document [9].

As a simple example this JDL can be provided to test the integration of the postprocessing monitoring plugin, which has been already implemented:

```

VirtualOrganisation = "cg";
Type                = "Job";
Executable          = "interactive_mpich-g2_app";
JobType             = { "interactive", "mpich-g2" };
NodeNumber         = 10;
ListenerPort       = 24100;
Arguments          = "-n";
FuzzyRank          = true;
InputSandBox       = {"interactive_mpich-g2_app"};
Requirements       = - other.GlueCEStateEstimatedResponseTime;
Rank               =
ppt_getClusterParameter("max", "idle_bogomips", other.GlueCeUniqueId, 0) *
ppt_getNetworkParameter("avg", "bandwidth", other.groupGlueCeUniqueId, 0);

```

Additional examples may be found in the ppt user guide.

Now we run a test from the user interface to see the results with different examples. The exact results of course depend on many factors like the user rank expressions and the testbed status for example:

1. First we run a job-list-match with a simple jdl to see what are the resources available

```
bash-2.05a$ edg-job-list-match hostname.jdl
```

```

Selected Virtual Organisation name (from UI conf file): cg
Connecting to host rb.fzk.de, port 7772

```

```

*****
                        COMPUTING ELEMENT IDs LIST
The following CE(s) matching your job requirements have been found:

      *CEId*
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-long
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-short
ce010.fzk.de:2119/jobmanager-pbs-long
ce010.fzk.de:2119/jobmanager-pbs-medium
ce010.fzk.de:2119/jobmanager-pbs-short
grid14.physics.auth.gr:2119/jobmanager-pbs-infinite
grid14.physics.auth.gr:2119/jobmanager-pbs-long
grid14.physics.auth.gr:2119/jobmanager-pbs-short
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-long
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-short
*****

```

We can see that we have 4 different CEs with their respective queues available

2. Now we want to test the ppt monplugin with a simple expression, ranking the resources according to the “idle bogomips” value provided by this tool. The jdl would be the next one:

```
Executable      = "/bin/hostname";
Arguments       = "";
StdOutput       = "std.out";
StdError        = "std.err";
OutputSandbox  = {"std.out","std.err"};
Rank = ppt_getClusterParameter("max","idle_bogomips",other.GlueCEUniqueID,0);
```

We obtain the list of available ranked CEs:

```
bash-2.05a$ edg-job-list-match --rank
../hostname_postproc_get_likeothergroupCeId.jdl
```

```
Selected Virtual Organisation name (from UI conf file): cg
Connecting to host rb.fzk.de, port 7772
```

```
*****
                        COMPUTING ELEMENT IDs LIST
The following CE(s) matching your job requirements have been found:

                *CEId*                                *Rank*
ce010.fzk.de:2119/jobmanager-pbs-long                86603.9
ce010.fzk.de:2119/jobmanager-pbs-medium             86603.9
ce010.fzk.de:2119/jobmanager-pbs-short              86603.9
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite 24971.8
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-long     24971.8
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-short    24971.8
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite    0
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-long        0
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-short       0
*****
```

We observe that the CE *grid14.physics.auth.gr* is not in the list, probably because the ppt monplugin did not have the information about this particular CE and cannot evaluate the rank expression requested. This is not the same *cagnode45.cs.tcd.ie*, whose value for idle bogomips is 0.

The list of the CEs is listed by their values, in descending order.

3. In the next example we can see the use of **other.GroupGlueCEUniqueID** in the context of an mpich-g2 job, which can run across sites.

The rank expression is slightly modified using this attribute, and we add a second term to test another function of the plugin that gives values of the network, in particular the bandwidth among pairs of resources in this example.

```

Executable      = "/bin/hostname";
JobType         = "mpich-g2";
NodeNumber      = 4;
Arguments       = "";
StdOutput       = "std.out";
StdError        = "std.err";
OutputSandbox   = {"std.out","std.err"};
Rank = ppt_getClusterParameter("max","idle_bogomips",other.GroupGlueCEUniqueID,0) +
ppt_getNetParameter("avg","bandwidth",other.GroupGlueCEUniqueID,0) ;
    
```

Results are the following...

Selected Virtual Organisation name (from UI conf file): cg
 Connecting to host rb.fzk.de, port 7772

GROUPS OF CE IDS LIST

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

| *Groups with 1 CEs* | *TotalCPUs* | *FreeCPUs* |
|---|-------------|------------|
| [Rank=86603.9] | | |
| ce010.fzk.de:2119/jobmanager-pbs-long | 16 | 16 |
| [Rank=86603.9] | | |
| ce010.fzk.de:2119/jobmanager-pbs-medium | 16 | 16 |
| [Rank=86603.9] | | |
| ce010.fzk.de:2119/jobmanager-pbs-short | 16 | 16 |
| [Rank=24971.8] | | |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite | 8 | 8 |
| [Rank=24971.8] | | |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-long | 8 | 8 |
| [Rank=24971.8] | | |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-short | 8 | 8 |
| [Rank=0] | | |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite | 6 | 6 |
| [Rank=0] | | |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-long | 6 | 6 |
| [Rank=0] | | |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-short | 6 | 6 |

Although this one is mpich-g2 job, we see that the rank numbers are the same as before because groups only have 1 CE. (we asked for 4 cpus and all CEs have it at least 4 free)

other.GroupGlueCEUniqueID contains the list of CEid for groups, but in this case will only contain the unique CEid as before.

The second part of rank expression (**ppt_getNetParameter**) will be always 0 as there is no bandwidth pairs to calculate.

4. Now we want to test the **GroupGlueCEUniqueID** attribute in the context of normal jobs. The rank expression is the same as the last example

```
Executable      = "/bin/hostname";
Arguments       = "";
StdOutput       = "std.out";
StdError        = "std.err";
OutputSandbox   = {"std.out", "std.err"};
Rank = ppt_getClusterParameter("max", "idle_bogomips", other.GroupGlueCEUniqueID, 0) +
ppt_getNetParameter("avg", "bandwidth", other.GroupGlueCEUniqueID, 0) ;
```

This jdl will give the following results:

```
Selected Virtual Organisation name (from UI conf file): cg
Connecting to host rb.fzk.de, port 7772
```

```
*****
```

COMPUTING ELEMENT IDs LIST

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

| *CEid* | *Rank* |
|---|---------|
| ce010.fzk.de:2119/jobmanager-pbs-long | 86603.9 |
| ce010.fzk.de:2119/jobmanager-pbs-medium | 86603.9 |
| ce010.fzk.de:2119/jobmanager-pbs-short | 86603.9 |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite | 24971.8 |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-long | 24971.8 |
| gtbcg01.ifca.unican.es:2119/jobmanager-pbs-short | 24971.8 |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite | 0 |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-long | 0 |
| cagnode45.cs.tcd.ie:2119/jobmanager-pbs-short | 0 |

```
*****
```

Results are same as before because:

other.GroupGlueCEUniqueID contains the list of CEid for groups, but in this case (not mpich-g2 job) will only contain the unique CEid as before.

The second part of rank expression (**ppt_getNetParameter**) will be always 0 as there is no bandwidth pairs to calculate.

In this example we show that we still can use the attribute **other.GroupGlueCEUniqueID** for normal jobs not requiring groups of CEs, but it will give the same result as using **other.GlueCEUniqueID**.

We can also see that the value provided for the second term of the expression is 0 because it should be used for groups of CEs.

5. In the last example we show how the rank is done for groups in the mpich-g2 jobs. The example is the same as the previous mpich-g2 job but requesting 10 cpus, in order to show some group.

```
Executable      = "/bin/hostname";
JobType         = "mpich-g2";
NodeNumber      = 10;
Arguments       = "";
StdOutput       = "std.out";
StdError        = "std.err";
OutputSandbox   = {"std.out","std.err"};
Rank = ppt_getClusterParameter("max","idle_bogomips",other.GroupGlueCEUniqueID,0) +
ppt_getNetParameter("avg","bandwidth",other.GroupGlueCEUniqueID,0) ;
```

Results are the following...

```
Selected Virtual Organisation name (from UI conf file): cg
Connecting to host rb.fzk.de, port 7772
```

```
*****
                        GROUPS OF CE IDs LIST
The following groups of CE(s) matching your job requirements have been found:

                *Groups with 1 CEs*                                *TotalCPUs* *FreeCPUs*
[Rank=86650.5]
ce010.fzk.de:2119/jobmanager-pbs-long                        16           16
[Rank=86650.5]
ce010.fzk.de:2119/jobmanager-pbs-medium                    16           16
[Rank=86650.5]
ce010.fzk.de:2119/jobmanager-pbs-short                     16           16

                *Groups with 2 CEs*                                *TotalCPUs* *FreeCPUs*

[Rank=24943.5 TotalCPUs=14 FreeCPUs=14]
cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite            6             6
gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite         8             8

*****
```

Now we are asking for 10 cpus and we have these results. Groups with 1 CE are the ones that fulfil the cpu requirements by themselves. In this case rank is not the same (the monitoring plugin part updated the cached data, and it is slightly different).

There is one available group with 2 CEs, and in this case the attribute **GroupGlueCEUniqueID** contains the CEid of the group. This is evaluated in the context of each group, so in the context of the evaluation of the rank of the group with 2 CEs:

GroupGlueCEUniqueID =

```
{ "cagnode45.cs.tcd.ie:2119/jobmanager-pbs-infinite", "gtbcg01.ifca.unican.es:2119/jobmanager-pbs-infinite" };
```

Now the second part of the expression that calculates bandwidth will not be 0, because the set has 2 CEs and the evaluation is done correctly by the `ppt_getNetParameter` function with this attribute.

It has to be noted that this is also the value of the attribute when internally calling the first function `ppt_getClusterParameter`. In this case will give the max value for the idle bogomips of the 2 CEs. We could use the attribute **GlueCEUniqueID** which would be evaluated in the context of every CE of the group with slightly different rank values.

Moreover Rank is weighted by their free cpus as explained in the document [9].

2.3.4. DAG support

A directed acyclic graph (DAG) is a directed graph (a graph with one-way edges) that does not contain cycles. This means that if there is a route from node A to node B then there is no way back.

A DAG can be used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies.

An example DAG is presented on the Figure 1. It consists of 5 nodes laying on 3 levels.

