

# **GT 4.2.1 GridWay: System Administrator's Guide**

---

## GT 4.2.1 GridWay: System Administrator'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><sup>1</sup>

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.

---

<sup>1</sup> <http://www.apache.org/licenses/LICENSE-2.0>

---

---

# Table of Contents

1. Introduction .....	1
1. GridWay Architecture .....	1
2. Meta-scheduling Infrastructures with GridWay .....	2
2. Building and installing .....	5
1. Verifying Globus Installation .....	5
2. Required Software .....	6
3. Platform Notes .....	6
4. Installing GridWay .....	7
3. Configuration Guide .....	12
1. Core Configuration Guide .....	12
2. Scheduler Configuration Guide .....	16
3. MAD Configuration Guide .....	26
4. Integration Guides .....	33
4. Deploying .....	34
5. Testing .....	35
1. Verifying the installation .....	35
2. Test Suite .....	36
3. DRMAA Test Suite .....	38
6. Security considerations .....	39
1. Security Considerations for GridWay .....	39
7. Debugging .....	40
1. Logging in Java WS Core .....	40
8. Troubleshooting .....	42
1. Errors .....	42
2. Debugging .....	42

---

## List of Figures

1.1. Components of the GridWay Meta-scheduler .....	1
1.2. Enterprise Grid deployment with GridWay. ....	2
1.3. Partner Grid deployment with GridWay. ....	3
1.4. Grid Federation with GridWay and GridGateWays. ....	4
3.1. Job Scheduling in GridWay .....	17
3.2. Job and resource prioritization policies in GridWay. ....	18
3.3. Dispatch priority of a job .....	20
3.4. Estimated execution time of a job on a resource .....	22
3.5. Banned time formula .....	22
3.6. Suitability priority of a resource .....	23

---

# List of Tables

- 2.1. Configure Options. .... 8
- 3.1. GWD Configuration File Options. .... 14
- 3.2. Built-in Scheduler Configuration File Options. .... 24
- 5.1. GridWay tests description. .... 37
- 8.1. Gridway Errors ..... 42

---

# Chapter 1. Introduction

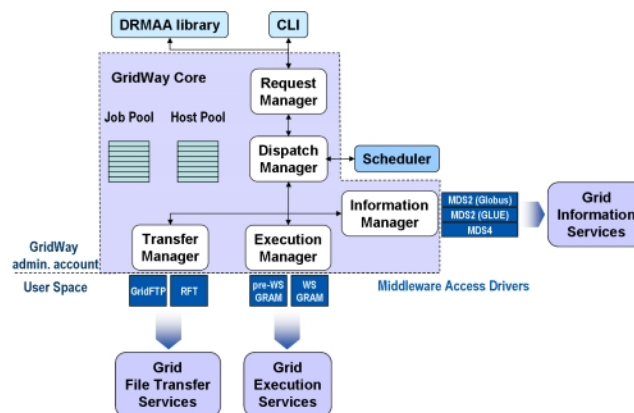
This guide contains installation and configuration information for system administrators installing GridWay. It explains how to install, configure and test the installation.

## ! Important

This information is in addition to the basic Globus Toolkit prerequisite, overview, installation, security configuration instructions in the [Installing GT 4.2.1](#). Read through this guide before continuing!

## 1. GridWay Architecture

Figure 1.1. Components of the GridWay Meta-scheduler.



GridWay 5 architecture consists of the following components:

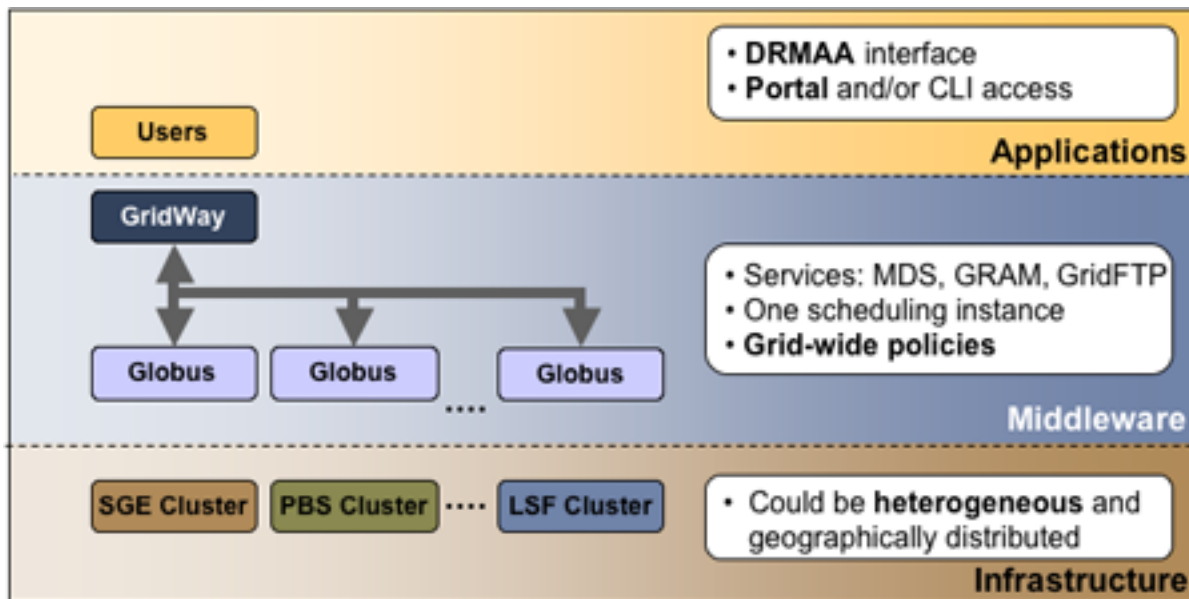
- *User Interface* provides the end user with DRM-like commands to submit, kill, migrate, monitor and synchronize jobs and includes DRMAA (Distributed Resource Management Application API) GGF (Global Grid Forum) standard support to develop distributed applications (C and JAVA bindings).
- *GridWay core* is responsible for job execution management and resource brokering, providing advanced scheduling, and job failure & recovery capabilities. The Dispatch Manager performs all submission stages and watches over the efficient execution of the job. The Information Manager, through its MADs (Middleware Access Driver), is responsible for host discovery and monitoring. The Execution Manager, through its MADs, is responsible job execution and management. The Transfer Manager, through its MADs, is responsible for file staging, remote working directory set-up and remote host clean-up.
- *Scheduler* makes scheduling decisions for jobs on available resources.
- *Information Manager MAD* interfaces with the monitoring and discovering services available in the Grid infrastructure.
- *Execution Manager MAD* interfaces with the Job Management Services available in the Grid resources.
- *Transfer Manager MAD* interfaces with the Data Management Services available in the Grid resources.

## 2. Meta-scheduling Infrastructures with GridWay

### 2.1. Enterprise Grid Infrastructures

Enterprise grids enable diverse resource sharing to improve internal collaboration and achieve a better return from information technology investment. Available resources within a company are better exploited and the administrative overhead is minimized by using Grid technology. The resources are part of the same administrative domain. These infrastructures require a centralized approach for scheduling and accounting. The administrator must be able to apply centralized usage policies and access to global reporting and accounting. Enterprise grid infrastructures require meta-schedulers to provide support for multiple users in a single scheduling instance.

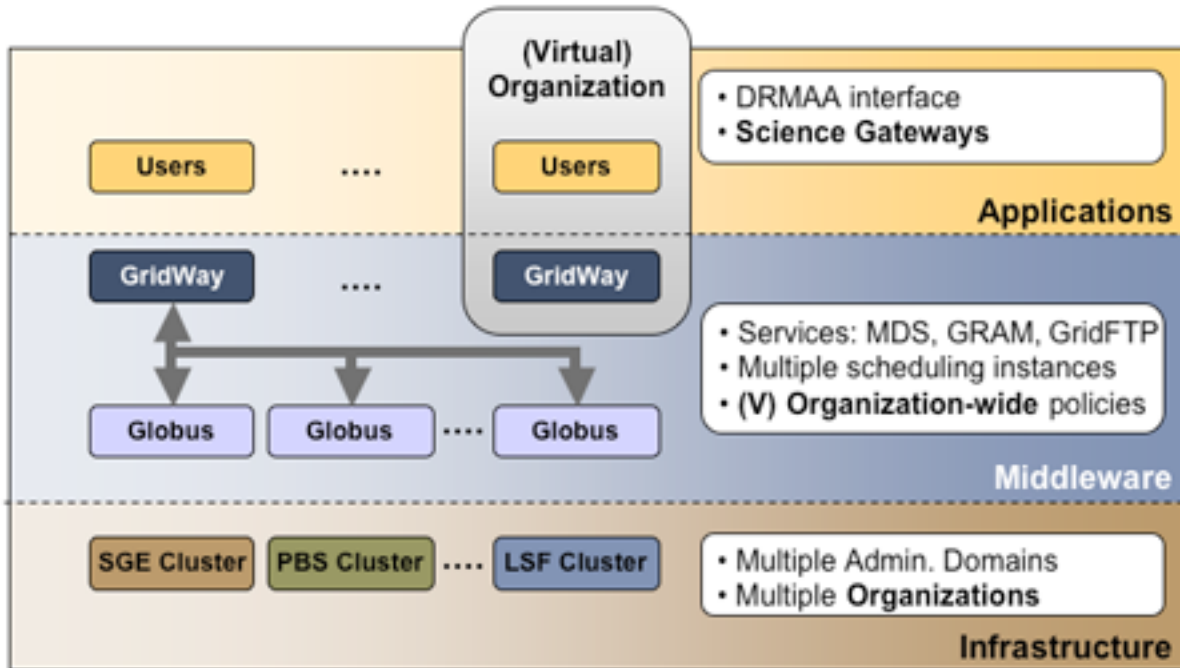
**Figure 1.2. Enterprise Grid deployment with GridWay.**



### 2.2. Partner Grid Infrastructures

Partner grid infrastructures of several scales are being deployed within the context of different research projects, whose final goal is to provide large-scale, secure and reliable sharing of resources among partner organizations and supply-chain participants. Such partner grids allow access to a higher computing performance to satisfy peak demands and also provide support to face collaborative projects. The multiple administration domains existing in a partner grid infrastructure prevent the deployment of centralized meta-schedulers, with total control over client requests and resource status. Organization-level meta-schedulers provide support for multiple intra-organization users in each scheduling instance. This means that there is one scheduling instance for each organization, and all scheduling instances compete with each other for the available resources.

Figure 1.3. Partner Grid deployment with GridWay.

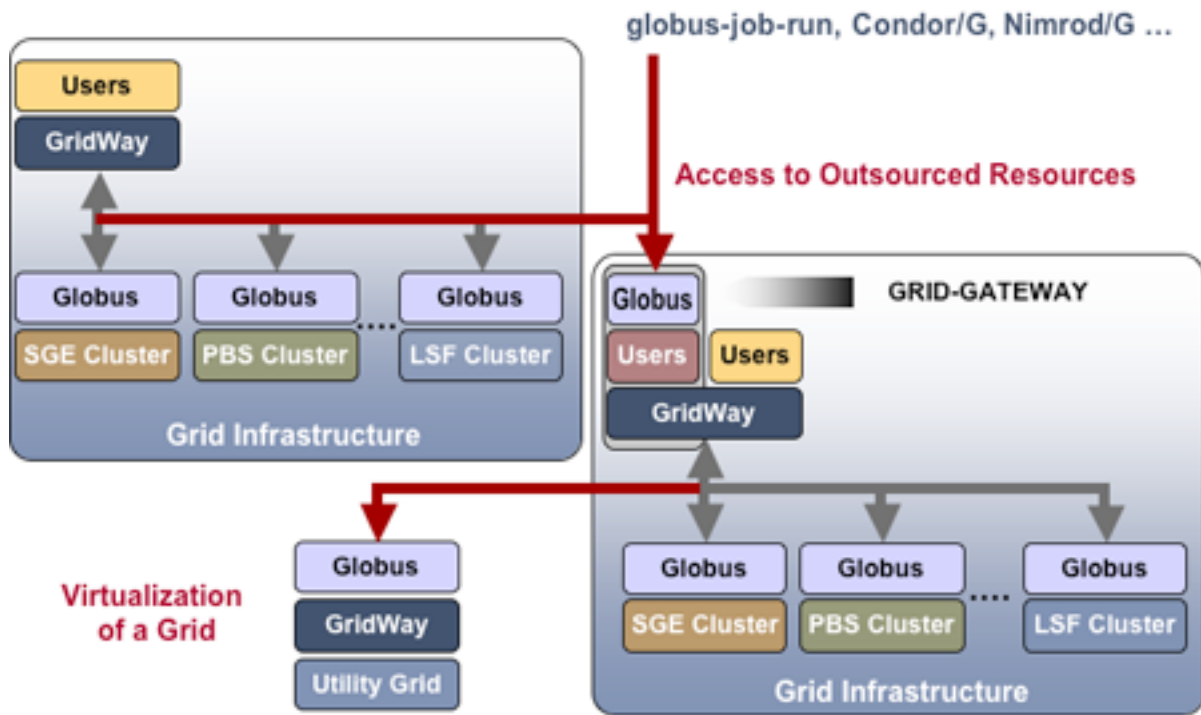


## 2.3. Federation of Grid Infrastructures

Please visit the web page about [Federation of Grid Infrastructures and Utility Computing](http://www.gridway.org/research/gridfederation.php)<sup>1</sup> to find out how to deploy federated grid infrastructures using Globus and GridWay. A WS-GRAM service hosting a GridWay meta-scheduler provides the standard functionality required to implement a gateway to a federated grid. That is to virtualize a whole grid, providing a powerful abstraction of the underlying grid resource management services.

<sup>1</sup> <http://www.gridway.org/research/gridfederation.php>

Figure 1.4. Grid Federation with GridWay and GridGateWays.



---

# Chapter 2. Building and installing

## 1. Verifying Globus Installation

As GridWay relies on Globus services, it is assumed that a Globus grid infrastructure has been installed and configured. You can perform the following tests to verify your Globus pre-WS installation, and to ensure that it will work with GridWay:

1. Authorization test:

```
$ globusrun -a -r localhost
```

You should receive the message "GRAM Authentication test successful".

2. Submission test:

```
$ globus-job-run localhost /bin/uname -a
```

You should see the output of the "/bin/uname -a" command.

3. File transfer test:

```
$ globus-url-copy file:///etc/hosts gsiftp://localhost/tmp/hosts1
```

```
$ globus-url-copy gsiftp://localhost/tmp/hosts1 file:///tmp/hosts2
```

The contents of files /etc/hosts, /tmp/hosts1 and /tmp/hosts2 should be identical.

4. Information retrieval test:

```
$ grid-info-search -x
```

You should see a lot of information in LDIF format.

Change localhost to the name of the host your want to test.

You can perform the following tests to verify your Globus WS installation, and to ensure that it will work with GridWay:

1. Submission test:

```
$ globusrun-ws -submit -F localhost -s -c /bin/uname -a
```

You should see the output of the "/bin/uname -a" command (along with other information about submission progress).

2. File transfer test:

```
$ globus-url-copy file:///etc/hosts gsiftp://localhost/tmp/hosts1
```

```
$ globus-url-copy gsiftp://localhost/tmp/hosts1 file:///tmp/hosts2
```

The contents of files `/etc/hosts`, `/tmp/hosts1` and `/tmp/hosts2` should be identical.

3. Information retrieval test:

```
$ wsrp-query -x -s https://localhost:8443/wsrp/services/DefaultIndexService
```

You should see a lot of information in XML format.



### Note

XML documents from `wsrp-query` should not contain any DEBUG messages. SOAP Message Logging for the client tools has to be disabled in `$GLOBUS_LOCATION/log4j.properties`.

Change `localhost` to the name of the host you want to test.

If a binary distribution of the Globus Toolkit is installed, you may be required to manually install `globus_core` (used to detect the compiler and platform settings of the computer that the Toolkit is installed on). The following command can be used to perform this operation:

```
$GLOBUS_LOCATION/sbin/gpt-build -nosrc <flavor>
```

More information about this procedure is available ([here](#)).

## 2. Required Software

GridWay is distributed as a source package, required software to compile it:

- C compiler: Tested versions gcc 3.4.2, 3.4.4, 4.0.3, 4.0.3 and 4.1.2
- Globus C libraries: `globus_gram_client`, `globus_ftp_client` and `globus_gass_copy`
- Globus JAVA development libraries
- J2SE versions 1.4.2\_10+ (Builds higher than 10) or 1.5.0+
- GNU Make
- Sudo command (only required for multiple-user mode)
- Berkeley Database library version 4.4.20 (only required to compile the accounting module)

## 3. Platform Notes

GridWay has been run on the following platforms:

### 3.1. Fedora Core

There are issues with Fedora Core 4 and Sun's Java 1.4.2. Please upgrade to Java 1.5+ on these platforms. Also problems have been reported on Fedora Core platforms when using 32 bit JSDK binaries on AMD64 architectures.

## 3.2. Debian Testing

No known issues.

## 3.3. Mac OS X

No known issues. Tested on Mac OS X 10.4 (Tiger).

## 3.4. Solaris 10

No known issues.

## 3.5. Other Linux/UNIX flavors

GridWay should run smoothly on any linux based distribution and it is also likely to work on any unix based operating system, although it just have been tested in the aforementioned platforms.

# 4. Installing GridWay

You can install GridWay in two different ways:

1. *Single-user installation.* GridWay will be installed configured and executed by each user. In this case, neither the installation nor the configuration require privilege access to the system. This installation mode will be useful if you want to set up a personal queue, or for testing purposes.
2. *Multiple-user installation.* GridWay will be installed, and configured by the system manager. Regular users are able to submit, control and monitor their jobs from a front-end (GridWay server host) or from client hosts. This installation mode is recommended for production use.

Next sections describe in detail the installation process for these two cases.

## 4.1. Single-User Mode Installation

In this scenario, GridWay is installed by each end-user in his client host.

Login as your user account and follow these steps:

1. Download the distribution file to the installation directory, for example your \$HOME directory
2. Unpack the distribution file and change to gw5 directory:

```
$ tar xzf gw5.tgz
```

```
$ cd gw5
```

3. Set up Globus development environment:

```
$ source $GLOBUS_LOCATION/etc/globus-devel-env.sh
```

or

```
$ . $GLOBUS_LOCATION/etc/globus-devel-env.csh
```

depending on the shell you are using.

4. Run `configure` to set up GridWay installation. Possible options for `configure` are:

**Table 2.1. Configure Options.**

Option	Description
<code>--prefix</code>	Sets final GridWay installation dir. Defaults to <code>/usr/local/gw</code> .
<code>--with-flavor=flavor</code>	The <code>configure</code> script will try to detect the flavor (eg. <code>gcc32dbg</code> ) of the Globus toolkit installed in your system. However, if the <code>configure</code> script is not able to detect it, specify it with this option.
<code>--disable-drmaa</code>	Don't build <code>drmaa</code> support. Default is enabled.
<code>--enable-drmaa-ruby</code>	Build <code>ruby drmaa</code> support. Default is disabled.
<code>--disable-prews</code>	Don't build <code>pre-web-services</code> support. Default is enabled.
<code>--disable-ws</code>	Don't build <code>web-services</code> support. Default is enabled.
<code>--enable-globus-scheme</code>	Adds <code>gridway</code> subdirectories to <code>etc</code> and <code>var</code> . Default is disabled.
<code>--enable-jsdl</code>	Does compile <code>jsdl</code> support. Default is enabled. Disabled for GT builds.
<code>--enable-globus-scheme</code>	Adds <code>gridway</code> subdirectories to <code>etc</code> and <code>var</code> . Default is disabled.
<code>--disable-gridftp</code>	Does not compile <code>gridftp</code> mad. Default is enabled.
<code>--with-db=path_to_db</code>	Specify the Berkeley Database path to build accounting support.
<code>--with-doc</code>	Install GridWay documentation
<code>--with-tests</code>	Install tests

If you want to install GridWay inside `$GLOBUS_LOCATION` you can use the option `--enable-globus-scheme` so GridWay specific `var`, `etc` and `test` files are relocated to `var/gridway`, `etc/gridway` and `test/gridway` respectively. This new directory scheme lets `globus` and GridWay share the same root directory without interfering each other.

The next line will configure GridWay to include documentation and accounting

```
$ ./configure --with-doc --with-db=/usr/local/db
```

5. Run `make`:

```
$ make
```

6. Run `make install`:

```
$ make install
```

7. Once installed, you should have the following directory tree in your GridWay location directory:

```
$GW_LOCATION/
|
+--- bin/          executables
|
+--- etc/          gwd.conf and job_template.default configuration files
```

```

|
+--- share/      Include examples and [Optionally] documentation
|
+--- include/    header files
|
+--- lib/        compiled libraries
|
+--- libexec/    wrapper and monitor scripts
|
+--- test/       test suite [Optional]
|
+--- var/        lock, port and log files

```

## 4.2. Multiple-User Mode Installation

In this scenario, the installation of GridWay is performed by the system manager and the users are able to submit, control and monitor their jobs from a front-end (GridWay server host) or from client hosts, which may not require a GridWay/Globus installation. This means that there is one GridWay installation for each organization that provides support for multiple intra-organization users.

### Important

The instructions here described assumes that you are going to install GridWay in its own directory (\$GW\_LOCATION, e.g. /usr/local/gw). Also it is assumed that the installation, configuration and service execution will be performed by an special account (gwadmin).

GridWay can be also installed within the Globus Toolkit tree. To do this you have to use the flag `--enable-globus-scheme` when calling configure script and set the prefix to \$GLOBUS\_LOCATION. In this case, the gwadmin user will be the user account that performed the Globus Toolkit installation.

When you install it this way you also have to note that \$GW\_LOCATION/var and \$GW\_LOCATION/etc directories will be \$GLOBUS\_LOCATION/var/gridway and \$GLOBUS\_LOCATION/etc/gridway.

### Important

Note that GridWay daemon SHOULD NOT be run as root. Only part of the installation will require privileged access.

Login as root account and follow the next steps:

1. All of the GridWay users must be members of the same UNIX group, <gwgroup>. We recommend to create a new group (call gwusers, for example), and assure that all GridWay user accounts are members of this new group.
2. The GridWay administrator account, <adminuser>, can be an existing administrative login or a new login. We recommend creating a new account for the GridWay administration user (call gwadmin, for example). This account will own all of the files in the GridWay installation, all of the daemons in the GridWay execution and it can be used to configure GridWay once it is installed. Primary group of <adminuser> should be <gwgroup>.

DO NOT use root account for the GridWay administrator account.

3. Download the distribution file to the installation directory, for example your /usr/local directory

4. Unpack the distribution file and change ownership:

```
# tar xzf gw5.tgz
```

```
# chown -R <adminuser>:<gwgroup> <gwlocation>
# chmod 755 <gwlocation>
```

Become GridWay administrator user, and change to gw5 directory:

```
# su gwadmin
$ cd gw5
```

5. Set up Globus development environment:

```
$ source $GLOBUS_LOCATION/etc/globus-devel-env.sh
```

or

```
$ . $GLOBUS_LOCATION/etc/globus-devel-env.csh
```

depending on the shell you are using.

6. Run configure to set up GridWay installation. Check above for possible options or just type **configure --help**.

7. Run make:

```
$ make
```

8. Run make install:

```
$ make install
```

9. Once installed, you should have the following directory tree in your GridWay location directory:

```
$GW_LOCATION/
|
+--- bin/          executables
|
+--- etc/          gwd.conf and job_template.default configuration files
|
+--- share/       Include examples and [Optionally] documentation
|
+--- include/     header files
|
+--- lib/         compiled libraries
|
+--- libexec/     wrapper and monitor scripts
|
+--- test/        test suite [Optional]
|
+--- var/         lock, port and log files
```

10. The `sudoers` file of the `sudo` command should include the following

```
...
# User alias specification
...
Runas_Alias      GW_USERS = %<gwgroup>
...
# GridWay entries
gadmin ALL=(GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_em_mad_prews *
gadmin ALL=(GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_em_mad_ws *
gadmin ALL=(GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_tm_mad_ftp *
```

Usually `sudo` clears all environment variables for security reasons. However MADs need the `GW_LOCATION` and `GLOBUS_LOCATION` variables to be set. To preserve those variables in the MAD environment, add the following line to your `sudoers` file:

```
Defaults>GW_USERS env_keep="GW_LOCATION GLOBUS_LOCATION"
```

Please refer to the `sudo` manual page for more information.

