Introduction to the GridWay Metascheduler

HPC SYSADMIN MEETING’12
Barcelona, Spain
October 15-16, 2012

Dr Ismael Marín Carrión
Distributed Systems Architecture Group
Universidad Complutense de Madrid
e-mail: i.marin@fdi.ucm.es / ismael.marin@ige-project.eu

Initiative for Globus in Europe
Contents

1. What is GridWay?
2. A Global Vision
3. Initiative for Globus in Europe
4. The GridWay Project
What is GridWay?

- The GridWay metascheduler enables large-scale, reliable and efficient sharing of computing resources over different grid middleware, providing a single point of access for them.

- GridWay provides a LRM-like CLI for submitting, monitoring, synchronizing and controlling jobs.

- GridWay implements the OGF standard DRMAA API, assuring compatibility of applications with LRM systems that implement the standard.

- GridWay provides BES- and GRAM-compliant interfaces, enabling the submission and monitoring of jobs through a standard interface.
Contents

1. What is GridWay?
2. A Global Vision
3. Initiative for Globus in Europe
4. The GridWay Project
A Global Vision

Global Architecture of a Computational Grid

- **Applications**
  - Standard API (OGF DRMAA)
  - Command Line Interface

- **Grid Meta-Scheduler**
  - Open source
  - Job execution management
  - Resource brokering

- **GridWay**

- **Grid Middleware**
  - Globus services
  - Standard interfaces
  - End-to-end (e.g. TCP/IP)

- **Infrastructure**
  - Highly dynamic & heterogeneous
  - High fault rate

- **Applications**
  - C, java

- **CLI**

- **DRMAA**

- **Results**
Contents

1. What is GridWay?
2. A Global Vision
3. Initiative for Globus in Europe
4. The GridWay Project
The IGE project is a EU FP7 project to coordinate European Globus activities.

Main objectives:
- Support the European computing infrastructures and their users
- Provide a central point of contact in Europe for Globus
- Strengthen the influence of European developers and users on the development of Globus

IGE serves as a comprehensive service provider for the European e-infrastructures regarding the development, customization, provisioning, support, and maintenance of components of the Globus Toolkit.
Initiative for Globus in Europe

Work Plan

Management and Overall Quality Assurance (WP1)

Community Involvement, Dissemination, and Training (WP2)

International Collaboration and Standardization (WP3)

Software Adaptation (WP4)

Infrastructure Support (WP5)

Test and Maintenance (WP6)
IGE Testbed

- GridWay instance provided by:
  - Universidad Complutense de Madrid (UCM), Spain

- Globus 5 resources provided by:
  - Technische Universität Dortmund (TUDO), Germany
  - Poznan Supercomputing and Networking Center (PSNC), Poland
  - Leibniz-Rechenzentrum, Bayerische Akademie der Wissenschaften (BADW-LRZ), Germany
  - Universitatea Technica Cluj-Napoca (UTCN), Romania
  - Stichting voor Fundamenteel Onderzoek der Materie – Institute for Subatomic Physics (FOM-NIKHEF), Netherlands
  - University of Edinburgh – Edinburgh Parallel Computing Centre (UEDIN-EPCC), United Kingdom

- Other resources:
  - University of Southampton (SOTON), United Kingdom
1. What is GridWay?
2. A Global Vision
3. Initiative for Globus in Europe
4. The GridWay Project
Some Projects and Infrastructures

- IRISGrid
- Politecnico di Torino
- CABGrid (Astrobiology Center)
- C2VO (Universidad de Castilla La Mancha)
- ESAC Grid (European Space Agency)
- CRO-GRID (Croatia)
- Sun Microsystems Solution Center World Grid
- EGEE Infrastructure
- BeinGRID Project
- GridX1 (Canadian Grid for HEP applications)
- Universidade do Porto
- Madras Institute of Technology
- National Center for High-Performance Computing

Some Application Porting Areas

- Life-Sciences
- Aerospace
- Fusion Physics
- Computational Chemistry
Development Process

• **Community** – Open Source Project. Globus Development Philosophy

• **Development Infrastructure:**
  - Mailing Lists (thanks to Globus Project!)
  - Bugzilla (dev.gridway.org)
  - Readmine
  - SVN

• **You are very welcome to contribute:**
  - Reporting Bugs (gridway-user@globus.org)
  - Making feature requests for the next GridWay release (gridway-user@globus.org)
  - Contributing your own developments (bug fixes, new features, documentation)

• **Detailed Roadmap:**
  - http://dev.gridway.org/projects/gridway/roadmap
Useful Links

- GridWay Development: http://dev.gridway.org/
- Documentation: http://gridway.org/doku.php?id=documentation
- Support: http://gridway.org/doku.php?id=support
  - User discussion: gridway-user@globus.org
  - Announcements: gridway-announce@globus.org
- The IGE Project: http://www.ige-project.eu/
Thank you for your attention!
Installation and Basic Configuration

HPC SYSADMIN MEETING'12
Barcelona, Spain
October 15-16, 2012

Dr Ismael Marín Carrión
Distributed Systems Architecture Group
Universidad Complutense de Madrid
e-mail: i.marin@fdi.ucm.es / ismael.marin@ige-project.eu

