Practical Work on EGEE – Using gLite

Peter Praxmarer
Markus Baumgartner
CERN openlab
Geneva/Switzerland

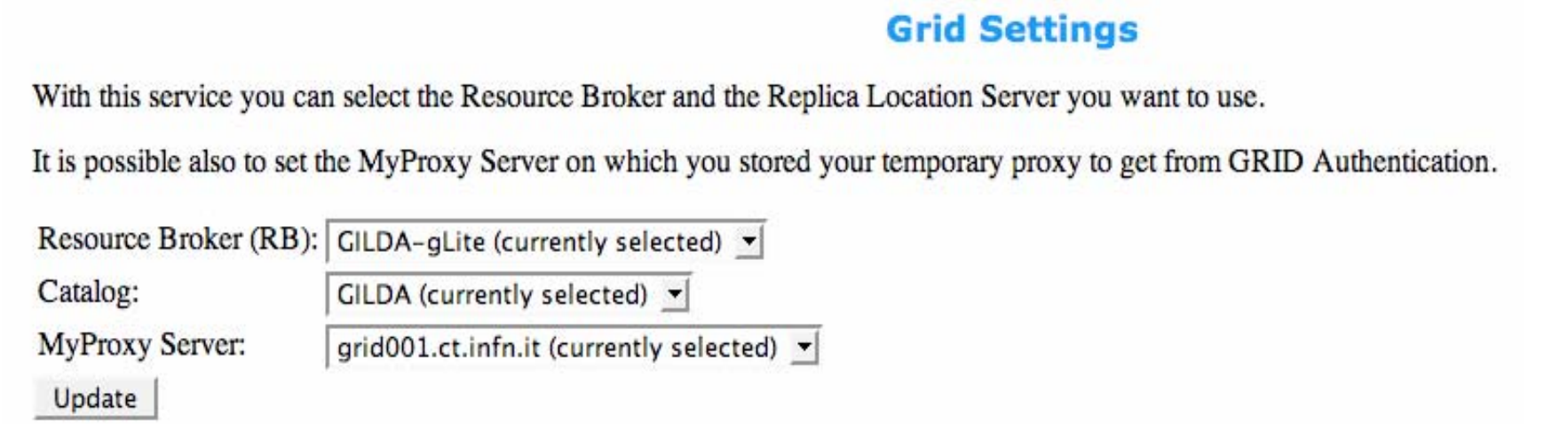
EGEE gLite Tutorial
CERN, 24 August 2005

Authentication on the GENIUS Portal

- **https://glite-tutor.ct.infn.it**
- **Choose 'Grid Settings' and log in to the portal**

- Username: cern01 ... cern50
- Password: GridCER01 ... GridCER50

Grid Settings



- **Resource Broker** knows all resources available and assigns jobs to them
- **Catalog** defines the Replica Catalog used by default
- **MyProxy Server** defines the server storing the myproxy certificate

Grid security (1)

- Each grid user needs valid credentials to use the grid
  - gLite uses X.509 certificates as credentials
  - Certificates are issued by a Certification Authority (CA)
  - X.509 credentials consist of
    - The certificate (public key) stored as ~/.globus/usercert.pem
    - The private key stored as ~/.globus/userkey.pem
    - The private key is protected by a passphrase
- To improve security, proxy certificates are used
    - The private key is NOT protected by a passphrase
    - A proxy certificate has a limited lifetime (default: 12 hours)
    - Proxy certificates are stored together with their private key

Grid security (2)

- In the following exercise you will:
  - Create a proxy certificate

Exercise 1: Grid security (1)

- Initialize your grid proxy certificate
  - Login to glite-tutor.ct.infn.it using ssh (putty)
    > ssh cernXX@glite-tutor.ct.infn.it
    - Substitute XX by your account number (00 to 50)
    - Use the password "GridCERXX"
  - Create a proxy certificate
    > voms-proxy-init
    - Provide the passphrase "**CERN**" to decrypt the user key

If successful the output will be "Your proxy is valid until…"

