Advanced Jobs

EELA-2 GRIS-2

Jérôme Verleyen, IBt - *UNAM (México)*

Queretaro, México, 28th september - 10th October 2009

CAPACITIES

Advanced job types

The Glite Middelware propose differents type of jobs:

- **Parametric**
- **DAG**
- **Collection**
- **Long time Job**

Parametric job

- It's a job whose JDL contains one or more parametric attributes (Arguments, Stdout, etc..)
- The attribute “Parameters” define the range of values for the parametrics attributes.
- It use the key word “_PARAM_ » as value of parameters.
- Several instances of a similar jobs only differing for the value of the parametrized attributes.
- An unique JobId as if the job is unique. Easy to control.

Parametric Job : Parameters

The PARAMETERS attribute could be :

- **The upper bound :**
  - Parameters = 1000
- **The lower bound**
  - Parameters = -100
- **A list of items :**
  - Parameters = { exe , f1 , f2 , f3 }
  - In this case, the value item don't have a type. Should not enclosed between quotes (“).

Parameterstep : size of each variation

Parameterstart : Initial value of the parameters attributes.

Numbers of jobs: (Parameters – ParameterStart) / ParameterStep

Parametric Job example (I)

- **JDL Example :**
  - 10 input files to analyse (md5sums).
  - 10 output files

```
[
    JobType = "Parametric";
    Executable = "/bin/sh";
    Arguments = "md5.sh input_PARAM_.txt";
    InputSandbox = {"md5.sh", "input_PARAM_.txt"};
    StdOutput = "out_PARAM_.txt";
    StdError = "err_PARAM_.txt";
    Parameters = 11;
    ParameterStart = 1;
    ParameterStep = 1;
    OutputSand3box = {"out_PARAM_.txt", "err_PARAM_.txt"};
]
```

Parametric Job example (II)

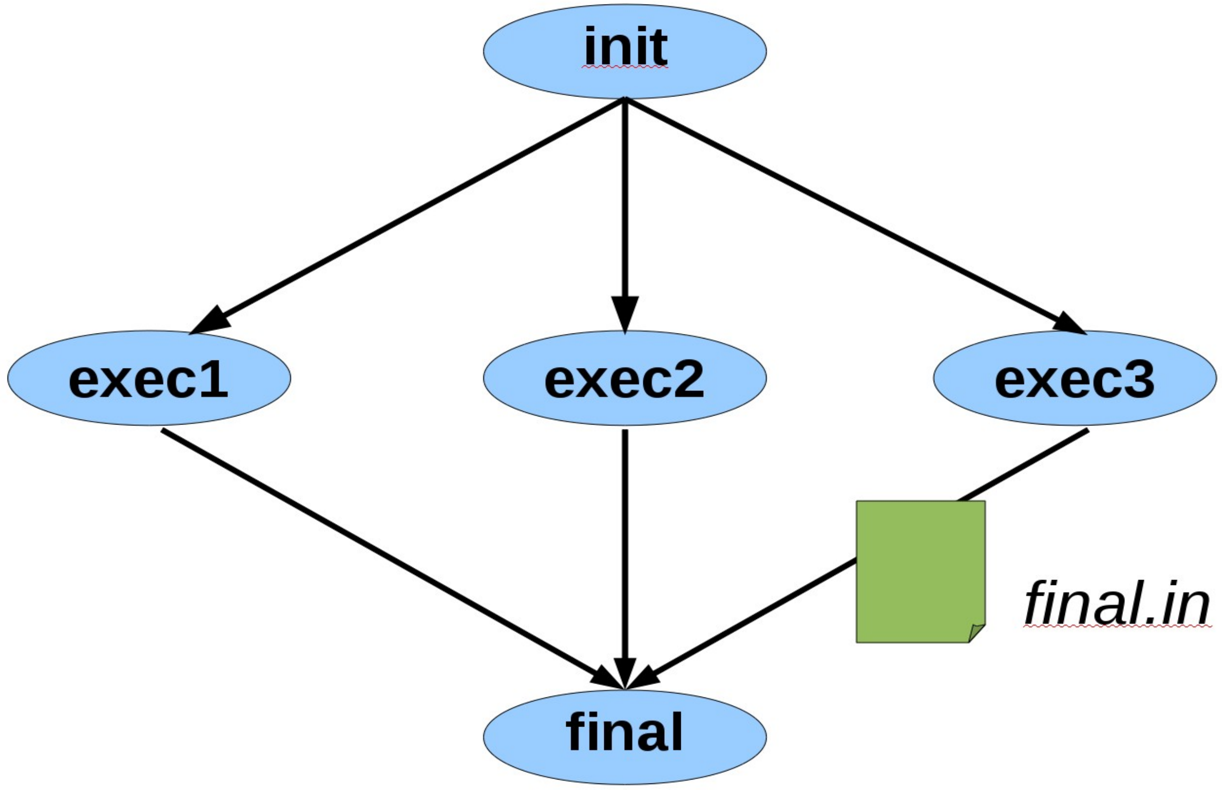
- **Need 10 input files (input_PARAM_.txt):**
  - input1.txt, input2.txt ... input10.txt
- **Obtain 10 output files:**
  - output1.txt output10.txt
- **Obtain 10 errors file:**
  - err1.txt error10.txt

