Advanced Job Submission on the Grid

Antun Balaz
Scientific Computing Laboratory
Institute of Physics Belgrade
http://www.scl.rs/

30 Nov – 11 Dec 2009
www.eu-egee.org

FP7-INFRA-222667 Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
• Again a little bit on WMS

• How are your jobs handled

• JDL attributes

• Which types of jobs exist

• Examples of complex JDLs
Why does the Workload Management System exist?

- **Grids have**
  - Many users
  - Many jobs – a “job” = an executable you want to run
  - Where many compute nodes are available
  - Workload Management System is a software service that makes running jobs easier for the user

- **It builds on the basic grid services**
  - E.g. Authorisation, Authentication, Security, Information Systems, Job submission

- **Terminology**: “Compute element”: defined as a batch queue - One cluster can have many queues
What WMS does?

- **WMS manages jobs on users’ behalf**
  - User doesn’t decide where jobs are run
  - User defines the job and its requirements, WMS matches this with available CEs

- **Effect:**
  - Easier submission
  - Users insulated from change in Compute elements
  - WMS – can optimise your jobs – e.g. which CE?
Managing jobs with gLite command line tools

User describes job in text file using Job Description Language
Submits job to WMS using (usually) the command-line interface
Using WMS

- Jobs run in batch mode on grids.
- Steps in running a job on a gLite grid with WMS:

1. Create a text file in “Job Description Language”
2. Optional check: list the compute elements that match your requirements (“list match” command)
3. Submit the job ~ “glite-wms-job-submit –a myfile.jdl”
   Non-blocking - Each job is given an id.
4. Occasionally check the status of your job
5. When “Done” retrieve output
Simple example of a JDL file

```plaintext
Type = "Job";
Executable = "/bin/hostname";
Arguments = "";
StdError = "stderr.txt";
StdOutput = "stdout.txt";
InputSandbox = "";
OutputSandbox = {"stderr.txt", "stdout.txt"};

$ glite-wms-job-submit -a my.jdl

Returns a “job-id” used to monitor job, retrieve output...
```
Standard JDL-file attributes

- **Type** – “Job” for sequential jobs; later more details
- **Executable** – sets the name of the executable file;
- **Arguments** – command line arguments of the program;
- **StdOutput, StdError** - files for storing the standard output and error messages output;
- **InputSandbox** – set of input files needed by the program, including the executable;
- **OutputSandbox** – set of output files which will be written during the execution, including standard output and standard error output; these are sent from the CE to the WMS for you to retrieve
- **ShallowRetryCount** – in case of grid error, retry job this many times (“Shallow”: before job is running)
A worker node is allocated by the local jobmanager.

- STD input stream is read from file
- STD out and err. streams are redirected into files

Logging and bookkeeping

Job status update

Job Submit Event

Submit

Input “sandbox”

Get output

Output “sandbox”

publish state

/stderr.txt
/stdout.txt
/bin/hostname

A worker node is allocated by the local jobmanager.
The “Executable”

- **Script:**
  - No compilation is necessary
  - Can invoke binary that is statically installed on the CE

- **Binary:**
  - Must be compiled on the User Interface ➔ binary compatibility with CEs is guaranteed
  - Statically linked ➔ to avoid errors caused by library versions

- **Coming from client side**
  - Part of InputSandbox

- **Installed on the CE**
  - Standard software in Linux
  - VO specific software: advertised in information system

- Use JDL to navigate job to such a site
<table>
<thead>
<tr>
<th>WMS version</th>
<th>LCG-2 WMS</th>
<th>gLite WMS via NS</th>
<th>gLite WMS via WMProxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delegate proxy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatible resources</td>
<td>edg-job-list-match jdlfile</td>
<td>glite-job-list-match jdlfile</td>
<td>glite-wms-job-list-match [-d delegID] [-a] jdlfile</td>
</tr>
</tbody>
</table>
Type = "Job";
Executable = "/bin/hostname";
Arguments = "";
StdError = "stderr.txt";
StdOutput = "stdout.txt";
InputSandbox = "";
OutputSandbox = {"stderr.txt", "stdout.txt"};