  - Verify the proxy certificate
    > voms-proxy-info

This will show you subject, issuer, etc. of your local proxy certificate

  - **After the practicals**, delete your temporary proxy
    > voms-proxy-destroy

Exercise 1: Grid security (2)

- Upload your proxy certificate to the MyProxy server:
  - Initialize and upload your MyProxy certificate
    > myproxy-init -s grid001.ct.infn.it
    - Provide the passphrase "**CERN**" to decrypt the user key
    - Choose a passphrase for the proxy on the server

If successful the output will be "A proxy valid for 168 hours…"

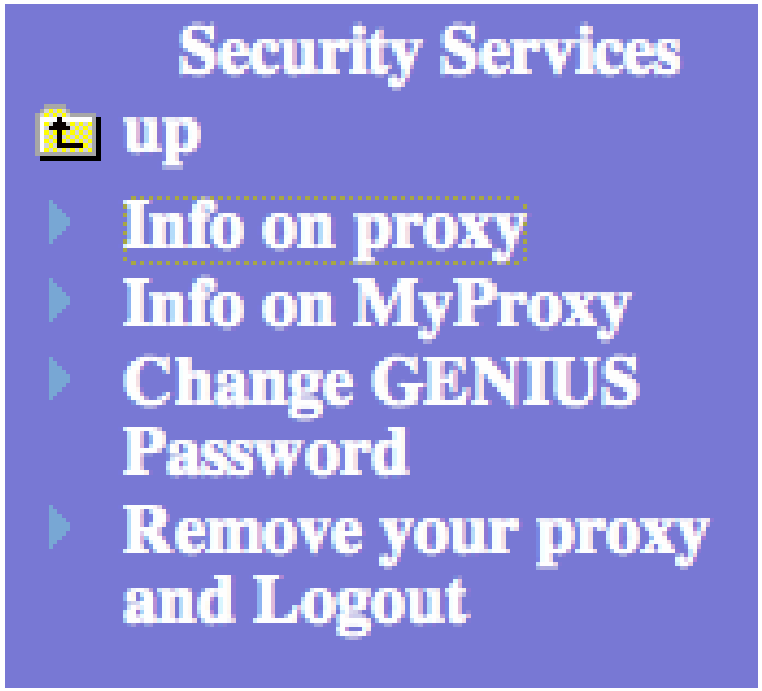
  - Verify the MyProxy certificate on the server
    > myproxy-info -s grid001.ct.infn.it

Shows you details about your proxy on the MyProxy server

  - **After the practicals**, remove your proxy certificate from the server
    > myproxy-destroy –s grid001.ct.infn.it

Grid security (GENIUS)

- **Get information on the currently used proxy certificates**
  - Info on the proxy certificate
  - Info on the certificate stored on the MyProxy server
- **Change the GENIUS login password**
- **Logout from GENIUS and remove the proxy certificate**



