GT 4.2.1 GridWay: User's Guide

Published May, 2008
Copyright © 2002-2008 GridWay Team, Distributed Systems Architecture Group, Universidad Complutense de Madrid (dsa-research.org).

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Any academic report, publication, or other academic disclosure of results obtained with the GridWay Metascheduler will acknowledge GridWay's use by an appropriate citation to relevant papers by GridWay team members.

1 http://www.apache.org/licenses/LICENSE-2.0
# Table of Contents

1. Introduction .................................................................................................................................... 1  
   1. Benefits for the end user ........................................................................................................... 1  
   2. How GridWay operates .............................................................................................................. 1  
   3. Job life-cycle in GridWay .......................................................................................................... 2  
   4. A grid-aware application model ................................................................................................. 3  
2. User environment configuration .................................................................................................. 4   
   1. Environment variables for GridWay ............................................................................................. 4  
3. Functionality ................................................................................................................................... 6  
   1. Job description overview ............................................................................................................ 6  
   2. Job Template options ................................................................................................................ 7  
   3. File definition in Job Templates ................................................................................................... 9  
   4. Variable substitution ................................................................................................................ 12  
   5. Resource selection expressions ................................................................................................. 12  
   6. Job Submission Description Language (JSDL) ............................................................................. 16  
4. Usage Scenarios ............................................................................................................................. 24  
   1. Single jobs: Submitting and monitoring the simplest job ............................................................ 24  
   2. Array jobs: Calculating the $\pi$ number ...................................................................................... 26  
   3. MPI jobs: Calculating the $\pi$ number again ............................................................................ 30  
   4. Workflows ..................................................................................................................................... 31  
I. GridWay Commands ........................................................................................................................ 36  
   Job and Array Job submission Command ........................................................................................ 37  
   DAG Job submission Command ..................................................................................................... 38  
   Job Monitoring Command ........................................................................................................... 39  
   Job History Command ................................................................................................................ 41  
   Host Monitoring Command .......................................................................................................... 42  
   Job Control Command ................................................................................................................. 44  
   Job Synchronization Command .................................................................................................... 45  
   User Monitoring Command ......................................................................................................... 46  
   Accounting Command ................................................................................................................ 47  
   JSDL To GridWay Job Template Parser Command ........................................................................... 48  
5. Troubleshooting ............................................................................................................................. 49  
   1. Debugging job execution .......................................................................................................... 49  
   2. Frequent problems .................................................................................................................. 49
List of Figures

1.1. Simplified state machine of the GridWay Metascheduler. ................................................................. 2
4.1. Workflow example. ............................................................................................................................ 32
4.2. Dag graph generated by the gwdag tool. ........................................................................................... 35
# List of Tables

3.1. Job Template options. ..................................................................................................................... 8
3.2. Substitution variables. ................................................................................................................... 12
3.3. Variables that can be used to define the job REQUIREMENTS and RANK. .............................................. 14
3.4. JSDL vs GWJT ................................................................. 19
5. Field options .................................................................................................................................. 40
6. Field information ............................................................................................................................ 41
7. Field information ............................................................................................................................ 42
8. Queue field information ................................................................................................................... 43
9. Field information ............................................................................................................................ 46
10. Field information .......................................................................................................................... 47
Chapter 1. Introduction

1. Benefits for the end user

GridWay, on top of Globus services, enables large-scale, secure and reliable sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by different resource management systems (PBS, SGE, LSF, Condor...), within a single organization (enterprise grid) or scattered across several administrative domains (partner or supply-chain grid). GridWay provides the end-user with a working environment and functionality similar to those found on local DRM systems, such as SGE, LSF or PBS. The end-user is able to submit, monitor and control his jobs by means of DRM-like commands (gwsubmit, gwwait, gwkill, gwhosts...) or the DRMAA API.

The benefits for the end user are:

• Reliable and unattended execution of jobs: Transparently to the end user, the scheduler is able to manage the different failure situations.

• Efficient execution of jobs: Jobs are executed on the faster available resources.

• Broad application scope: GridWay is not bounded to a specific class of application generated by a given programming environment and does not require application deployment on remote hosts, which extends its application range and allows reusing of existing software. GridWay allows submission of single, array or complex jobs consisting of task dependencies, which may require file transferring and/or database access.

• DRM Command Line Interface: The GridWay command line interface is similar to that found on Unix and resource management systems such as PBS or SGE. It allows users to submit, kill, migrate, monitor and synchronize jobs. Moreover, jobs can be specified in GridWay Job Template format or using the JSDL (Job Submission Description Language) OGF standard.

• DRMAA Application Programming Interface: GridWay provides full support for DRMAA (OGF standard) to develop distributed applications (C, Java, Perl, Ruby and Python bindings).

2. How GridWay operates

GridWay enables you to treat your jobs as if they were Unix processes. Each job is given a numerical identifier, analogous to the PID of a process. This value is called the Job identifier, JID for short. If the job belongs to an array job, it will also have an array identifier, AID for short. A job index within an array is called the task identifier, TID for short.

Jobs are submitted using the gwsubmit command. A job is described by its template file. Here you can specify the job's executable file, its command line arguments, input/output files, standard stream redirection as well as other aspects.

Jobs can be monitored using the gwps command. You can control your jobs at runtime using the gwkill command. You can synchronize your jobs using the gwwait command. You can find out what resources your job has used with the gwhistory command.

System monitoring commands allow you to gather information of the GridWay system and the Grids you are using. These commands are: gwuser to show information about the users using GridWay; gwhost to monitor the available hosts in the testbed; and gwacct to print usage (accounting) information per user or host.

Note

Every command has a -h option which shows its usage and available options.
3. Job life-cycle in GridWay

A job can be in one of the following dispatch states (DM state):

- **Pending** (*pend*): The job is waiting for a resource to run on. The job reaches this state when it is initially submitted by the user or when it is restarted after a failure, stop or self-migration.

- **Hold** (*hold*): The owner (or GridWay administrator) has held the job. It will not be scheduled until it receives a release signal.

- **Prolog** (*prol*): The job is preparing the remote system, by creating the execution directory in the remote host and transferring the input and restart (in case of migration) files to it.

- **Pre-wraper** (*prew*): The job is making some advanced preparation tasks in the remote resource, like getting some data from a service, obtaining software licenses, etc.

- **Wrapper** (*wrap*): The job is executing the Wrapper, which in turns executes the actual application. It also starts a self-monitoring program if specified. This monitor, watches the raw performance (CPU usage) obtained by the application.

- **Epilog** (*epil*): The job is finalizing. In this phase it transfers the output and restart (in case of failure, stop or self-migration) files and cleaning up the remote system directory.

- **Migrate** (*migr*): The job is migrating from one resource to another, by canceling the execution of Wrapper and performing finalization tasks in the old resource (like in Epilog state) and preparation tasks in the new resource (like in Prolog state).

- **Stopped** (*stop*): The job is stopped. If restart files have been defined in the Job Template, they are transferred back to the client, and will be used when the job is resumed.

- **Failed** (*fail*): The job failed.

- **Done** (*done*): The job is done and the user can check the exit status.

**Figure 1.1. Simplified state machine of the GridWay Metascheduler.**

When a job is in Wrapper dispatch state, it can be in one of the following execution states (EM state), which are a subset of the available Globus GRAM states:

- **Pending** (*pend*): The job has been successfully submitted to the local DRM system and it is waiting for the local DRM system to execute it.

- **Suspended** (*susp*): The job has been suspended by the local DRM system.
• Active (active): The job is being executed by the local DRM system
• Failed (fail): The job failed.
• Done (done): The job is done.

Finally, the following flags are associated with a job (RWS flags):
• Restarted (R): Number of times the job was restarted or migrated.
• Waiting (W): Number of clients waiting for this job to end.
• Rescheduled (S): 1 if this job is waiting to be rescheduled, 0 otherwise.

4. A grid-aware application model

In order to obtain a reasonable degree of both application performance and fault tolerance, a job must be able to adapt itself according to the availability of the resources and the current performance provided by them. Therefore, the classical application model must be extended to achieve such functionality.

The GridWay system assumes the following application model:

• **Executable**: The executable must be compiled for the remote host architecture. GridWay provides a straightforward method to select the appropriate executable for each host. The variable GW_ARCH, as provided by the Information MAD, can be used to define the executable in the Job Template (for example, EXECUTABLE=sim_code.$(GW_ARCH))

• **Input files**: These files are staged to the remote host. GridWay provides a flexible way to specify input files and supports Parameter Sweep like definitions. Please note that these files may be also architecture dependent.

• **Output files**: These files are generated on the remote host and transferred back to the client once the job has finished.

• **Standard streams**: The standard input (STDIN) file is transferred to the remote system previous to job execution. Standard output (STDOUT) and standard error (STDERR) streams are also available at the client once the job has finished. These files could be extremely useful for debugging.