Initiative for Globus in Europe
Contents

1. Installation
2. Configuration
3. Logging
4. Scheduling Policies
Installation

• Available accounts: ige_user001 – ige_user030

• Log in (SSH) the host: gridway.fdi.ucm.es
  • This host provides the GW instance of the IGE testbed (multi-user)
  • You can install and configure your own GridWay instance (single-user installation)

• Each account has a certificate for authentication within the IGE testbed

• Now you are ready!
Installation

Required Software for GridWay Core

- C Compiler
- JDK for building the DRMAA Java binding
- GNU Make

- Globus GSI Proxy Utility Programs
- Sudo command
  - Only required for multiple-user mode
- Berkeley Database Library
  - Only required to compile the accounting module
Installation

Installation Procedure

- **User** or **multi-user** installation?
- If multi-user installation, then create the *gwusers* group and the *gwadmin* user
- Download and uncompress *gridway-5.12.0.tar.gz*
- Change to the directory containing the GridWay distribution
- Set environment variables `$GW_LOCATION` and `$GLOBUS_LOCATION`
- Run `./configure --prefix=$GW_LOCATION`
  - There are more options -- check them out in the manual
- Run `'make'` and `'make install'`
- If multi-user installation, then some privileges are required to be set in the `/etc/sudoers` file
- Build and install the needed drivers
Installation

Required Software for GridWay Drivers

• C, C++ and Java Compilers, and GNU Make

• For Static (IM), GRAM5 (EM) and GridFTP (TM) drivers, install the software:
  – **Building**: Globus Common Library Development Files, GRAM Client Library, GRAM Client Library Development Files, Globus Gass Copy, GridFTP Client Library Development Files, Globus Gass Copy Development Files, GPT and Globus Common Library Programs
  – **Installation**: GridWay Core, GRAM Client Library, Globus Gass Copy, Globus Gass Copy Programs and Globus GSI Proxy Utility Programs

• For BDII (IM), CREAM (EM) and Dummy (TM) drivers, install the software:
  – **Building**: Development Files for the Client of the CREAM service
  – **Installation**: GridWay Core, Globus Gass Server_ez Programs, C/C++ Libraries for the Client of the CREAM service and the Plugin for gSoap to use glite-security-gss as the communication layer

• For Static (IM), BES (EM) and Dummy (TM) drivers, install the software:
  – **Building**: JDK, AXIS, Xalan, XMLBeans and the GridSAM schema
  – **Installation**: GridWay Core, JRE, AXIS, WSDL4J, Log4j, Xalan, XMLBeans and the GridSAM schema
Installation

Installation Procedure

- Change to src/{im_mad,em_mad,tm_mad}/{driver}
- Run 'make' and 'make install'
- Configure GridWay through the file $GW_LOCATION/etc/gwd.conf (see next point)
- Configure the available hosts in case of using the Static Files IM driver (see next point)
- If multi-user installation, then some privileges are required to be set in the /etc/sudoers file
- Launch the GridWay daemon from $GW_LOCATION/bin/gwd.
- Test it and enjoy!
Installation

RPM/DEB Installation Procedure

- Set up the IGE or UMD repository
  - http://repository.egi.eu/category/umd_releases/distribution/umd-2/

- Run 'yum install gridway-core' or 'apt-get install gridway-core'

- Install the needed drivers
  - GT5: Run 'yum install gridway-GT5' or 'apt-get install gridway-gt5'
  - gLite 3.2: Run 'yum install gridway-gLite'
  - BES: Run 'yum install gridway-BES' or 'apt-get install gridway-bes'

- Configure GridWay through the file /usr/etc/gwd.conf (see next point)

- Configure the available hosts in case of installing gridway-GT5 or gridway-BES (see next point)

- If multi-user installation, then some privileges are required to be set in the /etc/sudoers file

- Start GridWay:
  - Run '/etc/init.d/gwd start'

- Test it and enjoy!
Contents

1. Installation
2. Configuration
3. Logging
4. Scheduling Policies
Configuration

Overview

- `$GW_LOCATION/etc/gwd.conf`
  - Configuration options for GridWay daemon (GWD)

- `$GW_LOCATION/etc/sched.conf`
  - Configuration options for GridWay built-in scheduling policies

- `$GW_LOCATION/etc/job_template.default`
  - Default values for job templates

- `$GW_LOCATION/etc/gwrc`
  - Default environment variables for MADs

<option> = [value]
Connection Options

- **GWD_PORT**: TCP/IP port where GWD will listen for client requests.
  - TCP/IP port being used by GWD can be found at $GW_LOCATION/etc/gwd.port
- **MAX_NUMBER_OF_CLIENTS**: Max number of simultaneous client connections.

Pool Options

- **NUMBER_OF_JOBS**: Max number of jobs handled by GridWay.
- **NUMBER_OF_ARRAYS**: Max number of array-jobs handled by GridWay.
- **NUMBER_OF_HOSTS**: Max number of hosts handled by GridWay.
- **NUMBER_OF_USERS**: Max number of different users.

