

## Special jobs with the gLite WMS

*Emidio Giorgio*

*INFN*

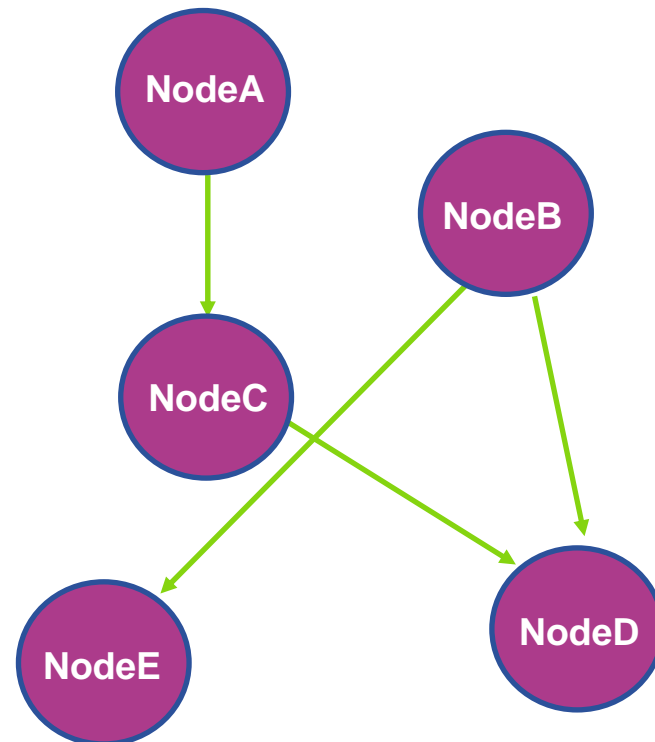
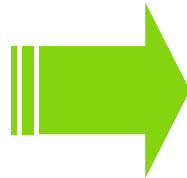
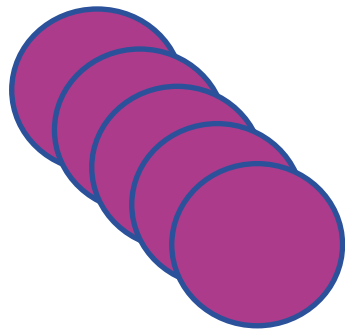
*Retreat between GILDA and ESR VO, Bratislava, 27-30.06.2005*



- **DAG**
- **Checkpointable**
- **MPI**
- **Interactive**
- **Bulk Submissions**

- A DAG represents a set of jobs:

*Nodes = Jobs*

*Edges = Dependencies*



- Type = "DAG"  *Mandatory*
- VirtualOrganisation = "yourVO"  *Mandatory*
- Max\_Nodes\_Running = int >0  *Optional*
- MyProxyServer = "... "  *Optional*
- Requirements = "... "  *Optional*
- Rank = "... "  *Optional*
- InputSandbox =  more later!  *Optional*
- ~~• OutSandbox = "... "~~
- Nodes = nodeX  more later!  *Mandatory*
- Dependencies  more later!  *Mandatory*

The *Nodes* attribute is the core of the DAG description;

```
....
Nodes = [ nodefilename1 = [...]
          nodefilename2 = [...]
          .....
          dependencies = ...
        ]
```



```
Nodefilename1 = [ file = "foo.jdl"; ]
Nodefilename2 =
  [ file = "/home/vardizzo/test.jdl";
    retry = 2;      ]
```



```
Nodefilename1 = [
  description = [ JobType = "Normal";
                 Executable = "abc.exe";
                 Arguments = "1 2 3";
                 OutputSandbox = [...];
                 InputSandbox = [...];
                 ..... ]
  retry = 2;
  ]
```

- It is a list of lists representing the dependencies between the nodes of the DAG.

```

....
Nodes = [ nodefilename1 = [...]
          nodefilename2 = [...]
          .....
          dependencies = ...
        ]
    
```



```
dependencies =
    {nodefilename1, nodefilename2}
```