• **Restart files**: Restart files are highly advisable if dynamic scheduling is performed. User-level checkpointing managed by the programmer must be implemented because system-level checkpointing is not possible among heterogeneous resources.

Migration is commonly implemented by restarting the job on the new candidate host. Therefore, the job should generate restart files at regular intervals in order to restart execution from a given point. However, for some application domains the cost of generating and transferring restart files could be greater than the saving in compute time due to checkpointing. Hence, if the checkpointing files are not provided the job is restarted from the beginning. In order not to reduce the number of candidate hosts where a job can migrate, the restart files should be architecture independent.
Chapter 2. User environment configuration

1. Environment variables for GridWay

⚠️ Important

You should include the following environment variables in your shell configuration file. (example $HOME/.bashrc)

In order to set the user environment, follow these steps:

1. Set up Globus user environment:

   $ source $GLOBUS_LOCATION/etc/globus-user-env.sh

   or

   $ . $GLOBUS_LOCATION/etc/globus-user-env.csh

   depending on the shell you are using.

2. Set up the GridWay user environment:

   $ export GW_LOCATION=<path_to_GridWay_installation>
   $ export PATH=$PATH:$GW_LOCATION/bin

   or

   $ setenv GW_LOCATION <path_to_WG_location>
   $ setenv PATH $PATH:$GW_LOCATION/bin

   depending on the shell you are using.

3. Optionally, you can set up your environment to use the GridWay DRMAA library:

   $ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GW_LOCATION/lib

   or:

   $ setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$GW_LOCATION/lib

4. If GridWay has been compiled with accounting support, you may need to set up the DB library. For example, if DB library has been installed in /usr/local/BerkeleyDB.4.4:

   $ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/BerkeleyDB.4.4/lib
5. DRMAA extensions for all the languages use the dynamic drmaa libraries provided by GridWay. To use these libraries, it is needed to tell the operating system where to look for them. Here are described the steps needed to do this in Linux and MacOS X.

- 1. In Linux, we have two ways to do this, one is using environment variables and the other one is modifying systemwide library path configuration. You only need to use one of these methods. If you do not have root access to the machine you are using or you do not want to setup it for every user in your system you have to use the environment variable method.

- 1.1 The environment variable you have to set so the extensions find the required DRMAA library is LD_LIBRARY_PATH with a line similar to:

```
export LD_LIBRARY_PATH=$GW_LOCATION/lib
```

If you want to setup this systemwide you can put this line alongside GW_LOCATION setup into /etc/profile. If you do not have root access or you want to do it per user the best place to do it is in the user's .bashrc.

You can also do this steps in the console before launching your scripts as it will have the same effect.

- Systems that use GNU/libc (GNU/Linux is one of them) do have a systemwide configuration file with the paths where to look for dynamic libraries. You have to add this line to /etc/ld.so.conf:

```
<path_to_gridway_installation>/lib
```

After doing this you have to rebuild the library cache issuing this command:

```
# ldconfig
```

- In MacOS X you have to use the environment variable method described for Linux but this time the name of the variable is DYLD_LIBRARY_PATH.
Chapter 3. Functionality

1. Job description overview

Job Templates allow you to configure your job requirements, in terms of needed files, generated files, requirements and ranks of execution hosts, as well as other options.

Syntax:

<VARIABLE> = ["]<VALUE>["]
# <Comments>

⚠️ Important

Default values for EVERY Job Template are read from $GW_LOCATION/etc/job_template.default.
2. Job Template options
Table 3.1. Job Template options.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Name of the job (filename of the Job Template by default).</td>
</tr>
<tr>
<td>Execution</td>
<td>The executable file. Example: EXECUTABLE = bin.${ARCH}</td>
</tr>
<tr>
<td></td>
<td>Arguments to the above executable. Example: ARGUMENTS = &quot;${TASK_ID}&quot;</td>
</tr>
<tr>
<td></td>
<td>User defined, comma-separated, environment variables. Example: ENVIRONMENT = SCRATCH_DIR /tmp, LD_LIBRARY_PATH=/usr/local/lib</td>
</tr>
<tr>
<td></td>
<td>Type of job. Possible values are &quot;single&quot; (default), &quot;multiple&quot; and &quot;mpi&quot;,</td>
</tr>
<tr>
<td></td>
<td>with similar behaviour to that of GRAM jobs.</td>
</tr>
<tr>
<td></td>
<td>Number of processors in MPI jobs. For &quot;multiple&quot; and &quot;single&quot; jobs it</td>
</tr>
<tr>
<td></td>
<td>defines the &quot;count&quot; parameter in the RSL.</td>
</tr>
<tr>
<td>I/O files</td>
<td>A comma-separated pair of &quot;local remote&quot; filenames. If the remote filename</td>
</tr>
<tr>
<td></td>
<td>is missing, the local filename will be preserved in the execution host.</td>
</tr>
<tr>
<td></td>
<td>Example: INPUT_FILES = param.${TASK_ID} param, inputfile</td>
</tr>
<tr>
<td></td>
<td>A comma-separated pair of remote filename local filename. If the local</td>
</tr>
<tr>
<td></td>
<td>filename is missing, the remote filename will be preserved in the client</td>
</tr>
<tr>
<td></td>
<td>host. Example: OUTPUT_FILES = outputfile, binary binary.${ARCH}.${TASK_ID}</td>
</tr>
<tr>
<td></td>
<td>Standard input file. Example: STDIN_FILE = /dev/null</td>
</tr>
<tr>
<td></td>
<td>Standard output file. Example: STDOUT_FILE = stdout_file.${JOB_ID}</td>
</tr>
<tr>
<td></td>
<td>Standard error file. Example: STDERR_FILE = stderr_file.${JOB_ID}</td>
</tr>
<tr>
<td></td>
<td>Checkpointing</td>
</tr>
<tr>
<td></td>
<td>Checkpoint Files. These files are managed by the programmer and should</td>
</tr>
<tr>
<td></td>
<td>be architecture independent (NO URLS HERE, you can use a checkpoint</td>
</tr>
<tr>
<td></td>
<td>server using CHECKPOINT_URL). Example: RESTART_FILES = checkpoint</td>
</tr>
<tr>
<td></td>
<td>How often (seconds) restart files are transferred from the execution host</td>
</tr>
<tr>
<td></td>
<td>to the checkpointing server.</td>
</tr>
<tr>
<td></td>
<td>GridFTP URL to store/access checkpoint files (Default is log directory in</td>
</tr>
<tr>
<td></td>
<td>localhost). Example: CHECKPOINT_URL = gsiftp://hydrus.ucm.es/var/checkpoints/</td>
</tr>
<tr>
<td></td>
<td>Resource selection (See Section 5, “Resource selection expressions”, for</td>
</tr>
<tr>
<td></td>
<td>more information)</td>
</tr>
<tr>
<td></td>
<td>A Boolean expression evaluated for each available host, if the evaluation</td>
</tr>
<tr>
<td></td>
<td>returns true the host will be considered to submit the job. Example:</td>
</tr>
<tr>
<td></td>
<td>REQUIREMENTS = ARCH = &quot;i686&quot; &amp; CPU_MHZ &gt; 1000;</td>
</tr>
<tr>
<td></td>
<td>A numerical expression evaluated for each candidate host (those for which</td>
</tr>
<tr>
<td></td>
<td>the requirement expression is true). Those candidates with higher ranks</td>
</tr>
<tr>
<td></td>
<td>are used first to execute your jobs. Example: RANK = (CPU_MHZ * 2) + FREE_MEM_MB;</td>
</tr>
</tbody>
</table>
### Scheduling

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESCHEDULING_INTERVAL</td>
<td>How often GridWay searches the Grid for better resources to run this job. (0 = never)</td>
</tr>
<tr>
<td>RESCHEDULING_THRESHOLD</td>
<td>If a better resource is found and the job has been running less than this threshold (in seconds), it will migrate to the new host.</td>
</tr>
<tr>
<td>DEADLINE</td>
<td>Deadline (format [[D[:H:]M] to start the job (0 = none).</td>
</tr>
</tbody>
</table>

### Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSPENSION_TIMEOUT</td>
<td>Maximum suspension time (seconds) in the local job management system. If exceeded the job is migrated to another host. (0 = never)</td>
</tr>
<tr>
<td>CPULOAD_THRESHOLD</td>
<td>If the CPU assigned to your job is less than this given percentage, the job will be migrated</td>
</tr>
<tr>
<td>MONITOR</td>
<td>Optional program to monitor job performance</td>
</tr>
</tbody>
</table>

### Fault tolerance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESCHEDULE_ON_FAILURE</td>
<td>Behavior in case of failure. Possible values are 'yes' or 'no'</td>
</tr>
<tr>
<td>NUMBER_OF_RETRIES</td>
<td>Number of retries in case of failure. GridWay follows a linear backoff strategy for re-trying file transfers and job submissions.</td>
</tr>
</tbody>
</table>

### Advanced job execution

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRAPPER</td>
<td>Script for wrapper. stdout and stderr streams of this program can be found in directory $GW_LOCATION/var/$GW_JOB_ID as files stdout.wrapper.$GW_RESTARTED and stderr.wrapper.$GW_RESTARTED</td>
</tr>
<tr>
<td>PRE_WRAPPER</td>
<td>Optional program that is executed before the execution of the job, to perform an additional remote setup (e.g. access a web service). This job is ALWAYS submitted to the FORK job-manager. stdout and stderr streams of this program can be found in directory $GW_LOCATION/var/$GW_JOB_ID as files stdout.pre_wrapper.$GW_RESTARTED and stderr.pre_wrapper.$GW_RESTARTED</td>
</tr>
<tr>
<td>PRE_WRAPPER_ARGUMENTS</td>
<td>Arguments to the pre-wrapper program.</td>
</tr>
</tbody>
</table>

### 3. File definition in Job Templates

Input and output files are in general specified in a comma-separated, source/destination pair.