Intervals

- **SCHEDULING_INTERVAL**: Seconds between two scheduling actions.
- **DISCOVERY_INTERVAL**: Seconds between searches for new hosts on the Grid (Information Manager).
- **MONITORING_INTERVAL**: Seconds between host information updates (Information Manager).
- **POLL_INTERVAL**: Seconds between underlying Grid middleware queries for job state.
GWD Configuration

Middleware Access Driver (MAD) Options

- **IM_MAD**: Information Manager MADs.

  \[
  \text{IM\_MAD} = \text{<mad\_name>:<path\_to\_mad>:[args]:[nice]:<tm\_mad\_name>:<em\_mad\_name>}
  \]

- **TM_MAD**: Transfer Manager MADs.

  \[
  \text{TM\_MAD} = \text{<mad\_name>:<path\_to\_mad>:[args]}
  \]

- **EM_MAD**: Execution Manager MADs.

  \[
  \text{EM\_MAD} = \text{<mad\_name>:<path\_to\_mad>:<args>:<rsl|rsl\_nsh|rsl2>}
  \]

- **MAX_ACTIVE_IM_QUERIES**: Max number (soft limit) of active IM queries
  - Each query spawns one process

**Scheduler Options**

- **DM_SCHED**: Scheduling module.
Contents

1. Installation
2. Configuration
3. Logging
4. Scheduling Policies
Logging

• At $GW_LOCATION/var/
  
  • `gwd.log`: System level log
    – MADs
    – Jobs (coarse-grain)
  
  • `sched.log`: Scheduler log
    – Fit scheduler policies to your organization needs
  
  • `$JOBID/job.log`: Detailed job log information
    – Details of job resource usage and performance
  
  • `acct`: Accounting information
    – `gwacct` accesses the databases (needs Berkeley DB Library version 4.4.20)
  
  • `.lock`: Prevents from running more than one instance of GWD
  
  • `gwd.port`: TCP/IP port listening for client connections
  
  • `globus-gw.log`: Used to encapsulate GridWay in a GRAM service

• GridWay offers the opportunity to write logs in the Syslog format. You can configure GridWay using `--with-syslog` option.
1. Installation

2. Configuration

3. Logging

4. Scheduling Policies
Scheduling Policies

Grid Scheduling = Job + Resource Policies

Resource Policies
- Rank Expressions
- Fixed Priority
- User Usage History
- Failure Rate

Job Policies
- Fixed Priority
- Urgent Jobs
- User Share
- Deadline
- Waiting Time

Pending Jobs

Matching Resources for each job (user)
Scheduling Policies

Jobs

Fixed Priority Policy (FP)

• Assigns fixed priority to each job (00 – 19).

Fair-Share Policy (SH)

• Allows to establish a dispatching ratio among users of a scheduling domain.

Waiting-time Policy (WT)

• Allows to prevent low-priority jobs to starve.

Deadline Policy (DL)

• Job priority will be increased when deadline approaches.
# Scheduling Policies

## Resources

### Fixed Resource Priority Policy (FP)
- Assigns fixed priority to each resource (01 – 99).

### Rank Policy (RA)
- Prioritizes resources suitable for a job from its point of view.

### Usage Policy (UG)
- Reflects behavior of Grid resources based on job execution statistics.

### Failure Rate Policy (FR)
- Exponential linear back-off strategy in case of resource failure.

## Rescheduling Policies

- A better resource is discovered.
- A job has been waiting in the remote queue system more than a given threshold.
- Requirements changed by application.
- Performance degradation is detected.
Thank you for your attention!
Submission, Monitoring and Control of Jobs

HPC SYSADMIN MEETING’12
Barcelona, Spain
October 15-16, 2012

Dr Ismael Marín Carrión
Distributed Systems Architecture Group
Universidad Complutense de Madrid
e-mail: i.marin@fdi.ucm.es / ismael.marin@ige-project.eu

Initiative for Globus in Europe
Contents

1. User Model Overview
2. Job Definition
3. Commands in Detail
4. Usage Scenarios
User Model Overview

A Grid-aware Application Model

- **Input Files**
  - STD input
  - STD error
  - STD output

- **Application**

- **Performance Profile**

- **Output Files**

- **Requirements + Rank**

- **Checkpoint**

- **Job Activity logging**

**Application execution restart**
Files are architecture independent

**Application requirements characterization**
User Model Overview

Life-cycle

- HOLD
- PENDING
- PROLOG
- WRAPPER
- MIGRATE
- EPILOG
- DONE
- PRE
- STOPPED

GridWay

Initiative for Globus in Europe

DSA Group
Contents

1. User Model Overview
2. Job Definition
3. Commands in Detail
4. Usage Scenarios
Job Definition

Job Template

Generic

• NAME = Name of the job.

Execution

• EXECUTABLE = Executable file.
• ARGUMENTS = Arguments for the executable.
• ENVIRONMENT = User defined, comma-separated, environment variables.
• TYPE = “Single”, “multiple” and “mpi” (like GRAM).
• NP = Number of processors in MPI jobs.