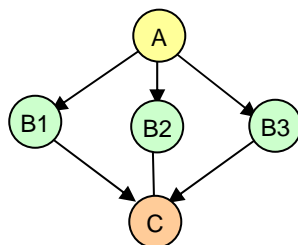


Figure 1. DAG.

The execution of the presented example DAG consists of three successive steps:

1. Execution of the first node (A) from the first level.
2. Parallel execution of three nodes (B1, B2, B3) from the second level. The execution can start if and only if the execution of the node from the level one (on which these nodes depend) is successful.
3. Execution of the last node (C) from the last (third) level. The execution can start if and only if the execution of all nodes from the level two (on which this node depends) is successful.

The DAG specification is based on the description presented in [5]. The implementation of DAG support is compatible with the prototype implementation available in [6] extending its functionalities. The mechanism that provides the execution and control of DAGs is called DAGMan (DAG Manager). This mechanism is based on the DAGMan software provided by Condor group [7] and is targeted for EDG.

DAGMan can be treated as an iterator on the DAG (graph of jobs) whose main purpose is to navigate through the graph (node by node), determine which nodes are free of dependencies, and follow the execution of corresponding jobs.

A DAG is considered as a single job unit of work and is represented in the Logging and Bookkeeping Service. A DAG can be managed as a normal single job, and additionally there are certain functionalities that can be performed on the nodes that form a DAG.

2.3.4.1. DAG specification

A DAG is written in the same Job Description Language (JDL) as in the case of normal single jobs. The specification of a DAG consists of the list of job descriptions. Each DAG must contain the attribute *type* set to *dag*. Then, the attribute *nodes* contain the specification of all nodes that form a DAG and dependencies between them. A node is specified using the usual attributes characteristic for a normal single job.

An example DAG is presented in Figure 2. Below we provide a JDL file (*dag.jdl*) that specifies a given DAG:

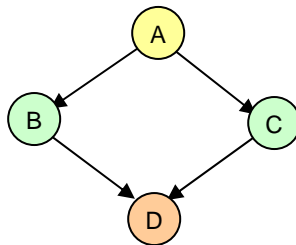


Figure 2. Example DAG consisted of 5 nodes.

```

[
  /* DAG that consists of 4 nodes
      A
      / \
     B  C
      \ /
      D
  */
  type = "dag";
  nodes = [
    dependencies = { {A, B}, {A, C}, {{B, C}, D} };
    A = [
      node_type = "edg-jdl";
      description = [
        Executable = "a.sh";
        StdOutput = "std.out";
        StdError = "std.err";
        InputSandbox = {"a.sh"};
        OutputSandbox = {"std.out", "std.err"};
      ];
    ];
  ];

```

```
B = [  
    node_type = "edg-jdl";  
    node_retry_count = 3;  
    file = "jobB.jdl";  
];  
C = [  
    node_type = "edg-jdl";  
    node_retry_count = 3;  
    description = [  
        Executable = "c.sh";  
        InputSandbox = {"c.sh"};  
        StdOutput = "std.out";  
        StdError = "std.err";  
        OutputSandbox = {"std.out", "std.err"};  
        job_exit_codes = { 10, 11 };  
    ];  
];  
D = [  
    node_type = "edg-jdl";  
    node_retry_count = 2;  
    description = [  
        Executable = "d.exe";  
        InputSandbox = {"d.exe"};  
    ];  
];  
]
```

2.3.4.2. Dependencies Between Nodes

Considering the example DAG there are few possibilities to specify the attribute *dependencies*:

- `dependencies = { {A, B}, {A, C}, {{B, C}, D} };`
B depends on A, C depends on A, D depends on B and C
- `dependencies = { {A, {B, C}}, {{B, C}, D} };`
B and C depends on A, D depends on B and C
- `dependencies = { {A, B}, {A, C}, {B, D}, {C, D} };`
B depends on A, C depends on A, D depends on B, D depends on C

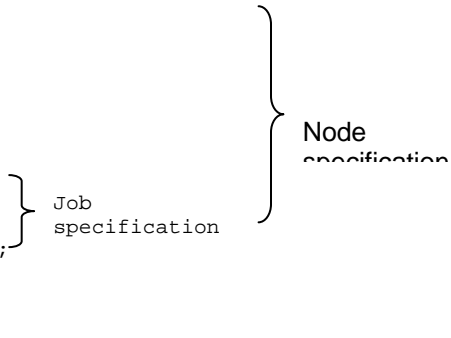
2.3.4.3. Node Description

The attribute *nodes* contain the list of nodes that form the considered DAG. Each node represents a job to be executed. An example node can be described as follows:

```

A = [
    node_type = "edg-jdl";
    node_retry_count = 3;
    app_exit_code = {10, 2};
    description = [
        Executable = "a.sh";
        InputSandbox = {"a.sh"};
    ];
];

```



Each node consists of general attributes, as well as the job specification. General attributes for a node are:

- node_type
- node_retry_count
- app_exit_code
- description / file

The type of the node (the attribute called *node_type*) must contain the value “edg-jdl” since the node is a normal job written in jdl. This attribute is mandatory.

Each node has the possibility to retry its execution when it fails. For that purpose the user can specify a value of the attribute called *node_retry_count*. If the user specifies this attribute, hence this particular node will be automatically retried in the case of failure. If this value is not specified (this attribute is optional), each node by default is retrying till 10 times (this value is configurable – by the administrator – in the file {EDG_WL_LOCATION}/etc/edg_wl.conf in the section JConfiguration, field DagmanRetry).

If a node fails because of the application failure (e.g. segmentation fault caused by division by 0), then a job should be aborted. However, when a job fails because some resources fail (machine has downed or queue is full) it should be retrying automatically. By default in both cases the node will be retried till *node_retry_count*. Therefore, the user has the possibility to set an attribute that controls the exit code of the job. The attribute is called *app_exit_code* and is specified as follows:

```
app_exit_codes = { 10, 11 };
```

If this attribute is set, it means that the node may return these values that for a user are well and do not cause the DAG failure. If the job returns one of the specified values, the node will not be retried anymore. If the job exit value is different from these values it implies the node failure and if possible (*node_retry_count* value does not reach the maximum of retries) its repetition. This attribute is optional but if given, it must be set inside the job specification.

There are two possibility to specify a job:

1. attribute called *description* where a job is specified directly inside this attribute in JDL, for example:

```
description = [  
    Executable = "a.sh";  
    StdOutput = "std.out";  
    StdError = "std.err";  
    InputSandbox = {"a.sh"};  
    OutputSandbox = {"std.out", "std.err"};  
  
];
```

2. attribute called *file* where a job is specified in the indicated file JDL.

```
file = "jobA.jdl";
```

jobA.jdl may contain for example:

```
Executable = "a.sh";  
StdOutput = "std.out";  
StdError = "std.err";  
InputSandbox = {"a.sh"};  
OutputSandbox = {"std.out", "std.err"};
```

A job can be a normal single job or MPI job. For example mpi.jdl file that specifies the mpi job can be written as follows:

```
Type = "Job";  
JobType = "MPICH";  
Executable = "menuMPI";  
NodeNumber = 2;  
InputSandbox = {"menuMPI"};  
StdOutput = "menuMPI.out";  
StdError = "menuMPI.err";  
OutputSandbox = {"menuMPI.out", "menuMPI.err"};  
VirtualOrganisation = "cg";  
Rank = other.GlueCEStateFreeCPUs;
```

2.3.4.4. Submitting DAGs

DAGs are submitted using the `edg-dag-submit-command`, for example:

```
edg-dag-submit dag.jdl
```

The output of the command will give the following results:

```
Selected Virtual Organisation name (from UI conf file): cg
Connecting to host aorbgrid.uab.es, port 7772
Logging to host aorbgrid.uab.es, port 9002
```

```
*****
                                JOB SUBMIT OUTCOME

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

- https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
```

When the DAG has been submitted, the daemon of DAGMan is executed. This daemon is treated as a parent of all the nodes from the DAG and manages the execution of all nodes of the DAG. DAGMan submits jobs to Condor in an order represented by the DAG and processes the results.

Job status can be checked using the `edg-job-status` command as with other kind of jobs:

```
edg-job-status <dag_id>
```

Having `dag_id` returned by the submission command, the user can type the status command in the following manner:

```
edg-job-status https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
```

The output of the command will give such results:

```
*****
BOOKKEEPING INFORMATION:

Status info for the Job : https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
Current Status:      Running
Status Reason:      unavailable
Destination:        dagman
reached on:         Tue May 25 09:23:04 2004
*****

- Nodes information for:
  Status info for the Job : https://aorbgrid.uab.es:9000/kBDo0SHPyHu_03hPcGZppg
  Current Status:      Waiting
  reached on:          Tue May 25 09:23:00 2004
  Parent Job:          https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
  Status info for the Job : https://aorbgrid.uab.es:9000/0RNry99KrDPNq8f64dbMyg
  Current Status:      Submitted
  reached on:          Tue May 25 09:22:35 2004
  Parent Job:          https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
  Status info for the Job : https://aorbgrid.uab.es:9000/vXYKlKdQtgObwZ4fPne90A
  Current Status:      Submitted
  reached on:          Tue May 25 09:22:35 2004
  Parent Job:          https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
  Status info for the Job : https://aorbgrid.uab.es:9000/fEXlvWI3xwp9saWEAi-Q3g
  Current Status:      Submitted
  reached on:          Tue May 25 09:22:35 2004
  Parent Job:          https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
  Status info for the Job : https://aorbgrid.uab.es:9000/yte6MVinUx2tykhRLdMFVw
  Current Status:      Submitted
  reached on:          Tue May 25 09:22:35 2004
  Parent Job:          https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww
*****
```

The output presents two types of information:

- about the submitted DAG that is treated as a parent job of all nodes
- the list of information for each node of the DAG

It is also possible to receive information about a particular node status:

edg-job-status <job_id>

For example for the first node of the DAG the command and the output will be as follow:

```
*****  
BOOKKEEPING INFORMATION:  
  Status info for the Job : https://aorbgrid.uab.es:9000/kBDo0SHPyHu_03hPcGZppg  
  Current Status:      Submitted  
  reached on:         Tue May 25 09:30:00 2004  
  Parent Job:        https://aorbgrid.uab.es:9000/FurH84XdFgIz00BEfxSIww  
*****
```

When a node fails DAGMan retries its execution according to the *node_retry_count* and *app_exit_code* value set by the user in the node specification. If a job is failed, DAGMan automatically tries to execute this job from the beginning but, if possible, the job will be automatically executed in other computing element (CE). It gives more probability to execute the job successfully, if the resources, where it was just executed, failed. If the user specifies CE in the job requirements hence DAGMan does not choose other site to execute the job but it repeats given CE according to the requirements. If a node available repetition was exhausted, DAGMan leaves this node as failed and does not try to execute it any more. In the case of a node failure, DAGMan continues the execution of nodes till it is possible (till there are no dependencies on the failed nodes). However if some nodes depend on the failed node, and there are no nodes free of dependencies on the failed nodes, DAGMan stops the execution of the whole DAG and returns error.