**SRC1 DST1, SRC2 DST2, ...**

We next describe the available alternatives and protocols that you can use to choose the best staging strategy for your applications.

### 3.1. Defining input files

Input files (if staged to the remote host) are always placed in the remote experiment directory. However you can specify the name used for the file in the remote directory with the destination name (DST above). This feature is very useful when your executable always expect a fixed input filename, and you want to process different input files (as is common in parametric computations).
### Important

The destination names for input files MUST be a single name, do not use absolute paths or URLs.

You can specify the input files using:

- **Absolute path**: In this case no staging is performed. File is assumed to be in that location in the remote host.
  
  **Example:**
  ```
  EXECUTABLE = /bin/ls  #Will use remote ls!
  ```

- **GridFTP URL**: The file will be downloaded from the given GridFTP url. If no destination is given the filename in the URL will be used in the remote host.

  **Example**, will copy file `input_exp1` from `/tmp` in machine to the remote host with name `input`.
  ```
  INPUT_FILES = gsiftp://machine/tmp/input_exp1 input
  ```

- **File URL**: The file will copied from an absolute path in the local host. If no destination is given the filename in the URL will be used in the remote host.

  **Example**:
  ```
  INPUT_FILES = file:///etc/passwd #Will copy local /etc/passwd file to remote dir, with
  ```

- **Name**: Use simple names to stage files in your local experiment directory (directory where the Job Template file is placed). If no destination is given the filename will be preserved.

  **Example**:
  ```
  INPUT_FILES = test_case.bin
  ```

#### Note

The executable is treated as an input file, so the same remarks are applicable for the `EXECUTABLE` Job Template parameter.

### 3.2. Defining output files

Output files are always copied FROM the remote experiment directory. However you can specify the destination of the output files of your applications.

### Important

The source names for output files MUST be a single name, do not use absolute paths or URLs.

You can specify the destination of output files using:
Functionality

- **Absolute path**: The remote file name will be copied to the absolute path in the local host. Note that you can also use `file:///` protocol.

  Example, to copy the output file `output.bin` in the `/tmp` directory of the local host with name `outfile`:

  ```
  OUTPUT_FILES  = output.bin /tmp/outfile
  ```

- **GridFTP URL**: The file will be copied to the given GridFTP url.

  Example, you can also use variable substitution in URLs, see next section.

  ```
  OUTPUT_FILES  = out gsiftp://storage_servere/~/output.${TASK_ID}
  ```

- **Name**: Use simple names to stage files to your local experiment directory (directory where the Job Template file is placed). If no destination is given the filename will be preserved.

  Example:

  ```
  OUTPUT_FILES  = test_case.bin
  ```

  Tip: You can organize your output files in directories in the experiment directory (They MUST exist!), using a relative path

  ```
  OUTPUT_FILES = outfile Out/file.${JOB_ID} #Directory Out must exist in the experiment directory
  ```

### 3.3. Defining standard streams

Standard streams includes the standard input for your executable (`STDIN_FILE`) and its standard output and error (`STDOUT_FILE` and `STDERR_FILE`).

The STDIN_FILE can be defined using any of the methods described above for the input files. However you can not specified a destination name for the standard input stream, as is internally handled by the system. Note also that only one standard input file can be specified.

Example:

```
STDIN_FILE = In/input.${JOB_ID} #Will use input from directory In
```

The STDOUT_FILE and STDERR_FILE parameters can be defined using any of the methods described above for the output files. However you can not specified a source name (only destination). Note also that only one standard output and error files can be specified.

Example:

```
STDOUT_FILE = Out/ofile #Will place stdout in Out directory with name ofile
```
3.4. Defining restart files

Restart files are periodically copied to the job directory ($GW_LOCATION/var/$JOB_ID/). Restart files are only specified with its name. Note also that you can defined a checkpointing server with the CHECKPOINT_URL Job Template parameter.

Example:

```bash
RESTART_FILES = tmp_file
```

4. Variable substitution

You can use variables in the value string of each option, with the format:

```
${GW_VARIABLE}
```

These variables are substituted at run time with its corresponding value. For example:

```bash
STDOUT_FILE = stdout.${JOB_ID}
```

will store the standard output of job 23 in the file `stdout.23`

The following table lists the variables available to define job options, along with their description.

### Table 3.2. Substitution variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>${JOB_ID}</td>
<td>The job identifier</td>
</tr>
<tr>
<td>${ARRAY_ID}</td>
<td>The job array identifier (-1 if the job does not belong to any)</td>
</tr>
<tr>
<td>${TASK_ID}</td>
<td>The task identifier within the job array (-1 if the job does not belong to any)</td>
</tr>
<tr>
<td>${TOTAL_TASKS}</td>
<td>The total number of tasks in the job array (-1 if the job does not belong to any)</td>
</tr>
<tr>
<td>${ARCH}</td>
<td>The architecture of the selected execution host</td>
</tr>
<tr>
<td>${PARAM}</td>
<td>Allows the assignment of arbitrary start and increment values for array jobs (start + increment*GW_TASK_ID). Useful to generate file naming patterns or task processing. The values for start and increment will be specified with options <code>-s</code> (for start, with 0 by default) and <code>-i</code> (for increment, with 1 by default) of the <code>gwsubmit</code> command.</td>
</tr>
<tr>
<td>${MAX_PARAM}</td>
<td>Upper bound of the ${PARAM} variable.</td>
</tr>
</tbody>
</table>

⚠️ Important

The above variables can be used in any job option.

5. Resource selection expressions

5.1. Requirement expression syntax

The syntax of the requirement expressions is defined as:
stmt ::= expr';
expr ::= VARIABLE '==' INTEGER
    | VARIABLE '>'; INTEGER
    | VARIABLE '<' INTEGER
    | VARIABLE '==' STRING
    | expr '&' expr
    | expr '|' expr
    | '!' expr
    | '(' expr ')'
Table 3.3. Variables that can be used to define the job REQUIREMENTS and RANK.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOSTNAME</td>
<td>FQDN (Fully Qualified Domain Name) of the execution host (e.g. &quot;hydrus.dacya.ucm.es&quot;)</td>
</tr>
<tr>
<td>ARCH</td>
<td>Architecture of the execution host (e.g. &quot;i686&quot;, &quot;alpha&quot;)</td>
</tr>
<tr>
<td>OS_NAME</td>
<td>Operating System name of the execution host (e.g. &quot;Linux&quot;, &quot;SL&quot;)</td>
</tr>
<tr>
<td>OS_VERSION</td>
<td>Operating System version of the execution host (e.g. &quot;2.6.9-1.66&quot;, &quot;3&quot;)</td>
</tr>
<tr>
<td>CPU_MODEL</td>
<td>CPU model of the execution host (e.g. &quot;Intel(R) Pentium(R) 4 CPU 2&quot;, &quot;PIV&quot;)</td>
</tr>
<tr>
<td>CPU_MHZ</td>
<td>CPU speed in MHZ of the execution host</td>
</tr>
<tr>
<td>CPU_FREE</td>
<td>Percentage of free CPU of the execution host</td>
</tr>
<tr>
<td>CPU_SMP</td>
<td>CPU SMP size of the execution host</td>
</tr>
<tr>
<td>NODECOUNT</td>
<td>Total number of nodes of the execution host</td>
</tr>
<tr>
<td>SIZE_MEM_MB</td>
<td>Total memory size in MB of the execution host</td>
</tr>
<tr>
<td>FREE_MEM_MB</td>
<td>Free memory in MB of the execution hosts</td>
</tr>
<tr>
<td>SIZE_DISK_MB</td>
<td>Total disk space in MB of the execution hosts</td>
</tr>
<tr>
<td>FREE_DISK_MB</td>
<td>Free disk space in MB of the execution hosts</td>
</tr>
<tr>
<td>LRMS_NAME</td>
<td>Name of local DRM system (job manager) for execution, usually not fork (e.g. &quot;jobmanager-pbs&quot;, &quot;PBS&quot;, &quot;jobmanager-sge&quot;, &quot;SGE&quot;)</td>
</tr>
<tr>
<td>LRMS_TYPE</td>
<td>Type of local DRM system for execution (e.g. &quot;pbs&quot;, &quot;sge&quot;)</td>
</tr>
<tr>
<td>QUEUE_NAME</td>
<td>Name of the queue (e.g. &quot;default&quot;, &quot;short&quot;, &quot;dteam&quot;)</td>
</tr>
<tr>
<td>QUEUE_NODECOUNT</td>
<td>Total node count of the queue</td>
</tr>
<tr>
<td>QUEUE_FREENODECOUNT</td>
<td>Free node count of the queue</td>
</tr>
<tr>
<td>QUEUE_MAXTIME</td>
<td>Maximum wall time of jobs in the queue</td>
</tr>
<tr>
<td>QUEUE_MAXCPUTIME</td>
<td>Maximum CPU time of jobs in the queue</td>
</tr>
<tr>
<td>QUEUE_MAXCOUNT</td>
<td>Maximum count of jobs that can be submitted in one request to the queue</td>
</tr>
<tr>
<td>QUEUE_MAXRUNNINGJOBS</td>
<td>Maximum number of running jobs in the queue</td>
</tr>
<tr>
<td>QUEUE_MAXJOBSINQUEUE</td>
<td>Maximum number of queued jobs in the queue</td>
</tr>
<tr>
<td>QUEUE_DISPATCHTYPE</td>
<td>Dispatch type of the queue (e.g. &quot;batch&quot;, &quot;inmediate&quot;)</td>
</tr>
<tr>
<td>QUEUE_PRIORITY</td>
<td>Priority of the queue</td>
</tr>
<tr>
<td>QUEUE_STATUS</td>
<td>Status of the queue (e.g. &quot;active&quot;, &quot;production&quot;)</td>
</tr>
</tbody>
</table>