I/O Files

• INPUT_FILES = A comma-separated pair of “local remote” filenames.
  – Absolute path, GridFTP URL, File URL, Name
• OUTPUT_FILES = A comma-separated pair of “remote local” filenames.
  – Absolute path, GridFTP URL, Name
Job Definition

Job Template

Standard Streams

- STDOUT_FILE = Standard Output file.
- STDERR_FILE = Standard Error file.

Resource Selection

- REQUIREMENTS = Boolean expression. If true, host will be considered for scheduling.
- RANK = Numerical expression evaluated for each host considered for scheduling.

Advanced Job Definition

Checkpointing
Scheduling
Performance
Fault Tolerance
Advanced Job Execution
Job Definition

Resource Selection

Two variables can be used to define valid resources for a given job.

- **REQUIREMENTS**: Express conditions that BAN resources
  
  ```
  # Only use pbs like jobmanagers
  REQUIREMENTS = LRMS_NAME = "pbs"
  
  # Only hosts in Spain
  REQUIREMENTS = HOSTNAME = "es"
  ```

- **RANK**: Express conditions over the PREFERENCE of resources
  
  ```
  # Use preferably the resources with the highest CPU_MHZ
  RANK = CPU_MHZ
  
  # Use preferably the resources with the highest value
  # obtained from (CPU_MHZ*2)+FREE_MEM_MB
  RANK = (CPU_MHZ * 2) + FREE_MEM_MB
  ```

- **Available Resource Selection Variables:**
  
  - HOSTNAME
  - ARCH
  - OS_NAME
  - OS_VERSION
  - CPU_MODEL
  - CPU_MHZ
  - CPU_FREE
  - CPU_SMP
  - NODECOUNT
  - SIZE_MEM_MB
  - FREE_MEM_MB
  - SIZE_DISK_MB
  - FREE_DISK Mb
  - LRMS_NAME
  - LRMS_TYPE
  - QUEUE_NAME
  - QUEUE_NODECOUNT
  - QUEUE_FREENNODECOUNT
  - QUEUE_MAXTIME
  - QUEUE_MAXCPUTIME
  - QUEUE_MAXRUNNINGJOBS
  - QUEUE_MAXJOBSINQUEUE
  - QUEUE_DISPATCHTYPE
  - QUEUE_PRIORITY
  - QUEUE_STATUS
Job Definition

Variable Substitution

Generics

• Variables can be used 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\}

Valid Variables

• $\{JOB_ID\}$  Job ID.
• $\{ARRAY_ID\}$  Job array ID. -1 if job is not in any.
• $\{TASK_ID\}$  Task ID within job array. -1 if job is not in any.
• $\{ARCH\}$  Architecture of selected execution hosts.
• $\{PARAM\}$  Allows assignment of arbitrary start and increment values for array jobs (e.g. file naming patterns).
• $\{MAX_PARAM\}$  Upper bound for the $\{PARAM\}$ variable.
Job Environment

- Job environment variables can be set with the `ENVIRONMENT` parameter.
- The variables defined in the `ENVIRONMENT` are "sourced" in a bash shell.
  
  - `ENVIRONMENT = VAR = "\`expr \${JOB_ID} + 3`" # will set VAR to JOB_ID + 3

- Available GW Variables:
  - GW_RESTARTED
  - GW_EXECUTABLE
  - 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
Contents

1. User Model Overview
2. Job Definition
3. Commands in Detail
4. Usage Scenarios
Commands in Detail

gwsubmit – submitting jobs

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

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_PARM = 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.
gwps – monitoring jobs

COMMANDS IN DETAIL

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

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.
Commands in Detail

**gwhistory – accessing job history**

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

**OPTIONS**

- `-h` - Prints help.
- `-n` - Do not print the header lines.
- `job_id` - Job identification as provided by gwps.
Commands in Detail

gwhost – monitoring hosts

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

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.
gwkill – signalling jobs

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

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.
Commands in Detail

gwwait – waiting for jobs

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

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).
gwuser – accessing user information

```
gwuser [-h] [-n]
```

**OPTIONS**

- `-h` - Prints help.
- `-n` - Do not print the header.
Commands in Detail

**gwacct – accessing accounting information**

```
$ gwacct [-h] [-n] [<d n | -w n | -m n | -t s>]

<-u user|-r host>
```

**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.
Contents

1. User Model Overview
2. Job Definition
3. Commands in Detail
4. Usage Scenarios
Usage Scenarios

Single Job

- Create your proxy

- Use `gwhost` command to see available resources:

<table>
<thead>
<tr>
<th>HID</th>
<th>PRIO</th>
<th>OS</th>
<th>ARCH</th>
<th>MHZ</th>
<th>%CPU</th>
<th>MEM(F/T)</th>
<th>DISK(F/T)</th>
<th>N(U/F/T)</th>
<th>LRMS</th>
<th>HOSTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Linux2.6.32.27- x86_64</td>
<td>2533</td>
<td>400</td>
<td>2007/2007</td>
<td>71G/71G</td>
<td>0/2/2</td>
<td>jobmanager-sge</td>
<td>gt5-ige.drg.lrz.de</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Linux2.6.18-238 x86_64</td>
<td>1995</td>
<td>600</td>
<td>2007/2007</td>
<td>92G/92G</td>
<td>0/44/44</td>
<td>jobmanager-pbs</td>
<td>udo-gt01.grid.tu-dortmund.de</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Linux2.6.18-194 x86_64</td>
<td>2993</td>
<td>800</td>
<td>2011/2011</td>
<td>367G/367G</td>
<td>0/8/8</td>
<td>jobmanager-fork</td>
<td>ve.nikhef.nl</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Linux2.6.18-194 x86_64</td>
<td>1600</td>
<td>100</td>
<td>1024/1024</td>
<td>40G/40G</td>
<td>0/1/1</td>
<td>jobmanager-fork</td>
<td>gt1.epcc.ed.ac.uk</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Linux2.6.18-194 x86_64</td>
<td>2328</td>
<td>400</td>
<td>2009/2009</td>
<td>4250G/4250G</td>
<td>0/4/4</td>
<td>jobmanager-fork</td>
<td>gt01.ige.pscn.pl</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Linux2.6.32-36 x86_64</td>
<td>2000</td>
<td>400</td>
<td>2009/2009</td>
<td>8683G/8683G</td>
<td>0/2/2</td>
<td>bes-pbs</td>
<td>147.96.25.29</td>
<td></td>
</tr>
</tbody>
</table>

- and get more detailed information specifying a Host ID:

<table>
<thead>
<tr>
<th>HID</th>
<th>PRIO</th>
<th>OS</th>
<th>ARCH</th>
<th>MHZ</th>
<th>%CPU</th>
<th>MEM(F/T)</th>
<th>DISK(F/T)</th>
<th>N(U/F/T)</th>
<th>LRMS</th>
<th>HOSTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Linux2.6.32.27- x86_64</td>
<td>2533</td>
<td>400</td>
<td>2007/2007</td>
<td>71G/71G</td>
<td>0/2/2</td>
<td>jobmanager-sge</td>
<td>gt5-ige.drg.lrz.de</td>
<td></td>
</tr>
</tbody>
</table>

- Describe the job. Create a job template:

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

- and save it as jt in directory example.

- Use `gwsubmit` command to submit the job:

```
$ gwsubmit -t examples/jt
```
Usage Scenarios

Single Job

• Check the resources that match job requirements with `gwhost -m <job_id>`:

```
$ gwhost -m 0

<table>
<thead>
<tr>
<th>HID</th>
<th>QNAME</th>
<th>RANK</th>
<th>PRI0</th>
<th>SLOTS</th>
<th>HOSTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>all.q</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>gt5-ige.drg.lrz.de</td>
</tr>
<tr>
<td>1</td>
<td>dgiseq</td>
<td>0</td>
<td>1</td>
<td>44</td>
<td>udo-gt01.grid.tu-dortmund.de</td>
</tr>
<tr>
<td>1</td>
<td>dgipar</td>
<td>0</td>
<td>1</td>
<td>44</td>
<td>udo-gt01.grid.tu-dortmund.de</td>
</tr>
<tr>
<td>2</td>
<td>default</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>ve.nikhef.nl</td>
</tr>
<tr>
<td>3</td>
<td>default</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>gt1.epcc.ed.ac.uk</td>
</tr>
<tr>
<td>4</td>
<td>globus</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>gt01.ige.psnc.pl</td>
</tr>
<tr>
<td>5</td>
<td>batch</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>gt-ige.utcluj.ro</td>
</tr>
<tr>
<td>6</td>
<td>pbs</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>147.96.25.29</td>
</tr>
</tbody>
</table>
```

• Follow the evolution of the job with `gwps` command:

```
$ gwps

<table>
<thead>
<tr>
<th>JID</th>
<th>DM</th>
<th>EM</th>
<th>START</th>
<th>END</th>
<th>EXEC</th>
<th>XFER</th>
<th>EXIT</th>
<th>NAME</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>imarin:0</td>
<td>gt5-ige.drg.lrz.de/jobman</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>imarin:0</td>
<td>udo-gt01.grid.tu-dortmund</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>imarin:0</td>
<td>gt5-ige.drg.lrz.de/jobman</td>
</tr>
</tbody>
</table>
```

• HINT: Use `gwps -c <seconds>` for continuous output.
Usage Scenarios

Single Job

• See the job history with `gwhistory` command:

```
$ gwhistory 0
HID  START     END     PROLOG  WRAPPER  EPILOG  MIGR  REASON  QUEUE   HOST
   0  10:38:35 10:39:04 0:00:03 0:00:21 0:00:05 0:00:00 ----   all.q  gt5-ige.drg.lrz.de/jobmanager-sge
```

• Once finished... time to retrieve the results:

```
$ ls -l example/
total 8
-rw-r--r-- 1 imarin imarin 21 2011-05-04 16:47 jt
-rw-r--r-- 1 imarin imarin 0 2011-05-05 10:39 stderr.0
-rw-r--r-- 1 imarin imarin 72 2011-05-05 10:39 stdout.0

$ vi example/stdout.0

job.env
stderr.execution
stderr.wrapper
stdout.execution
stdout.wrapper
```
Usage Scenarios