If a job specification contains OutputSandbox field it is possible to obtain the files generated by it. To get output files generated by all nodes of the DAG the user can type the command:

edg-job-get-output <dag_id>

This command gets output for all jobs of the DAG, except the DAG itself (parent job), as we cannot specify output for a DAG. Output is obtained only for a DAG with status Done. If status of a DAG is Submitted or Cancelled, this command will give an error. If a DAG consists of 2 nodes, where the execution of the second node depends on the first node, it may happen that although retrying the execution of the first node specified times, the node has failed. The second node is in the Submitted state and DAGMan has finished with the exit code different than zero. The status will be as follows:

```
*****  
BOOKKEEPING INFORMATION:  
Status info for the Job : https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw  
Current Status:      Done (Exit Code !=0)  
Exit code:          1  
Status Reason:      Warning: job exit code != 0  
Destination:        dagman  
Submitted:          Fri Jul 16 10:20:49 2004 CEST  
*****
```

- Nodes information for:

```
Status info for the Job : https://aow6grid.uab.es:9000/VIddqSxz89Z0Am4gX_kA_A
Current Status:      Done (Exit Code !=0)
Exit code:          10
Status Reason:      Warning: job exit code != 0
Destination:        cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite
Submitted:          Fri Jul 16 10:20:50 2004 CEST
Parent Job:          https://aow6grid.uab.es:9000/2O\_FkF79m8gEYjAKTjlycw
```

```
*****
Status info for the Job : https://aow6grid.uab.es:9000/Q2Idnw2XZ_L0Up6nlrFOXA
Current Status:      Submitted
Submitted:           Fri Jul 16 10:20:50 2004 CEST
Parent Job:          https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw
*****
```

If a job has other status than Done, the getting output command will give an error for that particular job. However, it will get output for the rest of jobs of DAG.

```
edg-job-get-output https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw
```

```
Retrieving files from host: aow6grid.uab.es ( for https://aow6grid.uab.es:9000/VIddqSxz89Z0Am4gX_kA_A )
```

```
**** Error: NS_JOB_OUTPUT_NOT_READY ****
```

```
The OutputSandbox files for job
```

```
"https://aow6grid.uab.es:9000/Q2Idnw2XZ_L0Up6nlrFOXA"
```

```
are not yet ready for retrieval. Please wait that the job enters the "Done" status.
```

```
*****
```

```
JOB GET OUTPUT OUTCOME
```

```
OutputSandbox not set in submitted JDL. No output files to be retrieved for the job:
```

```
- https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw
```

```
Children files retrieved/purged:
```

```
- https://aow6grid.uab.es:9000/VIddqSxz89Z0Am4gX_kA_A
```

```
- https://aow6grid.uab.es:9000/Q2Idnw2XZ_L0Up6nlrFOXA not retrieved!
```

```
*****
```

After retrieving the output for the DAG described above, the status command will give the following results:

```
*****
BOOKKEEPING INFORMATION:
Status info for the Job : https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw
Current Status:      Cleared
Status Reason:      user retrieved output sandbox
Destination:        dagman
Submitted:          Fri Jul 16 10:20:49 2004 CEST
*****
```

- Nodes information for:

```
Status info for the Job : https://aow6grid.uab.es:9000/VIdDqSxz89Z0Am4gX_kA_A
Current Status:      Cleared
Status Reason:      user retrieved output sandbox
Destination:        cgnode00.di.uoa.gr:2119/jobmanager-pbs-infinite
Submitted:          Fri Jul 16 10:20:50 2004 CEST
Parent Job:         https://aow6grid.uab.es:9000/2O\_FkF79m8gEYjAKTjlycw
```

```
*****
Status info for the Job : https://aow6grid.uab.es:9000/Q2Idnw2XZ_L0Up6nlrFOXA
Current Status:      Submitted
Submitted:          Fri Jul 16 10:20:50 2004 CEST
Parent Job:         https://aow6grid.uab.es:9000/2O_FkF79m8gEYjAKTjlycw
*****
```

It is also possible to get output for a particular job:

edg-job-get-output <job_id>

3. INTERFACE REFERENCE GUIDE

A complete description of the available user commands is depicted in the DataGrid WP1 WMS Software Administrator and User Guide [1]. For the sake of completeness the description of each command is repeated.

3.1. EDG-JOB-SUBMIT

Allows the user to submit a job for execution on remote resources in a grid.

SYNOPSIS

```
edg-job-submit [options] <jdl_file>
```

Options:

```
--help
--version
--vo <vo_name>
--input, -i <file_path>
--resource, -r <ce_id>
--hours, -h <hours_number>
--chkpt <file_path>
--nolisten
--nogui
--nomsg
--config, -c <file_path>
--config-vo <file_path>
--output, -o <file_path>
--noint
--debug
--logfile <file_path>
```

DESCRIPTION

edg-job-submit is the command for submitting jobs to the DataGrid and hence allows the user to run a job at one or several remote resources. **edg-job-submit** requires as input a job description file in which job characteristics and requirements are expressed by means of Condor class-ad-like expressions. While it does not matter the order of the other arguments, the job description file *has to be* the last argument of this command.

The job description file given in input to this command is syntactically checked and default values are assigned to some of the not provided mandatory attributes in order to create a meaningful class-ad. The resulting job-ad is sent to the NS, which then forwards it to the WM, which via the RB/Matchmaker finds the job best matching resource (*match-making*) and then the JC submits the job to it.

Upon successful completion this command returns to the user the submitted job identifier *edg_jobId* (a string that identifies unambiguously the job in the whole EDG), generated by the User Interface, that

can be later used as a handle to perform monitor and control operations on the job (e.g. see **edg-job-status** described later in this document). The format of the *edg_jobId* is as follows:

```
https://Lbserver_address[:port]/unique_string
```

The *unique_string* is an md5 string computed taking into account the following information:

- IP of the User Interface machine,
- timestamp,
- process ID (more UI instances may occur on the same machine),
- sequence or just random number (if the User Interface submits jobs in batches and more than one per second can be submitted),

The final md5 sum is encoded using modified Base64 encoding (":" is used instead of "/") ensuring reasonable uniqueness and compactness of job IDs.

The structure of the *edg_jobId* that could appear in some way complex and not easily readable, has been conceived in order to ensure uniqueness and at the same time contain information that is needed by the components of the WMS to fulfil user requests.

The `--vo` option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this attribute has not been specified in the JDL, then the default VO specified in the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` (*DefaultVo* field) configuration file is considered. Otherwise an error is returned to the user.

With the deployment of VOMS, the VO information will be encoded in the user's proxy certificate and this will make the described behaviour change.

The `--resource` option can be used to target the job submission to a specific known resource identified by the provided Computing Element identifier *ce_id* (returned by **edg-job-list-match** described later in this document). The *CE identifier* is a string published in the IS (the *GlueCEUniqueID* field in the Glue schema) that univocally identifies a resource belonging to the Grid. The admitted format for *CEId* is:

```
<full-hostname>:<port-number>/jobmanager-<service>-<queue-name>
```

where *<service>* is for example *lsf*, *pbs*, *bqs*, *condor* but can also be a different string as it is freely set by the site administrator when the queue is set-up.

When the `--resource` option is specified, the WMS skips completely the match making process and directly submits the job to the requested CE. It is important to note that in this case the *BrokerInfo* file is not generated even if data requirements have been specified in the JDL, so jobs submitted using this option should not rely on the *BrokerInfo* file information when running on the CE. The *BrokerInfo* file is a file generated by the RB/Matchmaker during matchmaking and contains information about the location where input data specified in the JDL are physically stored, the SEs that are "close" to the CE chosen for submitting the job etc. It is shipped within the *InputSandbox* to the CE where the job is going to run so that it can be used at run-time to get information (through the appropriate API) for accessing data.

A way for performing direct submission to a given CE and at the same time having the *BrokerInfo* file generated by RB and shipped to the CE is to not use the `--resource` option and specify the following requirements in the JDL:

```
Requirements = other.GlueCEUniqueID == <Ce_identifier>;
```

(e.g. Requirements = other.GlueCEUniqueID == "lxde01.pd.infn.it:2119/jobmanager-lsf-grid01");

It is also possible to specify the target CE to which submit the job using the `--input` option. With the `--input` option an *input_file* must be supplied containing a list of target CE ids. In this case the **edg-job-submit** command parses the *input_file* and displays on the standard output the list of CE Ids written in the *input_file*. The user is then asked to choose one *CEId* between the listed ones. The command will then behave exactly like already explained for the `--resource` option. The basic idea of this command is to use as *input_file* the output file generated by the **edg-job-list-match** command when used with the `--output` option (see **edg-job-list-match**) that contains the list of CE Ids (if any) matching the requirements specified in the *jobad.jdl* file. An example of a possible sequence of commands is:

```
>$ edg-job-list-match --output CEList.out jobad.jdl
>$ edg-job-submit --input CEList.out jobad.jdl
```

If *CEList.out* contains more than one *CEId* then the user is prompted for choosing one Id from the list.

It is possible to redirect the returned *edg_jobId* to an output file using the `--output` option. If the file already exists, a check is performed: if the file was previously created by the command **edg-job-submit** (i.e. it contains a well defined header), the returned *edg_jobId* is appended to the existing file every time the command is launched. If the file wasn't created by the command **edg-job-submit** the user will be prompted to choose if overwrite the file or not. If the answer is no the command will abort.