5.4. Job environment

Job environment variables can be easily set with the ENVIRONMENT parameter of the Job Template. These environment variables are parsed, so you can use the GridWay variables defined in Section 4, "Variable substitution", to set the job environment.

Note

The variables defined in the ENVIRONMENT are "sourced" in a bash shell. In this way you can take advantage of the bash substitution capabilities and built-in functions. For example:
ENVIRONMENT = VAR = "\`expr \${JOB_ID} + 3\"" # will set VAR to JOB_ID + 3

In addition to those variables set in the `ENVIRONMENT` parameter, GridWay set the following variables, that can be used by your applications:

- GW_RESTARTED
- GW_EXECUTABLE
- GW_HOSTNAME
- GW_ARCH
- GW_CPU_MHZ
- GW_MEM_MB
- GW_RESTART_FILES
- GW_CPULOAD_THRESHOLD
- GW_ARGUMENTS
- GW_TASK_ID
- GW_CPU_MODEL
- GW_ARRAY_ID
- GW_TOTAL_TASKS
- GW_JOB_ID
- GW_OUTPUT_FILES
- GW_INPUT_FILES
- GW_OS_NAME
- GW_USER
- GW_DISK_MB
- GW_OS_VERSION

### 5.5. Monitor and Wrapper Scripts

If you want to specify your own monitor or wrapper script you can do it using `WRAPPER` or `MONITOR` variables in your job template (or modifying the default one located in the etc directory of your GW installation). The file you specify must be a full path name of the script or a relative path if it is located inside `$GW_LOCATION`. Here you can not use gsiftp or file url protocol prefixes.
6. Job Submission Description Language (JSDL)

6.1. JSDL overview

GridWay supports Job Submission Description Language (JSDL). This specification is a language for describing the job requirements for submission to resources. The JSDL language specification is based on XML Schema that facilitate the expression of those requirements as a set of XML elements. More info at https://forge.gridforum.org/sf/projects/jsdl-wg.

6.2. JSDL document structure

The JSDL document structure is as follows:

```xml
<JobDefinition>
  |--<JobDescription>
    |---<JobIdentification>
    |    |--<JobName>?
    |    |--<Description>?
    |    |--<JobAnnotation>*
    |    |--<JobProject>*
    |    |--<xsd:any##other>*
    |    </JobIdentification>?
    |--<Application>
    |    |--<ApplicationName>?
    |    |--<ApplicationVersion>?
    |    |--<Description>?
    |    |--<xsd:any##other>*
    |    </Application>?
    |--<Resources>?
    |    |--<CandidateHosts>
    |    |    |--<HostName>+
    |    |    </CandidateHosts>?
    |    |--<FileSystem>
    |    |    |--<Description>?
    |    |    |--<MountPoint>?
    |    |    |--<MountSource>?
    |    |    |--<DiskSpace>?
    |    |    |--<FileSystemType>?
    |    |    |--<xsd:any##other>*
    |    |    </FileSystem>*
    |    |--<ExclusiveExecution>?
    |    |--<OperatingSystem>?
    |    |    |--<OperatingSystemType>
    |    |    |    |--<OperatingSystemName>
    |    |    |    |--<xsd:any##other>*
    |    |    |    </OperatingSystemName>
    |    |    |    |--<xsd:any##other>*
    |    |    </OperatingSystemType>*
    |    |--<OperatingSystemVersion>?
    |    |--<Description>?
    |    |--<xsd:any##other>*
    |    </OperatingSystem>?
</JobDefinition>
```

1 http://forge.gridforum.org/sf/projects/jsdl-wg
6.3. JSDL POSIX application

This schema defines the JSDL specification for describing an application executed on a POSIX compliance system. Due to GridWay Job Template specification, this schema MUST be included in the JSDL file. The JSDL POSIX Application Schema is as follow:

```xml
<POSIXApplication name="xsd:NCName">
    <Executable/>
    <Argument/>
    <Input/>
</POSIXApplication>
```

Note

The symbol "?" denotes zero or one occurrences, "*" denotes zero or more occurrences and "+" denotes one or more occurrences.
6.4. JSDL HPC Profile Application Extension

This schema defines the JSDL specification for describing a simple HPC application that is made up of an executable file running within an operating system process. It shares much in common with the JSDL POSIXApplication. The JSDL HPC Application Schema is as follow:

```xml
<HPCProfileApplication name="xsd:NCName">
    |-------<Executable>?
    |-------<Argument>*
    |-------<Input>?
    |-------<Output>?
    |-------<Error>?
    |-------<WorkingDirectory>?
    |-------<Environment>*
    |-------<WallTimeLimit>?
    |-------<FileSizeLimit>?
    |-------<CoreDumpLimit>?
    |-------<DataSegmentLimit>?
    |-------<LockedMemoryLimit>?
    |-------<MemoryLimit>?
    |-------<OpenDescriptorsLimit>?
    |-------<PipeSizeLimit>?
    |-------<StackSizeLimit>?
    |-------<CPUTimeLimit>?
    |-------<ProcessCountLimit>?
    |-------<VirtualMemoryLimit>?
    |-------<ThreadCountLimit>?
    |-------<UserName>?
    |-------<GroupName>?
</HPCProfileApplication name="xsd:NCName">
```

More details at https://forge.gridforum.org/sf/projects/jsdl-wg

6.5. Job Submission Description Language versus GridWay Job Template

Next table compares JSDL and GridWay Job Template schema, and the JSDL elements support by the current GridWay version.
Table 3.4. JSDL vs GWJT

<table>
<thead>
<tr>
<th>JSDL Element</th>
<th>GWJT Attribute</th>
<th>Adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobDefinition</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>JobDescription</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>JobIdentification</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>JobName</td>
<td>NAME</td>
<td>Supported</td>
</tr>
<tr>
<td>JobAnnotation</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>JobProject</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>Application</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>ApplicationVersion</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>Description</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>Resources</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>CandidateHosts</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>HostName</td>
<td>HOSTNAME</td>
<td>Supported</td>
</tr>
<tr>
<td>FileSystem</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>MountPoint</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>MountSource</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>DiskSpace</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>FileSysType</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>ExclusiveExecution</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>OperatingSystem</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>OperatingSysType</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>OperatingSysName</td>
<td>OS_NAME</td>
<td>Supported</td>
</tr>
<tr>
<td>OperatingSysVersion</td>
<td>OS_VERSION</td>
<td>Supported</td>
</tr>
<tr>
<td>CPUArchitecture</td>
<td>-</td>
<td>Supported</td>
</tr>
<tr>
<td>CPUArchName</td>
<td>ARCH</td>
<td>Supported</td>
</tr>
<tr>
<td>IndividualCPUSpeed</td>
<td>CPU_MHZ</td>
<td>Supported</td>
</tr>
<tr>
<td>IndividualCPUTime</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>IndividualCPUCount</td>
<td>NODECOUNT</td>
<td>Supported</td>
</tr>
<tr>
<td>IndividualNetworkBandwidth</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>IndividualPhysicalMemory</td>
<td>MEM_MB</td>
<td>Supported</td>
</tr>
<tr>
<td>IndividualVirtualMemory</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>IndividualDiskSpace</td>
<td>SIZE_DISK_MB</td>
<td>Supported</td>
</tr>
<tr>
<td>TOTALCPUTime</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>TOTALCPUCount</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>TOTALPhysicalMemory</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>TOTALVirtualMemory</td>
<td>-</td>
<td>Not supported</td>
</tr>
<tr>
<td>TOTALDiskSpace</td>
<td>-</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
6.6. Examples