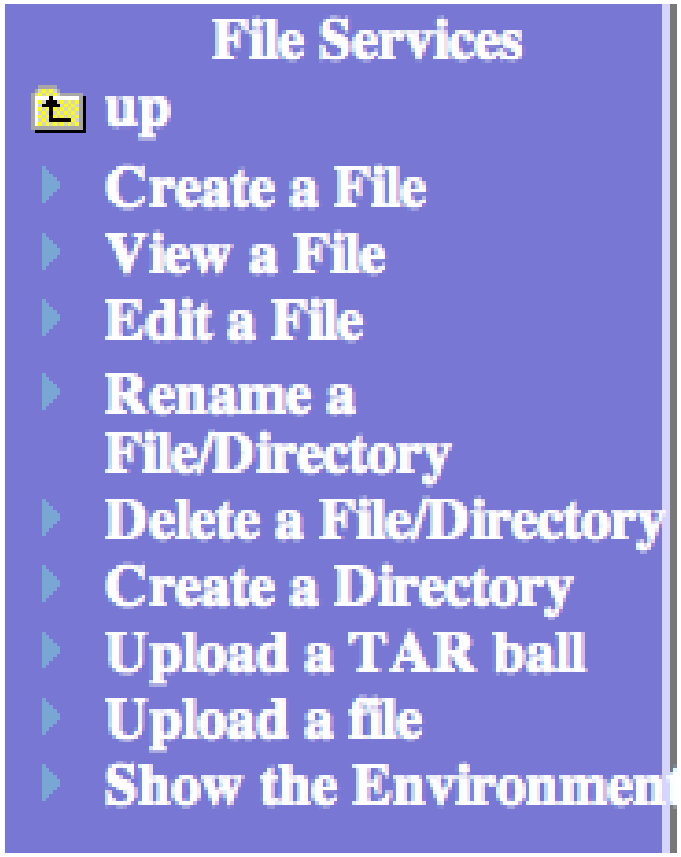
Exercise 2: Security Services

UI glite-tutor.ct.infn.it	Check with the GENIUS WebUI
> grid-proxy-info > myproxy-info -s grid001.ct.infn.it	Click on "Info on proxy" Click on "Info on MyProxy"
> myproxy-destroy -s grid001.ct.infn.it > myproxy-info -s grid001.ct.infn.it	Click on "Info on MyProxy" (Error)
> grid-proxy-destroy > grid-proxy-info	Click on "Info on proxy" (Error)
> myproxy-init -s grid001.ct.infn.it > myproxy-info -s grid001.ct.infn.it	Click on "Info on MyProxy"
> myproxy-get-delegation -s grid001.ct.infn.it	
> grid-proxy-info	Click on "Info on proxy"

Job submission

- A "job" is a program that runs in the grid
- Jobs are described using the "Job Description Language" (JDL)
- The JDL description can include input and output data, executables, requirements (more later)
- In the following exercise you will:
  - Create a program to be run in the grid (shell script)
  - Create a job description using the JDL
  - Submit the job to the grid
  - Inspect the output of the job

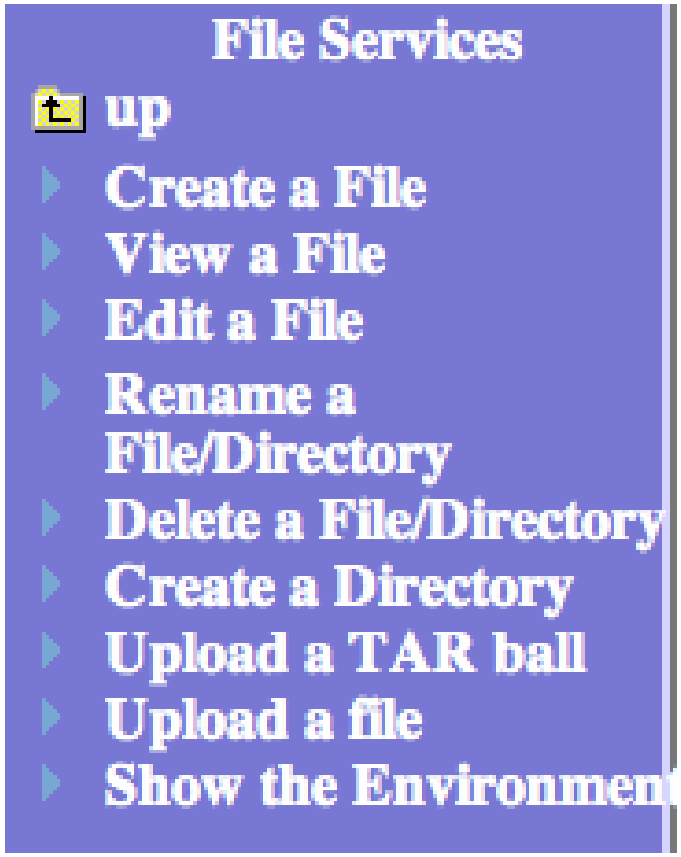
Exercise 3: Job submission (1)