The **edg-job-submit** command has a particular behaviour when the job description file contains the *InputSandbox* attribute whose value is a list of file paths on the UI machine local disk. The purpose of the introduction of the *InputSandbox* attribute is to stage, from the UI to the CE, files that are needed for the execution. Then the user can use the *InputSandbox* attribute to specify the files that have to be staged from the submitting machine to the executing CE. All of them are indeed transferred at job submission time together with the job class-ad to the NS that will store them temporarily on its local disk. The JobWrapper will then perform the staging of these files on the executing node. The size of files to be transferred to the "RB node" should be small since overfull of RB node local storage means that no more job of this type can be submitted.

This mechanism can also be used to stage a job executable available locally on the UI machine to the executing CE. Indeed in this case the user has to include this file in the *InputSandbox* list (specifying its absolute path in the file system of the UI machine) and as *Executable* attribute value has only to specify the file name. On the contrary, if the executable is already available in the file system of the executing machine, the user has to specify as *Executable* an absolute path name for this file (if necessary using environment variables). The same argument can be applied to the standard input file that is specified through the *StdInput* JDL attribute.

Since the *InputSandbox* expression can consist of a great number of file names, it is admitted the use of wildcards and environment variables to specify the value of this attribute.

It is important to note that since the gridftp protocol (the protocol used for the *InputSanbox* files staging) in general doesn't preserve the x flag, the script specified as *Executable* in the JDL (on which `chmod +x` is done automatically by the JobWrapper), should perform a `chmod +x` for all the files needing execution permission, that are transferred within the *InputSandbox* of the job.

For the standard output and error of the job the user shall instead always specify just file names (without any directory path) through the *StdOutput* and *StdError* JDL attributes. To have them staged

back on the UI machine it suffices to list them in the *OutputSandbox* and use after job completion the **edg-job-get-output** command described later in this document.

The list of data specification JDL attributes is completed by the *InputData* and *OutputData* attributes.

InputData refers to data used as input by the job that are not subjected to staging and are stored in one or more storage elements and published in replica catalogues. When the user specifies the *InputData* attribute then he/she also has to provide the protocol her/his application is able to “speak” for accessing data (*DataAccessProtocol* attribute). The *InputData* attribute should contain a list of Logical File Names (LFN) and/or Grid Unique Identifiers (GUID).

There is no need to specify the Replica Location Service to be contacted for resolving the logical files names and GUIDs to storage files names as it is automatically determined through the VO the user belongs to. This information is provided by in the *VirtualOrganisation* JDL attribute (filled by the UI).

It is worth noting that the usage for the ranking phase *getAccessCost* i.e. ranking CEs according to the cost for accessing data, can be triggered through the JDL by setting the rank as follows:

```
Rank = other.DataAccessCost;
```

The *OutputData* attribute allows instead the user to ask for the automatic upload and registration of datasets produced by the job on the WN.

Through this attribute it is possible to indicate for each output file the LFN to be used for registration and the SE on which the file has to be uploaded.

Both LFN and SE are optional in the sense that if no LFN is indicated then it is assigned automatically by the Replica Management services and if no SE is indicated, the close SE is considered.

OutputData is a list of classads, where each classad indicates the name of the file to be uploaded, the logical file to be used and the SE where the file has to be copied. E.g.:

```
OutputData = {
    [
        OutputFile = "dataset_1.out ";
        StorageElement = "se1.cnaf.infn.it";
        LogicalFileName = "lfn:LFN_1"
    ],
    [
        OutputFile = "dataset_2.out ";
        StorageElement = "se2.pd.infn.it";
        LogicalFileName = "lfn:LFN_2"
    ],
    [
        OutputFile = "dataset_3.out ";
        StorageElement = "se3.cesnet.cz";
        LogicalFileName = "lfn:LFN_3"
    ]
};
```

If the attribute *OutputData* is found in the JDL then the *JobWrapper* at the end of the job calls the “*copy And Register*” service that copies the file from the WN onto the specified SE and registers it with the given LFN. As usual, logical file names have to be prefixed with the string “*lfn:*”.

If the specified LFN is already in use, the Replica Manager registers the file with a newly generated identifier GUID (Grid Unique Identifier).

During this process the *JobWrapper* creates a file (named “*DSUpload_<unique_jobid_string>.out*”) that is put automatically in the *OutputSandbox* attribute list by the UI and can then be retrieved by the user. This file contains the results of the upload and registration process in the following format:

<FILE_NAME><LFN | ERROR>

e.g. in our case we could have:

```
dataset_1.out   LFN_1
dataset_2.out   <GUID2>
dataset_3.out   <error msg>
```

meaning that *dataset_1.out* was uploaded successfully and registered as *LFN_1*, *dataset_1.out* was uploaded successfully but with name *<GUID2>* (assigned by the ERM) since *LFN_2* was already in use and upload of *dataset_3.out* failed for the reason specified by the reported error message.

It is worth noting that the *StorageElement* attributes of the *OutputData* list are not taken into account by the RB for the matchmaking, so the job could have run on a CE that is not close to the specified SEs. Due to this it is suggested (unless the user has particular needs) to omit the *StorageElement* specification so that the close SEs are automatically taken into account for the datasets upload.

The *Arguments* attribute in the JDL allows the user to specify all the command line arguments needed to start the job. They have to be specified as a single string, e.g. the job *sum* that is started with:

```
$ sum N1 N2 -out result.out
```

is described by:

```
Executable = "sum";
Arguments = "N1 N2 -out result.out";
```

If you want to specify a quoted string inside the *Arguments* then you have to escape quotes with the `\` character. E.g. when describing a job like:

```
$ grep -i "my name" *.txt
```

you will have to specify:

```
Executable = "/bin/grep";
Arguments = "-i \"my name\" *.txt";
```

Analogously, if the job takes as argument a string containing a special character (e.g. the job is the *tail* command issued on a file whose name contains the quotes character, say *file1&file2*), since on the shell line you would have to write:

```
$ tail -f file1\&file2
```

in the JDL you'll have to write:

```
Executable = "/usr/bin/tail";  
Arguments = "-f file1\\\&file2";
```

i.e. a \ for each special character.

In general, special characters such as &, |, >, < are only allowed if specified inside a quoted string or preceded by triple \.

The character “” cannot be specified in the Arguments attribute of the JDL.

The *RetryCount* attribute allows setting the number of submission retries for a job upon failure due to some grid component (i.e. not to the job itself). *RetryCount* has to be a positive number and the actual number of submission retries for a job is represented by the minimum value between *RetryCount* itself and the value of the *MaxRetryCount* parameter in the WM configuration file. It suffices setting *RetryCount* to 0 to disable job resubmission.

The *--hours* allows the user to specify the user proxy duration *H*, in hours, needed for submitting the job. This option has to be used for long-lasting jobs, indeed a job when submitted needs to be accompanied by a valid proxy certificate during all its life-time and the default duration of user proxy created by UI commands is 12 hours that could in some case not be enough.

It is recalled that anyway a safer way for submitting long-running jobs is to use the proxy renewal feature provided by the WMS.

To do this the user should use the *myproxy-init* command before the **edg-job-submit**. The *myproxy-init* command registers indeed in a MyProxy server a valid long-term certificate proxy that will be used by WMS to perform a periodic credential renewal for the submitted job.

When using the *myproxy-init* command the user has to specify either through the *-s* option or the *MYPROXY_SERVER* environment variable the host name of the MyProxy server where to store the certificate proxy.

To trigger the proxy renewal mechanism, the same MyProxy server address has to be specified in the JDL through the *MyProxyServer* attribute.

An example of the JDL setting is provided hereafter:

```
MyProxyServer = "skurut.cesnet.cz";
```

Note that the port number must not be provided.

Interactive jobs are specified setting the JDL *JobType* attribute to “Interactive”. When an interactive job is submitted, the **edg-job-submit** command starts a grid console shadow process in the background that listens on a port for the job standard streams. Moreover the **edg-job-submit**

command opens a new window where the incoming job streams are forwarded. The port on which the shadow process listens is assigned by the OS, but can be forced through the *ListenerPort* attribute in the JDL.

As the command in this case opens a X window, the user should make sure the DISPLAY environment variable is correctly set, a X server is running on the local machine and if she/he is connected to the UI node from remote machine (e.g. with ssh) enable secure X11 tunneling.

If this is not possible, the user can specify the *--nogui* option that makes the command provide a simple standard non-graphical interaction with the running job.

Another option that is reserved for interactive jobs is *--nolisten*: it makes the command forward the job standard streams coming from the WN to named pipes on the UI machine whose names are returned to the user together with the OS id of the listener process. This allows the user to interact with the job through her/his own tools. It is important to note that when this option is specified, the UI has no more control over the launched listener process that has hence to be killed by the user (through the returned process id) when the job is finished.

For interactive jobs the UI automatically requires for the job outbound IP connectivity on the WN adding (in AND to the user defined expression) the *other.GlueHostNetworkAdapterOutboundIP* to the JDL *Requirements* expression.

Checkpointable jobs are specified setting the JDL *JobType* attribute to “Checkpointable”. When a checkpointable job is submitted the user can specify the number (or list) of steps in which the job can be logically decomposed and the step to be considered as the initial one. This can be done setting respectively the JDL attributes *JobSteps* and *CurrentStep*. *CurrentStep* is a mandatory attribute and if not provided by the user, it is set automatically to 0 by the UI.

The *--chkpt* option allows the submission of a checkpointable job specifying as input a checkpoint state generated by a previously submitted job. This option makes the submitted job start running from the checkpoint state given in input and not from the very beginning.

The initial checkpoint states to be used with this option can be retrieved by means of the **edg-job-get-chkpt** command. A checkpoint state is a JDL file.

MPI jobs are specified setting the JDL *JobType* attribute to “MPICH” or “MPICH-G2”. When a MPI job is submitted the presence of the *NodeNumber* attribute (it specifies the required number of CPUs) in the JDL is mandatory and the RB automatically requires the MPICH runtime environment installed on the CE and a number of CPUs at least equal to the required number of nodes.

Lastly the *--nomsg* option makes the command display neither messages nor errors on the standard output. Only the *edg_jobId* assigned to the job is printed to the user if the command was successful. Otherwise the location of the generated log file containing error messages is printed on the standard output. This option has been provided to make easier use of the **edg-job-submit** command inside scripts in alternative to the *--output* option.