**MANDATORY : YES!**

```
dependencies = {};
```

```
{ nodefilename1, nodefilename2 }
```

```
{ { nodefilename1, nodefilename2 }, nodefilename3 }
```

```
{ { { nodefilename1, nodefilename2 }, nodefilename3 }, nodefilename4 }
```

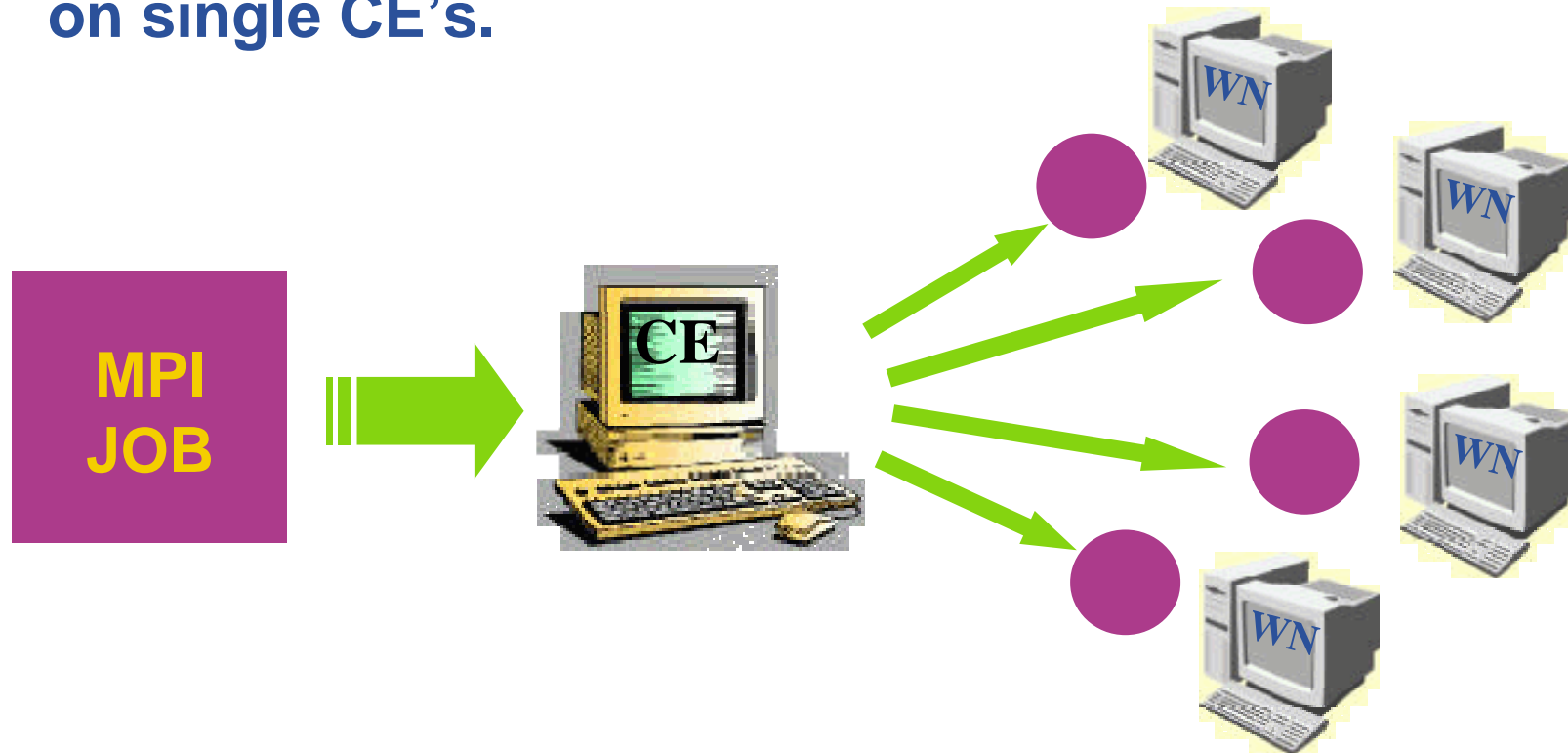
- All nodes inherit the value of the attributes from the one specified for the DAG.