- **Create a file named ‘myhelloworld.sh’. This will be the job to be run in the grid.**

```
#!/bin/sh

MYNAME="Your Name"

WORKER_NODE=`hostname`

echo "Hello ${MYNAME}"

echo "Greets from ${WORKER_NODE}!"
```

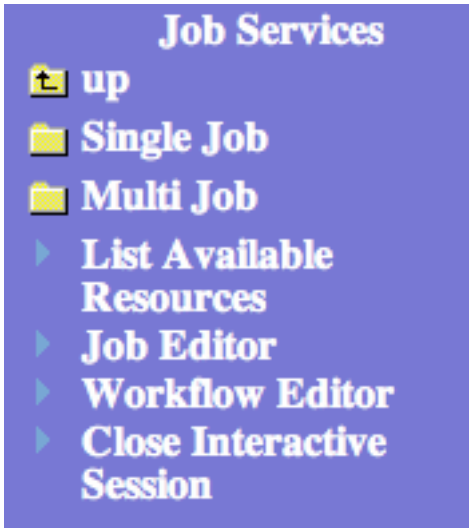
Exercise 3: Job submission (2)



- **Create a JDL file named ‘myhelloworld.jdl’**

```
[
Executable="myhelloworld.sh";
StdOutput="std.out";
StdError="std.err";
VirtualOrganisation="gilda";
InputSandbox={"myhelloworld.sh"};
OutputSandbox={"std.out", "std.err"};
]
```

Exercise 3: Job submission (3)



Finding available resources

- From the Main menu choose "Job Services"
  - Choose "List Available Resources"
  - Specify the previously created JDL file and query for matching resources

- Using the command line interface:
  - Login to your account on glite-tutor.ct.infn.it using ssh.
  - In the homedir you can find the previously created JDL (if not you did something wrong)
  - Issue the following command:

```
> glite-job-list-match <JDL-file>
```

Exercise 3: Job submission (4)

Submitting the job

- From the "Job Services" menu choose "Single Job"
  - Choose Job Submission and specify the JDL file
  - Proceed by pressing the "next" button
  - Submit the job by pressing the "Submit Job" button

- Submit the job using the command line interface by

  > glite-job-submit <JDL-file>

The glite-job-submit command returns the job identifier which is going to be used in subsequent slides.

Example: https://glite-rb2.ct.infn.it:9000/ZpmKtN2NtPJAny4G10_YmA

Exercise 3: Job submission (5)

Job Queue

#	Job ID	JDL Name	Last Update	Destination	Status	Exit Code	Action
6	bGVFFbVO93hxuZz0Rt0MkA	/home/pprax/myhelloworld.jdl	Aug 12 13:44:53 2005 CEST	gilda-ce-01.pd.infn.it:2119/jobmanager-lcgpbs-short	Done	0	Get Output

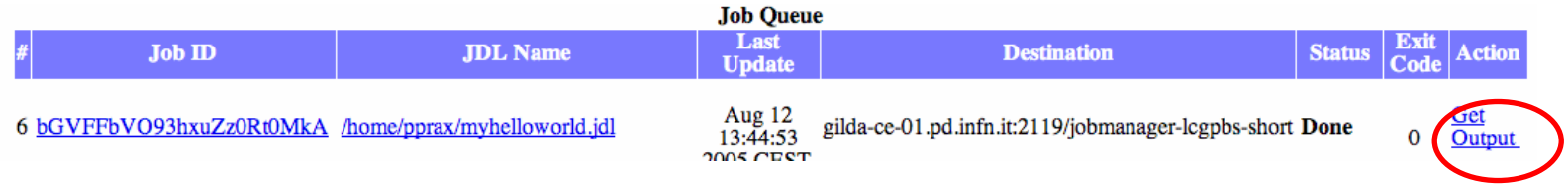
Checking the job status

- Query the job status by pressing "Job Queue"
  - The "Status" changes from "Ready" to "Scheduled" to "Running", and eventually to "Done"
- This function is equal to the

  > glite-job-status <JobID>

command.