Array Jobs

- Defining the problem - calculation of the $\pi$ Number:
Usage Scenarios

Array Jobs

• pi.c calculates each slice:

```c
#include <string.h>
#include <stdlib.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;
}
```

$ gcc -O3 pi.c -o pi

• pi arguments:
  • Task ID
  • Total tasks
  • Integral intervals
Usage Scenarios

Array Jobs

• Create a job template (pi.jt):

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

• Submit the array of jobs:

```plaintext
$ gwsubmit -v -t pi.jt -n 4
ARRAY ID: 0

TASK JOB
0  3
1  4
2  5
3  6
```

• Use the `gwwait` command to wait for the jobs:

```plaintext
$ gwwait -v -A 0
0 : 0
1 : 0
2 : 0
3 : 0
```
Usage Scenarios

Array Jobs

• At the end we have the following STDOUT files:

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

• Sum the contained values to get the value of π:

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

• IDEA: Embedding all in script? Check the examples directory …
Usage Scenarios

MPI Jobs

• With fine-grain parallelism apps (allow low latency communication)

• Again, we are going to use the π example
  – All the files needed can be found in $GW_LOCATION/examples/mpi

• Assuming an MPI aware pi.c, we use mpicc to compile it:

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

• Now we create a Job Template (mpi.jt)

  ```
  EXECUTABLE   = mpi
  STDOUT_FILE  = stdout.$(JOB_ID)
  STDERR_FILE  = stderr.$(JOB_ID)
  RANK          = CPU_MHZ
  TYPE          = "mpi"
  NP            = 2
  ```

• and then we submit it to GridWay as any other job
Usage Scenarios

Workflow Jobs

• GridWay can handle 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 specified by using the `-d` option of the `gwsubmit` command

```bash
• $ gwsubmit -v -t A.jt
  JOB ID: 5

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

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

• $ gwsubmit -t D.jt -d "6 7"
```

Hands on! Example 4
Thank you for your attention!
Programming with the DRMAA OGF Standard

HPC SYSADMIN MEETING’12
Barcelona, Spain
October 15-16, 2012

Dr Ismael Marín Carrión
Distributed Systems Architecture Group
Universidad Complutense de Madrid
e-mail: i.marin@fdi.ucm.es / ismael.marin@ige-project.eu

Initiative for Globus in Europe
Contents

1. Introduction
2. Program Structure and Compilation
3. DRMAA Functions
   – Session Handling
   – Job Template
   – Job Submission
   – Job Synchronize and Wait
   – Job Status and Control
   – Helper and Auxiliary Functions
4. DRMAA Java Binding
Introduction

What is DRMAA?

- Distributed Resource Management Application API
  - http://www.drmaa.org/
- Open Grid Forum Standard
- Homogeneous interface to different Distributed Resource Managers (DRM):
  - SGE
  - Condor
  - PBS/Torque
  - GridWay
    - C
    - JAVA
    - Perl
    - Ruby
    - Python

- GridWay now provides a C binding for the new OGF's DRMAA2 specification
Introduction

Application Profiles

**Embarrassingly Distributed**

```c
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);
```

**Master-Worker**

```c
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);
```
Contents

1. Introduction
2. Program Structure and Compilation
3. DRMAA Functions
   - Session Handling
   - Job Template
   - Job Submission
   - Job Synchronize and Wait
   - Job Status and Control
   - Helper and Auxiliary Functions
4. DRMAA Java Binding
Program Structure and Compilation

- Include the DRMAA library:
  
  ```
  #include "drmaa.h"
  ```

- Verify the following environment variable (.bashrc):
  
  ```
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GW_LOCATION/lib/
  ```

- Include the compiling and linking options for DRMAA:
  
  ```
  -L $GW_LOCATION/lib
  -I $G_LOCATION/include
  -ldrmaa
  ```

- Example:
  
  ```
  gcc example.c -L $GW_LOCATION/lib \ 
  -I $GW_LOCATION/include -ldrmaa -o example
  ```
Contents

1. Introduction
2. Program Structure and Compilation
3. DRMAA Functions
   - Session Handling
   - Job Template
   - Job Submission
   - Job Synchronize and Wait
   - Job Status and Control
   - Helper and Auxiliary Functions
4. DRMAA Java Binding
DRMAA Functions

DRMAA Sessions: Initialization

```c
int drmaa_init (const char *contact, char *error_diagnosis, size_t error_diag_len)
```

• **Parameters:**
  - contact: Must be NULL for GridWay
  - error_diagnosis: Buffer where error will be stored
  - error_diag_len: Error buffer size

• **Returns:**
  - DRMAA_ERRNO_SUCCESS
  - DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
  - DRMAA_ERRNO_INVALID_CONTACT_STRING
  - DRMAA_ERRNO_ALREADY_ACTIVE_SESSION
**DRMAA Functions**