- Nodes without any InputSandbox values, have to contain in their description an empty list:

**InputSandbox = { };**

```
NodeA= [
description = [
    JobType = "Normal";
    Executable = "abc.exe";
    OutputSandbox = {"myout.txt"};
    InputSandbox = {
        "/home/vardizzo/myfile.txt",
        root.InputSandbox; };
    ]
]
```

- The MPI job is run in parallel on several processors.
- Libraries supported for parallel jobs: MPICH.
- Currently, execution of parallel jobs is supported only on single CE's.





- **Type = “job”;** ➔ *Mandatory*
- **JobType = “**MPICH**”;** ➔ *Mandatory*
- **Executable = “...”;** ➔ *Mandatory*
- **NodeNumber = “**int > 1**”;** ➔ *Mandatory*
- **Argument = “...”;** ➔ *Optional*
- **Requirements =** ➔ *Mandatory*  
*Member(“MpiCH”, other.GlueHostApplicationSoftwareRunTimeEnvironment)*  
*&& other.GlueCEInfoTotalCPUs >= NodeNumber ;*
  
- **Rank = *other.GlueCEStateFreeCPUs*;** ➔ *Mandatory*

```
Type = "Job";
```

```
JobType = "MPICH";
```

```
Executable = "MPItest.sh";
```

```
Arguments = "cpi 2";
```

```
NodeNumber = 2;
```

```
StdOutput = "test.out";
```

```
StdError = "test.err";
```

```
InputSandbox = {"MPItest.sh","cpi"};
```

```
OutputSandbox = {"test.err","test.out","executable.out"};
```

```
Requirements = other.GlueCEInfoLRMSType == "PBS" ||  
other.GlueCEInfoLRMSType == "LSF";
```

- It is a mechanism by which a job can access at some information about the job...at execution time!
- Two ways for parsing elements from BrokerInfo file:
  - 1)Directly from the job;
  - 2)From the UI with CLI and “*.BrokerInfo*” file;

**edg-brokerinfo** | **glite-brokerinfo** [options] function param



## STEP 1

```
#!/bin/sh
```

```
#
```

```
# this parameter is the binary to be executed
```

```
EXE=$1
```

```
# this parameter is the number of CPU's to be reserved for parallel  
execution
```

```
CPU_NEEDED=$2
```

```
# prints the name of the master node
```

```
echo "Running on: $HOSTNAME"
```

```

if [ -f "$PWD/.BrokerInfo" ] ; then
    TEST_LSF=`edg-brokerinfo getCE | cut -d/ -f2 | grep lsf`
else
    TEST_LSF=`ps -ef | grep sbatchd | grep -v grep`
fi
if [ "x$TEST_LSF" = "x" ] ; then
    # prints the name of the file containing the nodes allocated for parallel execution
    echo "PBS Nodefile: $PBS_NODEFILE"
    # print the names of the nodes allocated for parallel execution
    cat $PBS_NODEFILE
    echo "*****"
    HOST_NODEFILE=$PBS_NODEFILE
else
    # print the names of the nodes allocated for parallel execution
    echo "LSF Hosts: $LSB_HOSTS"
    # loops over the nodes allocated for parallel execution
    HOST_NODEFILE=`pwd`/lsf_nodefile.$$

    for host in ${LSB_HOSTS}
    do
        echo $host >> ${HOST_NODEFILE}
    done
fi

```

STEP 2

```
echo "*****"
# prints the working directory on the master node
echo "Current dir: $PWD"
echo "*****"
```