Exercise 3: Job submission (6)



Job Queue

#	Job ID	JDL Name	Last Update	Destination	Status	Exit Code	Action
6	bGVFFbVO93hxuZz0Rt0MkA	/home/pprax/myhelloworld.jdl	Aug 12 13:44:53 2005 CEST	gilda-ce-01.pd.infn.it:2119/jobmanager-lcgpbs-short	**Done**	0	Get Output

Retrieving the job output

- Fetch the output by pressing "Get Output“
- You can inspect the files by clicking on them

- On the command line you can do this with

  > glite-job-output <JobID>

- The output is stored in the ~/JobOutput/ directory. The exact location is given to you as the command returns.
- Go there and inspect the files.

Job Description Language (1)

- **The JDL is used to define**
  - Job characteristics
  - Job requirements
  - Data requirements
- **Based on Condor's CLASSified ADvertisement language (ClassAd)**
  - Fully extensible
    - A ClassAd
      - *Constructed with the classad construction operator []*
      - *It is a sequence of attributes separated by semi-colons.*
      - *An attribute is a pair (key, value), where value can be a Boolean, an Integer, a list of strings, …*
        - <attribute> = <value>;

Job Description Language (2)

- **The supported attributes are grouped into two categories:**
  - Job Attributes
    - Define the job itself
  - Resources
    - Taken into account by the Workload Manager for carrying out the matchmaking algorithm (to choose the “best” resource where to submit the job)
    - ***Computing Resource***
      - *Used to build expressions of Requirements and/or Rank attributes by the user*
      - *Have to be prefixed with “other.”*
    - ***Data and Storage resources***
      - *Input data to process, Storage Element (SE) where to store output data, protocols spoken by application when accessing SEs*

Relevant Attributes (1)

- JobType **(mandatory)**
  - *Normal* (simple, sequential job), *DAG*, *Interactive*, *MPICH*, *Checkpointable*
- Executable **(mandatory)**
  - The command name
- Arguments **(optional)**
  - Job command line arguments
- StdInput, StdOutput, StdError **(optional)**
  - Standard input/output/error of the job
- Environment **(optional)**
  - List of environment settings

Relevant Attributes (2)

- InputSandbox **(optional)**
  - List of files on the UI's local disk needed by the job for running
  - The listed files will be staged automatically to the remote resource
- OutputSandbox **(optional)**
  - List of files, generated by the job, which have to be retrieved
- VirtualOrganisation **(mandatory)**
  - The virtual organisation the user submitting the job is working for
  - Can be omitted if preconfigured on the UI or in the VOMS proxy

Relevant Attributes (3)

- Requirements **(optional)**
  - Job **requirements on computing resources**
  - Specified using attributes of resources published in the Information Service
- Rank **(optional)**
  - **Expresses preference** (how to rank resources that have already met the Requirements expression)
  - Specified using attributes of resources published in the Information Service
  - If not specified, default value defined in the UI configuration file is considered
    - Default: *other.**GlueCEStateEstimatedResponseTime*** (the lowest estimated traversal time)
    - Default: *other.**GlueCEStateFreeCPUs*** (the highest number of free CPUs) for parallel jobs (see later)

Relevant Attributes (4)

- InputData **(optional)**
  - Refers to data used as input by the job: these data are published in the Replica Catalog and stored in the Storage Elements
  - LFNs and/or GUIDs
- DataAccessProtocol **(mandatory if InputData has been specified)**
  - The protocol or the list of protocols which the application is able to speak with for accessing *InputData* on a given Storage Element
- OutputSE **(optional)**
  - The Uniform Resource Identifier of the output Storage Element
  - RB uses it to choose a Computing Element that is compatible with the job and is close to the Storage Element