Additionally you can configure your drivers environment by using the `gwrc` interface, see section [Section 3.1, “MAD Environment Configuration”](#).

To test the `sudo` command configuration try to execute a MAD as a user in the `<gwgroup>` group, for example:

```
$ sudo -u <gw_user> /usr/local/gw/bin/gw_em_mad_prews
```

Following previous steps, the end-users must login to the GridWay server host to be able to execute GridWay commands and use the DRMAA libraries.

Additionally, client hosts, that are not required to have GridWay/Globus installed, could be deployed to remotely interface to the GridWay server host. In such a case, user accounts and home directories must be shared between the GridWay server and the client hosts, via for example NIS and NFS; and `<gwlocation>` directory should be readable on all client hosts. The `<gwlocation>` directory may be available via for example NFS by exporting `<gwlocation>` from GridWay server, creating `<gwlocation>` directory in the client hosts, changing its ownership to `<adminuser>:<gwgroup>` and mounting the `<gwlocation>` directory exported by the GridWay server on the `<gwlocation>` of the client hosts.

Following those steps, a user logged in a client hosts is able to interface to the GridWay daemon in the GridWay server host. However, the `grid-proxy-init` globus command must be executed in the server host in order to create a proxy by, for example, executing `ssh <GridWay server> grid-proxy-init .`

---

# Chapter 3. Configuration Guide

## 1. Core Configuration Guide

GridWay requires that the environment variables `GLOBUS_LOCATION` and `GW_LOCATION` are set. These are set to the base of your Globus installation and GridWay installation. In GT 4.2.1, GridWay is installed in the same place as Globus, so you can set both of these environment variables to the same location.

### Important

Note that the GridWay daemon **SHOULD NOT** be run as root. Only part of the installation will require privileged access.

Login as root account and follow the next steps:

1. All of the GridWay users must be members of the same UNIX group, `<gwgroup>`. We recommend creating a new group (called `gwusers`, for example), and make sure that all GridWay user accounts are members of this new group.
2. The GridWay administrator account, `<adminuser>`, can be an existing administrative login or a new login. We recommend using the Globus account for the GridWay administration user. This account will own all of the files in the GridWay installation plus all of the daemons in the GridWay execution and it can be used to configure GridWay once it is installed. Primary group of `<adminuser>` should be `<gwgroup>`.

DO NOT use root account for the GridWay administrator account.

3. The `sudoers` file of the **sudo** command should include the following:

```
...
# User alias specification
...
Runas_Alias      GW_USERS = %<gwgroup>
...
# GridWay entries
globus ALL=(GW_USERS)    NOPASSWD: /home/gwadmin/gw/bin/gw_em_mad_prews *
globus ALL=(GW_USERS)    NOPASSWD: /home/gwadmin/gw/bin/gw_em_mad_ws *
globus ALL=(GW_USERS)    NOPASSWD: /home/gwadmin/gw/bin/gw_tm_mad_ftp *
```

Usually **sudo** clears all environment variables for security reasons. However MADs need the **GW\_LOCATION** and **GLOBUS\_LOCATION** variables to be set. To preserve those variables in the MAD environment, add the following line to your `sudoers` file:

```
Defaults>GW_USERS env_keep="GW_LOCATION GLOBUS_LOCATION"
```

Please refer to the **sudo** manual page for more information.

To test the **sudo** command configuration try to execute a MAD as a user in the `<gwgroup>` group, for example:

```
$ sudo -u <gw_user> /home/gwadmin/gw/bin/gw_em_mad_prews
```

## 1.1. Configuration Interface

The configuration files for GridWay are read from the following locations:

- `$GW_LOCATION/etc/gridway/gwd.conf`: Configuration options for the GridWay daemon (GWD).
- `$GW_LOCATION/etc/gridway/sched.conf`: Configuration options for the GridWay built-in scheduling policies (see [Section 2.6, “Scheduler Configuration”](#) for more information).
- `$GW_LOCATION/etc/gridway/job_template.default`: Default values for job's templates (i.e. job definition files).
- `$GW_LOCATION/etc/gridway/gwrc`: Default environment variables for MADs.

Options are defined one per line, with the following format:

```
<option> = [value]
```

If the value is missing the option will fall back to its default. Blank lines and any character after a '#' are ignored. Note: Job template options can use job or host variables to define their value, these variables are substituted at run time with their corresponding values (see the [GridWay user guide](#)<sup>1</sup>).

## 1.2. GridWay Daemon (GWD) Configuration

The GridWay daemon (GWD) configuration options are defined in `$GW_LOCATION/etc/gridway/gwd.conf`. The table below summarizes the configuration file options, their description and default values. Note that blank lines and any character after a '#' are ignored.

---

<sup>1</sup> <http://www.gridway.org/documentation/stable/userguide/>

**Table 3.1. GWD Configuration File Options.**

Option	Description	Default
Connection Options		
GWD_PORT	TCP/IP Port where GWD will listen for client requests. If this port is in use, GWD will try to bind to the next port until it finds a free one. The TCP/IP port used by GWD can be found in \$GW_LOCATION/var/gridway/gwd.port	6725
MAX_NUMBER_OF_CLIENTS	Maximum number of simultaneous client connections. Note that only blocking client requests keeps its connection open.	20
Pool Options		
NUMBER_OF_JOBS	The maximum number of jobs that will be handled by the GridWay system	200
NUMBER_OF_ARRAYS	The maximum number of array-jobs that will be handled by the GridWay system	20
NUMBER_OF_HOSTS	The maximum number of hosts that will be handled by the GridWay system	100
NUMBER_OF_USERS	The maximum number of different users in the GridWay system	30
Intervals		
SCHEDULING_INTERVAL	Period (in seconds) between two scheduling actions	30
DISCOVERY_INTERVAL	How often (in seconds) the information manager searches the Grid for new hosts	300
MONITORING_INTERVAL	How often (in seconds) the information manager updates the information of each host	120
POLL_INTERVAL	How often (in seconds) the underlying grid middleware is queried about the state of a job.	60
Middleware Access Driver (MAD) Options		
IM_MAD	Information Manager MADs, see <a href="#">Section 3.4, “Information Driver Configuration”</a>	-
TM_MAD	Transfer Manager MADs, see <a href="#">Section 3.3, “File Transfer Driver Configuration”</a>	-
EM_MAD	Execution Manager MADs, see <a href="#">Section 3.2, “Execution Driver Configuration”</a>	-
MAX_ACTIVE_IM_QUERIES	Maximum number (soft limit) of active IM queries (each query spawns one process)	4
Scheduler Options		
DM_SCHED	Scheduling module, see <a href="#">Section 2.6, “Scheduler Configuration”</a>	-

Here is an example of a GWD configuration file:

```
#-----
# Example: GWD Configuration File
#-----
GWD_PORT          = 6725
MAX_NUMBER_OF_CLIENTS = 20
```

```

NUMBER_OF_ARRAYS = 20
NUMBER_OF_JOBS   = 200
NUMBER_OF_HOSTS  = 100
NUMBER_OF_USERS  = 30
JOBS_PER_SCHED  = 10
JOBS_PER_HOST    = 10
JOBS_PER_USER    = 30
SCHEDULING_INTERVAL = 30
DISCOVERY_INTERVAL = 300
MONITORING_INTERVAL = 120
POLL_INTERVAL    = 60
IM_MAD = mds4:gw_im_mad_mds4:-s hydrus.dacya.ucm.es:gridftp:ws
TM_MAD = gridftp:gw_tm_mad_ftp:
EM_MAD = ws:gw_em_mad_ws:rsl2
DM_SCHED = flood:gw_flood_scheduler:-h 10 -u 30 -c 5

```

## 1.3. Job Template Default Values

Default values for *every* job template option can be set in `$GW_LOCATION/etc/gridway/job_template.default`. You can use this file to set the value of advanced job configuration options and use them for all your jobs. Note that the values set in a job template file override those defined in `job_template.default`. See the [GridWay user guide](#)<sup>2</sup> for a detailed description of each job option.

## 1.4. Running gwd

GridWay reporting and accounting facilities provide information about overall performance and help troubleshoot configuration problems. GWD generates the following logs under the `$GW_LOCATION/var` directory:

- `$GW_LOCATION/var/gwd.log`: System level log. You can find log information of the activity of the middleware access drivers; and a coarse-grain log information about jobs.
- `$GW_LOCATION/var/sched.log`: Scheduler log. You can find log information to fit the scheduler policies to your organization needs.
- `$GW_LOCATION/var/$JOB_ID/job.log`: Detailed log information for each job, it includes details of job resource usage and performance.
- `$GW_LOCATION/var/acct`: Accounting information. Use the **gwacct** command to access the data bases. Note that you need Berkeley DB library (version 4.4.20).
- `$GW_LOCATION/var/.lock`: Used to prevent from running more than one instance of the daemon.
- `$GW_LOCATION/var/gwd.port`: TCP/IP port GWD is listening for client connection requests.
- `$GW_LOCATION/var/globus-gw.log`: Used to encapsulate GridWay in a GRAM service (please refer to the Grid4Utility project web page for more information). This log file follows the globus fork job starter's format (based on SEG, Scheduler Event Generator messages):

```
001;TIMESTAMP;JOBID;STATE;EXIT_CODE
```

---

<sup>2</sup> <http://www.gridway.org/documentation/stable/userguide/>

## 1.5. Daemon Failure Recovery

Since GridWay 4.9, when you start the daemon, **gwd** tries to recover its previous state. This is, any submitted job is stored in a persistent pool, and in case of a gwd (or client machine) crash these jobs are recovered. This includes, for jobs in wrapper state, contacting with the remote jobmanager.

Recovery actions are performed by default, if you do not want to recover the previous submitted jobs use the **-c** option.

For example, to start gwd in multi-user mode and clear its previous state, use:

```
$ gwd -c -m
```

## 2. Scheduler Configuration Guide

Grid scheduling consists of finding a suitable (in terms of a given target) assignment between a computational workload (jobs) and computational resources. The scheduling problem has been thoroughly studied in the past and efficient algorithms have been devised for different computing platforms. Although some of the experience gained in scheduling can be applied to the Grid, it presents some characteristics that differ dramatically from classical computing platforms (i.e. clusters or MPPs), namely: different administration domains, limited control over resources, heterogeneity and dynamism.

Grid scheduling is an active research area. The Grid scheduling problem is better understood today and several heuristics, performance models and algorithms have been proposed and evaluated with the aid of simulation tools. However, current working Grid schedulers are only based on match-making, and barely consider multi-user environments.

In this section, we describe the state-of-the-art scheduling policies implemented in the GridWay system. The contents of this guide reflect the experience obtained since GridWay version 4, and a strong feedback from the GridWay user community.

### 2.1. GridWay Scheduling Architecture

The scheduler is responsible for assigning jobs to Grid resources; therefore, it decides when and where to run a job. These decisions are made periodically in an endless loop. The frequency of the scheduler interventions can be adjusted with the `SCHEDULER_INTERVAL` configuration parameter (see [Section 1.2, “GridWay Daemon \(GWD\) Configuration”](#)).

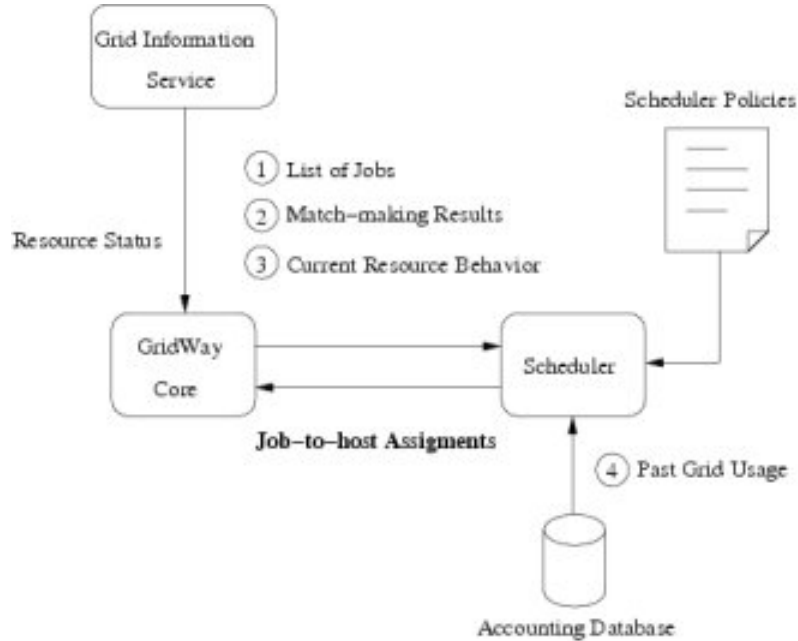
In order to make job to resource assignments the scheduler receives information from the following sources (see [Figure 3.1, “Job Scheduling in GridWay”](#)):

1. *List of jobs in the system*, which includes pending jobs as well as running jobs (those in wrapper state). Those jobs that cannot be started are filtered out from the list, i.e., jobs with unmet dependencies, stopped or held.
2. *Match-making results*: The Information Manager drivers query the Grid information services to track the availability and status of Grid resources. The discovery and monitoring processes can both be configured as static or dynamic, see [Section 3.4, “Information Driver Configuration”](#). This information is used by the GridWay core to build a list of suitable resources for each job, i.e., resources meeting the job requirements, and to compute their rank.
3. *Current resource behavior*: The scheduler considers the way a resource is behaving when making its decisions. In particular, it evaluates the migration and failure rates and execution statistics (transfer, execution and queue wait times).

4. *Past Grid Usage*: The scheduler also considers the past history (behavior) of Grid resources to issue schedules. Note that database support needs to be compiled in GridWay for this feature.

The information gathered from the previous sources is combined with a given scheduling policy to prioritize jobs and resources. Then, the scheduler dispatches the highest priority job to the best resource for it. The process continues until all jobs are dispatched, and those that could not be assigned wait for the next scheduling interval.

**Figure 3.1. Job Scheduling in GridWay**



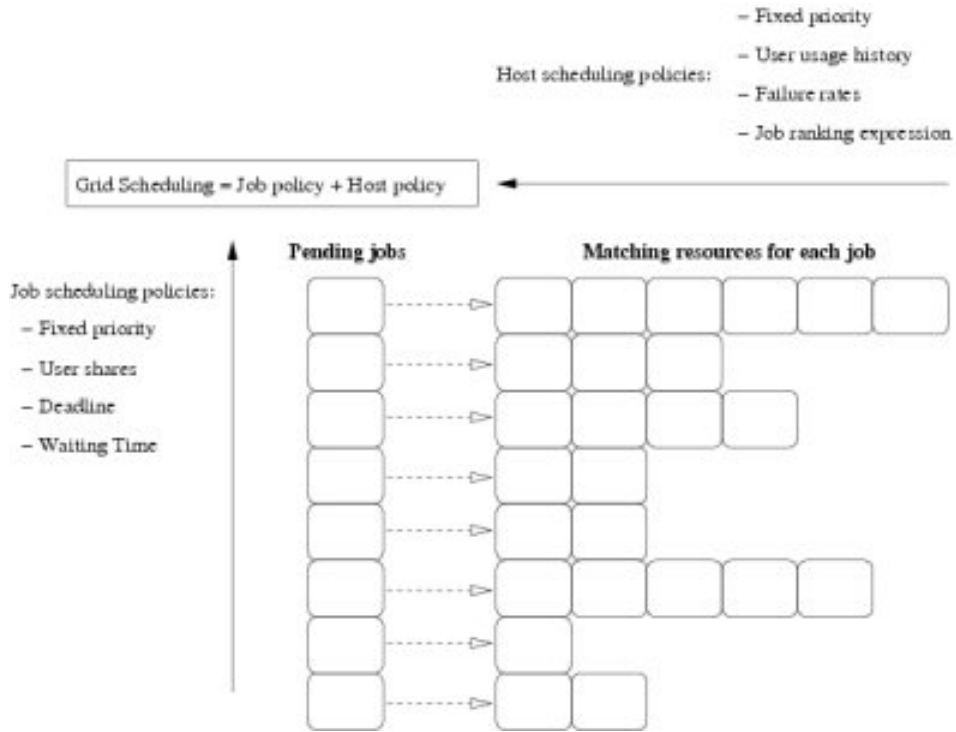
## 2.2. Scheduling Policies

A scheduling policy is used to assign a *dispatch priority* to each job and a *suitability priority* to each resource. Therefore, a Grid scheduling policy comprises two components:

- *Job prioritization policies*. Pending jobs are prioritized according to four policies: fixed, share, deadline and waiting-time. The job policies are used to sort the jobs of the users of a given scheduling domain (GridWay instance). Note that these policies are only enforced in the scheduling domain and not for the whole Grid infrastructure as discussed above.
- *Resource prioritization policies*. A given job can be executed on those resources that match its requirements. The resource policies are used to sort the matching resource list of each job. The matching resources are prioritized according to four policies: fixed, usage, failure and rank. Note that these policies do not only depend on the Grid resource but also on the job owner, as each Grid user (or VO member) has its own access rights and usage history.

These two top-level policies can be combined to implement a wide range of scheduling schemes (see [Figure 3.2, “Job and resource prioritization policies in GridWay.”](#)). The above scheduling policies are described in the following sections.

**Figure 3.2. Job and resource prioritization policies in GridWay.**



## 2.3. Job Prioritization Policies

The job prioritization policies allow Grid administrators to influence the dispatch order of the jobs, that is, to decide which job is sent to the Grid. Traditionally, DRMS implement different policies based on the owner of the job, the resources consumed by each user or the requirements of the job. Some of these scheduling strategies can be directly applied in a Grid, while others must be adapted because of their unique characteristics: dynamism, heterogeneity, high fault rate and site autonomy.

### 2.3.1. Fixed Priority Policy (FP)

This policy assigns a fixed priority to each job. The fixed priority ranges from 00 (lowest priority) to 19 (highest priority), so jobs with a higher priority will be dispatched first. The default priority values are assigned, by the Grid administrator, using the following criteria:

- *User.* All jobs of a User are given a fixed priority.
- *Group.* All jobs of a user belonging to a given Group get a fixed priority.

The user priority prevails over the group one. Also there is a special user (DEFAULT) to define the default priority value when no criteria apply.

The users can set the priority of their own jobs (**gws submit -p**) but without exceeding their limit set by the administrator in the scheduler configuration file.

Here is an example configuration for the fixed priority (see also [Section 2.5, "Built-in Scheduler Configuration File"](#)):

```
# Weight for the Fixed priority policy
```

```

FP_WEIGHT = 1

# Fixed priority values for David's and Alice's jobs
FP_USER[david] = 2
FP_USER[alice] = 12

# Fixed priority for every body in the staff group
FP_GROUP[staff] = 5

# Anyone else gets a default priority 3
FP_USER[DEFAULT] = 3

```

### 2.3.2. Urgent Job Policy

The Grid administrator can also set the fixed priority of a job to 20. When a job gets a fixed priority of 20, it becomes an *urgent job*. Urgent jobs are dispatched as soon as possible, bypassing all the scheduling policies.

### 2.3.3. Fair-Share Policy (SH)

The fair-share policy allows you to establish a dispatching ratio among the users of a scheduling domain. For example, a fair-share policy could establish that jobs from David and Alice must be dispatched to the Grid in a 2:5 ratio. In this case, the scheduler tracks the jobs submitted to the Grid by these two users and dispatches the jobs so they target a 2:5 ratio of job submissions.

This policy resembles the well-known fair-share of traditional LRMS. However, note that what GridWay users share is the ability to submit a job to the Grid and not resource usage. Resource usage share cannot be imposed at a Grid level, as Grid resources are shared with other Grid users and with local users from the remote organization. In addition, the set of resources that can be potentially used by each user is not homogeneous, as each user may belong to a different VO.

GridWay tracks the jobs submitted to the Grid by the users over time. Grid administrators can specify a timeframe over which user submissions are evaluated. The amount of time considered by GridWay is defined by a number of time intervals (`SH_WINDOW_DEPTH`) and the duration of each one (`SH_WINDOW_SIZE`, in days). The effective number of submissions in a given window is exponentially damped, so present events become more relevant.

Here is an example configuration for the share policy (see also [Section 2.5, “Built-in Scheduler Configuration File”](#)):

```

# Weight for the Fair-share policy
SH_WEIGHT = 1

# Shared values for David's and Alice's submissions
SH_USER[david] = 2
SH_USER[alice] = 5

# Anyone else gets a default share value of 1
SH_USER[DEFAULT] = 1

# Consider submissions in the last 5 days
SH_WINDOW_SIZE = 1
SH_WINDOW_DEPTH= 5

```

### 2.3.4. Waiting-time Policy (WT)

The goal of this policy is to prevent low-priority jobs from starving. So jobs in the pending state long enough will be eventually submitted to the Grid. This policy can be found in most of the DRMS today. In GridWay, the priority of a job is increased linearly with the waiting time.

Here is an example configuration for this policy:

```
# Weight for the Waiting-time policy
WT_WEIGHT = 1
```

### 2.3.5. Deadline Policy (DL)

GridWay includes support for specifying deadlines at job submission. The scheduler will increase the priority of a job as its deadline approaches.

#### Important

Note that this policy does not guarantee that a job is completed before the deadline.

Grid administrators should provide a way to qualify the remaining time to reach the job deadline by defining when a job should get half of the maximum priority assigned by this policy (DL\_HALF, in days).

Here is an example configuration for the deadline policy (see also [Section 2.5, “Built-in Scheduler Configuration File”](#)):

```
# Weight of the Deadline Policy
DL_WEIGHT = 1

# Assign half of the priority two days before the deadline
DL_HALF = 2
```

### 2.3.6. The Overall Dispatch Priority of a Job

The list of all pending jobs is sorted by the *dispatch priority*, which is computed as a weighted sum of the contribution from the previous policies. In this way, the Grid administrator can implement different scheduling schemes by adjusting the policy weights.

The *dispatch priority* of a job is therefore computed with:

#### Figure 3.3. Dispatch priority of a job

$$P_j = \sum_i w_i \cdot p_i \text{ where } i = \{\text{fixed, share, wait-time, deadline}\}.$$

where  $w_i$  is the weight for each policy (integer value) and  $p_i$  is the priority (normalized) contribution from each policy.

## 2.4. Resource Prioritization Policies

The resource prioritization policies allow Grid administrators to influence the usage of resources made by the users, that is, decide where to run a job. Usually, in classical DRMS, this resource usage is administered by means of the queue concept.

In GridWay, the scheduler builds a *meta-queue* (a queue consisting of the local queues of the Grid resources) for each job based on its requirements (e.g., operating system or architecture). Note that this *meta-queue* is not only built in terms of resource properties but is also based upon the owner of the job, (as each Grid user may belong to a different VO with its own access rights and usage privileges).

The *meta-queue* for a job consists of the queues of those resources that meet the job requirements specified in the job template and have at least one free slot. By default, this queue is sorted in a first-discovered first-used fashion. This order can be influenced by means of the subsequent resource prioritization policies.

### 2.4.1. Fixed Resource Priority Policy (RP)

Usually, GridWay is configured with several Information Managers (IM). Grid administrators can prioritize resources based upon the IM that discovered the resource. Grid administrators can also assign priorities to individual resources. For example, a fixed priority policy can specify that resources from the intranet (managed by an IM driver tagged **intranet**) should always be used before resources from other sites (managed by an IM driver tagged **grid**).

The priority of a resource ranges from 01 (lowest priority) to 99 (highest priority), so resources with a higher priority will be used first. Grid administrators can also prioritize individual resources based on business decisions.

When a resource gets the priority value 00, it becomes a **banned resource**, and will not be used for any job. So Grid administrators can virtually *unplug* resources from their scheduling domain.