Requirements = other.Architecture=="INTEL" &&
other.GlueCEInfoTotalCPUs > 480;
Rank = other.GlueCEStateTotalJobs;

InputData = "lfn:/grid/gridbox/antun/file.txt";
Basic services of gLite

User Interface
- Submit job
- Retrieve status & output
- Create credential
- Query

Resource Broker
- Submit job
- Query
- Retrieve output
- Logging
- Job status

Information System
- Submit job
- Publish state
- Query

File and Replica Catalog
- Query
- Create credential
- Retrieve status & output

Authorization Service (VOMS)
- Query

Logging and bookkeeping
- Job status

Site X
- Computing Element
- Storage Element
- Process

FP7-INFRA-222667  Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
Location of files

Characteristics of resources

Network Daemon

Workload Manager

Job Contr. - CondorG

WMS

LFC

Inform. Service

Computing Element

Storage Element

CE characters & status

SE characters & status

Characteristics of resources

Location of files

FP7-INFRA-222667 Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
Enabling Grids for E-sciencE

Network Daemon

LFC

Inform. Service

WMS

RB

storage

Workload Manager

CondorG

Storage Element

Computing Element

Daemon responsible for accepting incoming requests

Submitted

Waiting

JDL

Input Sandbox files

RB storage

glite-wms-job-submit myjob.jdl

CE characters & status

SE characters & status

Daemon responsible for accepting incoming requests

Network Daemon

Job Contr.

CondorG

Workload Manager

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting

glite-wms-job-submit myjob.jdl

RB

storage

JDL

Input Sandbox files

Submitted

Waiting
WM: responsible to take the appropriate actions to satisfy the request
Where this job can be executed?

Network Daemon

Match-Maker/Broker

LFC

Inform. Service

RB storage

Computing Element

Storage Element

CE characters & status

SE characters & status

WMS

Job Contr. - CondorG

Workload Manager

advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
Matchmaker: responsible to find the “best” CE where to submit a job.
Where is the needed InputData?

What is the status of the Grid?

CE characters & status

SE characters & status

Waiting

Submitted
JA: responsible for the final “touchs” to the job before performing submission (e.g. creation of wrapper script, etc.)
JC: responsible for the actual job management operations (done via CondorG)
glite-wms-get-output <jobID>
Diagram showing the workflow of a Grid computing system, with steps including:

- Submitted
- Waiting
- Ready
- Scheduled
- Running
- Done
- Cleared

Key components include:
- Network Daemon
- Workload Manager
- RB (Resource Broker) storage
- Job Contr. - CondorG
- LFC (Local File Catalog)
- Inform. Service

Output Sandbox files are directed to the RB storage. The Computing Element and Storage Element are also depicted.
Job monitoring

- glite-wms-job-status <jobID>
- glite-wms-job-logging-info <jobID>

Logging & Bookkeeping

LB: receives and stores job events; processes corresponding job status

Network Daemon

Workload Manager

Job Contr. - CondorG

WMS

Computing Element

Log of job events

FP7-INFRA-222667  Advanced School in High Performance and GRID Computing – Concepts and Applications, ICTP, Trieste, Italy
1. Meet CE requirements (defined by Requirements part of JDL)
2. Select CE which is close to InputData
   - “Close” relationship is defined between CEs and SEs by site administrators
   - “Close” is not necessarily physical distance – rather bandwidth
   - “Close” typically means same site
     - SE: se-3.grid.box
     - CE: ce-1.grid.box:2119/jobmanager-lcgpbs-gridbox
3. Select CE with highest rank
   (rank formula is defined by Rank part of JDL)
Some relevant CE attributes

- **GlueCEUniqueID** – Identifier of a CE
  Eliminating an erroneous CE:
  
  ```
  other.GlueCEUniqueID != 
  "ce-1.grid.box:2119/jobmanager-lcgpbs-gridbox"
  ```