It is important to note that the **edg-job-submit** is a sort of fire-and-forget command, i.e. it exits successfully once the JDL has been passed to the NS and the InputSandbox files have been transferred. It does not matter about what happens afterwards to the job. Understanding the reason of a job abort can however be accomplished by using the **edg-job-status** (especially looking at the “Status Reason” field) and **edg-job-get-logging-info** on the job identifier returned from the submission.

JOB DESCRIPTION FILE

A job description file contains a description of job characteristics and constraints in a class-ad style. A general description of the class-ad language is provided in document [8].

The job description file must be edited by the user to insert relevant information about the job that is later needed by the RB to perform the match-making. Job description file entries are strings having the format *attribute = expression* and are terminated by the semicolon character. Attribute expressions can span several lines provided the semicolon is put only at the end of the whole expression. 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 “*/”.

Being the class-ad an extensible language, it doesn't exist a fixed set of admitted attributes, i.e. the user can insert in the job description file whatever attribute he believes meaningful to describe her/his jobs, anyway only the attributes that can be in some way connected with the resource ones published in the IS are taken into account by the Matchmaker/RB for the match-making process. Unrelated attributes are simply ignored except when they are used to build the *Requirements* expression. In the latter case they are indeed evaluated and could affect the match-making result.

There is a small subset of JDL attributes that are compulsory, i.e. that have to be present in a job class-ad before it is sent to the Network Server in order to make possible the performing of the match making and submission.

They can be grouped in two categories: some of them **must** be provided by the user whilst some other, if not provided, are filled by the UI with configurable default values. The following Table 1 summarises what just stated.

| Attribute | Mandatory | Mandatory with default value (<i>default value</i>) |
|---------------------|--|--|
| Type | | ✓ “Job” |
| JobType | | ✓ “Normal” |
| Executable | ✓ | |
| Requirements | | ✓ other.GlueCEStateStatus == "Production" [configurable] |
| Rank | | ✓ other.GlueCEStateFreeCPUs (for MPICH jobs) other.GlueCEStateEstimatedResponseTime (for all other job types) [configurable] |
| NodeNumber | ✓ (only for MPICH or MPICH-G2 jobs) | |
| CurrentStep | | ✓ 0 (only for checkpointable jobs) |
| VirtualOrganisation | | ✓ |

| Attribute | Mandatory | Mandatory with default value (<i>default value</i>) |
|--------------------|---|---|
| | | [configurable] |
| DataAccessProtocol | ✓ (only if <i>InputData</i> has been specified) | |
| InputData | ✓ (only if <i>DataAccessProtocol</i> has been specified) | |

Table 1 Mandatory Attributes

In Table 1 the default values for *Requirements* and *Rank* can be interpreted respectively as follows:

- if the user has not provided job constraints then *Requirements* is set to (*other.GlueCEStateStatus* == "*Production*"), i.e. the target CE has to be active.
- Since in the JDL the greater is the value of *Rank* the better is considered the match, if no expression for *Rank* has been provided, then the resources where the jobs waits a shorter time to pass from the SCHEDULED to the RUNNING status are preferred, hence the *Rank* expression is set to (- *other.GlueCEStateEstimatedResponseTime*). MPICH jobs are an exception as they have as default rank *other.GlueCEStateFreeCPUs* meaning that the preferred resources are the ones having the higher number of free CPUs.

The default values for the *Requirements* and *Rank* attributes can be set in the *\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf* file.

As the classad language (and hence the JDL) is an extensible language, it allows the user to freely include new attributes within the job description. These attributes are ignored by the WMS for the scheduling but are passed-through by the UI (if their syntax is correct) since they could be relevant for the submitter or for some other component processing the JDL.

However if the job description file contains attributes that are unknown to the WMS, the UI will print a warning (when used with the *--debug* option) listing all of them.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--vo vo_name

This option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this JDL attribute hasn't been specified, then the default VO specified in the *\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf* (*DefaultVo* field) configuration file is considered. Otherwise an error is returned to the user.

--resource *ce_id***-r** *ce_id*

if the command is launched with this option, the job-ad sent to the NS contains a line of the type *SubmitTo = ce_id* and the job is submitted by the WMS to the resource identified by *ce_id* without going through the match-making process. Accepted format for the CEId is:

<full hostname>:<port number>/jobmanager-<service>-<queue name>

where *<service>* could be for example *lsf*, *pbs*, *bqs*, *condor* but can also be a different string as it is freely set by the site administrator when setting the queue.

Note that when this option is used, the “.BrokerInfo” file is not generated.

--input *file_path***-i** *input_file*

if this option is specified, the user will be asked to choose a *CEId* from a list of CEs contained in the *file_path*. Once a *CEId* has been selected the command behaves as explained for the **--resource** option. If this option is used together with the *-noint one* and the input file contains more than one *CEId*, then the first *CEId* in the list is taken into account for submitting the job.

--config *file_path***-c** *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--config-vo *file_path*

if the command is launched with this option, the vo-specific configuration file pointed to by *file_path* is used instead of the standard vo-specific configuration file.

--output *file_path***-o** *file_path*

writes the generated *edg_jobId* assigned to the submitted job in the file specified by *out_file*. *out_file* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *out_file* is created in the current working directory.

--chkpt *file_path*

This option can be used only for checkpointable jobs. The state specified as input is a checkpoint state generated by a previously submitted job. This option makes the submitted job start running from the checkpoint state given in input and not from the very beginning.

The initial checkpoint states to be used with this option can be retrieved by means of the **edg-job-get-chkpt** command (see 3.8).

--nogui

This option can be used only for interactive jobs. As the command for such jobs opens an X window, the user should make sure an X server is running on the local machine and if she/he is connected to the UI node from remote machine (e.g. with ssh) enable secure X11 tunneling. If this is not possible, the user can specify the `--nogui` option that makes the command provide a simple standard non-graphical interaction with the running job.

--nolisten

This option can be used only for interactive jobs. It makes the command forward the job standard streams coming from the WN to named pipes on the UI machine whose names are returned to the user together with the OS id of the listener process. This allows the user to interact with the job through her/his own tools. It is important to note that when this option is specified, the UI has no more control over the launched listener process that has hence to be killed by the user (through the returned process id) once the job is finished.

--hours *H***-h *H***

allows the user to specify the user proxy duration *H*, in hours, needed for submitting the job. When used with this option the `edg-job-submit` command behaves as follows:

- the command checks for user proxy existence and if the proxy does not exist a new proxy with *H* hours duration is created
- if the proxy exists then its duration is checked against the value specified with the `--hours` option. If proxy duration is greater than *H* hours then the job is submitted with the existing proxy, otherwise the old proxy is destroyed and a new one with *H* hours duration is created and used for submitting the job.

This mechanism allows the user to create before submission a proxy with a suitable duration for her/his job; moreover the user is not obliged to enter the PEM pass-phrase at each submission i.e. in all those cases where the existing proxy has a validity great enough for the job.

--nomsg

this option makes the command print on the standard output only the `edg_jobId` generated for the job if submission was successful; the location of the log file containing messages and diagnostics is printed otherwise.

--noint

if this option is specified every interactive question to the user is skipped, moreover only the `edg_jobId` is returned on the standard output. All warning messages and errors (if occurred) are written to the file `edg-job-submit_<UID>_<PID>_<timestamp>.log` under the `/tmp` directory. Log file location is configurable.

--debug

when this option is specified, information about parameters used for the API functions calls inside the command are displayed on the standard output and are written to `edg-job-submit_<UID>_<PID>_<timestamp>.log` file under the `/tmp` directory too. Log file location is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

jdl_file

this is the file containing the JDL describing the job to be submitted. It must be the last argument of the command.

EXIT STATUS

edg-job-submit exits with a status value of 0 (zero) upon success, and >0 (greater than zero) upon failure.

EXAMPLES

1. `$> edg-job-submit -vo cms myjob1.jdl`

where *myjob1.jdl* is as follows:

```
#####
#
# ----- Job description file -----
#
#####
[
  JobType = "Normal";
  Executable      = "$(CMS)/fpacini/exe/sum.exe";
  InputData      = "LF:testbed0-00019";

  DataAccessProtocol = "gridftp";
  Rank              = other.GlueCEPolicyMaxCPUTime;
  Requirements     = other.GlueCEInfoLRMSType == "Condor" && \
                    (!(RegExp("nikhef*", other.GlueCEUniqueID)));
```

submits *sum.exe* to a resource (supposed to contain the executable file) whose LRMS is Condor and not containing the string "nikhef" in the CE identifier. The command returns the following output to the user, containing the job handle (*edg_jobid*):

```
===== edg-job-submit Success =====
```

The job has been successfully submitted to the Network Server. Your job is identified by (edg_jobId):
`https://ibm139.cnaf.infn.it:9000/ZU9yOC7AP7AOEhMAHirG3`

Use `edg-job-status` command to display current job status.

```
=====
```

2. `$> edg-job-submit --chkpt /home/test/state10.chkpt myjob2.jdl`

Submits the checkpointable job described by *myjob2.jdl* that will start running from the initial state *state10.chkpt*.

SEE ALSO

`edg-job-list-match`, `edg-job-attach`, `edg-job-get-chkpt`.

3.2. EDG-JOB-GET-OUTPUT

This command retrieves the job output files (specified by the *OutputSandbox* attribute of the job-ad) from the RB node and stores them on the submitting machine local disk.

SYNOPSIS

```
edg-job-get-output [options] <job Id(s)>
```

Options:

```
--help  
--version  
--input, -i      <file_path>  
--dir            <directory_path>  
--config, -c    <file_path>  
--noint  
--debug  
--logfile       <file_path>
```

DESCRIPTION

The **edg-job-get-output** command can be used to retrieve the output files of a job that has been submitted through the **edg-job-submit** command with a job description file including the *OutputSandbox* attribute. After the submission, when the job has terminated its execution, the user can download the files generated by the job and temporarily stored on the RB machine as specified by the *OutputSandbox* attribute, issuing the **edg-job-get-output** with as input the *edg_jobId* returned by the **edg-job-submit**. It is also possible to specify a list of job identifiers when calling this command or an input file containing *edg_jobIds* by means of the *--input* option. When the *--input* is used, the user is requested to choose all, one or a subset of the job identifiers contained in the input file.