6.6.1. A simple example

This example shows the representation of a simple job in JSDL format and the translator of this example in Gridway Job Template format.
6.6.1.1. JSDL file

<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdl:JobDescription>
    <jsdl:JobName>Simple Application GW Template vs JSDL</jsdl:JobName>
    <jsdl:Description>This is a simple example to describe the main
differences between GW Template and the JSDL schema.</jsdl:Description>
  </jsdl:JobIdentification>
  <jsdl:Application>
    <jsdl:ApplicationName>ls</jsdl:ApplicationName>
    <jsdl-posix:POSIXApplication>
      <jsdl-posix:Executable>/bin/ls</jsdl-posix:Executable>
      <jsdl-posix:Argument>-la file.txt</jsdl-posix:Argument>
      <jsdl-posix:Environment name="LD_LIBRARY_PATH">/usr/local/lib</jsdl-posix:Environment>
      <jsdl-posix:Input>/dev/null</jsdl-posix:Input>
      <jsdl-posix:Output>stdout.${JOB_ID}</jsdl-posix:Output>
      <jsdl-posix:Error>stderr.${JOB_ID}</jsdl-posix:Error>
    </jsdl-posix:POSIXApplication>
  </jsdl:Application>
  <jsdl:Resources>
    <jsdl:CandidateHost>
      <jsdl:HostName>*.dacya.ucm.es</jsdl:HostName>
    </jsdl:CandidateHost>
    <jsdl:CPUArchitecture>
      <jsdl:CPUArchitectureName>x86_32</jsdl:CPUArchitectureName>
    </jsdl:CPUArchitecture>
  </jsdl:Resources>
  <jsdl:DataStaging>
    <jsdl:FileName>file.txt</jsdl:FileName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    <jsdl:Source>
      <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/file1.txt</jsdl:URI>
    </jsdl:Source>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>stdout.${JOB_ID}</jsdl:FileName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    <jsdl:Target>
      <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stdout.${JOB_ID}</jsdl:URI>
    </jsdl:Target>
  </jsdl:DataStaging>
  <jsdl:DataStaging>
    <jsdl:FileName>stderr.${JOB_ID}</jsdl:FileName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
    <jsdl:Target>
      <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stderr.${JOB_ID}</jsdl:URI>
    </jsdl:Target>
  </jsdl:DataStaging>
</jsdl:JobDefinition>
6.6.1.2. GridWay Job Template file

#This file was automatically generated by the JSDL2GWJT parser
EXECUTABLE=/bin/ls
ARGUMENTS=-la file.txt
STDIN_FILE=/dev/null
STDOUT_FILE=stdout.$(JOB_ID)
STDERR_FILE=stderr.$(JOB_ID)
ENVIRONMENT=LD_LIBRARY_PATH=/usr/local/lib
REQUIREMENTS=HOSTNAME="*.dacya.ucm.es" & ARCH="x86_32"
INPUT_FILES=file.txt

6.6.2. A HPC profile example

This example shows the representation of a HPC profile job in JSDL format and the translator of this example in Gridway Job Template format.

6.6.2.1. JSDL file

<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/">
  <jsdl:JobIdentification>
    <jsdl:JobName>Simple Application GW Template vs JSDL</jsdl:JobName>
    <jsdl:Description>This is a simple example to describe the main differences between GW Template and the JSDL schema.</jsdl:Description>
  </jsdl:JobIdentification>
  <jsdl:Application>
    <jsdl:ApplicationName>ls</jsdl:ApplicationName>
    <jsdl-hpcpa:HPCProfileApplication>
      <jsdl-hpcpa:Executable>/bin/ls</jsdl-hpcpa:Executable>
      <jsdl-hpcpa:Argument>-la file.txt</jsdl-hpcpa:Argument>
      <jsdl-hpcpa:Environment name="LD_LIBRARY_PATH">/usr/local/lib</jsdl-hpcpa:Environment>
      <jsdl-hpcpa:Input>/dev/null</jsdl-hpcpa:Input>
      <jsdl-hpcpa:Output>stdout.$(JOB_ID)</jsdl-hpcpa:Output>
      <jsdl-hpcpa:Error>stderr.$(JOB_ID)</jsdl-hpcpa:Error>
    </jsdl-hpcpa:HPCProfileApplication>
  </jsdl:Application>
</jsdl:JobDefinition>
<jsdl:Resources>
  <jsdl:CandidateHost>
    <jsdl:HostName>*.dacya.ucm.es</jsdl:HostName>
  </jsdl:CandidateHost>
  <jsdl:CPUArchitecture>
    <jsdl:CPUArchitectureName>x86_32</jsdl:CPUArchitectureName>
  </jsdl:CPUArchitecture>
</jsdl:Resources>

<jsdl:DataStaging>
  <jsdl:FileName>file.txt</jsdl:FileName>
  <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
  <jsdl:Source>
    <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/file1.txt</jsdl:URI>
  </jsdl:Source>
</jsdl:DataStaging>

<jsdl:DataStaging>
  <jsdl:FileName>stdout.${JOB_ID}</jsdl:FileName>
  <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
  <jsdl:Target>
    <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stdout.${JOB_ID}</jsdl:URI>
  </jsdl:Target>
</jsdl:DataStaging>

<jsdl:DataStaging>
  <jsdl:FileName>stderr.${JOB_ID}</jsdl:FileName>
  <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
  <jsdl:Target>
    <jsdl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stderr.${JOB_ID}</jsdl:URI>
  </jsdl:Target>
</jsdl:DataStaging>

<jsdl:JobDescription>
  EXECUTABLE=/bin/ls
  ARGUMENTS=-la file.txt
  STDIN_FILE=/dev/null
  STDOUT_FILE=stdout.${JOB_ID}
  STDERR_FILE=stderr.${JOB_ID}
  ENVIRONMENT=LD_LIBRARY_PATH=/usr/local/lib
  REQUIREMENTS=HOSTNAME="*.dacya.ucm.es" & ARCH="x86_32"
  INPUT_FILES=file.txt
</jsdl:JobDefinition>

6.6.2.2. GridWay Job Template file
Chapter 4. Usage Scenarios

1. Single jobs: Submitting and monitoring the simplest job

GWD should be configured and running. Check the Installation and Configuration Guide and Chapter 2, User environment configuration to do that. Do not forget to create a proxy with `grid-proxy-init`.

To submit a job, you will need a Job Template. The most simple Job Template in GridWay could be:

```
EXECUTABLE=/bin/ls
```

Save it as file `jt` in directory `example`.

Use the `gwsubmit` command to submit the job:

```
$ gwsubmit -t example/jt
```

Let see how many resources are available in our Grid, with `gwhost`:

```
HID  PRIO  OS              ARCH   MHZ  %CPU  MEM(F/T)     DISK(F/T)  N(U/F/T) LRMS  HOSTNAME
0   1    Linux2.6.17-2-6  x86   3215  100  923/2027 105003/118812  0/1/2  Fork  cygnus.dacya.ucm.es
1   1    Linux2.6.17-2-6  x86   3216  189  384/2027 105129/118812  0/2/2  Fork  draco.dacya.ucm.es
2   1    Linux2.6.18-3-a  x86_6  2211  100  749/1003  76616/77844  0/2/2  SGE   aquila.dacya.ucm.es
3   1                             0    0      0/0           0/0     0/0/0       hydrus.dacya.ucm.es
4   1    Linux2.6.18-3-a  x86_6  2009   74  319/878 120173/160796  0/1/1  Fork  orion.dacya.ucm.es
5   1    Linux2.6.16.13- x86   3200  100  224/256       114/312     0/6/6  SGE   ursa.dacya.ucm.es
```

Note that hydrus is down in this example, so no information is received at all and of course, this host won’t be considered in future scheduling decisions. If you want to retrieve more information about a single resource, issue the `gwhost` command followed by the host identification (HID):

```
$ gwhost 0
HID  PRIO  OS              ARCH   MHZ  %CPU  MEM(F/T)     DISK(F/T)  N(U/F/T) LRMS  HOSTNAME
5   1    Linux2.6.16.13- x86   3200  100  224/256     114/312  0/6/6  SGE   ursa.dacya.ucm.es
```

```
QUEUENAME SL(F/T) WALLT CPUT  COUNT MAXR  MAXQ  STATUS   DISPATCH   PRIORITY
all.q       6/6     0     0     0     6     0     enabled  NULL       0
```