- **GlueCEInfoTotalCPUs** – max number of CPUs at a CE
  ```
  Rank = other. GlueCEInfoTotalCPUs;
  ```

- **GlueCEStateWaitingJobs** – number of waiting jobs

- **GlueCEPolicyMaxCPUTime** – job will be killed after this number of minutes
  ```
  other.GlueCEPolicyMaxCPUTime > 300;
  ```

- **GlueHostMainMemoryRAMSize** – memory size
  ```
  other.GlueHostMainMemoryRAMSize > 1024;
  ```

http://glite.web.cern.ch/glite/documentation/ → JDL specification
• Rank =
  ( other.GlueCEStateWaitingJobs == 0 ?
  other.GlueCEStateFreeCPUs : -other.GlueCEStateWaitingJobs);

if there are no waiting jobs,
  – then the selected CE will be the one with the most free CPUs
  – else the one with the least waiting jobs.

• Requirements =
  ( Member(“IDL2.1”,
    other.GlueHostApplicationSoftwareRunTimeEnvironment) )
  &&
  (other.GlueCEPolicyMaxWallClockTime > 10000);

CE where,
  – IDL2.1 software is available
  – At least 10000s can be spent on the site (waiting + running)
• other.GlueHostMainMemoryRAMSize >= 512 * (other.GlueHostArchitectureSMPSize > 0 ? other.GlueHostArchitectureSMPSize : 1)

At least 512 MB of RAM memory per CPU core should be available

• other.GlueHostArchitecturePlatformType == "x86_64"

x86_64 arch requested
### Job states

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBMITTED</td>
<td>submission logged in the Logging &amp; Bookkeeping service</td>
</tr>
<tr>
<td>WAIT</td>
<td>job match making for resources</td>
</tr>
<tr>
<td>READY</td>
<td>job being sent to executing CE</td>
</tr>
<tr>
<td>SCHEDULED</td>
<td>job scheduled in the CE queue manager</td>
</tr>
<tr>
<td>RUNNING</td>
<td>job executing on a Worker Node of the selected CE queue</td>
</tr>
<tr>
<td>DONE</td>
<td>job terminated without grid errors</td>
</tr>
<tr>
<td>CLEARED</td>
<td>job output retrieved</td>
</tr>
<tr>
<td>ABORT</td>
<td>job aborted by middleware, check reason</td>
</tr>
</tbody>
</table>
**WMProxy (1)**

- Client on the UI communicates with the “WM Proxy”
- WMProxy can manage complex jobs
- WMProxy acts on your behalf to access WMS. It needs a “delegated proxy”:
  - Delegation once per session
  - Delegation at every submission

---

**UI (user interface)** has preinstalled client software

**Local Workstation**

**ssh**

**WMProxy**

**WMS**

**CEs**

**Workload Management System**

**UI (user interface)**
• **WMProxy is a SOAP Web service providing access to the Workload Management System (WMS)**

• **Job characteristics specified via JDL**
  - jobRegister
    - create id
    - map to local user and create job dir
    - register to L&B
    - return id to user
  - input files transfer
  - jobStart
    - register sub-jobs to L&B
    - map to local user and create sub-job dir’s
    - unpack sub-job files
    - deliver jobs to WM
Relevant JDL attributes 1

- **Type**
  - Job, DAG, Collection

- **JobType (only when the Type is set to Job!)**
  - Normal (sequential batch job), Parametric, Interactive, MPICH, Checkpointable, Partitionable

- **Executable**
  - The name of the executable (absolute path)

- **Arguments**
  - Job command line arguments

- **StdInput, StdOutput, StdError**
  - Standard input/output/error of the job
    (stdin absolute path; stdout & stderr relative path)

- **Environment**
  - List of environment variables to be set for the binary

- **InputSandbox**
  - List of files on the UI local disk needed by the job for running
    - The listed files will be staged to the remote resource