It is important to note that the *OutputSandbox* of a submitted job can only be retrieved when the job has reached the *Done* status indicating that the job has successfully terminated its execution and the *OutputSandbox* files are ready for retrieval on the RB node. **edg-job-get-output** will always fail for jobs that are not yet in the *Done* status.

The user can decide the local directory path on the UI machine where these files have to be stored by means of the *--dir* option, otherwise the retrieved files are put in a default location specified in the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` configuration file (OutputStorage parameter). In both cases a sub-directory will be added to the path supplied. The name of this sub-directory is the unique string of the *edg_jobId* identifier (see command **edg-job-submit** for details on the *edg_jobId* structure).

If the user wants to use her/his “private” configuration file, this can be done using option *--config path_name*. As a consequence the **edg-job-get-output** command looks for the file “*path_name*” instead of the standard configuration file. If this file does not exist the user is notified with an error message and the command is aborted.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--dir *directory_path*

retrieved files (previously listed by the user through the *OutputSandbox* attribute of the job description file) are stored in the location indicated by *directory_path*/*<edg_jobId unique string>*.

--config *file_path***-c** *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if occurred) are written to the file *edg-job-get-output-<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location of log file is configurable.

--debug

when this option is specified, information about parameters used for the API functions calls inside the command are displayed on the standard output and are written to *dg-get-job-output-<UID>_<PID>_<timestamp>.log* file under the */tmp* directory too. Location of log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. If a list of one or more job identifiers is specified, *edg_jobIds* have to be separated by a blank. Job identifiers must be last argument of the command.

--input *file_path***-i** *file_path*

this option makes the command return the *OutputSandbox* files for each *edg_jobId* contained in the *file_path*. This option can't be used if one (or more) *edg_jobIds* have been already

specified. The format of the input file must be as follows: one *edg_jobId* for each line and comment lines must begin with a “#” or a “*” character.

EXIT STATUS

edg-job-get-output exits with a status value of 0 (zero) upon success, >0 upon failure and <0 upon partial failure. An example of partial failure is when more than one job identifiers has been specified and the *OutputSandbox* could be retrieved only for some of them.

EXAMPLES

Let us consider the following command:

```
$> edg-job-get-output https://ibm139.cnaf.infn.it:9000/CiXMLojKC\_iLsvSHfEhqIQ --dir  
/home/data
```

It retrieves the files listed in the *OutputSandbox* attribute of job identified by https://ibm139.cnaf.infn.it:9000/CiXMLojKC_iLsvSHfEhqIQ from the RB node and stores them locally in */home/data/CiXMLojKC_iLsvSHfEhqIQ*.

3.3. EDG-JOB-LIST-MATCH

Returns the list of resources fulfilling job requirements specified in the JDL job description

SYNOPSIS

```
edg-job-list-match [options] <jdl file>
```

Options:

```
--help  
--version  
--verbose  
--rank  
--config, -c      <file_path>  
--config-vo      <file_path>  
--vo              <vo_name>  
--output, -o     <file_path>  
--noint  
--debug  
--logfile        <file_path>
```

edg-job-list-match displays the list of identifiers of the resources on which the user is authorized and satisfying the job requirements included in the job description file. The CE identifiers are returned either on the standard output or in a file according to the chosen command options, and are strings univocally identifying the CEs published in the IS.

The returned *CEIds* are listed in decreasing order of rank, i.e. the one with the best (greater) rank is in the first place and so on.

The `--rank` option makes the command also display the rank value for each found *CEId*.

The `--vo` option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this JDL attribute hasn't been specified, the default VO specified in the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` (*DefaultVo* field) configuration file is considered. Otherwise an error is returned to the user.

With the deployment of VOMS, the VO information will be encoded in the user's proxy certificate and this will make the described behaviour change.

edg-job-list-match requires a job description file in which job characteristics and requirements are expressed by means of a class-ad. The job description file is first syntactically checked and then used as the main command-line argument to **edg-job-list-match**. The Network Server is only contacted to find job compatible resources; the job is not submitted. See the **edg-job-submit** for general rules for building the job description file.

If the user wants to use his "private" configuration, file this can be done using option `--config path_name`.

The option `--verbose` of the **edg-job-list-match** command can be used to obtain on the standard output the class-ad sent to the RB generated from the job description.

The `--output` option makes the command save the list of compatible resources into the specified file. If the provided file name is not an absolute path, then the output file is created in the current working dir.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--verbose

-v

displays on the standard output the job class-ad that is sent to the Network Server generated from the job description file. This differs from the content of the job description file since the UI adds to it some attributes that cannot be directly inserted by the user (e.g., defaults for Rank and Requirements if not provided, VirtualOrganisation etc).

--rank

displays the "matching" *CEIds* and the associated ranking values.

--vo vo_name

This option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this JDL attribute has not been specified, then the default VO specified in the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` (*DefaultVo* field) configuration file is considered. Otherwise an error is returned to the user.

--config *file_path*

-c *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--config-vo *file_path*

if the command is launched with this option, the vo-specific configuration file pointed to by *file_path* is used instead of the standard vo-specific configuration file.

--output *file_path*

-o *file_path*

returns the *CEIDs* list in the file specified by *file_path*. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if any) are written to the file *edg-job-list-match* *<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location of the log file is configurable.

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-list-match_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location of the log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

jdl_file

this is the file containing the classad describing the job to be submitted. It must be the last argument of the command.

EXIT STATUS

edg-job-list-match exits with a status value of 0 upon success, and a >0 value upon failure.

EXAMPLES

3.4. EDG-JOB-CANCEL

Cancels one or more submitted jobs.

SYNOPSIS

```
edg-job-cancel [options] <job Id(s)>
```

Options:

```
--help
--version
--all
--input, -i      <file_path>
--config, -c     <file_path>
--config-vo     <file_path>
--vo            <vo_name>
--output, -o    <file_path>
--noint
--debug
--logfile       <file_path>
```

DESCRIPTION

This command cancels a job previously submitted using **edg-job-submit**. Before cancellation, it prompts the user for confirmation. The cancel request is sent to the Network Server that forwards it to the WM that fulfils it.

Edg-job-cancel can remove one or more jobs: the jobs to be removed are identified by their job identifiers (*edg_jobIds* returned by **edg-job-submit**) provided as arguments to the command and separated by a blank space. The result of the cancel operation is reported to the user for each specified *edg_jobId*.

If the *--all* option is specified, all the jobs owned by the user submitting the command are removed. When the command is launched with the *--all* option, no *edg_jobId* can be specified. It has to be remarked that only the owner of the job can remove the job. When the *--all* option is specified the UI queries each LB listed in the vo-specific configuration file `$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf` for getting the identifiers of all the jobs owned by the user identified by her/his certificate subject. Afterwards the UI sends a cancellation request to the NS for each job being in a status for which the cancellation is allowed.

Job states for which cancellation is allowed are:

- Submitted
- Waiting
- Ready
- Scheduled
- Running

- Unknown

For all the other job states the cancellation request will result in a failure.

If the user wants to use his “private” configuration file this could be done using option `--config file_path`.

The `--input` option permits to specify a file (*file_path*) that contains the *edg_jobIds* to be removed. The format of the file must be as follows: one *edg_jobId* for each line and comment lines must begin with a “#” or a “*” character. When using this option the user is interrogated for choosing among all, one or a subset of the listed job identifiers. If the *file_path* does not represent an absolute path the file will be searched in the current working directory.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--all

cancel all job owned by the user submitting the command. This option can't be used either if one or more *edg_jobIds* have been specified explicitly or with the `--input` option.

--input file_path

-i file_path

cancel *edg_jobId* contained in the *file_path*. This option can't be used neither if one or more *edg_jobIds* have been specified nor with the `--all` option.

--config file_path

-c file_path

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--config-vo file_path

if the command is launched with this option, the vo-specific configuration file pointed to by *file_path* is used instead of the standard vo-specific configuration file. This option is allowed only when used together with the `--all` one.

--vo vo_name

This option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this JDL attribute hasn't been specified, then the default VO specified in the `$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf` (*DefaultVo* field) configuration file is

considered. Otherwise an error is returned to the user. This option is allowed only when used together with the `--all` one.

--output *file_path*

-o *file_path*

writes the cancel results in the file specified by *file_path* instead of the standard output. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if occurred) are written to the file *edg-job-cancel_<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location of the log file is configurable.

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-cancel_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location of the log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. The job identifier list must be the last argument of this command.

EXIT STATUS

edg-job-cancel exits with a status value 0 if all the specified jobs were cancelled successfully, >0 if errors occurred for each specified job id and <0 in case of partial failure. An example of partial failure is when more than one job has been specified: some jobs could be successfully removed and some others could be not removed.

EXAMPLES

1. `$> edg-job-cancel --input joblist.txt`

where *joblist.txt* is a file containing 3 *edg_JobIds*, displays the following confirmation message:

```
Are you sure you want to remove all jobs specified? [y/n]n: y
```

```
===== edg-job-cancel Success =====  
The cancel request for the following job(s) has been successfully submitted  
to NS:  
  - https://ibm139.cnaf.infn.it:9000/nUbiIiMFmYloIusAaWxPhg  
  - https://ibm139.cnaf.infn.it:9000/VtMvhs8z7WGCptt92ZMPIQ  
  - https://ibm139.cnaf.infn.it:9000/yKTKyrdSgHKQlwwwSocJiw  
=====
```

\$>
In this case the command exit code is 0.

2. \$> **edg-job-cancel** --all --noint

removes all job owned by the user submitting the command.

SEE ALSO

edg-job-submit.

3.5. EDG-JOB-STATUS

Displays bookkeeping information about submitted jobs.

SYNOPSIS

edg-job-status [options] <job Id(s)>

Options:

```
--help  
--version  
--all  
--input, -i      <file_path>  
--verbosity, -v  <verbosity_value>  
--config, -c     <file_path>  
--config-vo      <file_path>  
--vo             <vo_name>  
--output, -o     <file_path>  
--noint  
--debug  
--logfile        <file_path>
```

DESCRIPTION

This command prints the status of a job previously submitted using **edg-job-submit**. The job status request is sent to the LB that provides the requested information. This can be done during the whole job life.

edg-job-status can monitor one or more jobs: the jobs to be checked are identified by one or more job identifiers (*edg_jobIds* returned by **edg-job-submit**) provided as arguments to the command and separated by a blank space.