You can also check the resources that match your requirements with `gwhost -m 0`:

```
HID  QNAME      RANK  PRIO  SLOTS  HOSTNAME
0   default    0     1     2     cygnus.dacya.ucm.es
1   default    0     1     0     draco.dacya.ucm.es
2   all.q      0     1     2     aquila.dacya.ucm.es
4   default    0     1     1     orion.dacya.ucm.es
5   all.q      0     1     6     ursa.dacya.ucm.es
```

24
Now, you can check the evolution of the job with the **gwps** command.

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>pend</td>
<td>----</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>--</td>
<td>jt</td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>prol</td>
<td>----</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:00</td>
<td>0:00:01</td>
<td>--</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>wrap</td>
<td>----</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:27</td>
<td>0:00:04</td>
<td>--</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>wrap</td>
<td>pend</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:27</td>
<td>0:00:04</td>
<td>--</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>wrap</td>
<td>actv</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:27</td>
<td>0:00:04</td>
<td>--</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>epil</td>
<td>----</td>
<td>000</td>
<td>10:42:09</td>
<td>--:--:--</td>
<td>0:00:31</td>
<td>0:00:05</td>
<td>--</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USER</th>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>RWS</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>jlvazquez</td>
<td>0</td>
<td>done</td>
<td>----</td>
<td>000</td>
<td>10:42:09</td>
<td>10:43:01</td>
<td>0:00:31</td>
<td>0:00:08</td>
<td>0</td>
<td>jt</td>
<td>cygnus.dacya.ucm.es/Fork</td>
</tr>
</tbody>
</table>

At the beginning, the job is in *pending* state and not allocated to any resource. Then, the job is allocated to cygnus.dacya.ucm.es/Fork and begins the *prolog* stage.

**Note**

You can use option `--c <delay>` to see a continuous output of the **gwps** command.

You can see the job history with the **gwhistory** command:

```
$ gwhistory 0
HID START    END      PROLOG  WRAPPER EPILOG  MIGR    REASON QUEUE    HOST
0 10:42:22 10:43:01 0:00:04 0:00:31 0:00:04 0:00:00 ----   default  cygnus.dacya.ucm.es/Fork
```

Now it's time to retrieve the results. As you specified by default, the results of the execution of this job will be in the same folder, in a text file called sdtout_file.$JOB_ID.

```
$ ls -lt example/
total 8
-rw-r--r-- 1 jlvazquez staff 0 2007-02-20 10:42 stderr.0
-rw-r--r-- 1 jlvazquez staff 72 2007-02-20 10:42 stdout.0
-rw-r--r-- 1 jlvazquez staff 19 2007-02-20 10:33 jt
```

```
$ cat example/stdout.0
job.env
stderr.execution
stderr.wrapper
stdout.execution
stdout.wrapper
```

Done! You have done your first execution with GridWay!
2. Array jobs: Calculating the number

2.1. Defining the problem

This is a well known exercise. For our purposes, we will calculate the integral of the following function:

\[
\int_{a}^{b} f(x) \, dx
\]

Being \( f(x) = \frac{4}{1+x^2} \). So, \( \pi \) will be the integral of \( f(x) \) in the interval \([0,1]\).

In order to calculate the whole integral, it's interesting to divide the function in several sections and compute them separately:

As you can see, the more sections you make, the more exact \( \pi \) will be:
So, you have a Grid with some nodes, you have GridWay... Why don't use them to calculate the π number by giving all the nodes a section to compute with only one command?

⚠️ Note

You will find all the files needed to perform this example in the $GW_LOCATION/examples/pi directory.

### 2.2. The coding part

For this example, we have chosen the C Programming Language. Create a text file called `pi.c` and copy inside the following lines:

```c
#include <stdio.h>
#include <string.h>
int main (int argc, char** args)
{
    int task_id;
    int total_tasks;
    long long int n;
    long long int i;
    double l_sum, x, h;

    task_id = atoi(args[1]);
    total_tasks = atoi(args[2]);
    n = atoll(args[3]);

    fprintf(stderr, "task_id=%d total_tasks=%d n=%lld\n", task_id, total_tasks, n);
    h = 1.0/n;
    l_sum = 0.0;

    for (i = task_id; i < n; i += total_tasks)
    {
        x = (i + 0.5)*h;
        l_sum += 4.0/(1.0 + x*x);
    }
}
```
Now it's time to compile it:

$ gcc -O3 pi.c -o pi

after this, you should have an executable called pi. This command receives three parameters:

- Task identifier: The identifier of the current task.
- Total tasks: The number of tasks the computation should be divided into.
- Number of intervals: The number of intervals over which the integral is being evaluated.

### 2.3. Defining the job

For making GridWay work with your program, you must create a Job Template. In this case, we will call it pi.jt. Copy the following lines inside:

```bash
EXECUTABLE  = pi
ARGUMENTS   = ${TASK_ID} ${TOTAL_TASKS} 100000
STDOUT_FILE = stdout_file.${TASK_ID}
STDERR_FILE = stderr_file.${TASK_ID}
RANK        = CPU_MHZ
```

### 2.4. Submitting the jobs

This time, we will submit an array of jobs. This is done by issuing the following command:

$ gws/submit -v -t pi.jt -n 4

ARRAY ID: 0

<table>
<thead>
<tr>
<th>TASK</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

In order to wait for the jobs to complete, you can use the `gwwait` command.

The argument passed to `gwwait` is the array identifier given by `gws/submit` when executed with the `-v` option. It could be also obtained through `gwps`.

This command will block and return when all jobs have been executed.
This command, when issued with option \(-v\) shows the exit codes for each job in the array (usually, 0 means success).

## 2.5. Result post-processing

The execution of these jobs has returned some output files with the result of each execution:

```
stdout_file.0
stdout_file.1
stdout_file.2
stdout_file.3
```

Now, we will need something to sum the results inside each file. For this, you can use an `awk` script like the following:

```
$ awk 'BEGIN {sum=0} {sum+=$1} END {printf "Pi is %0.12g\n", sum}' stdout_file.*
Pi is 3.1415926536
```

Well, not much precision, right? You could try it again, but this time with a much higher number of intervals (e.g. 10,000,000,000). Would you increment also the number of tasks? Which would be the best compromise?

Do you imagine how easy would be to implement these steps in a shell script in order to perform them unattendedly? Here you are the prove:

```
#!/bin/sh

AID=`gwsubmit -v -t pi.jt -n 4 | head -1 | awk '{print $3}'`

if [ $? -ne 0 ]
then
    echo "Submission failed!"
    exit 1
fi

gwait -v -A $AID

if [ $? -eq 0 ]
then
    awk 'BEGIN {sum=0} {sum+=$1} END {printf "Pi is %0.12g\n", sum}' stdout_file.*
else
    echo "Some tasks failed!"
fi
```
3. MPI jobs: Calculating the number again

When applications show fine-grain parallelism, with small computation to communication ratio, and thus need lower latencies, MPI (Message Passing Interface) jobs give a better choice.

Following the π example, we will now perform its computation using MPI in a single job. Notice that MPI jobs can also be part of an array or complex job.

Note

You will find all the files needed to perform this example in the $GW_LOCATION/examples/mpi directory.

Create a text file called mpi.c and copy inside the following lines:

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);
    printf("Process %d on %s\n", myid, processor_name);

    n = 100000000;
    startwtime = MPI_Wtime();

    h = 1.0 / (double) n;
    sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs)
    {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
    }
    mypi = h * sum;

    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (myid == 0)
    {
        printf("pi is approximately %.16f, Error is %.16f\n", pi, pi - PI25DT);
    }
}
```

Usage Scenarios
\begin{verbatim}
    pi, fabs(pi - PI25DT));
    endwtime = MPI_Wtime();
    printf("wall clock time = %f\n", endwtime-startwtime);
}

MPI_Finalize();

return 0;
\end{verbatim}

\section*{Note}

For more information about MPI, see http://www.mcs.anl.gov/mpi.

Notice that the above program already performs postprocessing in a single operand reduction operation MPI_Reduce which sums the partial results obtained by each processor.

Now it's time to compile it. Notice that you will need a compiler with MPI support like \texttt{mpicc}:

\$ mpicc -O3 mpi.c -o mpi

Now you must create a Job Template. In this case, we will call it \texttt{mpi.jt}:

\begin{verbatim}
EXECUTABLE    = mpi
STDOUT_FILE   = stdout.${JOB_ID}
STDERR_FILE   = stderr.${JOB_ID}
RANK          = CPU_MHZ
TYPE          = "mpi"
NP            = 2
\end{verbatim}

\section*{4. Workflows}

The powerful commands provided by GridWay to submit, control and synchronize jobs allow us to programmatically define complex jobs or workflows, where some jobs need data generated by other jobs. GridWay allows job submission to be dependent on the completion of other jobs. This new functionality provides support for the execution of workflows.

GridWay allows scientists and engineers to express their computational problems by using workflows. The capture of the job exit code allows users to define workflows, where each task depends on the output and exit code from the previous task. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the workflow of certain type of applications. The bash script flow control structures and the GridWay commands allow the development of workflows with the following functionality:

- Sequence, parallelism, branching and looping structures
- The workflow can be described in an abstract form without referring to specific resources for task execution
- Quality of service constraints and fault tolerance are defined at task level
Job dependencies can be specified at submission by using the `-d` option of the `gwsubmit` command. A job with dependencies will be submitted in the hold state, and once all the jobs on which it depends have successfully finished, it will be released. You can also release this job by hand with the `gwkill`.

### 4.1. A sample of DAG workflow

A DAG-based workflow consists of a temporal relationship between tasks, where the input, output or execution of one or more tasks depends on one or more other tasks. For this example we have chosen a simple workflow.

**Figure 4.1. Workflow example.**

In this example, job A generates a random number, jobs B and C add 1 to that number and, finally job D adds the result of these jobs. This is the final result is two times the number generated by A, plus two. In our case, the numbers are passed between jobs using the standard output files.

Job Template for job A (`A.jt`):