JDL: Specifying Requirements

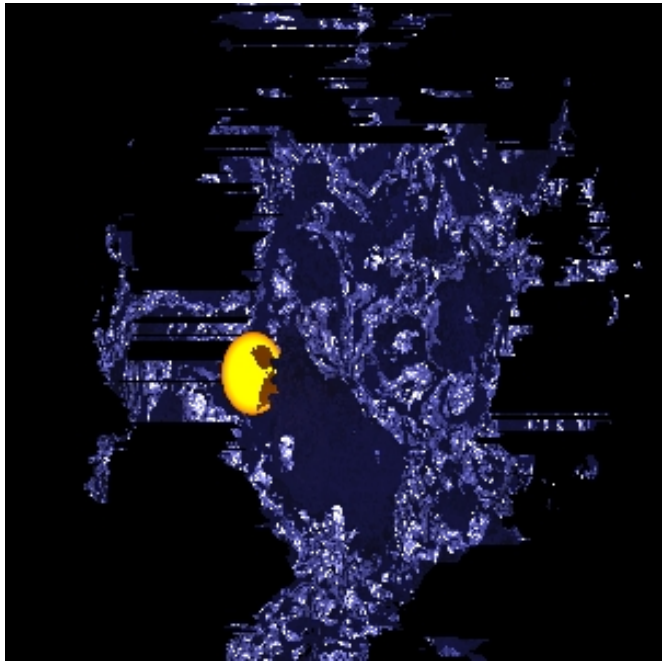
- **Requirements allow you to influence the matchmaking process**
  - Used to specify software requirements (e.g. a specific software has to be installed on the worker node in order to execute the job)
  - A resource must fulfill the constraints specified in order to be allowed to execute the job (e.g. a specified number of processors must be available; a job requires some minimum amount of RAM being available, etc.)
  - Example:

```
Requirements=other.GlueHostOperatingSystem==“LINUX” &&
other.GlueCEStateFreeCPUs>=4;
```

Allows you to influence the matchmaking process!

Job requirements

- In the following exercise you will:
  - Create a povray raytracing scene to be calculated in the grid
  - Create a JDL file specifying the requirements for that job
  - Submit the job to be run in the grid
  - Inspect the output of the job



Exercise 4: Job Requirements(1)

- **Objective**: Execute a POVRay rendering job
- **Requirements:** POVRay needs to be installed on the worker node

Create the POVRay scene description file "nuggets.pov":

```
// Thanks to „Dazza“
// http://astronomy.swin.edu.au/~pbourke/povray/scc3/final/
global_settings{radiosity{}}

#declare f=function{pigment{granite}};

union {
      isosurface {
               function{x*x+y*y+z*z-f(x,y,z).red}
               pigment{color rgb<1,1,2>/2}
               finish{specular 1}
      }
      sphere{<-.13,-.05,-.7> .05
      pigment{rgb<4,2,0>}
}
translate z*1.5
}
light_source{0,1}
```

Exercise 4: Job Requirements(2)

- Create a script named "startNuggets.sh" which calls "povray" and renders the scene on the compute element

```
#!/bin/sh
povray +Inuggets.pov +Onuggets.tga +FT +W300 +H300 +V -D +X
```

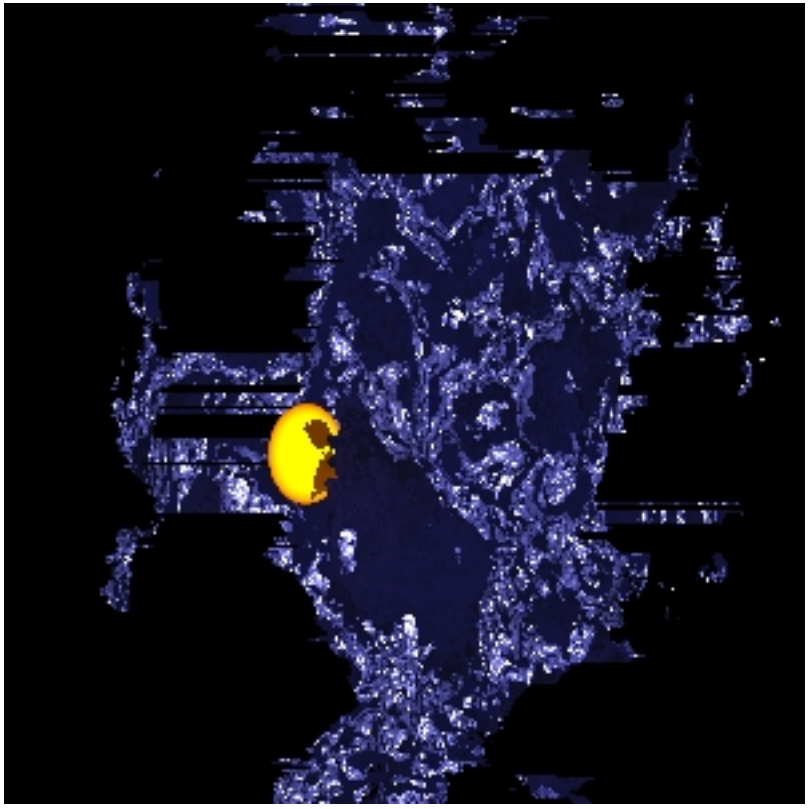
Exercise 4: Job Requirements(3)