**DRMAA Sessions: Finalization**

```c
int drmaa_exit (char *error_diagnosis, size_t error_diag_len)
```

- **Parameters:**
  - `error_diagnosis`: Buffer where error will be stored
  - `error_diag_len`: Error buffer size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`

---

**Auxiliary Functions: Getting Session Information**

```c
int drmaa_get_contact (char *contact, size_t contact_len, char *error_diagnosis, size_t error_diag_len);
int drmaa_version (unsigned int *major, unsigned int *minor, char *error_diagnosis, size_t error_diag_len);
int drmaa_get_DRM_system (char *drm_system, size_t drm_system_len, char *error_diagnosis, size_t error_diag_len);
int drmaa_get_DRMAA_implementation (char *drmaa_impl, size_t drmaa_impl_len, char *error_diagnosis, size_t error_diag_len);
```
Job Template: Assignation and deletion

int drmaa_allocate_job_template (drmaa_job_template_t **jt, char *error_diagnosis, size_t error_diag_len)

int drmaa_delete_job_template (drmaa_job_template_t **jt, char *error_diagnosis, size_t error_diag_len)

• **Parameters:**
  • jt: Pointer to Job Template
  • error_diagnosis: Buffer where error will be stored
  • error_diag_len: Error buffer size

• **Returns:**
  • DRMAA_ERRNO_SUCCESS
  • DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
  • DRMAA_ERRNO_INVALID_CONTACT_STRING
  • DRMAA_ERRNO_ALREADY_ACTIVE_SESSION
DRMAA Functions

Job Template: Set scalar / vector attribute

```c
int drmaa_set_attribute (drmaa_job_template_t *jt, const char *name, 
const char *value, char *error_diagnosis, size_t error_diag_len)
```

```c
int drmaa_set_vector_attribute (drmaa_job_template_t *jt, const char *name, 
const char *value[], char *error_diagnosis, size_t error_diag_len)
```

- **Parameters:**
  - `jt`: Pointer to Job Template
  - `name`: Attribute name
  - `value`: Attribute value
  - `error_diagnosis`: Buffer where error will be stored
  - `error_diag_len`: Error buffer size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_INVALID_ARGUMENT`
  - `DRMAA_ERRNO_NO_MEMORY`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`

- Similar function for **getting scalar** and **vector attributes**
DRMAA Functions

Job Submission: Simple job submission

```c
int drmaa_run_job (char *job_id, size_t job_id_len,
                   drmaa_job_template_t *jt, char *error_diagnosis, size_t error_diag_len)
```

**Parameters:**
- `jobid`: Job ID assigned by GridWay
- `job_id_len`: Job ID buffer size
- `jt`: Pointer to Job Template
- `error_diagnosis`: Buffer where error will be stored
- `error_diag_len`: Error buffer size

**Returns:**
- `DRMAA_ERRNO_SUCCESS`
- `DRMAA_ERRNO_INTERNAL_ERROR`
- `DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE`
- `DRMAA_ERRNO_TRY_LATER`
- `DRMAA_ERRNO_NO_ACTIVE_SESSION`
DRMAA Functions

Job Synchronize and Wait: Wait for job execution

```c
int drmaa_wait (const char *job_id, char *job_id_out, size_t job_id_out_len, int *stat, signed long timeout,
                 drmaa_attr_values_t **rusage, char *error_diagnosis, size_t error_diag_len)
```

• **Parameters:**
  - jobid: Job ID assigned by GridWay
  - job_id_out: Done Job ID
  - job_id_out_len: job_id_out buffer size
  - stat: job_id_out exit code
  - timeout: DRMAA_TIMEOUT_WAIT_FOREVER, DRMAA_TIMEOUT_NO_WAIT or n_seconds
  - rusage: Where remote resource usage values will be stored
  - error_diagnosis: Buffer where error will be stored
  - error_diag_len: Error buffer size

• **Returns:**
  - DRMAA_ERRNO_SUCCESS
  - DRMAA_ERRNO_INVALID_ARGUMENT
  - DRMAA_ERRNO_INVALID_JOB
  - DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
  - DRMAA_ERRNO_NO_RUSAGE
DRMAA Functions

Auxiliary Functions (for interpreting wait status code): Get exit code

```c
int drmaa_wexitstatus (int *exit_status, int stat, char *error_diagnosis, size_t error_diag_len)
```

- **Parameters:**
  - `exit_status`: Where the exit code will be stored
  - `stat`: Status code of a done job
  - `error_diagnosis`: Buffer where error will be stored
  - `error_diag_len`: Error buffer size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`

- Similar functions for checking if the job was signaled:
  - `drmaa_wifsignaled()`
  - The signal it received:
    - `drmaa_wtermsig()`
**DRMAA Functions**

**Helper Functions (for string lists)**

| int drmaa_get_next_attr_value (drmaa_attr_values_t *values, char *value, size_t value_len) |

- **Parameters:**
  - values: Value list
  - value: Actual value name
  - value_len: Actual value size

- **Returns:**
  - DRMAA_ERRNO_SUCCESS
  - DRMAA_INVALID_ARGUMENT
  - DRMAA_ERRNO_NO_ACTIVE_SESSION
  - DRMAA_ERRNO_NO_MORE_ELEMENTS