Example configuration for the resource Fixed Priority Policy:

```
# Weight for the Resource fixed priority policy
RP_WEIGHT = 1

# Fixed priority values for specific resources
RP_HOST[my.cluster.com] = 12
RP_HOST[slow.machine.com] = 02

# Fixed priority for every resource in the intranet
RP_IM[intranet] = 65

# Fixed priority for every resource discovered by the grid IM
RP_IM[grid] = 05

# Anyone else gets a default priority 04 (i.e. other IM)
RP_IM[DEFAULT] = 01
```

### 2.4.2. Rank Policy (RA)

The goal of this policy is to prioritize those resources more suitable for the job, from its own point of view. For example, the rank policy for a job can state that resources with faster CPUs should be used first. This policy is configured through the RANK attribute in the job template, please refer to the [GridWay user guide](http://www.gridway.org/documentation/stable/userguide/)<sup>3</sup>.

Example configuration for the Rank policy:

```
# Weight of the Rank policy
RA_WEIGHT = 1
```

---

<sup>3</sup> <http://www.gridway.org/documentation/stable/userguide/>

### 2.4.3. Usage Policy (UG)

This policy reflects the behavior of Grid resources based on job execution statistics. So, crucial performance variables, like the average queue wait time or network transfer times, are considered when scheduling a job. This policy is derived from the sum of two contributions: history and current.

- *History contribution.* Execution statistics on a given period of time (for example, average values in the last 3 days). This information is obtained from the accounting database, so GridWay must be compiled with the Berkeley DB libraries.
- *Last job contribution.* Execution statistics of the last job on that resource.

These values are used to compute an estimated execution time of a job on a given resource for a given user:

**Figure 3.4. Estimated execution time of a job on a resource**

$$T = (1 - w) \cdot (T_{exe}^h + T_{zfr}^h + T_{que}^h) + w \cdot (T_{exe}^c + T_{zfr}^c + T_{que}^c)$$

where  $T^c$  are the execution statistics of the last job (execution, transfer and queue wait-time),  $T^h$  are the execution statistics based on the history data; and  $w$  is the history ratio. Those resources with a lower estimated time are used first to execute a job.

The Usage policy can be configured with:

- UG\_HISTORY\_WINDOW. Number of days used to compute the execution statistics from the History contribution.
- UG\_HISTORY\_RATIO. The value of  $w$ , use 0 to use only data from the accounting database, and 1 to use only results from the last execution.

Example configuration for Usage policy:

```
# Weight of the Usage policy
UG_WEIGHT = 1

# Number of days in the history window
UG_HISTORY_WINDOW = 3

# Accounting database to last execution ratio
UG_HISTORY_RATIO = 0.25
```

### 2.4.4. Failure Rate Policy (FR)

When a resource fails, GridWay implements an exponential linear back-off strategy at resource level (and per each user); henceforth, resources with persistent failures are discarded (for a given user).

In particular, when a failure occurs a resource is *banned* for  $T$  seconds:

**Figure 3.5. Banned time formula**

$$T = T_{\infty} \cdot (1 - e^{-\frac{\Delta t}{\sigma}})$$

where  $T_\infty$  is the maximum time that a resource can be *banned*,  $\Delta t$  is the time since last failure, and  $C$  is a constant that determines how fast the  $T_\infty$  limit is reached.

The failure rate policy can be configured with the following parameters:

- **FR\_MAX\_BANNED\_TIME**. The value of  $T_\infty$ , use 0 to disable this policy.
- **FR\_BANNED\_C**. The value of the  $C$  constant in the above equation.

Example configuration for the Failure Rate policy:

```
# Maximum time that a resource will not be used, in seconds
FR_MAX_BANNED_TIME = 3600
# Exponential constant
FR_BANNED_C          = 650
```

### 2.4.5. The Overall Suitability Priority of a Resource

The list of all candidate resources is sorted by the *suitability priority*, which is computed as a weighted sum of the contribution from the previous policies. The *suitability priority* resource is therefore computed with:

**Figure 3.6. Suitability priority of a resource**

$$P_h = \sum_i w_i \cdot p_i \text{ where } i = \{\text{fixed, usage, rank}\}.$$

where  $w_i$  is the weight for each policy (integer value) and  $p_i$  is the priority (normalized) contribution from each policy.

### 2.4.6. Re-scheduling Policies

Also, the scheduler can migrate running jobs in the following situations:

- A better resource is discovered.
- A job has been waiting in the remote queue system more than a given threshold.
- The application changes its requirements.
- A performance degradation is detected.

See Section 1.2, “GridWay Daemon (GWD) Configuration” and the [GridWay user guide](#)<sup>4</sup>, for information on configuring these policies.

## 2.5. Built-in Scheduler Configuration File

The built-in scheduler configuration options are defined in `$GW_LOCATION/etc/sched.conf`. The table below summarizes the configuration file options, their description and default values. Note that blank lines and any character after a '#' are ignored.

---

<sup>4</sup> <http://www.gridway.org/documentation/stable/userguide/>

**Table 3.2. Built-in Scheduler Configuration File Options.**

Option	Description	Default
<b>Job Scheduling Policies.</b> Pending jobs are prioritized according to four policies:fixed (FP), share(SH), deadline (DL) and waiting-time (WT). The dispatch priority of a job is computed as a weighted sum of the contribution of each policy (normalized to one).		
DISPATCH_CHUNK	The maximum number of jobs that will be dispatched for each scheduling action	15 (0 to dispatch as many jobs as possible)
MAX_RUNNING_USER	The maximum number of simultaneous running jobs per user.	30 (0 to dispatch as many jobs as possible)
<i>Fixed Priority (FP) Policy:</i> Assigns a fixed priority to each job		
FP_WEIGHT	Weight for the policy (real numbers allowed).	1
FP_USER[<username>]	Priority for jobs owned by <username>. Use the special username DEFAULT to set default priorities. Priority range [0,19]	
FP_GROUP[<groupname>]	Priority for jobs owned by users in group <groupname>. Priority range [0,19]	
<i>Share (SH) Policy:</i> Allows you to establish a dispatch ratio among users.		
SH_WEIGHT	Weight for the policy (real numbers allowed).	
SH_USER[<username>]	Share for jobs owned by <username>. Use the special username DEFAULT to set default shares.	
SH_WINDOW_DEPTH	Number of intervals (windows) to "remember" each user's dispatching history. The submissions of each window are exponentially "forgotten".	5, the maximum value is 10.
SH_WINDOW_SIZE	The size of each interval in days (real numbers allowed).	1
<i>Waiting-time (WT) Policy:</i> The priority of a job is increased linearly with the waiting time to prevent job starvation		
WT_WEIGHT	Weight for the policy (real numbers allowed).	0
<i>Deadline (DL) Policy:</i> The priority of a job is increased exponentially as its deadline approaches.		
DL_WEIGHT	Weight for the policy (real numbers allowed).	1
DL_HALF	Number of days before the deadline when the job should get half of the maximum priority.	1
<b>Resource Scheduling Policies.</b> The resource policies allows grid administrators to influence the usage of resources made by the users, according to: fixed (FP), rank (RA), failure rate (FR), and usage (UG). The suitability priority of a resource is computed as a weighted sum of the contribution of each policy (normalized to one).		
MAX_RUNNING_RESOURCE	The maximum number of jobs that the scheduler submits to a given resource	10
<i>Fixed Priority (RP) Policy:</i> Assigns a fixed priority (range [01,99]) to each resource		
RP_WEIGHT	Weight for the policy (real numbers allowed).	1 (real numbers allowed)
RP_HOST[<FQDN>]	Priority for resource <FQDN>. Those resources with priority 00 WILL NOT be used to dispatch jobs.	

RP_IM[<im_tag>]	Priority for ALL resources discovered by the IM <im_tag> (as set in <code>gwd.conf</code> , see <a href="#">Section 1.2, “GridWay Daemon (GWD) Configuration”</a> ). Use the special tag <code>DEFAULT</code> to set default priorities for resources.	
<i>Usage (UG) Policy:</i> Resources are prioritized based on the estimated execution time of a job (on each resource).		
UG_WEIGHT	Weight for the policy (real numbers allowed).	1 (real numbers allowed)
UG_HISTORY_WINDOW	Number of days used to compute the history contribution.	3 (real numbers allowed)
UG_HISTORY_RATIO	Weight to compute the estimated execution time on a given resource.	0.25
<i>Rank (RA) Policy:</i> Prioritize resources based on their RANK (as defined in the job template)		
RA_WEIGHT	Weight for the policy.	0 (real numbers allowed)
<i>Failure Rate (FR) Policy:</i> Resources with persistent failures are banned		
FR_MAX_BANNED	The maximum time a resource is banned, in seconds. Use 0 TO DISABLE this policy.	3600
FR_BANNED_C	Exponential constant to compute banned time	650

## 2.6. Scheduler Configuration

GridWay uses an external and selectable scheduler module to schedule jobs. The following schedulers are distributed with GridWay:

- Built-in Scheduler (default), which implements the above policies.
- Round-robin/flood Scheduler. This is a simple scheduling algorithm. It maximizes the number of jobs submitted to the Grid. Available resources are flooded with user jobs in a round-robin fashion.



### Important

The flood (user round-robin) scheduler is included as an example, and should not be used in production environments.

The schedulers are configured with the `DM_SCHED` option in the `gwd.conf` file, with the format:

```
DM_SCHED = <sched_name>:<path_to_sched>:[args]
```

where:

- **sched\_name:** is a tag to further refer to this scheduler.
- **path\_to\_sched:** is the name of the Scheduler executable. Use an absolute path or include the Scheduler executable directory in the `PATH` variable (such directory is `$GW_LOCATION/bin` by default).
- **arg:** Additional arguments to be passed to the Scheduler executable.

## 2.6.1. Built-in Scheduler

By default, GridWay is configured to use the built-in policy engine described in the previous sections. If for any reason you need to recover this configuration, add the following line to `$GW_LOCATION/etc/gwd.conf`:

```
DM_SCHED = builtin:gw_sched:
```

Do not forget to adjust the scheduler policies to your needs by editing the `$GW_LOCATION/etc/sched.conf` file.

## 2.6.2. Flood Scheduler

To configure the round-robin/flood scheduler, first disable the built-in engine policy in the `$GW_LOCATION/etc/sched.conf` configuration file by adding the following line:

```
DISABLE = yes
```

Then add the following line to `$GW_LOCATION/etc/gwd.conf`:

```
DM_SCHED = flood:gw_flood_scheduler:-h 10 -u 30 -c 5 -s 15
```

where:

- **-h**: The max number of jobs that the scheduler submits to a given host. Default value is 10; use 0 to dispatch to each host as many jobs as possible.
- **-u**: The maximum number of simultaneous running jobs per user. Default value is 30; use 0 to dispatch as many jobs as possible.
- **-c**: Scheduling Chunk. Jobs of the same user are submitted in a round-robin fashion with the given chunk. Default value is 5.
- **-s**: Dispatch Chunk. The maximum number of jobs that will be dispatched each scheduling action. Default value is 15; use 0 to dispatch as many jobs as possible.

# 3. MAD Configuration Guide

GridWay uses several Middleware Access Drivers (MAD) to interface with different Grid services. The following MADs are part of the GridWay distribution:

- Execution Managers to interface with both pre-WS GRAM and WS GRAM services.
- Information Managers to interface with both MDS2 (MDS and GLUE schemas) and MDS4 services.
- Transfer Managers to interface with GridFTP servers.

These drivers are configured and selected via the GWD configuration interface described in [Section 1.2, “GridWay Daemon \(GWD\) Configuration”](#). Additionally you may need to configure your environment (see [Chapter 5, Testing](#)) in order to successfully load the MADs into the GWD core. To do so, you can also use global and per user environment configuration files (`gwr.c`).

## 3.1. MAD Environment Configuration

There is one global config file and per user configuration files that can be used to set environment variables for MADs. These files are standard shell scripts that are sourced into the MAD environment before it is loaded. It can be used, for example, to set the variable `X509_USER_PROXY` so you can have it located elsewhere instead of the standard place (`/tmp/x509_u<uid>`). Other variables can be set and you can even source other shell scripts, for instance, you can prepare another globus environment for MADs for some users, like this:

```
X509_USER_PROXY=$HOME/.globus/proxy.pem

GLOBUS_LOCATION=/opt/globus-4.2
. $GLOBUS_LOCATION/etc/globus-user-env.sh
```

The file for global MAD environment configuration is `$GW_LOCATION/etc/gridway/gwrc` and the user specific one is `$HOME/.gwrc`.

You have to take into account a couple of things:

- The global environment file is loaded *before* the user one, so the variables set by the user file take precedence over the global ones.
- The files are sourced so you need to export the variables to make them visible in the environment of the called MAD. Right now there is a mechanism so variables set as `VARIABLENAME=VALUE` are automatically exported (without spaces preceding the variable name). If you are sourcing other files or you put variables inside an indented block (for example, in an if statement) you have to explicitly export them. For example:

```
if [ -d /opt/globus-devel ]; then
export GLOBUS_LOCATION=/opt/globus-devel
```

## 3.2. Execution Driver Configuration

The Execution Driver interfaces with Grid Execution Services and is responsible for low-level job execution and management. The GridWay distribution includes the following Execution MADs:

- GRAM2 (Globus Toolkit 2.4 and above)
- GRAM4 (Globus Toolkit 4.0 and above)

Note that the use of these MADs requires a valid proxy.

Execution MADs are configured with the `EM_MAD` option in the `$GW_LOCATION/etc/gwd.conf` file, with the following format:

```
EM_MAD = <mad_name>:<path_to_mad>:<args>:<rs1|rs1_nsh|rs12>
```

where:

- **mad\_name**: is a tag to further refer to this Execution Driver, and it is also useful for logging purposes.
- **path\_to\_mad**: is the name of the Execution Driver executable, which *must* be placed in the `$GW_LOCATION/bin` directory.

- **args:** Parameters passed to the mad when it is executed.
- **rsl|rsl\_nsh|rsl2:** Selects the language that GWD will use to describe job requests. It can be *rsl* (intended to be used with pre-WS drivers), *rsl\_nsh* (intended to be used with pre-WS drivers over resources with non-shared home directories, like in LCG) and *rsl2* (intended to be used with WS drivers).

For example, the following line will configure GridWay to use the Execution Driver **gw\_em\_mad\_prews** using RSL syntax with name *prews*:

```
EM_MAD = prews:gw_em_mad_prews::rsl
```

To use WS-GRAM services, you can include the following line in your `$GW_LOCATION/etc/gwd.conf` file:

```
EM_MAD = ws:gw_em_mad_ws::rsl2
```

### Note

You can simultaneously use as many Execution Drivers as you need (up to 10). So GridWay allows you to simultaneously use pre-WS and WS Globus Services.

## 3.2.1. Port configuration in WS EM MAD

Now it is possible to specify a different gatekeeper port than the standard one (8443) in the Web Service driver. The line to configure EM MADs in `gwd.conf` has changed so you can add parameters to it. The parameter to change the port is the `-p` followed by the port number. For example:

```
EM_MAD = osg_ws:gw_em_mad_ws:-p 9443:rsl2
```

This line tells the EM MAD to use port 9443 to connect to the GT4 Gatekeeper.

## 3.3. File Transfer Driver Configuration

The File Transfer Driver interfaces with Grid Data Management Services and is responsible for file staging, remote working directory set-up and remote host clean up. The GridWay distribution includes:

- GridFTP server (version 1.1.2 and above)
- Dummy Transfer driver (to be used with clusters without shared home)

The use of this driver requires a valid Proxy.

File Transfer Managers are configured with the `TM_MAD` option in the `gwd.conf` file, with the format:

```
TM_MAD = <mad_name>:<path_to_mad>:[arg]
```

where:

- **mad\_name:** is a tag to further refer to this File Transfer Driver, and it is also useful for logging purposes.
- **path\_to\_mad:** is the name of the File Transfer Driver executable, which *must* be placed in the `$GW_LOCATION/bin` directory.
- **arg:** Additional arguments to be passed to the File Transfer executable.

To configure the Transfer Driver, add a line to `$GW_LOATION/etc/gwd.conf`, with the following format:

```
TM_MAD = <mad_name>:<path_to_mad>:[arguments]>
```

### 3.3.1. Configuring the GridFTP Transfer Driver

The GridFTP driver does not require any command line arguments. So to configure the driver, add the following line to `$GW_LOCATION/etc/gwd.conf`:

```
TM_MAD = gridftp:gw_tm_mad_ftp:
```

The name of the driver will be later used to specify the transfer mechanisms with Grid resource.

### 3.3.2. Configuring the Dummy Transfer Driver

The Dummy driver should be used with those resources (clusters) which do not have a shared home. In this case, transfer and execution are performed as follows:

- The Dummy Transfer MAD performs data movements from the cluster worker node and the client using a reverse server model.
- The `rsl_nsh` RSL generation function is used, which transfers the wrapper along with its stdout and stderr streams.
- The wrapper executing in the worker node automatically transfers job.env and input/output files from the client.

The following servers can be configured to access files on the client machine:

- *GASS Server*, started for each user.
- *GridFTP*, specified by its URL running on the GridWay server.

The Dummy driver behavior is specified with the following command line arguments:

- **-u <URL>**: URL of the GridFTP server.
- **-g**: Use a user GASS server to transfer files.

Sample configuration to use a GridFTP server:

```
TM_MAD = dummy:gw_tm_mad_dummy:-u gsiftp\://hostname
```

#### Important

You MUST escape the colon character in gsiftp URL. Also, `hostname` should be the host running the GridWay instance.

Sample configuration to use GASS servers:

```
TM_MAD = dummy:gw_tm_mad_dummy:-g
```

## 3.4. Information Driver Configuration

The Information Driver interfaces with Grid Monitoring Services and is responsible for host discovery and monitoring. The following Information Drivers are included in GridWay:

- Static host information data
- MDS2 with MDS schema (Globus Toolkit 2.4)

- MDS2 with GLUE schema (Globus Toolkit 2.4 and LCG middleware)
- MDS4 (Globus Toolkit 4.0 and above)

To configure an Information Driver, add a line to `$GW_LOCATION/etc/gwd.conf`, with the following format:

```
IM_MAD = <mad_name>:<path_to_mad>:[args]:<tm_mad_name>:<em_mad_name>
```

where:

- **mad\_name**: is a tag to further refer to this Information Driver.
- **path\_to\_mad**: is the name of the Information Driver executable. Use an absolute path or include the Information Driver directory in the `PATH` variable (such directory is `$GW_LOCATION/bin` by default).
- **arg**: Additional arguments to be passed to the Information Driver executable.
- **tm\_mad\_name**: File Transfer Driver to be used with the hosts managed by this Information Driver.
- **em\_mad\_name**: Execution Driver to be used with the hosts managed by this Information Driver.

For example, to configure GWD to access a MDS4 hierarchical information service:

```
IM_MAD = mds4:gw_im_mad_mds4:-s hydrus.dacya.ucm.es:gridftp:ws
```

All the Information Drivers provided with GridWay use a common interface to configure their operation mode. The arguments used by the Information Drivers are:

- **-s <server>**: The information server in a hierarchical configuration, i.e. MDS2 GIIS or MDS4 root IndexService.
- **-l <host list>**: A host list file to be used by GridWay, only relevant for static discovery and monitoring. See the Information Driver operation mode below (Relative path to `$GW_LOCATION`).
- **-b <base>**: The Virtual Organization name in the DN of the LDIF entries, i.e. the `Mds-Vo-name` attribute, only relevant for MDS2.
- **-f <filter>**: Additional requirements to be imposed on all the hosts managed by this Information Driver, in LDIF format.

These options allow you to configure your Information Drivers in the three operation modes, described below.

### 3.4.1. Static Discovery and Monitoring (SS mode)

In this mode, hosts are statically discovered by reading a host list file (note: each time it is read). Also the attributes of each host are read from files. Hint: Use this mode for testing purposes and not in a production environment. To configure a Information Driver in SS mode use the host list option, for example:

```
IM_MAD = static:gw_im_mad_static:-l examples/im/host.static:gridftp:ws
```

The host list file contains one host per line, with format:

```
FQDN    attribute_file
```

where:

- **FQDN**: is the Full Qualified Domain Name of the host.

- **attribute\_file**: is the name of the file with the static attributes of this host. Relative to the `GW_LOCATION` directory.

For example (you can find this file, `host.list`, in `$GW_LOCATION/examples/im/`)

```
hydrus.dacya.ucm.es examples/im/hydrus.attr
draco.dacya.ucm.es examples/im/draco.attr
```

The *attribute\_file* includes a *single line* with the host information and *other lines* with the information of each queue (one line per queue). Use the examples below as templates for your hosts.

Example of attribute file for a PBS cluster (you can find this file in `$GW_LOCATION/examples/im/`):

```
HOSTNAME="hydrus.dacya.ucm.es" ARCH="i686" OS_NAME="Linux" OS_VERSION="2.6.4"
CPU_MODEL="Intel(R) Pentium(R) 4 CPU 2" CPU_MHZ=2539 CPU_FREE=098 CPU_SMP=1
NODECOUNT=4 SIZE_MEM_MB=503 FREE_MEM_MB=188 SIZE_DISK_MB=55570
FREE_DISK_MB=39193 FORK_NAME="jobmanager-fork" LRMS_NAME="jobmanager-pbs"
LRMS_TYPE="pbs" QUEUE_NAME[0]="q4small" QUEUE_NODECOUNT[0]=1
QUEUE_FREENODECOUNT[0]=4 QUEUE_MAXTIME[0]=0 QUEUE_MAXCPU[0]=20
QUEUE_MAXCOUNT[0]=4 QUEUE_MAXRUNNINGJOBS[0]=0 QUEUE_MAXJOBSINQUEUE[0]=1
QUEUE_STATUS[0]="enabled" QUEUE_DISPATCHTYPE[0]="batch"
QUEUE_NAME[1]="q4medium" QUEUE_NODECOUNT[1]=4 QUEUE_FREENODECOUNT[1]=4
QUEUE_MAXTIME[1]=0 QUEUE_MAXCPU[1]=120 QUEUE_MAXCOUNT[1]=4
QUEUE_MAXRUNNINGJOBS[1]=0 QUEUE_MAXJOBSINQUEUE[1]=1
QUEUE_STATUS[1]="enabled" QUEUE_DISPATCHTYPE[1]="batch"
```

Example of attribute file for a Fork Desktop (you can find this file in `$GW_LOCATION/examples/im/`):

```
HOSTNAME="draco.dacya.ucm.es" ARCH="i686" OS_NAME="Linux" OS_VERSION="2.6-xen"
CPU_MODEL="Intel(R) Pentium(R) 4 CPU 3" CPU_MHZ=3201 CPU_FREE=185 CPU_SMP=2
NODECOUNT=2 SIZE_MEM_MB=431 FREE_MEM_MB=180 SIZE_DISK_MB=74312
FREE_DISK_MB=40461 FORK_NAME="jobmanager-fork" LRMS_NAME="jobmanager-fork"
LRMS_TYPE="fork" QUEUE_NAME[0]="default" QUEUE_NODECOUNT[0]=1
QUEUE_FREENODECOUNT[0]=1 QUEUE_MAXTIME[0]=0 QUEUE_MAXCPU[0]=0
QUEUE_MAXCOUNT[0]=0 QUEUE_MAXRUNNINGJOBS[0]=0 QUEUE_MAXJOBSINQUEUE[0]=0
QUEUE_STATUS[0]="0" QUEUE_DISPATCHTYPE[0]="Immediate"
```

To use the WS version of these files just change `jobmanager-fork` with `Fork` and `jobmanager-pbs` with `PBS`.

### 3.4.2. Static Discovery and Dynamic Monitoring (SD mode)

Hosts are discovered by reading a host list file. However, the information of each host is gathered by querying its information service (GRIS in MDS2 or the `DefaultIndexService` in MDS4). Hint: Use this mode if the resources in your Grid does not vary too much, i.e. resource are not added or removed very often. To configure an Information Driver in SD mode, use the host list option, for example:

```
IM_MAD = glue:gw_im_mad_mds2_glue:-1 examples/im/host.list:gridftp:prews
```

In this case the host list file contains one host per line, with the format:

```
FQDN
```

...  
FQDN

where:

- **FQDN**: is the Full Qualified Domain Name of the host.