```
EXECUTABLE=/bin/echo
ARGUMENTS="$RANDOM"
STDOUT_FILE=out.A
```

Job Template for jobs B and C (`B.jt` and `C.jt`):

```
EXECUTABLE=/usr/bin/expr
ARGUMENTS="`cat out.A` + 1"
INPUT_FILES=out.A
STDOUT_FILE=out.B #out.C for job C
```

Job Template for job D (`D.jt`):

```
EXECUTABLE=/usr/bin/expr
ARGUMENTS="`cat out.B` + "`cat out.C`"
INPUT_FILES=out.B, out.C
STDOUT_FILE=out.workflow
```

Once you have set up the previous Job Templates, the workflow can be easily submitted with the following commands:

```
$ gwsubmit -v -t A.jt
JOB ID: 5
```
$ gwsubmit -v -t B.jt -d "5"
JOB ID: 6

$ gwsubmit -v -t C.jt -d "5"
JOB ID: 7

$ gwsubmit -t C.jt -d "6 7"

Note

In the previous example, jobs B and C can be submitted as an array job using just one template with output, OUTPUT_FILES = out.${TASK_ID}. Therefore, input of job D will be INPUT_FILES = out.0, out.1.

The above steps can be easily implemented in a shell script.

```bash
#!/bin/sh
A_ID=`gwsubmit -v -t A.jt | cut -f2 -d':' | cut -f2 -d' '`
B_ID=`gwsubmit -v -t B.jt -d "$A_ID" | cut -f2 -d':'' | cut -f2 -d' ''`
C_ID=`gwsubmit -v -t C.jt -d "$A_ID" | cut -f2 -d':'' | cut -f2 -d' ''`
D_ID=`gwsubmit -v -t D.jt -d "$B_ID $C_ID"| cut -f2 -d':'' | cut -f2 -d' ''`

#Sync with last job of the workflow
gwwait $D_ID

echo "Random number `cat out.A`"
echo "Workflow computation `cat out.workflow`"
```

Note that when input and output files vary depending on the iteration or job id number, you should generate Job Templates dynamically before submitting each job. This can be done programmatically by using the DRMAA API, or via shell scripting.

4.1.1. Using gwdag tool

Alternatively you can describe DAG workflows using a file similar to the one used by Condor DAGMAN. In this case the dependencies are not managed by GW but by the gwdag tool. You only have to substitute Condor job descriptions with GridWay job templates. Here is a file describing the same DAG as the previous example:

```
JOB A A.jt
JOB B B.jt
JOB C C.jt
JOB D D.jt
PARENT A CHILD B C
PARENT B C CHILD D
```

To submit this job you only have to specify the file describing this DAG to gwdag tool:

```
$ gwdag <name of the file>
```
You can also get a DOT file for a DAG description that you can use later to generate a graph showing the flow using -d flag:

$ gwdag -d <name of the file> > <name of the dot file>
Figure 4.2. Dag graph generated by the gwdag tool.
GridWay Commands
Name

Job and Array Job submission Command -- job submission utility for the GridWay system

gwsubmit <-t template> [-n tasks] [-h] [-v] [-o] [-s start] [-i increment] [-d "id1 id2 ..."]

Description

Submit a job or an array job (if the number of tasks is defined) to gwd

Command options

- **h**
  Prints help.

- **t <template>**
  The template file describing the job.

- **n <tasks>**
  Submit an array job with the given number of tasks. All the jobs in the array will use the same template.

- **s <start>**
  Start value for custom param in array jobs. Default 0.

- **i <increment>**
  Increment value for custom param in array jobs. Each task has associated the value \( \text{PARAM} = \text{start} + \text{increment} \times \text{TASK_ID} \), and \( \text{MAX_PARM} = \text{start} + \text{increment} \times (\text{tasks}-1) \). Default 1.

- **d <"id1 id2...">**
  Job dependencies. Submit the job on hold state, and release it once jobs with id1,id2,.. have successfully finished.

- **v**
  Print to stdout the job ids returned by gwd.

- **o**
  Hold job on submission.

- **p <priority>**
  Initial priority for the job.
**Name**

DAG Job submission Command -- dag job submission utility for the GridWay system

`gwdag [-h] [-d] <DAG description file>`

**Description**

Submit a dag job to gwd

**Command options**

- `-h` Prints help.
- `-d` Writes to STDOUT a DOT description for the specified DAG job.
Name

Job Monitoring Command -- report a snapshot of the current jobs

gwps [-h] [-u user] [-r host] [-A AID] [-s job_state] [-o output_format] [-c delay] [-n] [job_id]

Description

Prints information about all the jobs in the GridWay system (default)

Command options

-h Prints help.
-u user Monitor only jobs owned by user.
-r host Monitor only jobs executed in host.
-A AID Monitor only jobs part of the array AID.
-s job_state Monitor only jobs in states matching that of job_state.
-o output_format Formats output information, allowing the selection of which fields to display.
-c <delay> This will cause gwps to print job information every <delay> seconds continuously (similar to top command).
-n Do not print the header.
-job_id Only monitor this job_id.
## Output field description

### Table 5. Field options

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>FIELD OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>u</td>
<td>owner of this job</td>
</tr>
<tr>
<td>JID</td>
<td>J</td>
<td>job unique identification assigned by the Gridway system</td>
</tr>
<tr>
<td>AID</td>
<td>i</td>
<td>array unique identification, only relevant for array jobs</td>
</tr>
<tr>
<td>TID</td>
<td>i</td>
<td>task identification, ranges from 0 to TOTAL_TASKS -1, only relevant for array jobs</td>
</tr>
<tr>
<td>FP</td>
<td>p</td>
<td>fixed priority of the job</td>
</tr>
<tr>
<td>TYPE</td>
<td>y</td>
<td>type of job (simple, multiple or mpi)</td>
</tr>
<tr>
<td>NP</td>
<td>n</td>
<td>number of processors</td>
</tr>
<tr>
<td>DM</td>
<td>s</td>
<td>dispatch Manager state, one of: pend, hold, pro, pro, wrap, epil, canl, stop, migr, done, fail</td>
</tr>
<tr>
<td>EM</td>
<td>e</td>
<td>execution Manager state (Globus state): pend, susp, actv, fail, done</td>
</tr>
<tr>
<td>RWS</td>
<td>f</td>
<td>flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S</td>
</tr>
<tr>
<td>START</td>
<td>t/T</td>
<td>the time the job entered the system</td>
</tr>
<tr>
<td>END</td>
<td>t/T</td>
<td>the time the job reached a final state (fail or done)</td>
</tr>
<tr>
<td>EXEC</td>
<td>t/T</td>
<td>total execution time, includes suspension time in the remote queue system</td>
</tr>
<tr>
<td>XFER</td>
<td>t/T</td>
<td>total file transfer time, includes stage-in and stage-out phases</td>
</tr>
<tr>
<td>EXIT</td>
<td>x</td>
<td>job exit code</td>
</tr>
<tr>
<td>TEMP-PLATE</td>
<td>j</td>
<td>filename of the job template used for this job</td>
</tr>
<tr>
<td>HOST</td>
<td>h</td>
<td>hostname where the job is being executed</td>
</tr>
</tbody>
</table>
Name

Job History Command -- shows history of a job

gwhistory [-h] [-n] <job_id>

Description

Prints information about the execution history of a job

Command options

-\h \ Ampersand  h \  Prints help.

-\n \ Ampersand  n \  Do not print the header lines

job\_id \  Job identification as provided by gwps.

Output field description

Table 6. Field information

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HID</td>
<td>host unique identification assigned by the Gridway system.</td>
</tr>
<tr>
<td>START</td>
<td>the time the job start its execution on this host.</td>
</tr>
<tr>
<td>END</td>
<td>the time the job left this host, because it finished or it was migrated.</td>
</tr>
<tr>
<td>PROLOG</td>
<td>total prolog (file stage-in phase) time.</td>
</tr>
<tr>
<td>WRAPPER</td>
<td>total wrapper (execution phase) time.</td>
</tr>
<tr>
<td>EPILOG</td>
<td>total epilog (file stage-out phase) time.</td>
</tr>
<tr>
<td>MIGR</td>
<td>total migration time.</td>
</tr>
<tr>
<td>REASON</td>
<td>the reason why the job left this host.</td>
</tr>
<tr>
<td>QUEUE</td>
<td>name of the queue.</td>
</tr>
<tr>
<td>HOST</td>
<td>FQDN of the host.</td>
</tr>
</tbody>
</table>
Name
Host Monitoring Command -- shows hosts information

gwhost [-h] [-c delay] [-nf] [-m job_id] [host_id]

Description
Prints information about all the hosts in the GridWay system (default)

Command options

-h  Prints help.
-c <delay>  This will cause gwhost to print job information every <delay> seconds continuously (similar to top command)
-n  Do not print the header.
-f  Full format.
-m <job_id>  Prints hosts matching the requirements of a given job.

host_id  Only monitor this host_id, also prints queue information.

Output field description

Table 7. Field information

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HID</td>
<td>host unique identification assigned by the Gridway system</td>
</tr>
<tr>
<td>PRIO</td>
<td>priority assigned to the host</td>
</tr>
<tr>
<td>OS</td>
<td>operating system</td>
</tr>
<tr>
<td>ARCH</td>
<td>architecture</td>
</tr>
<tr>
<td>MHZ</td>
<td>CPU speed in MHZ</td>
</tr>
<tr>
<td>%CPU</td>
<td>free CPU ratio</td>
</tr>
<tr>
<td>MEM(F/T)</td>
<td>system memory: F = Free, T = Total</td>
</tr>
<tr>
<td>DISK(F/T)</td>
<td>secondary storage: F = Free, T = Total</td>
</tr>
<tr>
<td>N(U/F/T)</td>
<td>number of slots: U = used by GridWay, F = free, T = total</td>
</tr>
<tr>
<td>LRMS</td>
<td>local resource management system, the jobmanager name</td>
</tr>
<tr>
<td>HOSTNAME</td>
<td>FQDN of this host</td>
</tr>
</tbody>
</table>
Table 8. Queue field information

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUEUENAME</td>
<td>name of this queue</td>
</tr>
<tr>
<td>SL(F/T)</td>
<td>slots: F = Free, T = Total</td>
</tr>
<tr>
<td>WALLT</td>
<td>queue wall time</td>
</tr>
<tr>
<td>CPUT</td>
<td>queue cpu time</td>
</tr>
<tr>
<td>COUNT</td>
<td>queue count number</td>
</tr>
<tr>
<td>MAXR</td>
<td>max. running jobs</td>
</tr>
<tr>
<td>MAXQ</td>
<td>max. queued jobs</td>
</tr>
<tr>
<td>STATUS</td>
<td>queue status</td>
</tr>
<tr>
<td>DISPATCH</td>
<td>queue dispatch type</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>queue priority</td>
</tr>
</tbody>
</table>
Name

Job Control Command -- controls job execution

gwkill [-h] [-a] [-k | -t | -o | -s | -r | -l | -9] <job_id [job_id2 ...] | -A array_id>

Description

Sends a signal to a job or array job

Command options

-h Prints help.

-a Asynchronous signal, only relevant for KILL and STOP.

-k Kill (default, if no signal specified).

-t Stop job.

-r Resume job.

-o Hold job.

-l Release job.

-s Re-schedule job.

-9 Hard kill, removes the job from the system without synchronizing remote job execution or cleaning remote host.

job_id [job_id2 ...] Job identification as provided by gwps. You can specify a blank space separated list of job ids.

-A <array_id> Array identification as provided by gwps.
Name
Job Synchronization Command -- synchronize a job

gwait [-h] [-a] [-v] [-k] <job_id ...| -A array_id>

Description
Waits for a job or array job

Command options
- h  Prints help.
- a  Any, returns when the first job of the list or array finishes.
- v  Prints job exit code.
- k  Keep jobs, they remain in fail or done states in the GridWay system. By default, jobs are killed and their resources freed.
- A <array_id>     Array identification as provided by gwps.
job_id ...       Job ids list (blank space separated).
Name

User Monitoring Command -- monitors users in GridWay

gwuser [-h] [-n]

Description

Prints information about users in the GridWay system

Command options

- h  Prints help.
- n  Do not print the header.

Output field description

Table 9. Field information

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>user unique identification assigned by the GridWay system</td>
</tr>
<tr>
<td>NAME</td>
<td>name of this user</td>
</tr>
<tr>
<td>JOBS</td>
<td>number of Jobs in the GridWay system</td>
</tr>
<tr>
<td>RUN</td>
<td>number of running jobs</td>
</tr>
<tr>
<td>IDLE</td>
<td>idle time, (time with JOBS = 0)</td>
</tr>
<tr>
<td>EM</td>
<td>execution manager drivers loaded for this user</td>
</tr>
<tr>
<td>TM</td>
<td>transfer manager drivers loaded for this user</td>
</tr>
<tr>
<td>PID</td>
<td>process identification of driver processes</td>
</tr>
</tbody>
</table>
Name
Accounting Command -- prints accounting information

gwacct [-h] [-n] [<-d n | -w n | -m n | -t s>] <-u user|-r host>

Description
Prints usage statistics per user or resource. Note: accounting statistics are updated once a job is killed.

Command options
-h Prints help.
-n Do not print the header.
<-d n | -w n | -m n | -t s> Take into account jobs submitted after certain date, specified in number of days (-d), weeks (-w), months (-m) or an epoch (-t).
-u user Print usage statistics for user.
-r hostname Print usage statistics for host.

Output field description

Table 10. Field information

<table>
<thead>
<tr>
<th>FIELD</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST/USER</td>
<td>host/user usage summary for this user/host</td>
</tr>
<tr>
<td>XFR</td>
<td>total transfer time on this host (for this user)</td>
</tr>
<tr>
<td>EXE</td>
<td>total execution time on this host (for this user), without suspension time</td>
</tr>
<tr>
<td>SUSP</td>
<td>total suspension (queue) time on this host (for this user)</td>
</tr>
<tr>
<td>TOTS</td>
<td>total executions on this host (for this user). Termination reasons:</td>
</tr>
<tr>
<td></td>
<td>• SUCC success</td>
</tr>
<tr>
<td></td>
<td>• ERR error</td>
</tr>
<tr>
<td></td>
<td>• KILL kill</td>
</tr>
<tr>
<td></td>
<td>• USER user requested</td>
</tr>
<tr>
<td></td>
<td>• SUSP suspension timeout</td>
</tr>
<tr>
<td></td>
<td>• DISC discovery timeout</td>
</tr>
<tr>
<td></td>
<td>• SELF self migration</td>
</tr>
<tr>
<td></td>
<td>• PERF performance degradation</td>
</tr>
<tr>
<td></td>
<td>• S/R stop/resume</td>
</tr>
</tbody>
</table>
Name
JSDL To GridWay Job Template Parser Command -- parser to translate JSDL file into GridWay Job Template file

```
jsdl2gw [-h] input_jsdl [output_gwjt]
```

Description
Converts a jsdl document into a gridway job template. If no output file is defined, it defaults to the standard output. This enables the use of pipes with gwsubmit in the following fashion:

```
jsdl2gw jsdl-job.xml | gwsubmit
```

Command options

- `-h` Prints help.

- `input_jsdl` Reads the jsdl document from the input_jsdl

- `output_gwjt` Stores the GridWay Job Template specification in the output_gwjt.jt file
Chapter 5. Troubleshooting

1. Debugging job execution

GridWay reporting and accounting facilities provide information about overall performance and help debug job execution. GWD generates the following files under the $GW\_LOCATION/var directory:

- **gwd.log**: System level log. You can find log information of the activity of the middleware access drivers; and a coarse-grain log information about jobs.

- **$JOB\_ID/job.log**: Detailed log information for each job, it includes details of job state transitions, resource usage and performance.

- **$JOB\_ID/stdout.wrapper**: Standard output of the wrapper executable.

- **$JOB\_ID/stderr.wrapper**: Standard error output of the wrapper executable. By default, wrapper is executed with shell debugging options (-xv) active, so this is usually the best source of information in case of failure.

2. Frequent problems

Currently, many errors are handled silently and are only shown in the job.log file.

Also, there is a number of failures related to the underlying middleware (Globus in this case) that could make some jobs fail. It is a good idea to perform some basic testing of Globus when some jobs unexpectedly fail (see the Installation and Configuration Guide to learn how to verify a Globus installation)

Use the GridWay lists (see www.gridway.org/support.php), to submit your problems and they will be eventually appear in this guide.