- **NOTE:** Values must be released with `drmaa_release_attr_values()`

---

**Hands on!**

**Howto2:**

Single Job Submission
DRMAA Functions

Job Status and Control: Get job status

```c
int drmaa_job_ps (const char *job_id, int *remote_ps, char *error_diagnosis, size_t error_diag_len)
```

- **Parameters:**
  - `job_id`: Job ID
  - `remote_ps`: Actual job status
  - `error_diagnosis`: Buffer where error will be stored
  - `error_diag_len`: Error buffer size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_INTERNAL_ERROR`
  - `DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE`
  - `DRMAA_ERRNO_INVALID_JOB`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`

- **Remote_ps values:**
  - `DRMAA_PS_QUEUED_ACTIVE`
  - `DRMAA_PS_RUNNING`
  - `DRMAA_PS_USER_ON_HOLD`
  - `DRMAA_PS_DONE`
  - `DRMAA_PS_FAILED`
  - `DRMAA_PS_UNDETERMINED`

- **Translation of remote_ps:**
  - `const char * drmaa_gw_status (int drmaa_state)`
DRMAA Functions

Job Status and Control: Job control

```c
int drmaa_control (const char *jobid, int action, char *error_diagnosis, size_t error_diag_len)
```

- **Parameters:**
  - `job_id`: Job ID
  - `action`: Control signal
  - `error_diagnosis`: Buffer where error will be stored
  - `error_diag_len`: Error buffer size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_INTERNAL_ERROR`
  - `DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`
  - `DRMAA_ERRNO_INVALID_ARGUMENT`
  - `DRMAA_ERRNO_INVALID_JOB`
  - `DRMAA_ERRNO_HOLD_INCONSISTENT_STATE`
  - `DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE`
  - `DRMAA_ERRNO_RESUME_INCONSISTENT_STATE`
  - `DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE`

- **Actions:**
  - `DRMAA_CONTROL_SUSPEND`
  - `DRMAA_CONTROL_RESUME`
  - `DRMAA_CONTROL_TERMINATE`
  - `DRMAA_CONTROL_HOLD`
  - `DRMAA_CONTROL_RELEASE`
DRMAA Functions

Job Synchronize and Wait: Synchronize jobs

int drmaa_synchronize (const char *job_ids[], signed long timeout, int dispose, char *error_diagnosis, size_t error_diag_len)

• Parameters:
  • job_ids[]: Job ID list (ends with NULL)
  • timeout: Max waiting time
  • dispose: Kill (1) or not (0) the job
  • error_diagnosis: Buffer where error will be stored
  • error_diag_len: Error buffer size

• Returns:
  • DRMAA_ERRNO_SUCCESS
  • DRMAA_ERRNO_INVALID_ARGUMENT
  • DRMAA_ERRNO_INVALID_JOB
  • DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
  • DRMAA_ERRNO_NO_ACTIVE_SESSION

Hands on! Howto3: Job Status and Control
DRMAA Functions

Job Submission: Submit job array

```c
int drmaa_run_bulk_jobs (drmaa_job_ids_t **jobids, drmaa_job_template_t *jt, int start,
   int end, int incr, char *error_diagnosis, size_t error_diag_len)
```

**Parameters:**
- `jobids`: Vector where Job IDs will be stored
- `jt`: Pointer to Job Template
- `start`: First job index
- `end`: Last job index
- `incr`: Increment used for obtaining job total number (GridWay uses 1)
- `error_diagnosis`: Buffer where error will be stored
- `error_diag_len`: Error buffer size

**Returns:**
- `DRMAA_ERRNO_SUCCESS`
- `DRMAA_ERRNO_INTERNAL_ERROR`
- `DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE`
- `DRMAA_ERRNO_TRY_LATER`
- `DRMAA_ERRNO_NO_ACTIVE_SESSION`
- `DRMAA_ERRNO_NO_MEMORY`
DRMAA Functions

Helper Functions (for string lists)

```c
int drmaa_get_next_job_id (drmaa_job_ids_t *values, char *value, size_t value_len)
```

- **Parameters:**
  - `values`: Job ID list
  - `value`: Actual Job ID
  - `value_len`: Actual Job ID size

- **Returns:**
  - `DRMAA_ERRNO_SUCCESS`
  - `DRMAA_ERRNO_INVALID_ARGUMENT`
  - `DRMAA_ERRNO_NO_ACTIVE_SESSION`
  - `DRMAA_ERRNO_NO_MORE_ELEMENTS`

- **NOTE:** Job ID lists must be released with `drmaa_release_job_ids()`
Contents

1. Introduction
2. Program Structure and Compilation
3. DRMAA Functions
   - Session Handling
   - Job Template
   - Job Submission
   - Job Synchronize and Wait
   - Job Status and Control
   - Helper and Auxiliary Functions
4. DRMAA Java Binding
DRMAA Java Binding

- Import the GridWay DRMAA Java package:

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

- Include the following option to the javac:

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

- Java code example (Howto1):

```java
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();
        }
    }
}
```
Thank you for your attention!