For example (you can find this file in `$GW_LOCATION/examples/im/`)

```
hydrus.dacya.ucm.es
ursa.dacya.ucm.es
draco.dacya.ucm.es
```



### Note

The information services of each host (GRIS or/and DefaultIndexServices) must be properly configured to use this mode.



### Important

You can configure your IMs to work in a dynamic monitoring mode but get some static information from an attributes file (as described in the SS mode). This configuration is useful when you want to add some host attributes missing from the IndexService (like software availability, special hardware devices...). You can see a useful use of this mode in section [Chapter 8, Troubleshooting](#).

## 3.4.3. Dynamic Discovery and Monitoring (DD mode)

In this mode, hosts are discovered and monitored by directly accessing the Grid Information Service. Hint: Use this mode if the resources in your Grid does vary too much, i.e. resource are added or removed very often. To configure a Information Driver in SD mode, use the server option, for example:

```
IM_MAD = mds4:gw_im_mad_mds4:-s hydrus.dacya.ucm.es:gridftp:ws
```



### Note

A hierarchical information service (GIIS or/and DefaultIndexService) must be properly configured to use this mode.

If you are using an MDS2 information service you may need to specify the Virtual Organization name in the DN of the LDIF entries (*Mds-vo-name*) with the base option described above.



### Note

You can simultaneously use as many Information Drivers as you need (up to 10). So GridWay allows you to simultaneously use MDS2 and MDS4 Services. You can also use resources from different Grids at the same time.



### Note

You can mix SS, SD and DD modes in the same Information Driver.

### 3.4.4. Separate Storage and Computing Element

There is a way to specify a different machine to be used as gsiftp endpoint than the one that has the gatekeeper installed. This is useful when the CE machine does not have gsiftp server configured but there is another machine that works as a Storage Element. Right now, this information could be set statically but the rest of the information can be updated dynamically. To use this feature you have to create a file for each host you want to configure with extra information and another file with pairs of host and file name (as described above for the SS mode). The filename can be a full path or a relative path to `GW_LOCATION`. Then in the IM MAD you must specify the list file with `-l`, like this (in `gwd.conf`):

```
IM_MAD = mds4:gw_im_mad_mds4:-l etc/gridway/host.list:gridftp:ws
```

The file list should look like this:

```
wsgram-host1.domain.com etc/gridway/wsgram-host1.attr
wsgram-host2.domain.com etc/gridway/wsgram-host2.attr
```

And the attributes file for each node should look like this:

```
SE_HOSTNAME="gridftp-host1.domain.com"
```

## 4. Integration Guides

GridWay architecture flexibility allows it to interoperate with grids based on different middleware stacks. The following documents states how to configure GridWay for the following infrastructures:

- Integration of GridWay within the EGEE infrastructure[\[HTML\]](#)<sup>5</sup>[\[PDF\]](#)<sup>6</sup>
- Integration of GridWay within the OSG infrastructure[\[HTML\]](#)<sup>7</sup>[\[PDF\]](#)<sup>8</sup>
- Integration of GridWay within the Teragrid infrastructure[\[HTML\]](#)<sup>9</sup>[\[PDF\]](#)<sup>10</sup>
- Integration of GridWay within the Nordugrid infrastructure[\[HTML\]](#)<sup>11</sup>[\[PDF\]](#)<sup>12</sup>
- Creating MADs: SSH Example[\[HTML\]](#)<sup>13</sup>[\[PDF\]](#)<sup>14</sup>

<sup>5</sup> <http://www.gridway.org/documentation/stable/egeehowto/>

<sup>6</sup> <http://www.gridway.org/documentation/stable/egeehowto.pdf>

<sup>7</sup> <http://www.gridway.org/documentation/stable/osghowto/>

<sup>8</sup> <http://www.gridway.org/documentation/stable/osghowto.pdf>

<sup>9</sup> <http://www.gridway.org/documentation/stable/tghowto/>

<sup>10</sup> <http://www.gridway.org/documentation/stable/tghowto.pdf>

<sup>11</sup> <http://www.gridway.org/documentation/stable/norduhowto>

<sup>12</sup> <http://www.gridway.org/documentation/stable/norduhowto.pdf>

<sup>13</sup> <http://www.gridway.org/documentation/stable/sshowto>

<sup>14</sup> <http://www.gridway.org/documentation/stable/sshowto.pdf>

---

# Chapter 4. Deploying

[TBD] GridWay is installed in the Globus directory when you issue the **make install**.

---

# Chapter 5. Testing

## 1. Verifying the installation

In order to test the GridWay installation, login as your user account, in the single-mode installation, or as the <gwad-min> account, in the multiple-user installation, and follow the steps listed below:

1. Set up the environment variables GW\_LOCATION and PATH:

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

or

```
$ setenv GW_LOCATION <path_to_GridWay_installation>
$ setenv PATH $PATH:$GW_LOCATION/bin
```

depending on the shell you are using.

2. Generate a valid proxy

```
$ grid-proxy-init
Your identity: /O=Grid/OU=GRIDWAY/CN=GRIDWAY User
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Mon Oct 29 03:29:17 2005
```

3. Show the GridWay license:

```
$ gwd -v
Copyright 2002-2008 GridWay Team, Distributed Systems Architecture
Group, Universidad Complutense de Madrid

GridWay 5.4 is distributed and licensed for use under the terms of the
Apache License, Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0).
```

4. Start the GridWay daemon (GWD) (in multiple-mode add the **-m** option):

```
$ gwd
```

5. Check the connection to GWD:

```
$ gwps
USER          JID DM   EM   START   END     EXEC   XFER   EXIT NAME          HOST
$ gwghost
HID PRIO  OS          ARCH  MHZ %CPU  MEM(F/T)  DISK(F/T)  N(U/F/T) LRMS
```

6. Stop GWD:

```
$ pkill gwd
```

To perform more sophisticated tests, check the *User Guide*. If you experience problems, check [Chapter 8, Troubleshooting](#).

## 2. Test Suite

GridWay is shipped with a test suite, available in the test directory. The test suite exercises different parts of GridWay, and can be used to track functionality bugs. However you need a working GridWay installation and testbed to execute the suite. Usage information is available with "gwtest -h". Tests can be performed individually (using the test id) or all together automatically.

**Table 5.1. GridWay tests description.**

Test #	Test Name	Description
1	Normal Execution (SINGLE)	Submits a single job and verifies it is executed correctly
2	Normal Execution (BULK)	Submits an array of 5 jobs and verifies that all of them are executed correctly.
3	Pre Wrapper	Verifies that GridWay is able to execute the pre wrapper functionality.
4	Prolog Fail (Fake Stdin) No Reschedule	Submits a single job that fails in the prolog state due to a wrong input file for stdin.
5	Prolog Fail (Fake Stdin) Reschedule	Equal to the previous one, but GridWay tries to reschedule the job up to 2 times.
6	Prolog Fail (Fake Input File) No Reschedule	Same as #4 with a wrong input file for the executable.
7	Prolog Fail (Fake Executable) No Reschedule	Same as #4 with a wrong filename for the executable.
8	Prolog Fail (Fake Executable) No Reschedule	Same as #4 with a wrong filename for the executable.
9	Prolog Fail (Fake Stdin) No Reschedule (BULK)	Same as #4 submitting an array of 5 jobs.
10	Execution Fail No Reschedule	Submits a single job designed to fail (bad exit code) and verifies the correctness of the final state (failed).
11	Execution Fail Reschedule	Same as #9 but GridWay tries to reschedule the job up to 2 times.
12	Hold Release	Submits a single job on hold, releases it and verifies that it is executed correctly.
13	Stop Resume	Submits a single job, stops it (in Wrapper state), resumes it and verifies that it is executed correctly.
14	Kill Sync	Submits a job and kills it using a synchronous signal.
15	Kill Async	Submits a job and kills it using an asynchronous signal.
16	Kill Hard	Submits a job and hard kills it.
17	Migrate	Submits a job and sends a migrate signal when it reaches the Wrapper state. It then verifies the correct execution of the job.
18	Checkpoint local	Submits a job which creates a checkpoint file and verifies the correct execution of the job and the correct creation of the checkpoint file.
19	Checkpoint remote server	Same as #17 but the checkpoint file is created in a remote gsiftp server.
20	Wait Timeout	Submits a job and waits for it repeatedly using short timeouts until it finishes correctly.
21	Wait Zerotimeout	Same as #19 but with zero timeout (effectively, an asynchronous wait).
22	Input Output files	Tests the different methods GridWay offers to stage files (both input and output).
23	Epilog Fail (Fake Output) No Reschedule	Submits a single job that fails in the epilog state due to a wrong output filename.

24	Epilog Fail (Fake Output) Resched- Same as #22 but GridWay tries to reschedule the job up to 2 times. ule
25	Epilog Fail (Fake Output) No Res- Same as #22 but submitting an array of 5 jobs. chedule (BULK)

### 3. DRMAA Test Suite

GridWay also ships with a [DRMAA test suite](#)<sup>1</sup>, conceived to test the DRMAA Java implementations. Download and untar the following tarball, then follow the instructions found in the README file.

---

<sup>1</sup> <http://www.gridway.org/documentation/stable/drmaaTestSuite.tgz>

---

# Chapter 6. Security considerations

## 1. Security Considerations for GridWay

Access authorization to the GridWay server is done based on the Unix identity of the user (accessing GridWay directly or through a Web Services GRAM, as in [GridGateWay<sup>1</sup>](#)). Hence, security in GridWay has the same implications as the Unix accounts of their users.

Also, GridWay uses proxy certificates to use Globus services, so the security implications of managing certificates also must be taken into account

---

<sup>1</sup> <http://www.grid4utility.org>

---

# Chapter 7. Debugging

The following is sys admin logging information (based on Java WS Core):

## 1. Logging in Java WS Core

The following information applies to Java WS Core and all services built on Java WS Core.

Java WS Core server side has two types of loggers. One logger is used for development logging and by default writes to standard out. The other logger includes system administration information and is [CEDPs best practices](#)<sup>1</sup> compliant.

On client side, only developer logging is available and is configured using `log4j.properties`.

### 1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)<sup>2</sup> API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)<sup>3</sup> as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)<sup>4</sup>.

#### 1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)<sup>5</sup>. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

#### 1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

---

<sup>1</sup> <http://cedps.net/index.php/LoggingBestPractices>

<sup>2</sup> <http://jakarta.apache.org/commons/logging/>

<sup>3</sup> <http://logging.apache.org/log4j/>

<sup>4</sup> [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

<sup>5</sup> <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

## 1.2. Configuring system administration logs

The specific logger to edit will be `log4j.logger.sysadmin` in `$GLOBUS_LOCATION/container-log4j.properties`. There you can configure the following properties:

```
log4j.appender.infoCategory=org.apache.log4j.RollingFileAppender
log4j.appender.infoCategory.Threshold=INFO
log4j.appender.infoCategory.File=var/containerLog
log4j.appender.infoCategory.MaxFileSize=10MB
log4j.appender.infoCategory.MaxBackupIndex=2
```

Above implies the logging file is rolling with each file size limited to 10MB and the logging information is stored in `$GLOBUS_LOCATION/var/containerLog`.

## 1.3. Sample log file

The [sample log file](#)<sup>6</sup> contains many log entries for various scenarios in the Java WS container.

---

<sup>6</sup> <http://www.globus.org/toolkit/docs/4.2/4.2.1/common/javawscore/sample-container-log.txt>

---

# Chapter 8. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

## 1. Errors

**Table 8.1. Gridway Errors**

Error Code	Definition	Possible Solutions
Lock file exists	Another GWD may be running.	Be sure that no other GWD is running, then remove the lock file and try again.
Error in MAD initialization	There may be problems with the proxy certificate, bin directory, or the executable name of a MAD may not be in the correct location.	Check that you have generated a valid proxy (for example with the <b>grid-proxy-info</b> command). Also, check that the directory <code>\$GW_LOCATION/bin</code> is in your path, and the executable name of all the MADs is defined in <code>gwd.conf</code> .
Could not connect to gwd	GridWay may not be running or there may be something wrong with the connection.	Be sure that GWD is running; for example: <pre>pgrep -l gwd</pre> If it is running, check that you can connect to GWD; for example: <pre>telnet `cat \$GW_LOCATION/var/gwd.port`</pre>

## 2. Debugging

For more detailed developer debugging information, see [Debugging](#). For information about sys admin logging, see [Chapter 7, Debugging](#).

# **GT 4.2.1 GridWay: User's Guide**

---

## 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><sup>1</sup>

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.

---

<sup>1</sup> <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 <i>REQUIREMENTS</i> and <i>RANK</i> . ....	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 (**gwsu**bs**mit**, **gww**ai**t**, **gww**ki**ll**, **gww**h**osts**...) 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 **gwsu**bs**mit** 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 **gww**s**t** command. You can control your jobs at runtime using the **gww**ki**ll** command. You can synchronize your jobs using the **gww**ai**t** command. You can find out what resources your job has used with the **gww**h**istory** command.

System monitoring commands allow you to gather information of the GridWay system and the Grids you are using. These commands are: **gww**u**ser** to show information about the users using GridWay; **gww**h**ost** to monitor the available hosts in the testbed; and **gww**ac**ct** to print usage (accounting) information per user or host.



### Note

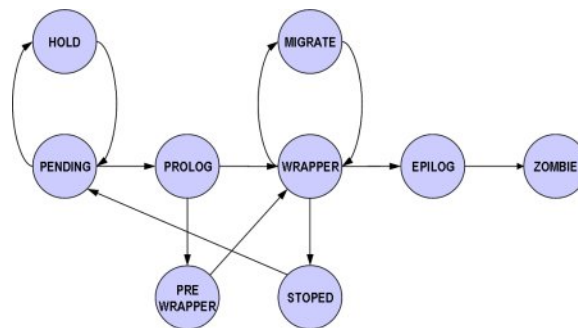
Every command has a **-h** option which shows its usage and available options.

### 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 (`prolog`): 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-wrapper (`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 (`epilog`): 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 (`actv`): 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_GW_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
```



## Note

This step is only needed if your environment has not been configured, ask your administrator.

5. DRMAA extensions for all the languages use the dynamic drmaa libraries provided by GridWay. To use this 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 this 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.**

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

Scheduling	
RESCHEDULING_INTERVAL	How often GridWay searches the Grid for better resources to run this job. (0 = never)
RESCHEDULING_THRESHOLD	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.
DEADLINE	Deadline (format [[D:]H:]M) to start the job (0 = none).
Performance	
SUSPENSION_TIMEOUT	Maximum suspension time (seconds) in the local job management system. If exceeded the job is migrated to another host. (0 = never)
CPULOAD_THRESHOLD	If the CPU assigned to your job is less than this given percentage, the job will be migrated
MONITOR	Optional program to monitor job performance
Fault tolerance	
RESCHEDULE_ON_FAILURE	Behavior in case of failure. Possible values are 'yes' or 'no'
NUMBER_OF_RETRIES	Number of retries in case of failure. GridWay follows a linear backoff strategy for re-trying file transfers and job submissions.
Advanced job execution	
WRAPPER	Script for wrapper. stdout and stderr streams of this program can be found in directory \$GW_LOCATION/var/\$GW_JOB_ID as files <code>stdout.wrapper.\$GW_RESTARTED</code> and <code>stderr.wrapper.\$GW_RESTARTED</code>
PRE_WRAPPER	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 <code>stdout.pre_wrapper.\$GW_RESTARTED</code> and <code>stderr.pre_wrapper.\$GW_RESTARTED</code>
PRE_WRAPPER_ARGUMENTS	Arguments to the pre-wrapper program.

### 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:

- *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_server/~/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 di
```

### 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 define a checkpointing server with the `CHECKPOINT_URL` Job Template parameter.

Example:

```
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:

```
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.**

<code>\${JOB_ID}</code>	The job identifier
<code>\${ARRAY_ID}</code>	The job array identifier (-1 if the job does not belong to any)
<code>\${TASK_ID}</code>	The task identifier within the job array (-1 if the job does not belong to any)
<code>\${TOTAL_TASKS}</code>	The total number of tasks in the job array (-1 if the job does not belong to any)
<code>\${ARCH}</code>	The architecture of the selected execution host
<code>\${PARAM}</code>	Allows the assignment of arbitrary start and increment values for array jobs ( <code>start + increment*GW_TASK_ID</code> ). 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>gws submit</code> command.
<code>\${MAX_PARAM}</code>	Upper bound of the <code>\${PARAM}</code> variable.

### 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 ')'

```

Each expression is evaluated to 1 (TRUE) or 0 (FALSE). Only those hosts for which the requirement expression is evaluated to TRUE will be considered to execute the job.

Logical operators are as expected ( less '<', greater '>', '&' AND, '|' OR, '!' NOT), '=' means equals with integers. When you use '=' operator with strings, it performs a shell wildcard pattern matching.

Examples:

```

REQUIREMENTS = LRMS_NAME = "*pbs*"; # Only use pbs like jobmanagers
REQUIREMENTS = HOSTNAME = "*.es"; #Only hosts in Spain
REQUIREMENTS = HOSTNAME = "hydrus.dacya.ucm.es"; #Only use hydrus.dacya.ucm.es

```

## 5.2. Rank expression syntax

The syntax of the rank expressions is defined as:

```

stmt ::= expr ';'
expr ::= VARIABLE
      | INTEGER
      | expr '+' expr
      | expr '-' expr
      | expr '*' expr
      | expr '/' expr
      | '-' expr
      | '(' expr ')'

```

Rank expressions are evaluated using each host information. '+', '-', '\*', '/' and '-' are arithmetic operators, so only integer values should be used in rank expressions.

## 5.3. Requirement and rank variables

To set the *REQUIREMENTS* and *RANK* parameter values the following extended set of variables, provided by the Information Manager, can be used:

**Table 3.3. Variables that can be used to define the job *REQUIREMENTS* and *RANK*.**

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

## 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><sup>1</sup>

### 6.2. JSDL document structure

The JSDL document structure is as follows:

```

<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>*
|-----</OperatingSystemType>?
|-----<OperatingSystemVersion>?
|-----<Description>?
|-----<xsd:any##other>*
|-----</OperatingSystem>?

```

<sup>1</sup> <http://forge.gridforum.org/sf/projects/jsdl-wg>

```

|-----<CPUArchitecture>
|-----<CPUArchitectureName>
|-----<xsd:any##other>*
|-----</CPUArchitecture>?
|-----<IndividualCPUSpeed>?
|-----<IndividualCPUTime>?
|-----<IndividualCPUCount>?
|-----<IndividualNetworkBandwidth>?
|-----<IndividualPhysicalMemory>?
|-----<IndividualVirtualMemory>?
|-----<IndividualDiskSpace>?
|-----<TOTALCPUTime>?
|-----<TOTALCPUCount>?
|-----<TOTALPhysicalMemory>?
|-----<TOTALVirtualMemory>?
|-----<TOTALDiskSpace>?
|-----<TOTALResourceCount>?
|-----<xsd:any##other>*
|-----</Resources>?
|-----<DataStaging>
|-----<FileName>
|-----<FileSystemName>?
|-----<CreationFlag>
|-----<DeleteOnTermination>?
|-----<Source>
|-----<URI>?
|-----<xsd:any##other>*
|-----</Source>?
|-----<Target>
|-----<URI>?
|-----<xsd:any##other>*
|-----</Target>?
|-----<xsd:any##other>*
|-----</DataStaging>*
|-----<xsd:any##other>*
</JobDefinition>

```

 **Note**

The symbol "?" denotes zero or one occurrences, "\*" denotes zero or more occurrences and "+" denotes one or more occurrences.

### 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:

```

<POSIXApplication name="xsd:NCName" ?>
|-----<Executable>?
|-----<Argument>*
|-----<Input>?

```

```

|-----<Output>?
|-----<Error>?
|-----<WorkingDirectory>?
|-----<Environment>*
|-----<WallTimeLimit>?
|-----<FileSizeLimit>?
|-----<CoreDumpLimit>?
|-----<DataSegmentLimit>?
|-----<LockedMemoryLimit>?
|-----<MemoryLimit>?
|-----<OpenDescriptorsLimit>?
|-----<PipeSizeLimit>?
|-----<StackSizeLimit>?
|-----<CPULimit>?
|-----<ProcessCountLimit>?
|-----<VirtualMemoryLimit>?
|-----<ThreadCountLimit>?
|-----<UserName>?
|-----<GroupName>?
</POSIXApplication name="xsd:NCName" ?>

```

More details at <https://forge.gridforum.org/sf/projects/jsdl-wg><sup>2</sup>

## 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:

```

<HPCProfileApplication name="xsd:NCName" ?>
|-----<Executable>?
|-----<Argument>*
|-----<Input>?
|-----<Output>?
|-----<Error>?
|-----<WorkingDirectory>?
|-----<Environment>*
|-----<UserNamet>?
</HPCProfileApplication name="xsd:NCName" ?>

```

More details at <https://forge.gridforum.org/sf/projects/jsdl-wg><sup>3</sup>

## 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.

<sup>2</sup> <http://forge.gridforum.org/sf/projects/jsdl-wg>

<sup>3</sup> <http://forge.gridforum.org/sf/projects/jsdl-wg>

**Table 3.4. JSDL vs GWJT**

<b>JSDL Element</b>	<b>GWJT Attribute</b>	<b>Adoption</b>
JobDefinition	-	Supported
JobDescription	-	Supported
JobIdentification	-	Supported
JobName	NAME	Supported
JobAnnotation	-	Not supported
JobProject	-	Not supported
Application	-	Supported
ApplicationName	-	Supported
ApplicationVersion	-	Supported
Description	-	Supported
Resources	-	Supported
CandidateHosts	-	Supported
HostName	HOSTNAME	Supported
FileSystem	-	Not supported
MountPoint	-	Not supported
MountSource	-	Not supported
DiskSpace	-	Not supported
FileSystemType	-	Not supported
ExclusiveExecution	-	Not supported
OperatingSystem	-	Supported
OperatingSystemType	-	Supported
OperatingSystemName	OS_NAME	Supported
OperatingSystemVersion	OS_VERSION	Supported
CPUArchitecture	-	Supported
CPUArchitectureName	ARCH	Supported
IndividualCPUSpeed	CPU_MHZ	Supported
IndividualCPUTime	-	Not supported
IndividualCPUCount	NODECOUNT	Supported
IndividualNetworkBandwidth	-	Not supported
IndividualPhysicalMemory	MEM_MB	Supported
IndividualVirtualMemory	-	Not supported
IndividualDiskSpace	SIZE_DISK_MB	Supported
TOTALCPUTime	-	Not supported
TOTALCPUCount	-	Not supported
TOTALPhysicalMemory	-	Not supported
TOTALVirtualMemory	-	Not supported
TOTALDiskSpace	-	Not supported

TOTALResourceCount	-	Not supported
DataStaging	-	Supported
FileName	-	Supported
FileSystemName	-	Not supported
CreationFlag	-	Supported
DeleteOnTermination	-	Supported
Source	INPUT_FILES	Supported
Target	OUTPUT_FILES	Supported
URI	-	Supported
POSIXApplication	-	Supported
Executable	EXECUTABLE	Supported
Argument	ARGUMENTS	Supported
Input	STDIN_FILE	Supported
Output	STDOUT_FILE	Supported
Error	STDERR_FILE	Supported
WorkingDirectory	-	Not supported
Environment	ENVIRONMENT	Supported
WallTimeLimit	-	Not supported
FileSizeLimit	-	Not supported
CoreDumpLimit	-	Not supported
DataSegmentLimit	-	Not supported
LockedMemoryLimit	-	Not supported
MemoryLimit	-	Not supported
OpenDescriptorsLimit	-	Not supported
PipeSizeLimit	-	Not supported
StackSizeLimit	-	Not supported
CPUTimeLimit	-	Not supported
ProcessCountLimit	-	Not supported
VirtualMemoryLimit	-	Not supported
ThreadCountLimit	-	Not supported
UserName	-	Not supported
GroupName	-	Not supported

## 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"?>

<jsdsl:JobDefinition xmlns="http://www.example.org/"
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdsl:JobDescription>
    <jsdsl:JobIdentification>
      <jsdsl:JobName>Simple Application GW Template vs JSDL</jsdl:JobName>
      <jsdsl:Description> This is a simple example to describe the main
        differences between GW Template and the JSDL schema.
      </jsdl:Description>
    </jsdl:JobIdentification>
    <jsdsl:Application>
      <jsdsl:ApplicationName>ls</jsdl:ApplicationName>
      <jsdsl-posix:POSIXApplication>
        <jsdsl-posix:Executable>/bin/ls</jsdl-posix:Executable>
        <jsdsl-posix:Argument>-la file.txt</jsdl-posix:Argument>
        <jsdsl-posix:Environment name="LD_LIBRARY_PATH">/usr/local/lib</jsdl-posix:Environment>
        <jsdsl-posix:Input>/dev/null</jsdl-posix:Input>
        <jsdsl-posix:Output>stdout.${JOB_ID}</jsdl-posix:Output>
        <jsdsl-posix:Error>stderr.${JOB_ID}</jsdl-posix:Error>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdsl:Resources>
      <jsdsl:CandidateHost>
        <jsdsl:HostName>*.dacya.ucm.es</jsdl:HostName>
      </jsdl:CandidateHost>
      <jsdsl:CPUArchitecture>
        <jsdsl:CPUArchitectureName>x86_32</jsdl:CPUArchitectureName>
      </jsdl:CPUArchitecture>
    </jsdl:Resources>
    <jsdsl>DataStaging>
      <jsdsl:FileName>file.txt</jsdl:FileName>
      <jsdsl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdsl>DeleteOnTermination>>true</jsdl>DeleteOnTermination>
      <jsdsl:Source>
        <jsdsl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/file1.txt</jsdl:URI>
      </jsdl:Source>
    </jsdl>DataStaging>
    <jsdsl>DataStaging>
      <jsdsl:FileName>stdout.${JOB_ID}</jsdl:FileName>
      <jsdsl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdsl>DeleteOnTermination>>true</jsdl>DeleteOnTermination>
      <jsdsl:Target>
        <jsdsl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stdout.${JOB_ID}</jsdl:URI>
      </jsdl:Target>
    </jsdl>DataStaging>
    <jsdsl>DataStaging>
      <jsdsl:FileName>stderr.${JOB_ID}</jsdl:FileName>