- Create the JDL file nuggets.jdl for rendering the scene:

```
[
    Executable="startNuggets.sh";
    StdOutput="nuggets.out";
    StdError="nuggets.err";
    InputSandbox={"startNuggets.sh", "nuggets.pov" };
    OutputSandbox={"nuggets.out", "nuggets.err", "nuggets.tga"};
    Requirements= Member("POVRAY-3.5",
    other.GlueHostApplicationSoftwareRunTimeEnvironment);
]
```

Exercise 4: Job Requirements(4)

- Submit the job
- Fetch the output as soon as it is available
- View the resulting image with your preferred image viewer



Submitting your own program

- In the following exercise you will:
  - Write and compile a C++ program to be run in the grid
  - Create a JDL file specifying the requirements for that job
  - Submit the job to be run in the grid
  - Inspect the output of the job

Exercise 5: Submitting your own Program (1)

- Create a file "myprogram.cpp" with the following content:

```
#include <iostream>
int main() {
        std::cout << "Hello World!" << std::endl;
}
```

- Compile and run

```
> g++ -o myprogram myprogram.cpp && ./myprogram
```

Exercise 5: Submitting your own Program (2)

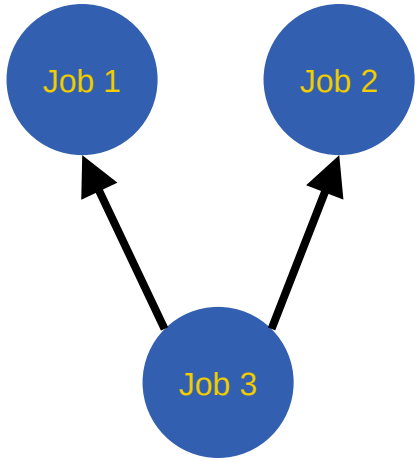
- Create the JDL file ‚myprogram.jdl‘ with the following content:

```
[
        Executable="myprogram";
        StdOutput="std.out";
        StdError="std.err";
        InputSandbox={"myprogram"};
        OutputSandbox={"std.out","std.err"};
]
```

- Submit it to the Grid
- Check the status
- Fetch and inspect the output

Grid workflow using DAG jobs

- A DAG (directed acyclic graph) is a set of nodes and edges that can be used to describe dependencies between grid jobs
- DAG jobs are used to specify workflows on the grid
- In the following exercise you will:
  - Write the JDL for a DAG job
  - Submit the job to the grid
  - Inspect and verify the output



Exercise 6: DAG jobs (1)

- Copy the DAG JDL file to your account

  > mkdir ~/dag

  > cp ~cern01/dag/* ~/dag

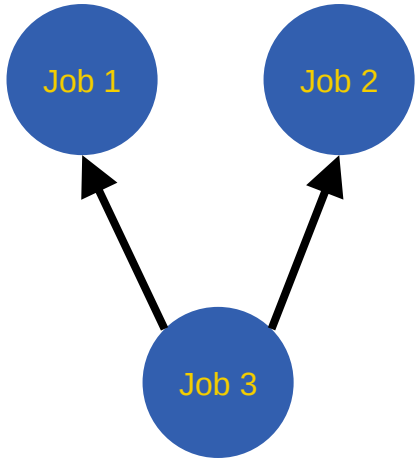
- Inspect the JDL file

  > cd ~/dag

  > cat dag1.jdl

  The DAG JDL file contains sections for three sub-jobs. Also note the dependencies section:

```
dependencies = {
  { {node1, node2 }, node3 }
}
```