DAG Job

- **A DAG (directed acrylic graph) represents a set of jobs where input, ouput or execution of one or more depends on others jobs.**
- **The dependencies is represented by a graph where nodes a jobs and edges identify the dependencies.**
- **Improvement of the management with:**
  - Shared outputSandbox
  - Common attributes
  - Unique ID to control all of the sub jobs
- **An example should to explain. In this exemple, we show how to « transfert » file between jobs**

DAG Job example

A simple graph to represents our DAG job:

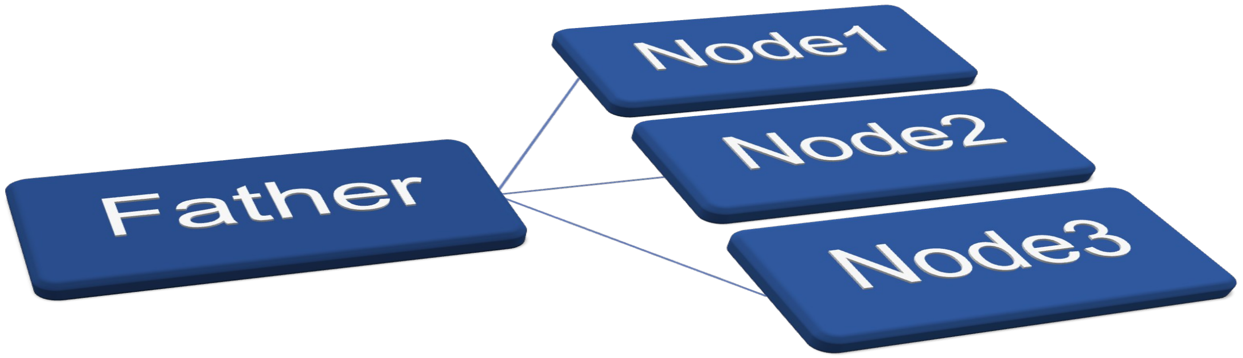


DAG Job Example (II)

```
[ type = "dag";
  InputSandbox = {“init.txt”};
nodes = [
 exec1 = [.........];
.../...
 exec3 = [
  description = [
    JobType = “Normal”;
     .../...
     InputSandbox={root.InputSandbox};
     OutputSandbox={“exec3.out”,”exec.err”,“final.in”} ;
  ];
 ];
final = [
  description = [
     .../...
     InputSandbox={“final-init”,root.nodes.exec3.description.OutputSandbox[2]} ;
  ];
];
dependencies = {
{init , exec1}, {init , exec2}, {init , exec3},
{{exec1, exec2 , exec3} , final}
  };
]
```

Job Collection

- A collection of jobs is a set of independent jobs that have to be monitored and controlled as a single request.
- For example, a same program to use with different inputs (could use the parametric job too).



Job Collection (II)

- The id of the principal node permits to check the state of all the jobs (sub jobs).
- Each sub jobs have an Id, that could be use to control each of one.
- The output of the collection of jobs is get back when the job state is done:
  - One unique command to download the results.
  - Each sub job output go in a sub-directory.

Job Collection (III)

There is two way to submit a collection of jobs:

- **First one: using the "*--collection*" argument:**

```
$ glite-wms-job-submit –a –collection <Directory>
```

Directory : contains only jdl files.

Collective Job (III)

- **Second way: a jdl file describing the collection of job (E2GRIS1 example).**

```
[
  type = “collection”;
  RetryCount = 3;
 Nodes = {
     [
       Executable = "autodock.sh";
       Arguments = "y7vyNzRqAVsBgZ9vDLJx ZINC00000480 ";
       StdOutput = "autodock.out";
       StdError = "autodock.err";
       OutputSandbox = {"autodock.err","autodock.out"};
       InputSandbox = {"autodock.sh","UserData/userinput.tgz"};
     ],
     [
       Executable = "autodock.sh";
       Arguments = "y7vyNzRqAVsBgZ9vDLJx ZINC00000481 ";
       StdOutput = "autodock.out";
       StdError = "autodock.err";
       OutputSandbox = {"autodock.err","autodock.out"};
       InputSandbox = {"autodock.sh","UserData/userinput.tgz"};
     ]
  }
]
```

Long-running job

- **The proxy generated has a 12 hours durability.**
- **Any job running after the expiration of the proxy will be canceled**
- **The solution: use of myproxy server :**
  - Used to store a long-lived certificate of a user.
  - WMS use this server to renew automaticaly the expired proxy of a user.
  - The user's jobs are authorized to continue.

Long-running job (II)

- Unset the GT_PROXY_MODE variable:
  - `$ unset GT_PROXY_MODE`
- Create a long-term proxy on the server:

```
myproxy-init --voms prod.vo.eu-eela.eu -s [server hostname] -d -n
```

  - -s : if you want to use a specific myproxy server (in a well installed UI, this server is defined)
  - -n : don't prompt for passphrase to register on the myproxy Server.
  - -d : use the proxy certificate subject (DN) as the default username
- Create your proxy normaly :

```
voms-proxy-init --voms prod.vo.eu-eela.eu
```

- Check the long-term proxy:

  myproxy-info -d
- Add in your JDL file the reference of the Mproxy server:

```
MyProxyServer = “px.eela.ufrj.br“;
```

- Delegate you proxy and send your job:

```
glite-wms-job-delegate-proxy -d $USER
glite-wms-job-submit -d $USER job.jdl
```

References

- **Quicstart for complex jobs (gilda wiki)**
  - https://grid.ct.infn.it/twiki/bin/view/GILDA/WmProxyUse
- **Specification of the JDL attributes**
  - https://edms.cern.ch/document/590869/1

Questions ...