```

```

<jsd1:CreationFlag>overwrite</jsdl:CreationFlag>
<jsd1>DeleteOnTermination>>true</jsdl>DeleteOnTermination>
<jsd1:Target>
  <jsd1:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stderr.${JOB_ID}</jsdl:URI>
</jsdl:Target>
</jsdl:DataStaging>
</jsdl:JobDescription>
</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"?>

<jsd1:JobDefinition xmlns="http://www.example.org/"
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsd1:JobDescription>
    <jsd1:JobIdentification>
      <jsd1:JobName>Simple Application GW Template vs JSDL</jsdl:JobName>
      <jsd1:Description> This is a simple example to describe the main
        differences between GW Template and the JSDL schema.
      </jsdl:Description>
    </jsdl:JobIdentification>
    <jsd1:Application>
      <jsd1:ApplicationName>ls</jsdl:ApplicationName>
      <jsd1-hpcpa:HPCProfileApplication>
        <jsd1-hpcpa:Executable>/bin/ls</jsdl-hpcpa:Executable>
        <jsd1-hpcpa:Argument>-la file.txt</jsdl-hpcpa:Argument>
        <jsd1-hpcpa:Environment name="LD_LIBRARY_PATH">/usr/local/lib</jsdl-hpcpa:Environment>
        <jsd1-hpcpa:Input>/dev/null</jsdl-hpcpa:Input>
        <jsd1-hpcpa:Output>stdout.${JOB_ID}</jsdl-hpcpa:Output>
        <jsd1-hpcpa:Error>stderr.${JOB_ID}</jsdl-hpcpa:Error>
      </jsdl-hpcpa:HPCProfileApplication>
    </jsdl:Application>
  </jsdl:JobDescription>
</jsdl:JobDefinition>

```

```

<jsdsl:Resources>
  <jsdsl:CandidateHost>
    <jsdsl:HostName>*.dacya.ucm.es</jsdl:HostName>
  </jsdl:CandidateHost>
  <jsdsl:CPUArchitecture>
    <jsdsl:CPUArchitectureName>x86_32</jsdl:CPUArchitectureName>
  </jsdl:CPUArchitecture>
</jsdl:Resources>
<jsdsl:DataStaging>
  <jsdsl:FileName>file.txt</jsdl:FileName>
  <jsdsl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdsl>DeleteOnTermination>true</jsdl>DeleteOnTermination>
  <jsdsl:Source>
    <jsdsl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/file1.txt</jsdl:URI>
  </jsdl:Source>
</jsdl:DataStaging>
<jsdsl:DataStaging>
  <jsdsl:FileName>stdout.${JOB_ID}</jsdl:FileName>
  <jsdsl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdsl>DeleteOnTermination>true</jsdl>DeleteOnTermination>
  <jsdsl:Target>
    <jsdsl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stdout.${JOB_ID}</jsdl:URI>
  </jsdl:Target>
</jsdl:DataStaging>
<jsdsl:DataStaging>
  <jsdsl:FileName>stderr.${JOB_ID}</jsdl:FileName>
  <jsdsl:CreationFlag>overwrite</jsdl:CreationFlag>
  <jsdsl>DeleteOnTermination>true</jsdl>DeleteOnTermination>
  <jsdsl:Target>
    <jsdsl:URI>gsiftp://hydrus.dacya.ucm.es/home/jose/stderr.${JOB_ID}</jsdl:URI>
  </jsdl:Target>
</jsdl:DataStaging>
</jsdl:JobDescription>
</jsdl:JobDefinition>

```

### 6.6.2.2. 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

```

---

# 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 **gws submit** command to submit the job:

```
$ gws submit -t example/jt
```

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

HID	PRIOR	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
1	1	Linux2.6.17-2-6	x86	3216	189	384/2027	105129/118812	0/2/2	Fork	draco.dacya
2	1	Linux2.6.18-3-a	x86_6	2211	100	749/1003	76616/77844	0/2/2	SGE	aquila.dacya
3	1			0	0	0/0	0/0	0/0/0		hydrus.dacya
4	1	Linux2.6.18-3-a	x86_6	2009	74	319/878	120173/160796	0/1/1	Fork	orion.dacya
5	1	Linux2.6.16.13-	x86	3200	100	224/256	114/312	0/6/6	SGE	ursa.dacya

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	PRIOR	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

QUEUE	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	PRIOR	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

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

```

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  pend  ---- 000 10:42:09  ---:---:--- 0:00:00  0:00:00  --  jt      --

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  prol  ---- 000 10:42:09  ---:---:--- 0:00:00  0:00:01  --  jt      cygnus.dacya.u

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  wrap  ---- 000 10:42:09  ---:---:--- 0:00:27  0:00:04  --  jt      cygnus.dacya.u

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  wrap  pend 000 10:42:09  ---:---:--- 0:00:27  0:00:04  --  jt      cygnus.dacya.u

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  wrap  actv 000 10:42:09  ---:---:--- 0:00:27  0:00:04  --  jt      cygnus.dacya.u

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  epil  ---- 000 10:42:09  ---:---:--- 0:00:31  0:00:05  --  jt      cygnus.dacya.u

USER      JID DM   EM   RWS START      END          EXEC      XFER      EXIT NAME  HOST
jlvazquez 0  done  ---- 000 10:42:09  10:43:01  0:00:31  0:00:08  0  jt      cygnus.dacya.u

```

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/

```

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

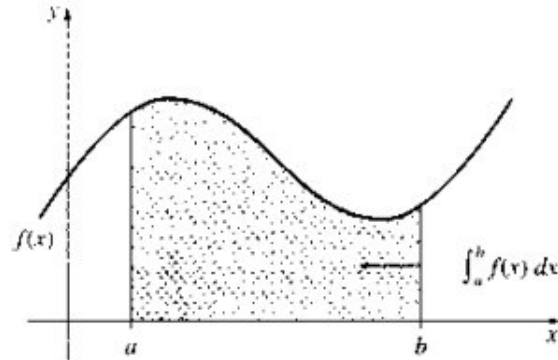
```

Done! You have done your first execution with GridWay!

## 2. Array jobs: Calculating the number

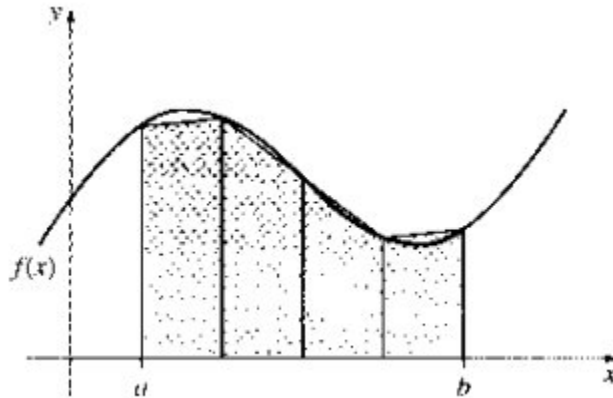
### 2.1. Defining the problem

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

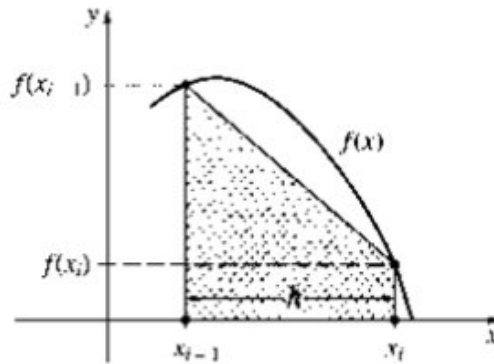


Being  $f(x) = 4/(1+x^2)$ . So,  $\pi$  will be the integral of  $f(x)$  in the interval  $[0,1]$ .

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  $\pi$  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:

```
#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);
    }
}
```

```
l_sum *= h;

printf("%0.12g\n", l_sum);

return 0;
}
```

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:

```
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
```

```
TASK JOB
0 0
1 1
2 2
3 3
```

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:

```
$ gwwait -v -A 0
0 : 0
1 : 0
2 : 0
3 : 0
```

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

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

gwwait -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  $\pi$  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:

```
#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, fabs(pi - PI25DT));
    endwtime = MPI_Wtime();
    printf("wall clock time = %f\n", endwtime-startwtime);
}

MPI_Finalize();

return 0;
}

```



## 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 **mpicc**:

```
$ mpicc -O3 mpi.c -o mpi
```

Now you must create a Job Template. In this case, we will call it `mpi.jt`:

```

EXECUTABLE    = mpi

STDOUT_FILE   = stdout.${JOB_ID}
STDERR_FILE   = stderr.${JOB_ID}

RANK          = CPU_MHZ

TYPE          = "mpi"
NP            = 2

```

## 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:

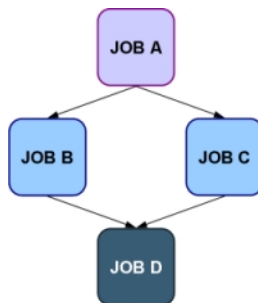
- Sequence, parallelism, branching and looping structures
- The workflow can be described in an abstract form without referring to specific resources for task execution
- Quality of service constraints and fault tolerance are defined at task level

Job dependencies can be specified at submission by using the **-d** option of the **gwsuubmit** 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 **gckill**.

## 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:

```

$ gwsuubmit -v -t A.jt
JOB ID: 5
  
```

```
$ gws submit -v -t B.jt -d "5"
JOB ID: 6
```

```
$ gws submit -v -t C.jt -d "5"
JOB ID: 7
```

```
$ gws submit -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.

```
#!/bin/sh

A_ID=`gws submit -v -t A.jt | cut -f2 -d':' | cut -f2 -d' '`
B_ID=`gws submit -v -t B.jt -d "$A_ID" | cut -f2 -d':' | cut -f2 -d' '`
C_ID=`gws submit -v -t C.jt -d "$A_ID" | cut -f2 -d':' | cut -f2 -d' '`
D_ID=`gws submit -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

```
gws submit <-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 $PARAM=start + increment * TASK\_ID$ , and $MAX\_PARAM = start+increment*(tasks-1)$ . 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**

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

---

# 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 Prints help.

-n Do not print the header lines

job\_id Job identification as provided by gwps.

# Output field description

**Table 6. Field information**

NAME	DESCRIPTION
HID	host unique identification assigned by the Gridway system.
START	the time the job start its execution on this host.
END	the time the job left this host, because it finished or it was migrated.
PROLOG	total prolog (file stage-in phase) time.
WRAPPER	total wrapper (execution phase) time.
EPILOG	total epilog (file stage-out phase) time.
MIGR	total migration time.
REASON	the reason why the job left this host.
QUEUE	name of the queue.
HOST	FQDN of the host.

---

# 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**

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

**Table 8. Queue field information**

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

---

# Name

Job Control Command -- controls job execution

```
gkill [-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

```
gwwait [-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**

<b>FIELD</b>	<b>DESCRIPTION</b>
UID	user unique identification assigned by the Gridway system
NAME	name of this user
JOBS	number of Jobs in the GridWay system
RUN	number of running jobs
IDLE	idle time, (time with JOBS = 0)
EM	execution manager drivers loaded for this user
TM	transfer manager drivers loaded for this user
PID	process identification of driver processes

---

# 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**

FIELD	DESCRIPTION
HOST/USER	host/user usage summary for this user/host
XFR	total transfer time on this host (for this user)
EXE	total execution time on this host (for this user), without suspension time
SUSP	total suspension (queue) time on this host (for this user)
TOTS	total executions on this host (for this user). Termination reasons: <ul style="list-style-type: none"><li>• SUCC success</li><li>• ERR error</li><li>• KILL kill</li><li>• USER user requested</li><li>• SUSP suspension timeout</li><li>• DISC discovery timeout</li><li>• SELF self migration</li><li>• PERF performance degradation</li><li>• S/R stop/resume</li></ul>

---

## 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 gws submit in the following fashion:

```
jsdl2gw jsdl-job.xml | gws submit
```

## 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](http://www.gridway.org/support.php)), to submit your problems and they will be eventually appear in this guide.

## **GT 4.2.1 GridWay: Developer's Guide**

---

## GT 4.2.1 GridWay: Developer's Guide

Published May, 2008

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

### Introduction

This guide is intended to help a developer interested in extending GridWay's functionality to understand its architecture, mainly to show how to develop new Middleware Access Drivers (MADs). It is also a good start point and reference for anyone interested in programming applications using GridWay's implementation of the DRMAA library (C, Java, Perl, Python and Ruby bindings).

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><sup>1</sup>

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.

---

<sup>1</sup> <http://www.apache.org/licenses/LICENSE-2.0>

---

---

# Table of Contents

1. Before you begin .....	1
1. Feature summary .....	1
2. Tested platforms .....	2
3. Backward compatibility summary .....	2
4. Technology dependencies .....	2
5. Security Considerations for GridWay .....	3
2. Usage scenarios .....	4
1. Simple Applications .....	4
3. Tutorials .....	8
1. DRMAA Program Structure and Compilation .....	8
2. DRMAA Programming Howto .....	9
3. External Tutorials and Material .....	23
4. Architecture and design overview .....	24
1. Architecture .....	24
2. Programming Model Overview .....	25
5. APIs .....	28
1. DRMAA bindings for C and Java .....	28
2. DRMAA bindings for scripting languages .....	28
6. Non-WSDL protocols .....	31
1. Information manager MAD .....	31
2. Execution manager MAD .....	33
3. Transfer manager MAD .....	33
4. Dispatch manager Scheduler .....	34
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
7. Debugging .....	49
1. Debugging in Java WS Core .....	49
2. Debugging in GridWay .....	49
8. Troubleshooting .....	51
1. Errors .....	51
2. Debugging .....	51
9. GT 4.2.1 Samples for GridWay .....	52
1. DRMAA examples .....	52

---

## List of Figures

4.1. Components of the GridWay Meta-scheduler. ....	24
4.2. Grid Development Model with DRMAA .....	25
4.3. Embarrassingly Distributed Applications schema .....	26
4.4. Master-Worker Applications schema .....	26

---

## List of Tables

5.1. Translation from C to Scripting language .....	29
6.1. Attributes that should be defined by the Information MADs. ....	32
3. Field options .....	40
4. Field information .....	41
5. Field information .....	42
6. Queue field information .....	43
7. Field information .....	46
8. Field information .....	47
8.1. Gridway Errors .....	51

---

# Chapter 1. Before you begin

## 1. Feature summary

Advanced Scheduling Capabilities	<p>GridWay implements several state-of-the-art Grid-aware scheduling policies, comprising <i>job prioritization</i> policies (fixed priority, urgency, share, deadline and waiting-time) and <i>resource prioritization</i> policies (fixed priority, usage, failure and rank).</p> <p>These policies are combined with:</p> <ul style="list-style-type: none"><li>• <i>Adaptive Scheduling</i>, to periodically adapt the schedule considering applications' demands and Grid resource characteristics.</li><li>• <i>Adaptive Execution</i> to migrate running applications in terms of resource availability, capacity or cost, and new application requirements or preferences.</li></ul>
Transparent Grid Access	<p>GridWay interfaces infrastructures with different middleware stacks. With GridWay, users can access heterogeneous resources in a transparent way. For example, it can access resources configured with both GRAM pre web services and GRAM web services. It also permits the use of different grids with different software stacks, for example, Teragrid with Globus and EGEE with gLite.</p>
Flexible Deployment Capabilities	<p>GridWay supports multiple-user operation mode, and does not require additional middleware installation (apart from standard Globus services). Globus installation is not required in each end-user system.</p> <p>GridWay allows different Grid deployment strategies, like Enterprise Grids, Partner Grids or Utility Grids.</p>
Different Application Profiles	<p>GridWay executes different Grid application profiles:</p> <ul style="list-style-type: none"><li>• Array (Bulk) jobs, for parameter sweep applications</li><li>• DAG Workflows</li><li>• Single-site MPI applications</li></ul>
Fault Detection and Recovery	<p>GridWay is able to detect several problems that can occur when executing a remote job. It also implements mechanisms that make the execution more reliable. It can detect a remote system crash, a job failure (via the job exit code) or even a network disconnection (using the polling mechanism) and migrate the problematic job to another resource.</p> <p>GridWay also performs periodic saves of its state in order to recover from local failure.</p>
Reporting and Accounting	<p>GridWay provides detailed statistics of Grid usage. In this way, the Grid administrator can properly plan usage policies and forecast workload. In addition, these statistics can be used by the scheduler to predict (per user) Grid resource response time.</p>

Standard Compliance

GridWay is an open-source project, flexible and completely based on standards to leverage its usability and interoperability. For example, users can describe their jobs using JSDL. Similarly, programmers can build grid enabled applications using the DRMAA standard.

User Interface

GridWay provides end-users with a familiar environment similar to that found on classical LRM systems. So GridWay CLI eases the adoption of Grid technologies.

## 2. Tested platforms

Tested platforms for GridWay:

- GridWay builds successfully for the following platforms:
  - Linux
  - Tru64
  - Mac OS X
  - Solaris
  - Aix

In addition, GridWay has been tested with the major Grid infrastructures. Click the following links to find more information on how to use GridWay with [EGEE<sup>1</sup>](#), [TeraGrid<sup>2</sup>](#), [OSG<sup>3</sup>](#) and [Nordug<sup>4</sup>](#).

## 3. Backward compatibility summary

The following information regards compatibility with the previous stable version of GridWay included in GT 4.0 series:

API changes since GT 4.0:

- Added DRMAA scripting language bindings for Python, Ruby and Perl.

CLI changes since GT 4.0:

- Added gwdag, a tool to run Condor DAGMAN compatible workflows.

Configuration interface changes since GT 4.0:

- None

## 4. Technology dependencies

GridWay uses different Globus services to perform the tasks of information gathering, job execution and data transfer.

- GridWay depends on the following GT components for job execution:

---

<sup>1</sup> <http://www.gridway.org/documentation/stable/egeehowto>

<sup>2</sup> <http://www.gridway.org/documentation/stable/tghowto/>

<sup>3</sup> <http://www.gridway.org/documentation/stable/osghowto/>

<sup>4</sup> <http://www.gridway.org/documentation/stable/nghowtoo/>

- GRAM: GridWay can interface with both GRAM 2 (pre web services) and GRAM 4 (web services)
- GridWay depends on the following GT components for data staging:
  - RFT
  - GridFTP
- GridWay depends on the following GT component for information gathering:
  - MDS
- GridWay relies on the Globus basic security infrastructure for authentication and authorization. On top of that, GridWay can use other Globus services and components to complement this infrastructure:
  - Delegation Service
  - MyProxy

## 5. Security Considerations for GridWay

Access authorization to the GridWay server is done based on the Unix identity of the user (accessing GridWay directly or through a Web Services GRAM, as in [GridGateWay<sup>5</sup>](#)). Hence, security in GridWay has the same implications as the Unix accounts of their users.

Also, GridWay uses proxy certificates to use Globus services, so the security implications of managing certificates also must be taken into account

---

<sup>5</sup> <http://www.grid4utility.org>

---

# Chapter 2. Usage scenarios

## 1. Simple Applications

### 1.1. A simple computational problem

This is a well known exercise. For our purposes, we will calculate the integral of the following function  $f(x) = 4/(1+x^2)$ . 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 tasks and compute its area. The following program computes the area of a set of intervals, assigned to a given task:

```
#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);
    }

    l_sum *= h;

    printf("%0.12g\n", l_sum);

    return 0;
}
```

We will use this program (`pi`) to develop our DRMAA distributed version.

## 1.2. The DRMAA code

### 1.2.1. Setting up the job template

Let us start with the definition of each tasks. As you can see, the previous program needs the number of intervals, total tasks, and the task number. These variables are available to compile job templates through the `DRMAA_GW_TASK_ID` and `DRMAA_GW_TOTAL_TASKS` predefined strings.