If the *--all* option is specified, information about all the jobs owned by the user submitting the command is printed on the standard output. When the command is launched with the *--all* option, neither can an *edg_jobId* be specified nor can the *--input* option be specified.

The *--input* option permits to specify a file (*file_path*) that contains the *edg_jobIds* to monitor. The format of the file must be as follows: one *edg_jobId* for each line and comment lines have to begin with a “#” or a “*” character. When using this option the user is requested for choosing among all, one or a subset of the listed job identifiers. If the *file_path* does not represent an absolute path, it will be searched in the current working directory.

If the user wants to use his “private” configuration file, this can be done using option *--config file_path*.

The same applies for the vo-specific configuration file and the *--config-vo* option.

The *--verbosity* option allows setting the detail level of the returned information. This option can be specified with three values, 0, 1 and 2. The default level of verbosity is 0 unless otherwise specified in the UI configuration file *\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf* (*DefaultStatusLevel* parameter).

Hereafter are listed the information displayed according to the verbosity level.

Verbosity equal 0:

- *edg_jobId* (the job unique identifier)
- Current Status (the job current status)
- *exit_code* (Unix exit code – if applicable)
- Status Reason (reason for being in this state)
- Reached on (date/time when the job entered actual state)
- *destination* (ID of CE where the job has been submitted – if applicable)

With verbosity equal 1, some additional information fields are added such as:

- *cancelling* (boolean indicating if a cancellation request for the job is in progress)
- *cancelReason* (Reason of cancel)
- *ce_node* (Worker node where the job is executed)
- *children_hist* (summary -- histogram -- of children job states)
- *children_num* (number of subjobs)
- *subjob_failed* (Subjob failed -- the parent job will fail too)
- *condorId* (Id within Condor-G)
- *cpuTime* (Consumed CPU time)
- *expectUpdate* (Boolean indicating that some logged information has not arrived yet)

-
- expectFrom (Sources of the missing information)
 - jobtype (Type of the request: 0 = Job, 1 = DAG)
 - lastUpdateTime (Last known event of the job)
 - location (location Where the job is being processed)
 - network_server (Network server handling the job)
 - owner (certificate subject of Job owner)
 - resubmitted (boolean indicating that the job was resubmitted)

Lastly, with verbosity equal 2 there are the following additional fields:

- jdl (User submitted job description)
- matched_jdl (Full job description after matchmaking)
- condor_jdl (ClassAd passed to Condor-G for job submission)
- rsl (Job RSL sent to Globus)
- stateEnterTimes (When all previous states were entered)

Information fields that are not available (i.e. not returned by the LB because not applicable for the given status) are not printed at all to the user.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--all

displays status information about all job owned by the user submitting the command. This option can't be used either if one or more *edg_jobIds* have been specified or if the *--input* option has been specified. All LBs listed in the vo-specific UI configuration file *\$EDG_WL_LOCATION/etc/<vo_name>/edg_wl_ui.conf* are contacted to fulfil this request.

--input *input_file*

-i *input_file*

displays bookkeeping info about *dg_jobIds* contained in the *input_files*. When using this option the user is interrogated for choosing among all, one or a subset of the listed job identifiers. This option can't be used either if one or more *edg_jobIds* have been specified or if the *--all* option has been specified.

--verbosity *verb_level*

--v *verb_level*

sets the detail level of information about the job displayed to the user. Possible values for *verb_level* are 0,1 and 2.

--config *file_path*

-c *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--config-vo *file_path*

if the command is launched with this option, the vo-specific configuration file pointed to by *file_path* is used instead of the standard vo-specific configuration file. This option is allowed only when used together with the *--all* one.

--vo *vo_name*

This option allows the user to specify the Virtual Organisation she/he is currently working for. If this option is not used, then the *VirtualOrganisation* attribute in the JDL is considered. If this attribute hasn't been specified, then the default VO specified in the *\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf* (*DefaultVo* field) configuration file is considered. Otherwise an error is returned to the user. This option is allowed only when used together with the *--all* one.

--output *file_path*

-o *file_path*

writes the bookkeeping information in the file specified by *file_path* instead of the standard output. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if any) are written to the file *edg-job-status_<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location of log file is configurable.

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-status_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location of log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. Job identifiers must always be provided as last arguments of the command.

EXIT STATUS

edg-job-status exits with a value of 0 if the status of all the specified jobs is retrieved correctly, >0 if errors occurred for each specified job id and <0 in case of partial failure. An example of partial failure is when more then one job is specified: status info could be successfully retrieved for some jobs and not retrieved for some others.

EXAMPLES

\$> **edg-job-status -v 0** https://ibm139.cnaf.infn.it:9000/_tO6hdgToYKGCuV68q-gqQ

displays the following lines:

```
*****
BOOKKEEPING INFORMATION:

Printing status info for the Job:
https://ibm139.cnaf.infn.it:9000/_tO6hdgToYKGCuV68q-gqQ
Current Status:      Scheduled
Destination:        bbq.mi.infn.it:2119/jobmanager-pbs-dque
Status Reason:      Job successfully submitted to Globus
reached on:         Tue May 6 16:14:59 2003
*****

$>
```

SEE ALSO

edg-job-submit.

3.6. EDG-JOB-GET-LOGGING-INFO

Displays logging information about submitted jobs.

SYNOPSIS

```
edg-job-get-logging-info [options] <job Id(s)>
```

Options:

```
--help
--version
--input, -i      <file_path>
--verbosity, -v <verbosity_value>
--config, -c     <file_path>
--output, -o     <file_path>
--noint
--debug
--logfile        <file_path>
```

DESCRIPTION

This command queries the LB persistent DB for logging information about jobs previously submitted using **edg-job-submit**. The job logging information are stored permanently by the LB service and can be retrieved also after the job has terminated its life-cycle, differently from the bookkeeping information that are in some way “consumed” by the user during the job existence.

The **edg-job-get-logging-info** request is sent to the LB service that queries the DB and returns the retrieved information. Content of the logging information varies according to the type of the event they are related to. The most common information fields are:

- Event *(event type)*
- source *(WMS component which generated the event)*
- result *(result of the attempt)*
- destination *(destination where the job is being transferred to)*
- timestamp *(timestamp of event generation)*

The *--verbosity* option allows setting the detail level of the returned information. This option can be specified with three values, 0, 1 and 2. The default level of verbosity is 0 unless otherwise specified in the UI configuration file *\$EDG_WL_LOCATION/etc/edg_wl_ui_cmd_var.conf* (*DefaultLoggingLevel* parameter).

The information listed above is displayed when the chosen verbosity level is 0. If the command is issued with 1 as verbosity flag, then the following additional information is shown:

-
- host (hostname of the machine where the event was generated)
 - dest_host (destination hostname)
 - dest_instance (instance of destination WMS component)
 - user (identity -- cert. subj. -- of the generator)
 - dest_jobid (destination internal jobid)
 - node (worker node where the executable is run)
 - ns (Network server handling the job)
 - nsubjobs (number of subjobs)
 - local_jobid (new jobId assigned by the receiving component)
 - queue (destination queue name)
 - status_code (way of job termination/classification of the cancel)

Lastly if the command is issued with *verbosity* level 2, additional information mostly consisting in the job description within the WMS component that has logged the event, is printed to the user:

- jdl (job description)
- job (job description in receiver language)
- descr (description of current job transformation -- output of helper)
- classad (checkpoint state value)
- seqcode (sequence code assigned to the event)
- level (logging level -- system, debug, ...)

Data on several jobs can be queried by specifying a list of job identifiers separated by a blank space as arguments of the command. Moreover the *--input* option permits to specify a file (*file_path*) which contains the *edg_jobIds* whose information are requested. The format of the file must be as follows: one *edg_jobId* for each line and comment lines have to begin with a “#” or a “*” character. When using this option the user is interrogated for choosing among all, one or a subset of the listed job identifiers. If the *file_path* does not represent an absolute path, it will be searched in the current working directory.

Each event logged in the LB has an associated *log level* according to “Universal Format for Logger Messages” (see [draft-abela-ulg-05.txt](http://www-didc.lbl.gov/NetLogger/draft-abela-ulg-05.txt) available at <http://www-didc.lbl.gov/NetLogger/draft-abela-ulg-05.txt>). Default value for the log level used by WMS components is *System*, anyway there could be special situations in which problems investigation is needed and additional events are logged with the *Debug* log level. The *--output* option can be used to have the retrieved information written in the file identified by *file_path* instead of the standard output. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory.

If the user wants to use his “private” configuration file this could be done using option *--config file_path*.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--input *file_path*

-i *file_path*

retrieves logging info for all *edg_jobIds* contained in the *file_path*. This option can't be used if one or more *edg_jobIds* have been specified.

--verbosity *verb_level*

--v *verb_level*

sets the detail level of information about the job displayed to the user. Possible values for *verb_level* are 0,1 and 2.

--config *file_path*

-c *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--output *file_path*

-o *file_path*

writes the logging information in the file specified by *file_path* instead of the standard output. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if occurred) are written to the file *edg-job-logging_<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location for log file is configurable.

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-logging_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location for log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. Job identifiers must always be provided as last arguments for this command.

EXIT STATUS

edg-job-get-logging-info exits with a value of 0 if the status of all the specified jobs is retrieved correctly, >0 if errors occurred for each specified job and <0 in case of partial failure. An example of partial failure is when more then one job is specified: some job's logging info could be successfully retrieved and some others could be not retrieved.

EXAMPLES

1. `$> edg-job-get-logging-info \ https://ibm139.cnaf.infn.it:9000/GMUJtnNqe6Lq7w7MfOzeQw - output mylog.txt`

writes in file *mylog.txt* in the current working directory logging information about the job identified by *https://ibm139.cnaf.infn.it:9000/GMUJtnNqe6Lq7w7MfOzeQw*.