- **OutputSandbox**
  - List of files, generated by the job, which have to be retrieved
    - Files will be transferred back
Relevant JDL attributes 2

- **Input Data**
  - For the broker, WMS does not transfer these files

- **Output Data**
  - For the broker, WMS does not transfer these files

- **Requirements**
  - Required CE characteristics

- **Rank**
  - “Goodness” value for compatible CEs

- **ShallowRetryCount**
  - in case of grid error, retry job this many times
  - “Shallow”: before job is running

- **RetryCount**
  - resubmit if the job failed in Running mode
  - If job fails after it has already done something (e.g. creating a Grid file) then resubmission can generate inconsistencies

- **MyProxyServer**
  - where to download proxy from in case of the existing proxy expires
  - Done by WMS
Complex jobs 1: Job collection

- A set of independent jobs
- For some reason must be managed as a single unit
- Possible reasons:
  - Belong to the same experiment
  - Share common input files
  - Optimize network traffic
- Sharing of sandboxes
Type = "Collection";

InputSandbox = {
    "sharedFile1"; . . .; "sharedFileM" 
};

nodes = {
    [ JobType = "Normal";
        InputSandbox = {root.InputSandbox, . . .}
        ... ];
    . . .

    [ JobType = "Normal";
        ... ];
    . . .
};
• **Direct Acyclic Graph (DAG)** is a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs

• Sharing and inheritance of sandboxes
  - **Include OutputSandbox in the next InputSandbox**

• Dependencies defined between pairs of jobs
JDL of a DAG

```plaintext
Type = "DAG";  # Transfer from UI only once

InputSandbox = {
    "sharedFile1", ..., "sharedFileN"
};

nodes = [
    job1 = [
        description = [
            JobType = "Normal";
            ...;
        ];
    ];
    ...;
];

dependencies = {
    {job1, {job2, job3}}, {job2, job4}, {job3, job4}
};

Graph structure
```
[ Type = "DAG"; 
...

job4 = [

description = [

  JobType = "Normal";
  InputSandbox = {
    root.nodes.job1.description.OutputSandbox[0],
    root.nodes.job2.description.OutputSandbox,
    ...
  };
  ...
};

];

...]

]
Complex jobs 3: parametric jobs

- A set of jobs generated from one JDL
- Useful where many similar (but not identical) jobs must be executed
  - Parameter study, parametric sweep applications
  - Majority of grid applications are parametric!
- One or more parametric attributes in the JDL:
  - Use the _PARAM_ keyword
  - E.g. InputSandbox = “input(PARAM);”
JDL of a parametric job 1

```
[ 
  Type = "Parametric";
  ...

  ParameterStart = 0;
  ParameterStep = 2;
  Parameters = 6; → _PARAM_: 0, 2, 4, 6, 8, 10

  Arguments = "inputfigure_PARAM_.jpg";
  StdOutput = "transformed_PARAM_.jpg";
  OutputSandbox = {" transformed_PARAM_.jpg ",...};
  ...
]
```
[  
Type = "Parametric";
...

Parameters = {alpha, beta, gama};

Arguments = "inputfigure_PARAM_.jpg";
StdOutput = "transformed_PARAM_.jpg";
OutputSandbox  = {" transformed_PARAM_.jpg ",...};
  ...
]


For simple jobs: `glite-wms-…` is the recommended way to use the WMS.

History:
- Before the `glite-wms-` commands we had `glite-` commands
  - used the WMS without WMProxy
- Before the `glite-` commands we had
  - `edg-` commands (`edg-job-submit….`)
    - *European Data Grid – project before EGEE*
  - Used the “resource broker”
  - Still very widely used
- You might see these commands still in use.

Status
- Complex jobs with WMProxy: first stable version just released. Not yet in routine production use
- Watch for news!
Practicals on advanced job submission

- Create and submit a JDL file with different requirements and rankings
- Create and submit a JDL file for a collection of jobs
- Create and submit a JDL file for a parametric job
- Create and submit a JDL file for a DAG job