Also, each task must generate a different standard output file, with its partial result. We can use the standard `DRMAA_PLACEHOLDER_INCR` predefined string to set up different filenames for each task, so they will not overwrite each others output.

```
void setup_job_template( drmaa_job_template_t **jt)
{
    char          error[DRMAA_ERROR_STRING_BUFFER];
    int           rc;
    char          cwd[DRMAA_ATTR_BUFFER];

    const char   *args[4] = {DRMAA_GW_TASK_ID,
                             DRMAA_GW_TOTAL_TASKS,
                             "10000000",
                             NULL};

    rc = drmaa_allocate_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER);

    getcwd(cwd, DRMAA_ATTR_BUFFER)

    rc = drmaa_set_attribute(*jt,
                             DRMAA_WD,
                             cwd,
                             error,
                             DRMAA_ERROR_STRING_BUFFER);

    rc = drmaa_set_attribute(*jt,
                             DRMAA_JOB_NAME,
                             "pi.drmaa",
                             error,
                             DRMAA_ERROR_STRING_BUFFER);

    rc = drmaa_set_attribute(*jt,
                             DRMAA_REMOTE_COMMAND,
                             "pi",
                             error,
                             DRMAA_ERROR_STRING_BUFFER);

    rc = drmaa_set_vector_attribute(*jt,
                                    DRMAA_V_ARGV,
                                    args,
```

```
        error,  
        DRMAA_ERROR_STRING_BUFFER);  
  
    rc = drmaa_set_attribute(*jt,  
        DRMAA_OUTPUT_PATH,  
        "stdout."DRMAA_PLACEHOLDER_INCR,  
        error,  
        DRMAA_ERROR_STRING_BUFFER);  
}
```

## 1.2.2. The main DRMAA routine

The DRMAA program just submits a given number of tasks, each one will compute its section of the previous integral. Then we will synchronize these tasks, aggregate the partial results to get the total value. Note that this results are passed to the DRMAA program through the tasks standard output.

```
int main(int argc, char **argv)  
{  
    int rc;  
    int end, i;  
  
    char error[DRMAA_ERROR_STRING_BUFFER];  
    char value[DRMAA_ATTR_BUFFER];  
    char attr_buffer[DRMAA_ATTR_BUFFER];  
  
    const char *job_ids[1] = {DRMAA_JOB_IDS_SESSION_ALL};  
  
    drmaa_job_template_t *jt;  
    drmaa_job_ids_t      *jobids;  
  
    FILE *fp;  
    float pi, pi_t;  
  
    if ( argc != 2)  
    {  
        fprintf(stderr, "Usage drmaa_pi <number_of_tasks>\n");  
        return -1;  
    }  
    else  
        end = atoi(argv[1]) - 1;  
  
    rc = drmaa_init(NULL, error, DRMAA_ERROR_STRING_BUFFER-1);  
  
    setup_job_template(&jt);  
  
    rc = drmaa_run_bulk_jobs(&jobids,  
        jt,  
        0,  
        end,  
        1,
```

```
        error,
        DRMAA_ERROR_STRING_BUFFER);

fprintf(stderr,"Waiting for bulk job to finish...\n");

rc = drmaa_synchronize(job_ids,
        DRMAA_TIMEOUT_WAIT_FOREVER,
        DISPOSE,
        error,
        DRMAA_ERROR_STRING_BUFFER);

fprintf(stderr,"All Jobs finished\n");

pi = 0.0;

for(i=0;i<=end;i++)
{
    snprintf(attr_buffer,DRMAA_ATTR_BUFFER,"stdout.%s",i);

    fp = fopen(attr_buffer,"r");
    fscanf(fp,"%f",&pi_t);
    fprintf(stderr,"Partial computed by task %i = %1.30f\n",i,pi_t);
    fclose(fp);

    pi += pi_t;
}

drmaa_release_job_ids(jobids);

fprintf(stderr,"\nPI=%1.30f\n",pi);

drmaa_exit(NULL, 0);

return 0;
}
```

---

# Chapter 3. Tutorials

This chapter is a tutorial for getting started programming DRMAA applications with GridWay. Although is not necessary that you already know how GridWay works, prior experience submitting and controlling jobs with GridWay will come in handy. This tutorial shows the use of the most important functions in the DRMAA standard, and gives you some hints to use the GridWay DRMAA library.

The source code for the following examples can be found in the `$GW_LOCATION/examples` directory.

## 1. DRMAA Program Structure and Compilation

### 1.1. DRMAA C Binding

You have to include the following file in your C sources to use the GridWay DRMAA C bindings library:

```
#include "drmaa.h"
```

Also add the following compiler options to link your program with the DRMAA library:

```
-L $GW_LOCATION/lib  
-I $GW_LOCATION/include  
-ldrmaa
```

### 1.2. DRMAA JAVA Binding

You have to import the GridWay DRMAA JAVA package:

```
import org.ggf.drmaa.*;
```

Add the following option to the **javac**:

```
-classpath $(CLASSPATH):$GW_LOCATION/lib/drmaa.jar
```

Also do not forget to update your shared library path:

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

## 2. DRMAA Programming Howto

### 2.1. DRMAA Sessions

Let us start with the initialization steps of every DRMAA program. Before any call to a DRMAA function you must start a DRMAA session. The session is used to manage the jobs submitted in your Grid application. Before ending the program you should disengage from the previously started session, to free the internal session structures.

Also, as you will see every DRMAA function returns an error code that allows you to check whether the call was successful or not (DRMAA\_ERRNO\_SUCCESS, if everything goes well). Through out this tutorial, for the shake of clarity, we will get rid of error checking code in most cases. But remember that you should check return codes.

The following example shows you how to manage a DRMAA session. It also shows some information about the DRMAA implementation and the DRMS you are using.

```
char          error[DRMAA_ERROR_STRING_BUFFER];
int           result;
char          contact[DRMAA_ATTR_BUFFER];
unsigned int  major;
unsigned int  minor;
char          drm[DRMAA_ATTR_BUFFER];
char          impl[DRMAA_ATTR_BUFFER];

result = drmaa_init (NULL,
                    error,
                    DRMAA_ERROR_STRING_BUFFER-1);           (See 1)

if ( result != DRMAA_ERRNO_SUCCESS)
{
    fprintf(stderr,"drmaa_init() failed: %s\n", error);
    return -1;
}
else
    printf("drmaa_init() success \n");

drmaa_get_contact(contact,
                 DRMAA_ATTR_BUFFER-1,
                 error,
                 DRMAA_ERROR_STRING_BUFFER-1);           (See 2)

drmaa_version(&major,
             &minor,
             error,
             DRMAA_ERROR_STRING_BUFFER);

drmaa_get_DRM_system(drm,
                    DRMAA_ATTR_BUFFER-1,
                    error,
                    DRMAA_ERROR_STRING_BUFFER-1);

drmaa_get_DRMAA_implementation(impl,
```

```
        DRMAA_ATTR_BUFFER-1,
        error,
        DRMAA_ERROR_STRING_BUFFER-1);

printf("Using %s, details:\n",impl);
printf("\t DRMAA version %i.%i\n",major,minor);
printf("\t DRMS %s (contact: %s)\n",drm,contact);

result = drmaa_exit (error, DRMAA_ERROR_STRING_BUFFER-1);  (See 3)

if ( result != DRMAA_ERRNO_SUCCESS)
{
    fprintf(stderr,"drmaa_exit() failed: %s\n", error);
    return -1;
}

printf("drmaa_exit() success \n");
```

1. `drmaa_init()` is the first DRMAA function in a DRMAA program, and must be called only once per DRMAA program. Almost every DRMAA function uses to special arguments for error reporting purposes: a string buffer (`error`), and its length (`DRMAA_ERROR_STRING_BUFFER-1`). As you can see DRMAA has some predefined buffer sizes for `char *` variables that you can use, although they are not mandatory.

The first parameter of the `drmaa_init()` function is an implementation dependent string which may be used to specify which DRM system to use. In the GridWay library it can be used to select which GW daemon you want to connect to, `NULL` means the default option, `localhost`.

If you try to call a DRMAA function outside a session, it will return the error code `DRMAA_ERRNO_NO_ACTIVE_SESSION`.

2. This program also shows some information about the DRMAA implementation you are using, by calling the functions `drmaa_get_contact()`, `drmaa_version()`, `drmaa_get_DRM_system()` and `drmaa_get_DRMAA_implementation()`. You should see something like this:

```
Using DRMAA for GridWay 4.7, details:
DRMAA version 1.0
DRMS GridWay (contact: localhost)
```

3. At the end of the program you should call `drmaa_exit()` to clean up the library internal structures. Beyond this point you should not be calling any DRMAA function (there are some exceptions like the information functions above, like `drmaa_get_contact()`,...)

This example shows the same program using the DRMAA JAVA bindings.

```
import org.ggf.gridway.drmaa.*;

import java.util.*;

public class Howto1
{
```

```
public static void main (String[] args)
{
    SessionFactory  factory = SessionFactory.getFactory();
    Session        session = factory.getSession();

    try
    {
        session.init(null);

        System.out.println("Session Init success");

        System.out.println ("Using " + session.getDRMAAImplementation()
                             + ", details:");

        System.out.println ("\t DRMAA version " + session.getVersion());

        System.out.println ("\t DRMS " + session.getDRMSInfo() +
                             "(contact: " + session.getContact() + ")");

        session.exit();

        System.out.println("Session Exit success");

    }
    catch (DrmaaException e)
    {
        e.printStackTrace();
    }
}
```

## 2.2. Job Template Compilation

So we already have an active session, how do we use it to submit jobs. The first thing to do is to provide a description of your job. A job is described by its job template (a `drmaa_job_template_t` variable), which in turns is a structure to store information about your job (things like the executable, its arguments or the output files).

The DRMAA standard provides several pre-defined strings to refer to common job template attributes (those starting with `DRMAA_` like `DRMAA_REMOTE_COMMAND`). The GridWay library also defines some macros to refer to GridWay specific attributes (those starting with `DRMAA_GW_` like `DRMAA_GW_INPUT_FILES`).

There are two kind of attributes: scalar and vector. Scalar attributes are simple strings (`char *`) and corresponds to template attributes in the form:

```
attribute = value
```

You can use the `drmaa_set_attribute()` and `drmaa_get_attr_value()` to manage these scalar attributes. On the other hand, vector attributes corresponds to job template variables with one or more values i.e:

```
attribute = value1 value2 ... valueN
```

A vector attribute is NULL terminated array of strings (char \*\*). You can use the `drmaa_set_vector_attribute()` and `drmaa_get_next_attr_value()` to deal with vector attributes.

We will use a common job template for the rest of the tutorial, so we will make a function to set up this job template. Remember to check the return codes of the DRMAA functions.

```
void setup_job_template(drmaa_job_template_t **jt)
{
    char          error[DRMAA_ERROR_STRING_BUFFER];
    int           rc;
    char          cwd[DRMAA_ATTR_BUFFER];

    const char   *args[3] = {"-l", "-a", NULL};           (See 1)

    rc = drmaa_allocate_job_template(jt, error, DRMAA_ERROR_STRING_BUFFER-1);
                                                    (See 2)

    if ( rc != DRMAA_ERRNO_SUCCESS)
    {
        (See 3)
        fprintf(stderr, "drmaa_allocate_job_template() failed: %s\n", error);
        exit(-1);
    }

    if ( getcwd(cwd, DRMAA_ATTR_BUFFER) == NULL )
    {
        perror("Error getting current working directory");
        exit(-1);
    }

    rc = drmaa_set_attribute(*jt,                               (See 4)
                             DRMAA_WD,
                             cwd,
                             error,
                             DRMAA_ERROR_STRING_BUFFER-1);

    if ( rc != DRMAA_ERRNO_SUCCESS )
    {
        fprintf(stderr, "Error setting job template attribute: %s\n", error);
        exit(-1);
    }

    rc = drmaa_set_attribute(*jt,
                             DRMAA_JOB_NAME,
                             "ht2",
                             error,
                             DRMAA_ERROR_STRING_BUFFER-1);

    rc = drmaa_set_attribute(*jt,                               (See 5)
                             DRMAA_REMOTE_COMMAND,
                             "/bin/ls",
```

```
        error,
        DRMAA_ERROR_STRING_BUFFER-1);

rc = drmaa_set_vector_attribute(*jt,                (See 6)
                               DRMAA_V_ARGV,
                               args,
                               error,
                               DRMAA_ERROR_STRING_BUFFER-1);

if ( rc != DRMAA_ERRNO_SUCCESS )
{
    fprintf(stderr,"Error setting remote command arguments: %s\n",error);
    exit(-1);
}

rc = drmaa_set_attribute(*jt,                    (See 7)
                        DRMAA_OUTPUT_PATH,
                        "stdout."DRMAA_GW_JOB_ID,
                        error,
                        DRMAA_ERROR_STRING_BUFFER-1);

rc = drmaa_set_attribute(*jt,
                        DRMAA_ERROR_PATH,
                        "stderr."DRMAA_GW_JOB_ID,
                        error,
                        DRMAA_ERROR_STRING_BUFFER-1);
```

1. As stated before, every DRMAA function returns an error code that should be checked. Check the DRMAA Reference Guide to find out the error codes returned by each function.
2. In this code we set up a job template for a simple "ls" command. The arguments are a vector attribute, in this case we are using two arguments "-l" and "-a". Note that we use an array with 3 elements, the last one must be always NULL.
3. Before using a job template you must allocate it. This function allocates memory for the library internal structures of your job template. Do not forget to free the job template with `drmaa_delete_job_template()` function.
4. The GridWay DRMAA implementation will generate a job template file in the job working directory (`DRMAA_WD`). This attribute value should be defined, it defaults to the program current working directory (`DRMAA_PLACEHOLDER_WD`). Please note that all files are named relative to the working directory. `DRMAA_WD` is a local path name, this directory will be "recreated" (by the use of the `DRMAA_GW_INPUT_FILES` variable) in the remote host, and it will be the working directory of the job on the execution host.
5. This is the executable we are going to submit to the Grid, a simple "ls".
6. Here we set the arguments, note we are using the vector attribute function.
7. The standard output of our "ls" command will be stored (under the `DRMAA_WD` directory) with name "stdout."DRMAA\_GW\_JOB\_ID. In this case we are using an specific GridWay define `DRMAA_GW_JOB_ID`, which corresponds to `#{JOB_ID}`. So, if our job is assigned with id 34 by GridWay, we will have its standard output in "stdout.34". Use always this naming pattern not to over-write previously generated files.

If everything went well, the following job template will be generated:

```
#This file was automatically generated by the GridWay DRMAA library
EXECUTABLE=/bin/ls
ARGUMENTS= -l -a
STDOUT_FILE=stdout.${JOB_ID}
STDERR_FILE=stderr.${JOB_ID}
RESCHEDULE_ON_FAILURE=no
NUMBER_OF_RETRIES=3
```

This fragment of code shows you how to construct the same template using the DRMAA JAVA bindings.

```
jt = session.createJobTemplate();

jt.setWorkingDirectory(java.lang.System.getProperty("user.dir"));

jt.setJobName("ht2");

jt.setRemoteCommand("/bin/ls");

jt.setArgs(new String[] {"-l", "-a"});

jt.setOutputPath("stdout." + SessionImpl.DRMAA_GW_JOB_ID);

jt.setErrorPath ("stderr." + SessionImpl.DRMAA_GW_JOB_ID);
```

## 2.3. Single Job Submission

We can now submit our "ls" to the Grid. The next example shows you how to submit your job, and how to synchronize its execution. The resource usage made by your job is also shown.

```
int main(int argc, char *argv[])
{
    char          error[DRMAA_ERROR_STRING_BUFFER];
    int           result;
    drmaa_job_template_t * jt;
    char          job_id[DRMAA_JOBNAME_BUFFER];
    char          job_id_out[DRMAA_JOBNAME_BUFFER];
    drmaa_attr_values_t * rusage;
    int           stat;
    char          attr_value[DRMAA_ATTR_BUFFER];

    result = drmaa_init (NULL, error, DRMAA_ERROR_STRING_BUFFER-1);

    if ( result != DRMAA_ERRNO_SUCCESS)
    {
        fprintf(stderr, "drmaa_init() failed: %s\n", error);
        return -1;
    }
}
```

```
setup_job_template(&jt);                                (See 1)

drmaa_run_job(job_id,
              DRMAA_JOBNAME_BUFFER-1,                  (See 2)
              jt,
              error,
              DRMAA_ERROR_STRING_BUFFER-1);

fprintf(stderr,"Job successfully submitted ID: %s\n",job_id);

result = drmaa_wait(job_id,
                   job_id_out,                          (See 3)
                   DRMAA_JOBNAME_BUFFER-1,
                   &stat,
                   DRMAA_TIMEOUT_WAIT_FOREVER,
                   &rusage,
                   error,
                   DRMAA_ERROR_STRING_BUFFER-1);

if ( result != DRMAA_ERRNO_SUCCESS)
{
    fprintf(stderr,"drmaa_wait() failed: %s\n", error);
    return -1;
}

drmaa_wexitstatus(&stat,stat,error,DRMAA_ERROR_STRING_BUFFER-1);
                                                         (See 4)
fprintf(stderr,"Job finished with exit code %i, usage: %s\n",stat,job_id);

while (drmaa_get_next_attr_value(rusage,
                                 attr_value,              (See 5)
                                 DRMAA_ATTR_BUFFER-1) != DRMAA_ERRNO_NO_MORE_ELEMENTS )
    fprintf(stderr,"\t%s\n",attr_value);

drmaa_release_attr_values (rusage);                      (See 6)

drmaa_delete_job_template(jt,
                          error,
                          DRMAA_ERROR_STRING_BUFFER-1);

drmaa_exit (error, DRMAA_ERROR_STRING_BUFFER-1);

return 0;
}
```

1. Let us setup the job template with the previous function
2. Now we can submit a single job to the Grid, with this job template. The job id assigned by GridWay to our job is returned in the job\_id variable. So let us print it and you can check the progress of your ls job with the gwps command.

3. This function will block (`DRMAA_TIMEOUT_WAIT_FOREVER`) until the job has finished or failed. There are two interesting values returned by the `drmaa_wait()` function: `stat` (the exit status of your job), and `rusage` (the resource usage made by the "ls" command, execution and transfer times). If the job is killed the `drmaa_wait()` will return with `DRMAA_ERRNO_NO_RUSAGE`.
4. DRMAA provides some functions to deal with the `stat` value. In this case we want to get the exit status of our job (you can also check if the job was signaled, the signal it received...)
5. Lets print the resource usage. As this is of type `drmaa_attr_values_t` we have to iterate over the list to get all the values. We finish when we find the `DRMAA_ERRNO_NO_MORE_ELEMENTS` return code.
6. Do not forget to free the DRMAA variables, like the job template, or the `rusage` list allocated by the `drmaa_wait()` function.

This example shows the same program using the DRMAA JAVA bindings.

```
try
{
    session.init(null);

    String id = session.runJob(jt);

    System.out.println("Job successfully submitted ID: " + id);

    JobInfo info = session.wait(id, Session.DRMAA_TIMEOUT_WAIT_FOREVER);

    System.out.println("Job usage:");

    Map rmap = info.getResourceUsage();
    Iterator r = rmap.keySet().iterator();

    while(r.hasNext())
    {
        String name2 = (String) r.next();
        String value = (String) rmap.get(name2);
        System.out.println(" " + name2 + "=" + value);
    }

    session.deleteJobTemplate(jt);

    session.exit();
}
catch (DrmaaException e)
{
    e.printStackTrace();
}
```

## 2.4. Job Status and Control

But you can do more things with a job than just submit it. The DRMAA standard allows you to control your jobs (kill, hold, release, stop,...) even they are not submitted within a DRMAA session. See the following example:

```
int main(int argc, char *argv[])
{
    char                error[DRMAA_ERROR_STRING_BUFFER];
    int                 rc;
    drmaa_job_template_t * jt;
    char                job_id[DRMAA_JOBNAME_BUFFER];
    const char          *job_ids[2]={DRMAA_JOB_IDS_SESSION_ALL,NULL};
    int                 status;

    drmaa_init (NULL, error, DRMAA_ERROR_STRING_BUFFER-1);

    setup_job_template(&jt);

    drmaa_set_attribute(jt,                                (See 1)
                        DRMAA_JS_STATE,
                        DRMAA_SUBMISSION_STATE_HOLD,
                        error,
                        DRMAA_ERROR_STRING_BUFFER-1);

    drmaa_run_job(job_id,
                  DRMAA_JOBNAME_BUFFER,
                  jt,
                  error,
                  DRMAA_ERROR_STRING_BUFFER-1);

    fprintf(stdout,"Your job has been submitted with id: %s\n", job_id);

    sleep(5);

    drmaa_job_ps(job_id, &status, error, DRMAA_ERROR_STRING_BUFFER);
                                                (See 2)
    fprintf(stdout,"Job state is: %s\n",drmaa_gw_strstatus(status));

    sleep(1);

    fprintf(stdout,"Releasing the Job\n");

    rc = drmaa_control(job_id,                                (See 3)
                       DRMAA_CONTROL_RELEASE,
                       error,
                       DRMAA_ERROR_STRING_BUFFER-1);

    if ( rc != DRMAA_ERRNO_SUCCESS)
    {
        fprintf(stderr,"drmaa_control() failed: %s\n", error);
        return -1;
    }
}
```

```
drmaa_job_ps(job_id, &status, error, DRMAA_ERROR_STRING_BUFFER);

fprintf(stdout, "Job state is: %s\n", drmaa_gw_strstatus(status));

fprintf(stdout, "Synchronizing with job...\n");

rc = drmaa_synchronize(job_ids,                                (See 4)
                       DRMAA_TIMEOUT_WAIT_FOREVER,
                       0,
                       error,
                       DRMAA_ERROR_STRING_BUFFER-1);

if ( rc != DRMAA_ERRNO_SUCCESS)
{
    fprintf(stderr, "drmaa_synchronize failed: %s\n", error);
    return -1;
}

fprintf(stdout, "Killing the Job\n");

drmaa_control(job_id,                                       (See 5)
              DRMAA_CONTROL_TERMINATE,
              error,
              DRMAA_ERROR_STRING_BUFFER-1);

if ( rc != DRMAA_ERRNO_SUCCESS)
{
    fprintf(stderr, "drmaa_control() failed: %s\n", error);
    return -1;
}

fprintf(stdout, "Your job has been deleted\n");

drmaa_delete_job_template(jt,
                          error,
                          DRMAA_ERROR_STRING_BUFFER-1);

drmaa_exit (error, DRMAA_ERROR_STRING_BUFFER-1);

return 0;
}
```

1. We are adding a new attribute to our job template the job submission state (DRMAA\_JS\_STATE). Our job will be held on submission
2. Let's check that the job is in the HOLD state. Note that we use a GridWay specific function to show the state `drmaa_gw_strstatus()`. This function is not part of the DRMAA standard.
3. OK, so we can now release our job, so it will begin its execution (i.e. will enter the QUEUED\_ACTIVE state)
4. Now let us wait for this job, note that we are not using `drmaa_wait()` in this case. `drmaa_synchronize()` can be used to wait for a set of jobs. Its first argument is a NULL terminated array of job ids; we are using an special job

name: DRMAA\_JOB\_IDS\_SESSION\_ALL, to wait for all the jobs in submitted within your DRMAA session (other special job name is DRMAA\_JOB\_IDS\_SESSION\_ANY). The third argument (dispose) if equal to 1 will dispose (kill) the job. In this example we will do it by hand.