2. `$> edg-job-get-logging-info -v 0 -input $HOME/myIds.txt`
where

`$HOME/myjobs.txt` contains two job identifiers, displays the following output

```
-----
1 : https://ibm139.cnaf.infn.it:9000/D4S_i25ffAsPnKB3iCqeaA
2 : https://ibm139.cnaf.infn.it:9000/2qzyCbPW7pDY3rNh9PuXA
a : all
q : quit
-----
Choose one or more edg_jobId(s) in the list - [1-2]all: 2
```

LOGGING INFORMATION:

Printing info for the Job : *https://ibm139.cnaf.infn.it:9000/2qzyCbPW7pDY3rNh9PuXA*

```
---
Event: RegJob
- source      =  UserInterface
- timestamp   =  Wed May 14 10:55:35 2003
---
```

Event: Transfer

```
- destination = NetworkServer
- result      = START
- source      = UserInterface
- timestamp   = Wed May 14 10:55:36 2003
```

Event: Transfer

```
- destination = NetworkServer
- result      = OK
- source      = UserInterface
- timestamp   = Wed May 14 10:55:44 2003
```

Event: Accepted

```
- source      = NetworkServer
- timestamp   = Wed May 14 10:56:42 2003
```

Event: EnQueued

```
- result      = OK
- source      = NetworkServer
- timestamp   = Wed May 14 10:56:45 2003
```

...

...

SEE ALSO

edg-job-submit.

3.7. EDG-JOB-ATTACH

This command starts an interactive session for a previously submitted interactive job.

SYNOPSIS

edg-job-attach [options] <job Id>

Options:

```
--help
--version
--port, -p      <port_num>
--nogui
--nolisten
--config, -c    <file_path>
```

```
--input, -i      <file_path>
--noint
--debug
--logfile        <file_path>
```

DESCRIPTION

This command starts a listener process on the UI machine (*grid_console_shadow*) that allows attaching to the standard streams of a previously submitted interactive job and displays them on a dedicated window.

As the command opens a X window, the user should make sure the DISPLAY environment variable is correctly set, a X server is running on the local machine and if she/he is connected to the UI node from remote machine (e.g. with ssh) enable secure X11 tunneling.

The listener process and the window are started automatically by the **edg-job-submit** command for interactive jobs, so this command can be used for example in case a problem occurred on the UI machine that made the interactive session be lost or in case the user needs to follow the job from another machine or another port on the same machine (*--port* option).

This command can only be invoked for interactive jobs.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--port *port_num*

-p *port_num*

makes the command start a listener on the local machine on the specified port and logs these information to the LB associated to the job.

--nogui

As the **edg-job-attach** command opens a X window, the user should make sure a X server is running on the local machine and if she/he is connected to the UI node from remote machine (e.g. with ssh) enable secure X11 tunneling. If this is not possible, the user can specify the *--nogui* option that makes the command provide a simple standard non-graphical interaction with the running job.

--nolisten

This option makes the command forward the job standard streams coming from the WN to named pipes on the UI machine whose names are returned to the user together with the OS id of the listener process. This allows the user to interact with the job through her/his own tools. It is important to note that when this option is specified, the UI has no more control over the

launched listener process that has hence to be killed by the user (through the returned process id) once the job is finished.

--config *file_path***-c** *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--input *file_path***-i** *file_path*

allows the user to attach to one (just one) of the *edg_jobIds* contained in the *file_path*. This option can't be used if one *edg_jobIds* has been specified.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if occurred) are written to the file *edg-job-attach_<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location for log file is configurable.

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-attach_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location for log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. Job identifiers must always be provided as last arguments for this command.

EXIT STATUS

edg-job-attach exits with a value of 0 on success and >0 on failure.

EXAMPLES

\$> **edg-job-attach** https://ibm139.cnaf.infn.it:9000/t3KwW8qhXhkYs-ZfNCFidg

displays the following information message:

```
*****
JOB ATTACHED:
The Interactive Session Listener has been successfully launched
with the following parameters:
    ---
Host:                10.1.1.90
Port:                40713
Pid:                 18575
*****
```

and opens a window allowing interaction with the job through the standard streams.

3.8. EDG-JOB-GET-CHKPT

This command retrieves checkpoint states saved by a previously submitted checkpointable job.

SYNOPSIS

edg-job-get-chkpt [options] <job Id>

Options:

```
--help
--version

--cs                <state_num>
--config, -c       <file_path>
--input, -i        <file_path>
--output, -o       <file_path>
--noint
--debug
--logfile          <file_path>
```

DESCRIPTION

This command allows the user to retrieve one or more checkpoint states saved by a previously submitted job. Checkpoint states are retrieved from the LB server and are saved locally into a file in JDL format.

The `--cs` option allows the user to select the checkpoint state she/he wants to be retrieved. Indeed specifying the command with “`--cs N`” makes the command retrieve the last but N job checkpoint state. Last saved state is retrieved otherwise.

The retrieved state is saved in a file in JDL format. The output file path can be set through the `--output` option of the command.

This command can be used only for checkpointable jobs.

OPTIONS

--help

displays command usage.

--version

displays UI version.

--config *file_path***-c** *file_path*

if the command is launched with this option, the configuration file pointed to by *file_path* is used instead of the standard configuration file.

--cs *state_num*

if the command is launched with this option then it retrieves the “*last but state_num*” state saved by the job. Last saved state is returned if the option is not used (equivalent to *state_num* = 0).

--input *file_path***-i** *file_path*

allows the user to select one (just one) of the *edg_jobIds* contained in the *file_path* for retrieval of the saved checkpoint state. This option can't be used if one *edg_jobIds* has been specified.

--output *file_path***-o** *file_path*

saves the retrieved state in the file specified by *file_path*. *file_path* can be either a simple name or an absolute path (on the submitting machine). In the former case the file *file_path* is created in the current working directory. If this option is not used the retrieved state is displayed on the standard output.

--noint

if this option is specified every interactive question to the user is skipped. All warning messages and errors (if occurred) are written to the file *edg-job-get-chkpt_<UID>_<PID>_<timestamp>.log* under the */tmp* directory. Location for log file is configurable

--debug

when this option is specified, information about the API functions called inside the command are displayed on the standard output and are written to the file *edg-job-attach_<UID>_<PID>_<timestamp>.log* under the */tmp* directory too. Location for log file is configurable.

--logfile *file_path*

when this option is specified, the command log file is relocated to the location pointed by *file_path*

edg_jobId

job identifier returned by **edg-job-submit**. Job identifiers must always be provided as last arguments for this command.

EXIT STATUS

edg-job-get-chkpt exits with a value of 0 on success and >0 on failure.

EXAMPLES

The following commands retrieve the last but 3 saved checkpoint state of the job and save it in the file specified by the user:

```
$> edg-job-get-chkpt -o myjob.chk -cs 3 https://ibm139.cnaf.infn.it:9000/LNn4rOX17LL30e34hSqGjQ
```

```
===== edg-job-get-chkpt Success =====  
The checkpointable Job state has been successfully retrieved from LB  
Server and stored in the file: /home/fpacini/CLI/bin/myjob.chk  
=====
```

```
$> more /home/fpacini/CLI/bin/myjob.chk
```

```
# Job State Retrieved for  
#edg_jobId: https://ibm139.cnaf.infn.it:9000/LNn4rOX17LL30e34hSqGjQ  
[  
  UserData =  
  [  
    distribution = false;  
    hsum_filename =  
      "gsiftp://lxde01.pd.infn.it/tmp/root_test/hsum_lxde04_1200000.root";  
    first_event = 1200001  
  ];  
];
```

4. CONTACT INFORMATION AND CREDITS

People involved in Grid Resource Management are:

- Álvaro Fernández Alvaro.Fernandez@ific.uv.es, IFIC
- Enol Fernández enol@aomail.uab.es, UAB
- Elisa Heymann Elisa.Heymann@uab.es, UAB
- Antonio Hervás Antonio.Hervas@aows10.uab.es, UAB
- Anna Morajko ania@aomail.uab.es, UAB
- Miquel A. Senar MiquelAngel.Senar@uab.es, UAB
- Marco Sottilaro marco.sottilaro@datamat.it, DATAMAT

For bug reporting use GridPortal bug tools at <http://savannah.fzk.de/bugs/?group=cg-wp3-2>

5. THE EDG LICENSE AGREEMENT

Copyright (c) 2005 CrossGrid. All rights reserved.

This software includes voluntary contributions made to the CrossGrid Project. For more information on CrossGrid, please see <http://www.eu-crossgrid.org>.

Installation, use, reproduction, display, modification and redistribution of this software, with or without modification, in source and binary forms, are permitted. Any exercise of rights under this license by you or your sub-licensees is subject to the following conditions:

1. Redistributions of this software, with or without modification, must reproduce the above copyright notice and the above license statement as well as this list of conditions, in the software, the user documentation and any other materials provided with the software.

2. The user documentation, if any, included with a redistribution, must include the following notice: “This product includes software developed by the CrossGrid Project (<http://www.eu-crossgrid.org>).”

Alternatively, if that is where third-party acknowledgments normally appear, this acknowledgment must be reproduced in the software itself.

3. The names “CrossGrid” and “CG” may not be used to endorse or promote software, or products derived therefrom, except with prior written permission by cgooffice@cyfronet.krakow.pl.

4. You are under no obligation to provide anyone with any bug fixes, patches, upgrades or other modifications, enhancements or derivatives of the features, functionality or performance of this software that you may develop. However, if you publish or distribute your modifications, enhancements or derivative works without contemporaneously requiring users to enter into a separate written license agreement, then you are deemed to have granted participants in the CrossGrid Project a worldwide, non-exclusive, royalty-free, perpetual license to install, use, reproduce, display, modify, redistribute and sub-license your modifications, enhancements or derivative works, whether in binary or source code form, under the license conditions stated in this list of conditions.

5. DISCLAIMER

THIS SOFTWARE IS PROVIDED BY THE CROSSGRID PROJECT AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, OF SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE OR USE ARE DISCLAIMED. THE CROSSGRID PROJECT AND CONTRIBUTORS MAKE NO REPRESENTATION THAT THE SOFTWARE, MODIFICATIONS, ENHANCEMENTS OR DERIVATIVE WORKS THEREOF, WILL NOT INFRINGE ANY PATENT, COPYRIGHT, TRADE SECRET OR OTHER PROPRIETARY RIGHT.

6. LIMITATION OF LIABILITY

THE CROSSGRID PROJECT AND CONTRIBUTORS SHALL HAVE NO LIABILITY TO LICENSEE OR OTHER PERSONS FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, OR PUNITIVE DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA OR PROFITS, OR BUSINESS INTERRUPTION, HOWEVER CAUSED AND ON ANY THEORY OF CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE), PRODUCT LIABILITY OR OTHERWISE, ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.