Exercise 6: DAG jobs (2)

- Submit the DAG job to the grid and follow its status

  > glite-job-submit dag1.jdl

  > glite-job-status <JOBID>

- As soon as it is done, fetch the output

  > glite-job-output <JOBID>

- Verify the output of the three sub-jobs
  - The time stamp in the output of job 3 must not preceed those of jobs 1 and 2

File management

- In the following exercise you will:
  - Browse the FireMan catalogue
  - Create and remove directories in the catalogue
  - Publish files in the catalogue

Exercise 7: File management (1)

- **Browse the file catalog using**

  > glite-catalog-ls -l /

- **Create a new directory in the catalog's root**

  > glite-catalog-mkdir /<account>

- **Explore the properties of the newly created directory**

  > glite-catalog-stat /<account>

- **Remove the directory you have just created**

  > glite-catalog-rmdir /<account>

Exercise 7: File management (2)

- **Store a local file in the root directory on the storage element and assign it a LFN (Logical File Name)**

  > glite-put ~/myhelloworld.sh lfn:///myhelloworld_<account>.sh

- **Browse the file catalog again**

  > glite-catalog-ls -l /

- **Retrieve the file from the storage element**

  > glite-get lfn:///myhelloworld_<account>.sh test.sh

- **Remove it from the storage element**

  > glite-rm lfn:///myhelloworld_<account>.sh

- **Browse the file catalog**

  > glite-catalog-ls -l /

Information System: R-GMA

- In the following exercise you will:
  - Browse the R-GMA information system

Exercise 8: Using the gLite Information System (1)

- Metainformation about the state of the Grid is provided through R-GMA (Relational - Grid Monitoring Architecture)
- In order to use the R-GMA commands you will need a working proxy.
- To start working with R-GMA just run the command rgma from the prompt [glite-tutor] /home/pprax > rgma

```
Welcome to the R-GMA virtual database for Virtual Organisations.
================================================================

Your local R-GMA server is:

  https://rgmasrv.ct.infn.it:8443/R-GMA

You are connected to the following R-GMA Registry services:

  https://rgmasrv.ct.infn.it:8443/R-GMA/RegistryServlet

You are connected to the following R-GMA Schema service:

  https://rgmasrv.ct.infn.it:8443/R-GMA/SchemaServlet

Type "help" for a list of commands.

rgma>
```

Exercise 8: Using the gLite Information System (2)

- Displays help for a specific command

  rgma> help <command>

- Exit the R-GMA command line

  rgma> exit or quit

- To show a list of all table names:

  rgma> show tables

- To show information about a table MyTable :

  rgma> describe MyTable

- To show a table of properties for the current session:

  rgma> show properties

- To show a list of all R-GMA producers that produce the table MyTable :

  rgma> show producers of MyTable

Exercise 8: Using the gLite Information System (3)

- Show all tables that are present in the Schema:

  rgma> show tables

- Choose one table where you want insert your information and see its attribute:

  rgma> describe <NameTable>

- Some example queries:

  rgma> select * from Site

  rgma> select SysAdminContact,UserSupportContact from Site

  rgma> select Name from Site where Latitude > 0

  rgma> select Endpoint, Type from Site,Service where Site.Name = Service.Site_Name and Latitude > 0

Further information

- http://www.glite.org/
- http://gilda.ct.infn.it/
- http://www.eu-egee.org/
- http://www.egee.nesc.ac.uk/
- http://www.cern.ch/

Resources

- http://www.mozilla.org/
- http://www.chiark.greenend.org.uk/~sgtatham/putty/