5. Kill the job with the TERMINATE control action.

Let see the same program in JAVA

```
try
{
    session.init(null);

    setup_job_template();

    String id = session.runJob(jt);

    System.out.println("Job successfully submitted ID: " + id);

    try
    {
        Thread.sleep(5 * 1000);
    }
    catch (InterruptedException e)
    { // Don't care }

    printJobStatus(session.getJobProgramStatus(id));

    try
    {
        Thread.sleep(1000);
    }
    catch (InterruptedException e)
    { // Don't care }

    System.out.println("Releasing the Job");

    session.control(id, Session.DRMAA_CONTROL_RELEASE);

    printJobStatus(session.getJobProgramStatus(id));

    System.out.println("Synchronizing with job...");

    session.synchronize(
        Collections.singletonList(Session.DRMAA_JOB_IDS_SESSION_ALL),
        Session.DRMAA_TIMEOUT_WAIT_FOREVER,
        false);

    System.out.println("Killing the Job");

    session.control(id, Session.DRMAA_CONTROL_TERMINATE);

    session.deleteJobTemplate(jt);
```

```
    session.exit();
}
catch (DrmaaException e)
{
    e.printStackTrace();
}
```

## 2.5. Array Jobs (bulk)

Bulk jobs are a direct way to express parametric computations. A bulk job is a set of independent (and very similar) tasks that use the same job template. You can use the `DRMAA_PLACEHOLDER_INCR` constat to assign different input/output files for each task. The `DRMAA_PLACEHOLDER_INCR` is a unique identifier of each job (task) in the bulk (array) job. In the GridWay DRMAA library it corresponds to the `#{TASK_ID}` parameter.

```
int main(int argc, char *argv[])
{
    char                error[DRMAA_ERROR_STRING_BUFFER];
    int                rc;
    int                stat;

    drmaa_job_template_t * jt;
    drmaa_attr_values_t * rusage;
    drmaa_job_ids_t     * jobids;

    char                value[DRMAA_ATTR_BUFFER];
    const char *       job_ids[2] = {DRMAA_JOB_IDS_SESSION_ALL, NULL};
    char                job_id_out[DRMAA_JOBNAME_BUFFER];

    int                rcj;

    drmaa_init (NULL, error, DRMAA_ERROR_STRING_BUFFER-1);

    setup_job_template(&jt);

    drmaa_set_attribute(jt,
                       DRMAA_OUTPUT_PATH,
                       "stdout."DRMAA_PLACEHOLDER_INCR,    (See 1)
                       error,
                       DRMAA_ERROR_STRING_BUFFER-1);

    rc = drmaa_run_bulk_jobs(&jobids,
                             jt,
                             0,          (See 2)
                             4,
                             1,
                             error,
                             DRMAA_ERROR_STRING_BUFFER-1);

    if ( rc != DRMAA_ERRNO_SUCCESS)
```

```
{
    fprintf(stderr,"drmaa_run_bulk_job() failed: %s\n", error);
    return -1;
}

fprintf(stderr,"Bulk job successfully submitted IDs are:\n");

do          (See 3)
{
    rc = drmaa_get_next_job_id(jobids, value, DRMAA_ATTR_BUFFER-1);

    if ( rc == DRMAA_ERRNO_SUCCESS )
        fprintf(stderr,"\t%s\n", value);

}while (rc != DRMAA_ERRNO_NO_MORE_ELEMENTS);

fprintf(stderr,"Waiting for bulk job to finish...\n");

drmaa_synchronize(job_ids,                                (See 4)
                  DRMAA_TIMEOUT_WAIT_FOREVER,
                  0,
                  error,
                  DRMAA_ERROR_STRING_BUFFER-1);

fprintf(stderr,"All Jobs finished\n");

do
{
    rcj = drmaa_get_next_job_id(jobids, value, DRMAA_ATTR_BUFFER-1);

    if ( rcj == DRMAA_ERRNO_SUCCESS )
    {
        drmaa_wait(value,                                (See 5)
                   job_id_out,
                   DRMAA_JOBNAME_BUFFER-1,
                   &stat,
                   DRMAA_TIMEOUT_WAIT_FOREVER,
                   &rusage,
                   error,
                   DRMAA_ERROR_STRING_BUFFER-1);

        drmaa_wexitstatus(&stat,stat,error,DRMAA_ERROR_STRING_BUFFER-1);

        fprintf(stderr,"Rusage for task %s (exit code %i)\n", value, stat);

        do
        {
            rc = drmaa_get_next_attr_value(rusage, value, DRMAA_ATTR_BUFFER-1);

            if ( rc == DRMAA_ERRNO_SUCCESS )
                fprintf(stderr,"\t%s\n", value);

        }while (rc != DRMAA_ERRNO_NO_MORE_ELEMENTS);
    }
}
```

```
        drmaa_release_attr_values(rusage);
    }
}while (rcj != DRMAA_ERRNO_NO_MORE_ELEMENTS);

drmaa_release_job_ids(jobids);

drmaa_delete_job_template(jt,
                          error,
                          DRMAA_ERROR_STRING_BUFFER-1);

drmaa_exit (error,DRMAA_ERROR_STRING_BUFFER-1);

return 0;
}
```

1. The output file of each task in the bulk job will be "stdout."DRMAA\_PLACEHOLDER\_INCR. So the tasks will not over-write each others outputs.
2. We submit a bulk job with 5 tasks (0,1,2,3,4), note that we will get five different output files (stdout.0,stdout.1,...). The job ids assigned to each task are stored in the first argument of the function, a `drmaa_job_ids_t` list.
3. In order to get each job id you must use the `drmaa_get_next_job_id()` function. We iterate through the list until `DRMAA_ERRNO_NO_MORE_ELEMENTS` is returned.Do not forget to free the job id list when you no longer need it.
4. We wait for all the jobs in the array. Each you will be disposed with a call to the `drmaa_wait()` function.
5. We now use the `drmaa_wait()` function to get each task exit status and resource usage. As we have previously synchronize the bulk job this function will not block. `drmaa_wait()` will also remove each task from the GridWay system.

Finally a bulk job in JAVA.

```
try
{
    session.init(null);

    int start = 0;
    int end   = 4;
    int step  = 1;

    int i;

    String id;

    java.util.List        ids = session.runBulkJobs(jt, start, end, step);
    java.util.Iterator    iter = ids.iterator();

    System.out.println("Bulk job successfully submitted IDs are: ");

    while(iter.hasNext())
    {
```

```
        System.out.println("\t" + iter.next());
    }

    session.deleteJobTemplate(jt);

    session.synchronize(
        Collections.singletonLsectionist(Session.DRMAA_JOB_IDS_SESSION_ALL),
        Session.DRMAA_TIMEOUT_WAIT_FOREVER,
        false);

    for (int count = start; count <= end; count += step)
    {
        JobInfo info = session.wait(Session.DRMAA_JOB_IDS_SESSION_ANY,
            Session.DRMAA_TIMEOUT_WAIT_FOREVER);

        System.out.println("Job usage:");
        Map rmap = info.getResourceUsage();
        Iterator r = rmap.keySet().iterator();

        while(r.hasNext())
        {
            String name2 = (String) r.next();
            String value = (String) rmap.get(name2);
            System.out.println(" " + name2 + "=" + value);
        }
    }

    session.exit();
}
catch (DrmaaException e)
{
    System.out.println("Error: " + e.getMessage());
}
```

### 3. External Tutorials and Material

- [GridWay Tutorial](http://www.gridway.org/documentation/tutorials.php)<sup>1</sup>
- [Tutorial from drmaa.org](https://forge.gridforum.org/sf/docman/do/listDocuments/projects.drmaa-wg/docman.root.ggf_13_drmaa_tutorial)<sup>2</sup>

---

<sup>1</sup> <http://www.gridway.org/documentation/tutorials.php>

<sup>2</sup> [https://forge.gridforum.org/sf/docman/do/listDocuments/projects.drmaa-wg/docman.root.ggf\\_13\\_drmaa\\_tutorial](https://forge.gridforum.org/sf/docman/do/listDocuments/projects.drmaa-wg/docman.root.ggf_13_drmaa_tutorial)

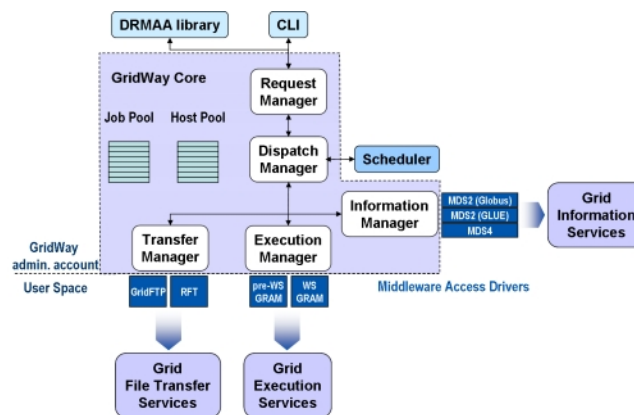
---

# Chapter 4. Architecture and design overview

## 1. Architecture

In GridWay 4.0.2, we introduced an architecture for the execution manager module based on a MAD (Middleware Access Driver) to interface Grid execution services. In this release we have taken advantage of this architecture to implement an information manager module with a MAD that interfaces Grid information services, and a transfer manager module with a MAD that interfaces Grid data services. Moreover, we have decoupled the scheduling process from the dispatch manager through the use of an external and selectable scheduler module.

**Figure 4.1. Components of the GridWay Meta-scheduler.**



GridWay architecture consists of the following components:

- *User Interface* provides the end user with DRM-like commands to submit, kill, migrate, monitor and synchronize jobs and includes DRMAA (Distributed Resource Management Application API) GGF (Global Grid Forum) standard support to develop distributed applications (C and JAVA bindings).
- *GridWay core* is responsible for job execution management and resource brokering providing advanced scheduling and job failure & recovery capabilities. The Dispatch Manager performs all submission stages and watches over the efficient execution of the job. The Information Manager, through its MADs (Middleware Access Driver), is responsible for host discovery and monitoring. The Execution Manager, through its MADs, is responsible job execution and management. The Transfer Manager, through its MADs, is responsible for file staging, remote working directory set-up and remote host clean-up.
- *Scheduler* takes scheduling decisions of jobs on available resources.
- *Information Manager MAD* interfaces to the monitoring and discovering services available in the Grid infrastructure.
- *Execution Manager MAD* interfaces to the Job Management Services available in the Grid resources.
- *Transfer Manager MAD* interfaces to the Data Management Services available in the Grid resources.

GWD communicates with MADs through the standard input/output streams. This makes easier the debugging process of new MADs.

## 2. Programming Model Overview

### 2.1. Programming Model

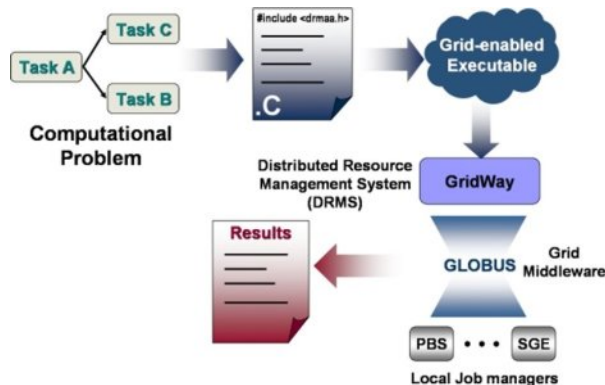
One of the most important aspects of Grid Computing is its potential ability to execute distributed communicating jobs. The Distributed Resource Management Application API (DRMAA) specification constitutes a homogeneous interface to different Distributed Resource Management Systems (DRMS) to handle job submission, monitoring and control, and retrieval of finished job status.

In this way, DRMAA could aid scientists and engineers to express their computational problems by providing a portable direct interface to DRMS. There are several projects underway to implement the DRMAA specification on different DRMS, like Sun Grid Engine (SGE), Condor or Torque.

GridWay provides a full-featured native implementation of the DRMAA standard to interface DRMS through the Globus Toolkit. The GridWay DRMAA library can successfully compile and execute the DRMAA test suite (version 1.4.0). Please check the GridWay DRMAA Reference Guide for a complete description of the DRMAA routines.

Although DRMAA could interface with DRMS at different levels, for example at the intranet level with SGE or Condor, with GridWay we will only consider its application at Grid level. In this way, the DRMS (GridWay in our case) will interact with the local job managers (Condor, PBS, SGE...) through the Grid middleware (Globus). This development and execution scheme with DRMAA, GridWay and Globus is depicted in the next figure.

**Figure 4.2. Grid Development Model with DRMAA**

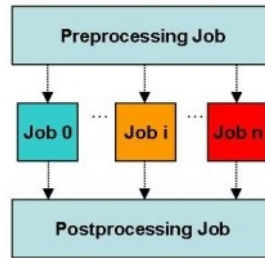


### 2.2. Application Profiles

DRMAA allows scientists and engineers to express their computational problems in a Grid environment. The capture of the job exit code allows users to define complex jobs, where each depends on the output and exit code from the previous job. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the workflow of certain type of applications. Let's review some typical scientific application profiles that can benefit from DRMAA.

#### 2.2.1. Embarrassingly distributed

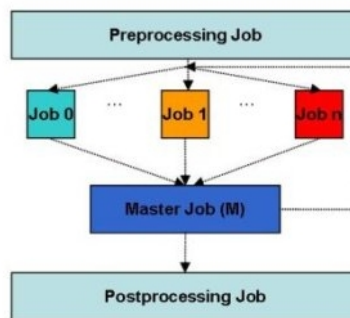
Applications that can be obviously divided into a number of independent tasks. The application is asynchronous when it requires distinct instruction streams and so different execution times. A sample of this schema with its DRMAA implementation is showed in the following figure.

**Figure 4.3. Embarrassingly Distributed Applications schema**

```
rc = drmaa_init (contact, err);
// Execute initial job and wait for it
rc = drmaa_run_job(job_id, jt, err);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err);
// Execute n jobs simultaneously and wait
rc = drmaa_run_bulk_jobs(job_ids, jt, 1, JOB_NUM, 1, err);
rc = drmaa_synchronize(job_ids, timeout, 1, err);
// Execute final job and wait for it
rc = drmaa_run_job(job_id, jt, err);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err);
rc = drmaa_exit(err_diag);
```

## 2.2.2. Master-worker

A Master task assigns a description (input less) of the task to be performed by each Worker. Once all the Workers are completed, the Master task performs some computations in order to evaluate a stop criterion or to assign new tasks to more workers. Again, it could be synchronous or asynchronous. The following figure shows a example of Master-worker optimization loop and a DRMAA implementation sample.

**Figure 4.4. Master-Worker Applications schema**

```
rc = drmaa_init(contact, err_diag);
// Execute initial job and wait for it
rc = drmaa_run_job(job_id, jt, err_diag);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag);
while (exitstatus != 0)
{
```

```
// Execute n Workers concurrently and wait
rc = drmaa_run_bulk_jobs(job_ids, jt, 1, JOB_NUM, 1, err_diag);
rc = drmaa_synchronize(job_ids, timeout, 1, err_diag);
// Execute the Master, wait and get exit code
rc = drmaa_run_job(job_id, jt, err_diag);
rc = drmaa_wait(job_id, &stat, timeout, rusage, err_diag);
rc = drmaa_wexitstatus(&exitstatus, stat, err_diag);
}
rc = drmaa_exit(err_diag);
```

---

# Chapter 5. APIs

## 1. DRMAA bindings for C and Java

The following links reference the API for the C and Java bindings of GridWay's implementation of DRMAA:

- [DRMAA C reference](#)<sup>1</sup>
- [DRMAA JAVA reference](#)<sup>2</sup>

## 2. DRMAA bindings for scripting languages.

Most functions in DRMAA C library use reference parameters to get results when calling them. Using scripting languages makes this way of getting information unfeasible. For example python strings are immutable so it is not possible to fill an empty string with the information needed from the extension. The way the functions are called are the same as Perl and Python SGE DRMAA bindings, reference variables are omitted and what functions return is an array of result code, reference variables and error string. For example in C this call:

```
result=drmaa_version(&major,&minor,error,DRMAA_ERROR_STRING_BUFFER-1);
```

is translated to ruby as:

```
(result, major, minor, error)=drmaa_version
```

---

<sup>1</sup> [http://www.gridway.org/documentation/stable/drmaa\\_c/](http://www.gridway.org/documentation/stable/drmaa_c/)

<sup>2</sup> [http://www.gridway.org/documentation/stable/drmaa\\_java/](http://www.gridway.org/documentation/stable/drmaa_java/)

**Table 5.1. Translation from C to Scripting language**

C	Scripting Language
result=drmaa_get_next_attr_name(values, &value, value_len)	(result, value)=drmaa_get_next_attr_name(values)
result=drmaa_get_next_attr_value(values, &value, value_len)	(result, value)=drmaa_get_next_attr_value(values)
result=drmaa_get_next_job_id(values, &value, value_len)	(result, value)=drmaa_get_next_job_id(values)
result=drmaa_get_num_attr_names(values, &size)	(result, size)=drmaa_get_num_attr_names(values)
result=drmaa_get_num_attr_values(values, &size)	(result, size)=drmaa_get_num_attr_values(values)
result=drmaa_get_num_job_ids(values, &size)	(result, size)=drmaa_get_num_job_ids(values)
drmaa_release_attr_names(values)	drmaa_release_attr_names(values)
drmaa_release_attr_values(values)	drmaa_release_attr_values(values)
drmaa_release_job_ids(values)	drmaa_release_job_ids(values)
result=drmaa_init(contact, error, error_len)	(result, error)=drmaa_init(nil)
result=drmaa_exit(error, error_len)	(result, error)=drmaa_exit()
result=drmaa_allocate_job_template(&jt, error, error_len)	(result, jt, error)=drmaa_allocate_job_template()
result=drmaa_delete_job_template(jt, error, error_len)	(result, error)=drmaa_delete_job_template(jt)
result=drmaa_set_attribute(jt, name, value, error, error_len)	(result, error)=drmaa_set_attribute(jt, name, value)
result=drmaa_get_attribute(jt, name, &value, error, error_len)	(result, value, error)=drmaa_get_attribute(jt, name)
result=drmaa_set_vector_attribute(jt, name, value, error, error_len)	(result, error)=drmaa_set_vector_attribute(jt, name, value)
result=drmaa_get_vector_attribute(jt, name, &values, error, error_len)	(result, values, error)=drmaa_get_vector_attribute(jt, name)
result=drmaa_get_attribute_names(&values, error, error_len)	(result, values, error)=drmaa_get_attribute_names()
result=drmaa_get_vector_attribute_names(&values, error, error_len)	(result, values, error)=drmaa_get_vector_attribute_names()
result=drmaa_run_job(job_id, job_id_len, jt, error, error_len)	(result, job_id, error)=drmaa_run_job(jt)
result=drmaa_run_bulk_jobs(&jobids, jt, start, end, incr, error, error_len)	(result, jobids, error)=drmaa_run_bulk_jobs(jt, start, end, incr)
result=drmaa_control(jobid, action, error, error_len)	(result, error)=drmaa_control(jobid, action)
result=drmaa_job_ps(job_id, &remote_ps, error, error_len)	(result, remote_ps, error)=drmaa_job_ps(job_id)
result=drmaa_synchronize(job_ids, timeout, dispose, error, error_len)	(result, error)=drmaa_synchronize(job_ids, timeout, dispose)
result=drmaa_wait(job_id, job_id_out, job_id_out_len, &stat, timeout, &rusage, error, error_len)	(result, job_id_out, stat, rusage, error)=drmaa_wait(job_id, timeout)
result=drmaa_wifexited(&exited, stat, error, error_len)	(result, exited, error)=drmaa_wifexited(stat)

C	Scripting Language
result=drmaa_wexitstatus(&exit_status, stat, error, error_len)	(result, exit_status, error)=drmaa_wexitstatus(stat)
result=drmaa_wifsignaled(&signaled, stat, error, error_len)	(result, signaled, error)=drmaa_wifsignaled(stat)
result=drmaa_wtermsig(signal, signal_len, stat, error, error_len)	(result, signal, error)=drmaa_wtermsig(stat)
result=drmaa_wcoredump(&core_dumped, stat, error, error_len)	(result, core_dumped, error)=drmaa_wcoredump(stat)
result=drmaa_wifaborted(&aborted, stat, error, error_len)	(result, aborted, error)=drmaa_wifaborted(stat)
error_string=drmaa_strerror(drmaa_errno)	error_string=drmaa_strerror(drmaa_errno)
result=drmaa_get_contact(contact, contact_len, error, error_len)	(result, contact, error)=drmaa_get_contact()
result=drmaa_version(&major, &minor, error, error_len)	(result, major, minor, error)=drmaa_version()
result=drmaa_get_DRM_system(drm_system, drm_system_len, error, error_len)	(result, drm_system, error)=drmaa_get_DRM_system()
result=drmaa_get_DRMAA_implementation(drmaa_impl, drmaa_impl_len, error, error_len)	(result, drmaa_impl, error)=drmaa_get_DRMAA_implementation()
str_status=drmaa_gw_strstatus(drmaa_state)	str_status=drmaa_gw_strstatus(drmaa_state)



## Note

In `drmaa_init` call we have to pass a **NULL** argument as GridWay DRMAA library requires it. Here as an example I used `nil` as it is the ruby object describing **NULL** but in perl you have to use `undef` and in python `None`.

---

# Chapter 6. Non-WSDL protocols

## 1. Information manager MAD

In order to provide an abstraction with the monitoring and discovery middleware layer (or Grid Information System), GridWay uses a Middleware Access Driver (MAD) module to discover and monitor hosts. This module provides basic operations with the monitoring and discovery middleware.

The format to send a request to the Execution MAD, through its standard input, is:

```
OPERATION HID HOST -
```

Where:

- **OPERATION:** Can be one of the following:
  - **INIT:** Initializes the MAD.
  - **DISCOVER:** Discovers hosts.
  - **MONITOR:** Monitors a host.
  - **FINALIZE:** Finalizes the MAD.
- **HID:** If the operation is **MONITOR**, it is a host identifier, chosen by GridWay. Otherwise it is ignored.
- **HOST:** If the operation is **MONITOR** it specifies the host to monitor. Otherwise it is ignored.

On the other side, the format to receive a response from the MAD, through its standard output, is:

```
OPERATION HID RESULT INFO
```

Where:

- **OPERATION:** Is the operation specified in the request that originated the response.
- **HID:** It is the host identifier, as provided in the submission request.
- **RESULT:** It is the result of the operation. Could be **SUCCESS** or **FAILURE**.
- **INFO:** If **RESULT** is **FAILURE**, it contains the cause of failure. Otherwise, if **OPERATION** is **DISCOVER**, it contains a list of discovered host, or if **OPERATION** is **MONITOR**, it contains a list of host attributes.

**Table 6.1. Attributes that should be defined by the Information MADs.**

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

The information drivers interface to the grid information services to collect the resource attributes. These attributes can be used by the end-user to set requirement and rank expressions (job template), for filtering, prioritizing and selecting the candidate hosts. GridWay can simultaneously use as many Information drivers as needed. For example, GridWay allows you to simultaneously use MDS2 and MDS4 services, so you can also use resources from different Grids at the same time. Drivers for MDS 2 and MDS 4 provide the variables described in Table 2-1. However, the information manager is able to receive from the driver other parameters. The GridWay team has used other information parameters that could be very important to improve application efficiency (HTC apps) and for job migration: BANDWIDTH, LATENCY, SPEC\_INT, SPEC\_FLOAT...

## 2. Execution manager MAD

In order to provide an abstraction with the resource management middleware layer, GridWay uses a Middleware Access Driver (MAD) module to submit, control and monitor the execution of jobs. This module provides basic operations with the resource management middleware.

The format to send a request to the Execution MAD, through its standard input, is:

```
OPERATION JID HOST/JM RSL
```

Where:

- **OPERATION:** Can be one of the following:
  - **INIT:** Initializes the MAD.
  - **SUBMIT:** Submits a job.
  - **POLL:** Polls a job to obtain its state.
  - **CANCEL:** Cancels a job.
  - **FINALIZE:** Finalizes the MAD.
- **JID:** Is a job identifier, chosen by GridWay.
- **HOST:** If the operation is **SUBMIT**, it specifies the resource contact to submit the job. Otherwise it is ignored.
- **JM:** If the operation is **SUBMIT**, it specifies the job manager to submit the job. Otherwise it is ignored.
- **RSL:** If the operation is **SUBMIT**, it specifies the resource specification to submit the job. Otherwise it is ignored.

On the other side, the format to receive a response from the MAD, through its standard output, is:

```
OPERATION JID RESULT INFO
```

Where:

- **OPERATION:** Is the operation specified in the request that originated the response or **CALLBACK**, in the case of an asynchronous notification of a state change.
- **JID:** It is the job identifier, as provided in the submission request.
- **RESULT:** It is the result of the operation. Could be **SUCCESS** or **FAILURE**.
- **INFO:** If **RESULT** is **FAILURE**, it contains the cause of failure. Otherwise, if **OPERATION** is **POLL** or **CALLBACK**, it contains the state of the job.

## 3. Transfer manager MAD

In order to provide an abstraction with the file transfer management middleware layer, GridWay uses a Middleware Access Driver (MAD) module to transfer job files. This module provides basic operations with the file transfer middleware.

The format to send a request to the Transfer MAD, through its standard input, is:

OPERATION JID TID EXE\_MODE SRC\_URL DST\_URL

Where:

- OPERATION: Can be one of the following:
  - INIT: Initializes the MAD, JID should be max number of jobs.
  - START: Init transfer associated with job JID
  - END: Finish transfer associated with job JID
  - MKDIR: Creates directory SRC\_URL
  - RMDIR: Removes directory SRC\_URL
  - CP: start a copy of SRC\_URL to DST\_URL, with identification TID, and associated with job JID.
  - FINALIZE: Finalizes the MAD.
- JID: Is a job identifier, chosen by GridWay.
- TID: Transfer identifier, only relevant for command CP.
- EXE\_MODE: If equal to 'X' file will be given execution permissions, only relevant for command CP.

On the other side, the format to receive a response from the MAD, through its standard output, is:

OPERATION JID TID RESULT INFO

Where:

- OPERATION: Is the operation specified in the request that originated the response or CALLBACK, in the case of an asynchronous notification of a state change.
- JID: It is the job identifier, as provided in the START request.
- TID: It is the transfer identifier, as provided in the CP request.
- RESULT: It is the result of the operation. Could be SUCCESS or FAILURE.
- INFO: If RESULT is FAILURE, it contains the cause of failure.

## 4. Dispatch manager Scheduler

In order to decouple the scheduling process, GridWay uses a Scheduler module to schedule jobs.

The format to send a scheduling request to the Scheduler, through its standard input, is:

SCHEDULE - - -

On the other side, the format to receive a response from the Scheduler, through its standard output, is:

OPERATION JID RESULT INFO

Where:

- **OPERATION:** Is the operation requested to the Dispatch Manager. The Dispatch Manager only supports the `SCHEDULE_JOB` operation.
- **JID:** It is a job identifier.
- **RESULT:** It is the result of the operation. Could be `SUCCESS` or `FAILURE`.
- **INFO:** If **RESULT** is `FAILURE`, it contains the cause of failure. Otherwise, if **OPERATION** is `SCHEDULE_JOB` it contains a resource specification in the form `HID:QNAME:RANK`, where:
  - **HID:** It is the host identifier, as provided by `gwhosts` command.
  - **QNAME:** It is the queue name.
  - **RANK:** It is the rank of the host.

GridWay includes a scheduler template (`gw_scheduler_skel.c`) to develop custom schedulers. As an example a Round-Robin/Flooding scheduler can be found in the GridWay distribution (`gw_flood_scheduler.c`, in `$GW_LOCATION/src/sched/`). This is a very simple scheduling algorithm, which maximizes the number of jobs submitted to the Grid.

---

# GridWay Commands

---

# Name

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

```
gws submit <-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 $PARAM=start + increment * TASK\_ID$ , and $MAX\_PARAM = start+increment*(tasks-1)$ . 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 3. Field options**

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

---

# 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 Prints help.

-n Do not print the header lines

job\_id Job identification as provided by gwps.

# Output field description

**Table 4. Field information**

NAME	DESCRIPTION
HID	host unique identification assigned by the Gridway system.
START	the time the job start its execution on this host.
END	the time the job left this host, because it finished or it was migrated.
PROLOG	total prolog (file stage-in phase) time.
WRAPPER	total wrapper (execution phase) time.
EPILOG	total epilog (file stage-out phase) time.
MIGR	total migration time.
REASON	the reason why the job left this host.
QUEUE	name of the queue.
HOST	FQDN of the host.

---

# 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 5. Field information**

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

**Table 6. Queue field information**

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

---

# Name

Job Control Command -- controls job execution

```
gkill [-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

```
gwwait [-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 7. Field information**

<b>FIELD</b>	<b>DESCRIPTION</b>
UID	user unique identification assigned by the Gridway system
NAME	name of this user
JOBS	number of Jobs in the GridWay system
RUN	number of running jobs
IDLE	idle time, (time with JOBS = 0)
EM	execution manager drivers loaded for this user
TM	transfer manager drivers loaded for this user
PID	process identification of driver processes

---

# 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 8. Field information**

FIELD	DESCRIPTION
HOST/USER	host/user usage summary for this user/host
XFR	total transfer time on this host (for this user)
EXE	total execution time on this host (for this user), without suspension time
SUSP	total suspension (queue) time on this host (for this user)
TOTS	total executions on this host (for this user). Termination reasons: <ul style="list-style-type: none"><li>• SUCC success</li><li>• ERR error</li><li>• KILL kill</li><li>• USER user requested</li><li>• SUSP suspension timeout</li><li>• DISC discovery timeout</li><li>• SELF self migration</li><li>• PERF performance degradation</li><li>• S/R stop/resume</li></ul>

---

## 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 gws submit in the following fashion:

```
jsdl2gw jsdl-job.xml | gws submit
```

## 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 7. Debugging

We'll begin with debugging the underlying Java WS Core and then discuss debugging in GridWay in particular. For information about sys admin logging, see [Chapter 7, Debugging](#) in the Admin Guide.

## 1. Debugging in Java WS Core

As GridWay relies on Globus services, it is assumed that a Globus grid infrastructure has been installed and configured. Failures related to Globus services (e.g. GRAM or MDS) can be debugged as described in [Chapter 10, Debugging](#).

### 1.1. Development Logging in Java WS Core

The following information applies to Java WS Core and those services built on it.

Logging in the Java WS Core is based on the [Jakarta Commons Logging](#)<sup>1</sup> API. Commons Logging provides a consistent interface for instrumenting source code while at the same time allowing the user to plug-in a different logging implementation. Currently we use [Log4j](#)<sup>2</sup> as a logging implementation. Log4j uses a separate configuration file to configure itself. Please see Log4j documentation for details on the [configuration file format](#)<sup>3</sup>.

#### 1.1.1. Configuring server side developer logs

Server side logging can be configured in `$GLOBUS_LOCATION/container-log4j.properties`, when the container is stand alone container. For tomcat level logging, refer to [Logging for Tomcat](#)<sup>4</sup>, . The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

Additional logging can be enabled for a package by adding a new line to the configuration file. Example:

```
#for debug level logging from org.globus.package.FooClass
log4j.category.org.globus.package.name.FooClass=DEBUG
#for warnings from org.some.warn.package
log4j.category.org.some.warn.package=WARN
```

#### 1.1.2. Configuring client side developer logs

Client side logging can be configured in `$GLOBUS_LOCATION/log4j.properties`. The logger `log4j.appender.A1` is used for developer logging and by default writes output to the system output. By default it is set for all warnings in the Globus Toolkit package to be displayed.

## 2. Debugging in GridWay

Due to GridWay's architecture, mainly its MADs components, debugging it is not a trivial task. The most obvious way to see what is going on is to monitor what happens in the GridWay log files. Here are the files to look into in case of trouble:

---

<sup>1</sup> <http://jakarta.apache.org/commons/logging/>

<sup>2</sup> <http://logging.apache.org/log4j/>

<sup>3</sup> [http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure\(java.lang.String,org.apache.log4j.spi.LoggerRepository\)](http://logging.apache.org/log4j/docs/api/org/apache/log4j/PropertyConfigurator.html#doConfigure(java.lang.String,org.apache.log4j.spi.LoggerRepository))

<sup>4</sup> <http://tomcat.apache.org/tomcat-5.5-doc/logging.html>

- `$GW_LOCATION/var/gwd.log`: This is the general log file, where the gwd daemon logs whatever the Resource, Dispatch, Transfer, Execution, Information managers inform it about.
- `$GW_LOCATION/var/sched.log`: The scheduler is a separate process that communicates with the daemon using the standard input/output. It writes log information to this file.
- `$GW_LOCATION/var/<job_id>/job.log`: Each job has its own log file, with information regarding its context (input/output files, MADs, resource) and it's life cycle. In this folder also reside the `job.template`, the `job.env` with the environment variable and the standard output and error of the wrapper script (`stdout.wrapper` and `stderr.wrapper`)

In order to get the maximum amount of debug information in the `gwd.log` file (especially more information about what the MADs are doing), you should compile GridWay with the following configure option:

```
./configure --enable-debug
```

If there is a problem with GridWay that makes any MAD crash, it will be useful to have a coredump. To tell the MADs that they should write to a coredump file when they crash, use the following environment variable before you execute your first job:

```
export MADDEBUG=yes
```

Sometimes it is the daemon (the gwd process) that crashes. In order to obtain a coredump of the daemon, run the following command before executing the daemon:

```
ulimit -c unlimited
```

The coredump file will be written to the `$GW_LOCATION/var` directory, with a filename corresponding to the process PID, that is,

```
$GW_LOCATION/var/core.<process_pid>
```

If you cannot figure out what is wrong, you can always use the mailing list [gridway-user](mailto:gridway-user)<sup>5</sup> to get support. Please provide a detailed explanation of your problem so the community can reproduce it and give advice. Also, send along:

- `$GW_LOCATION/var/gwd.log`
- `$GW_LOCATION/var/sched.log`
- `$GW_LOCATION/var/<job_id>/{job.log,stderr.wrapper}`: If relevant. The `stderr.wrapper` file is specially useful for debugging; it shows step by step the wrapper script being executed.

---

<sup>5</sup> [http://dev.globus.org/wiki/GridWay#Mailing\\_Lists](http://dev.globus.org/wiki/GridWay#Mailing_Lists)

---

# Chapter 8. Troubleshooting

For a list of common errors in GT, see [Error Codes](#).

## 1. Errors

**Table 8.1. Gridway Errors**

Error Code	Definition	Possible Solutions
Lock file exists	Another GWD may be running.	Be sure that no other GWD is running, then remove the lock file and try again.
Error in MAD initialization	There may be problems with the proxy certificate, bin directory, or the executable name of a MAD may not be in the correct location.	Check that you have generated a valid proxy (for example with the <b>grid-proxy-info</b> command). Also, check that the directory <code>\$GW_LOCATION/bin</code> is in your path, and the executable name of all the MADs is defined in <code>gwd.conf</code> .
Could not connect to gwd	GridWay may not be running or there may be something wrong with the connection.	Be sure that GWD is running; for example: <pre>pgrep -l gwd</pre> If it is running, check that you can connect to GWD; for example: <pre>telnet `cat \$GW_LOCATION/var/gwd.port`</pre>

## 2. Debugging

For more detailed developer debugging information, see [Chapter 7, Debugging](#). For information about sys admin logging, see [Chapter 7, Debugging](#).

---

# Chapter 9. GT 4.2.1 Samples for GridWay

## 1. DRMAA examples

The following links contain samples demonstrating GridWay's DRMAA bindings:

- [DRMAA C Howtos](#)<sup>1</sup>
- [DRMAA Java Howtos](#)<sup>2</sup>
- [DRMAA Python Howtos](#)<sup>3</sup>
- [DRMAA Ruby Howtos](#)<sup>4</sup>
- [DRMAA Perl Howtos](#)<sup>5</sup>

---

<sup>1</sup> <http://www.gridway.org/files/examples/drmaaC.tgz>

<sup>2</sup> <http://www.gridway.org/files/examples/drmaaJ.tgz>

<sup>3</sup> [http://www.gridway.org/files/examples/drmaa\\_python.tar.gz](http://www.gridway.org/files/examples/drmaa_python.tar.gz)

<sup>4</sup> [http://www.gridway.org/files/examples/drmaa\\_ruby.tar.gz](http://www.gridway.org/files/examples/drmaa_ruby.tar.gz)

<sup>5</sup> [http://www.gridway.org/files/examples/drmaa\\_perl.tar.gz](http://www.gridway.org/files/examples/drmaa_perl.tar.gz)

---

# GT 4.2.1 GridWay: Quality Profile

## Table of Contents

1. Test coverage reports .....	1
2. Code analysis reports .....	1
3. Outstanding bugs .....	1
4. Bug Fixes .....	1
5. Performance reports .....	2

<titleabbrev>Quality Profile</titleabbrev>

## 1. Test coverage reports

- [Not available](#)<sup>1</sup>

## 2. Code analysis reports

- [list]

## 3. Outstanding bugs

- [Bug 5718](#):<sup>2</sup> Transfer Errors
- [Bug 5719](#):<sup>3</sup> Timeouts for EM and TM Drivers
- [Bug 6003](#):<sup>4</sup> socket remains open when connection to gwd failed
- [Bug 6132](#):<sup>5</sup> GridGateWay install fails with a GridWay Globus Toolkit install
- [Bug 6401](#):<sup>6</sup> SSH Mads incompatible with ruby ssh drivers current version

## 4. Bug Fixes

- [Bug 6231](#):<sup>7</sup> GridWay fails to compile against Globus Toolkit 4.0.X
- [Bug 6254](#):<sup>8</sup> EM\_MAD\_WS doesn't load
- [Bug 6171](#):<sup>9</sup> IM\_MAD cannot cope with more than two level hierarchy in MDS4

---

<sup>1</sup> path

<sup>2</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5718](http://bugzilla.globus.org/globus/show_bug.cgi?id=5718)

<sup>3</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5719](http://bugzilla.globus.org/globus/show_bug.cgi?id=5719)

<sup>4</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6003](http://bugzilla.globus.org/globus/show_bug.cgi?id=6003)

<sup>5</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6132](http://bugzilla.globus.org/globus/show_bug.cgi?id=6132)

<sup>6</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6401](http://bugzilla.globus.org/globus/show_bug.cgi?id=6401)

<sup>7</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6231](http://bugzilla.globus.org/globus/show_bug.cgi?id=6231)

<sup>8</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6254](http://bugzilla.globus.org/globus/show_bug.cgi?id=6254)

<sup>9</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6171](http://bugzilla.globus.org/globus/show_bug.cgi?id=6171)

## 5. Performance reports

- [GridWay Report: Scalability](#)<sup>10</sup>

---

<sup>10</sup> [http://www.gridway.org/documentation/files/GridWay\\_Internal\\_Report\\_Scalability.pdf](http://www.gridway.org/documentation/files/GridWay_Internal_Report_Scalability.pdf)

---

# GT 4.2.1 Release Notes: GridWay

## Table of Contents

1. Component Overview .....	1
2. Feature summary .....	1
3. Summary of Changes in GridWay .....	2
4. Bug Fixes .....	2
5. Known Problems .....	3
6. Technology dependencies .....	4
7. Tested platforms .....	4
8. Backward compatibility summary .....	4
9. Associated Standards .....	5
10. For More Information .....	5

<titleabbrev>4.2.1 Release Notes</titleabbrev>

## 1. Component Overview

The GridWay Metascheduler enables large-scale, reliable and efficient sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by different LRM (Local Resource Management) systems, such as PBS, SGE, LSF, Condor..., within a single organization (enterprise grid) or scattered across several administrative domains (partner or supply-chain grid).

## 2. Feature summary

**Advanced Scheduling Capabilities**      GridWay implements several state-of-the-art Grid-aware scheduling policies, comprising *job prioritization* policies (fixed priority, urgency, share, deadline and waiting-time) and *resource prioritization* policies (fixed priority, usage, failure and rank).

These policies are combined with:

- *Adaptive Scheduling*, to periodically adapt the schedule considering applications' demands and Grid resource characteristics.
- *Adaptive Execution* to migrate running applications in terms of resource availability, capacity or cost, and new application requirements or preferences.

**Transparent Grid Access**      GridWay interfaces infrastructures with different middleware stacks. With GridWay, users can access heterogeneous resources in a transparent way. For example, it can access resources configured with both GRAM pre web services and GRAM web services. It also permits the use of different grids with different software stacks, for example, Teragrid with Globus and EGEE with gLite.

**Flexible Deployment Capabilities**      GridWay supports multiple-user operation mode, and does not require additional middleware installation (apart from standard Globus services). Globus installation is not required in each end-user system.

	GridWay allows different Grid deployment strategies, like Enterprise Grids, Partner Grids or Utility Grids.
Different Application Profiles	GridWay executes different Grid application profiles: <ul style="list-style-type: none"><li>• Array (Bulk) jobs, for parameter sweep applications</li><li>• DAG Workflows</li><li>• Single-site MPI applications</li></ul>
Fault Detection and Recovery	GridWay is able to detect several problems that can occur when executing a remote job. It also implements mechanisms that make the execution more reliable. It can detect a remote system crash, a job failure (via the job exit code) or even a network disconnection (using the polling mechanism) and migrate the problematic job to another resource.  GridWay also performs periodic saves of its state in order to recover from local failure.
Reporting and Accounting	GridWay provides detailed statistics of Grid usage. In this way, the Grid administrator can properly plan usage policies and forecast workload. In addition, these statistics can be used by the scheduler to predict (per user) Grid resource response time.
Standard Compliance	GridWay is an open-source project, flexible and completely based on standards to leverage its usability and interoperability. For example, users can describe their jobs using JSDL. Similarly, programmers can build grid enabled applications using the DRMAA standard.
User Interface	GridWay provides end-users with a familiar environment similar to that found on classical LRM systems. So GridWay CLI eases the adoption of Grid technologies.

## 3. Summary of Changes in GridWay

GridWay version shipped with GT4.2.1 is based on GridWay 5.4, a new stable version with increased functionality, including support to access the NorduGrid infrastructure, new bindings for DRMAA and support for DAGMAN workflows.

New to 4.2.1: Since this is an incremental release, only bug fixes have been included.

## 4. Bug Fixes

- [Bug 6231](http://bugzilla.globus.org/globus/show_bug.cgi?id=6231):<sup>1</sup> GridWay fails to compile against Globus Toolkit 4.0.X
- [Bug 6254](http://bugzilla.globus.org/globus/show_bug.cgi?id=6254):<sup>2</sup> EM\_MAD\_WS doesn't load
- [Bug 6171](http://bugzilla.globus.org/globus/show_bug.cgi?id=6171):<sup>3</sup> IM\_MAD cannot cope with more than two level hierarchy in MDS4

---

<sup>1</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6231](http://bugzilla.globus.org/globus/show_bug.cgi?id=6231)

<sup>2</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6254](http://bugzilla.globus.org/globus/show_bug.cgi?id=6254)

<sup>3</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6171](http://bugzilla.globus.org/globus/show_bug.cgi?id=6171)

## 5. Known Problems

The following problems and limitations are known to exist for GridWay at the time of the 4.2.1 release:

### 5.1. Limitations

- [Bug 5308](#):<sup>4</sup> gwd doesn't recover AIDs and TIDs
- [Bug 5626](#):<sup>5</sup> GridWay should offer the ability to start from a specified job ID
- [Bug 5722](#):<sup>6</sup> Autotool for the new doc
- [Bug 5727](#):<sup>7</sup> Improve DB checks in autoconf
- [Bug 5735](#):<sup>8</sup> Update development guide with the binding 1.0 for DRMAA JAVA
- [Bug 5883](#):<sup>9</sup> MAX\_RUNNING\_RESOURCE should be configurable for each host
- [Bug 6037](#):<sup>10</sup> implementation of pre\_wrapper option in libdrmaa
- [Bug 6334](#):<sup>11</sup> Customized Gridway JOB IDs
- [Bug 6343](#):<sup>12</sup> improve gridway error messages and logging
- [Bug 6346](#):<sup>13</sup> MADs should check for java executable if used

### 5.2. Outstanding bugs

- [Bug 5718](#):<sup>14</sup> Transfer Errors
- [Bug 5719](#):<sup>15</sup> Timeouts for EM and TM Drivers
- [Bug 6003](#):<sup>16</sup> socket remains open when connection to gwd failed
- [Bug 6132](#):<sup>17</sup> GridGateWay install fails with a GridWay Globus Toolkit install
- [Bug 6401](#):<sup>18</sup> SSH Mads incompatible with ruby ssh drivers current version

---

<sup>4</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5308](http://bugzilla.globus.org/globus/show_bug.cgi?id=5308)

<sup>5</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5626](http://bugzilla.globus.org/globus/show_bug.cgi?id=5626)

<sup>6</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5722](http://bugzilla.globus.org/globus/show_bug.cgi?id=5722)

<sup>7</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5727](http://bugzilla.globus.org/globus/show_bug.cgi?id=5727)

<sup>8</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5735](http://bugzilla.globus.org/globus/show_bug.cgi?id=5735)

<sup>9</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5883](http://bugzilla.globus.org/globus/show_bug.cgi?id=5883)

<sup>10</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6037](http://bugzilla.globus.org/globus/show_bug.cgi?id=6037)

<sup>11</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6334](http://bugzilla.globus.org/globus/show_bug.cgi?id=6334)

<sup>12</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6343](http://bugzilla.globus.org/globus/show_bug.cgi?id=6343)

<sup>13</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6346](http://bugzilla.globus.org/globus/show_bug.cgi?id=6346)

<sup>14</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5718](http://bugzilla.globus.org/globus/show_bug.cgi?id=5718)

<sup>15</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=5719](http://bugzilla.globus.org/globus/show_bug.cgi?id=5719)

<sup>16</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6003](http://bugzilla.globus.org/globus/show_bug.cgi?id=6003)

<sup>17</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6132](http://bugzilla.globus.org/globus/show_bug.cgi?id=6132)

<sup>18</sup> [http://bugzilla.globus.org/globus/show\\_bug.cgi?id=6401](http://bugzilla.globus.org/globus/show_bug.cgi?id=6401)

## 6. Technology dependencies

GridWay uses different Globus services to perform the tasks of information gathering, job execution and data transfer.

- GridWay depends on the following GT components for job execution:
  - GRAM: GridWay can interface with both GRAM 2 (pre web services) and GRAM 4 (web services)
- GridWay depends on the following GT components for data staging:
  - RFT
  - GridFTP
- GridWay depends on the following GT component for information gathering:
  - MDS
- GridWay relies on the Globus basic security infrastructure for authentication and authorization. On top of that, GridWay can use other Globus services and components to complement this infrastructure:
  - Delegation Service
  - MyProxy

## 7. Tested platforms

Tested platforms for GridWay:

- GridWay builds successfully for the following platforms:
  - Linux
  - Tru64
  - Mac OS X
  - Solaris
  - Aix

In addition, GridWay has been tested with the major Grid infrastructures. Click the following links to find more information on how to use GridWay with [EGEE](#)<sup>19</sup>, [TeraGrid](#)<sup>20</sup>, [OSG](#)<sup>21</sup> and [Nordug](#)<sup>22</sup>.

## 8. Backward compatibility summary

The following information regards compatibility with the previous stable version of GridWay included in GT 4.0 series:

API changes since GT 4.0:

---

<sup>19</sup> <http://www.gridway.org/documentation/stable/egeehowto>

<sup>20</sup> <http://www.gridway.org/documentation/stable/tghowto/>

<sup>21</sup> <http://www.gridway.org/documentation/stable/osghowto/>

<sup>22</sup> <http://www.gridway.org/documentation/stable/nghowtoo/>

- Added DRMAA scripting language bindings for Python, Ruby and Perl.

CLI changes since GT 4.0:

- Added gwdag, a tool to run Condor DAGMAN compatible workflows.

Configuration interface changes since GT 4.0:

- None

## 9. Associated Standards

GridWay is an open project, flexible and completely based on standards to leverage its usability and interoperability. GridWay supports several standards developed by the Open Grid Forum,<sup>23</sup> namely:

Distributed Resource Management Application API Working Group (DRMAA-WG)

The DRMAA working group<sup>24</sup> has developed an API specification for the submission and control of jobs to one or more Distributed Resource Management (DRM) systems. The goal is to facilitate the direct interfacing of applications to today's DRM systems by application's builders, portal builders, and Independent Software Vendors (ISVs). The Distributed Resource Management Application API (DRMAA) provides a generalized API to distributed resource management systems (DRMSs) in order to facilitate integration of application programs.

The scope of DRMAA is limited to job submission, job monitoring and control, and retrieval of the finished job status. DRMAA provides application developers and distributed resource management builders with a programming model that enables the development of distributed applications tightly coupled to an underlying DRMS. DRMAA preserves flexibility and choice in system design.

GridWay provides support for the DRMAA API (JAVA and C bindings) to develop distributed applications. OGF document GFD.104: GridWay DRMAA 1.0 Implementation - Experience Report<sup>25</sup> provides details of the implementation.

Job Submission Description Language (JSDL-WG)

The goal of the JSDL working group<sup>26</sup> is to produce a language that describes the requirements of jobs for submission to Grids. JSDL 1.0, published as OGF recommendation GFD-R-P.056<sup>27</sup> is an XML-based language that focuses mainly on computational jobs. The JSDL-WG is working on extending this language to address a wider class of jobs, including Web service invocations.

Last GridWay release provides support for JSDL, the POSIX Application profile and HTC Profile schemas can be used to describe jobs.

## 10. For More Information

Click here<sup>28</sup> for more information about this component.

---

<sup>23</sup> <http://www.ogf.org>

<sup>24</sup> <http://drmaa.org/wiki/>

<sup>25</sup> <http://www.ogf.org/documents/GFD.104.pdf>

<sup>26</sup> <http://forge.gridforum.org/sf/projects/jsdl-wg>

<sup>27</sup> <http://www.ggf.org/documents/GFD.56.pdf>

<sup>28</sup> <http://www.gridway.org>