**STEP 3**

```
for i in `cat $HOST_NODEFILE` ; do
    echo "Mirroring via SSH to $i"
    # creates the working directories on all the nodes allocated for parallel execution
    ssh $i mkdir -p `pwd`
    # copies the needed files on all the nodes allocated for parallel execution
    /usr/bin/scp -rp ./ * $i:`pwd`
    # checks that all files are present on all the nodes allocated for parallel execution
    echo `pwd`
    ssh $i ls `pwd`
    # sets the permissions of the files
    ssh $i chmod 755 `pwd`/$EXE
    ssh $i ls -alR `pwd`
    echo "#####"
done
```

## STEP 4

```
# execute the parallel job with mpirun
echo "*****"
echo "Executing $EXE"
chmod 755 $EXE
ls -l
mpirun -np $CPU_NEEDED -machinefile
    $HOST_NODEFILE `pwd`/$EXE > executable.out
echo "*****"
```

```
[glite-tutor] /home/vardizzo > cat executable.out
```

```
Process 0 of 2 on grid037.ct.infn.it
```

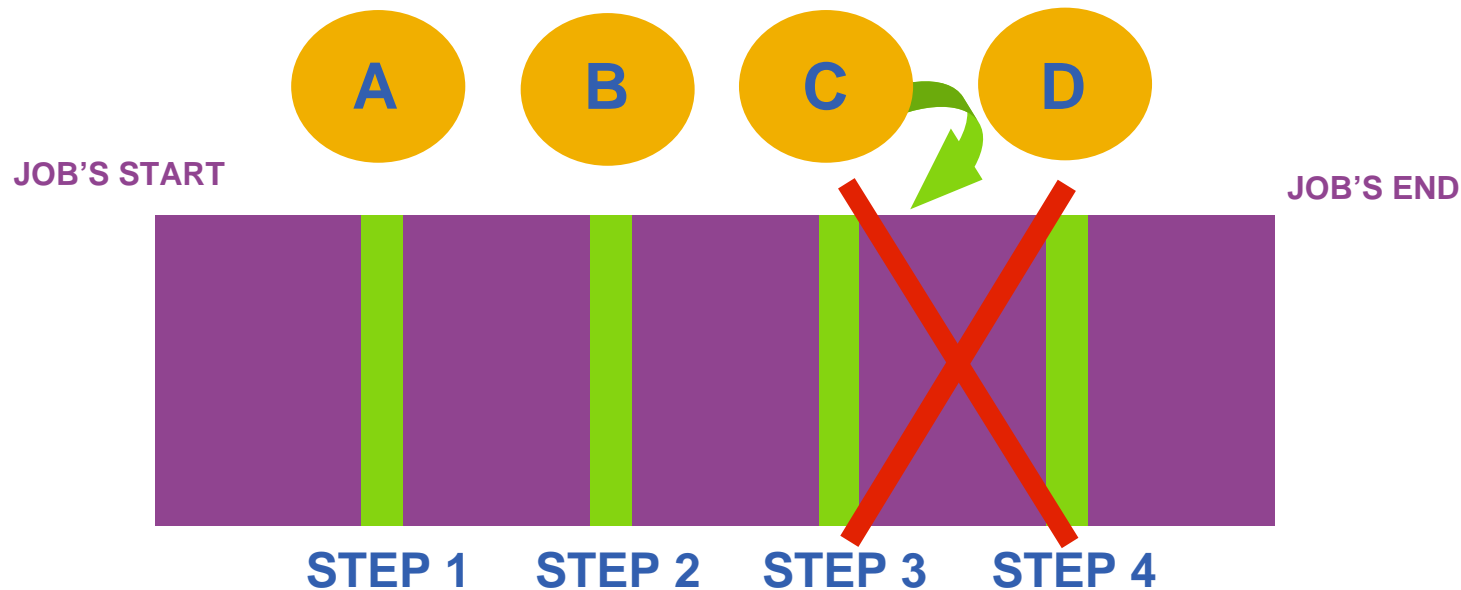
```
pi is approximately 3.1415926544231318, Error is  
0.0000000008333387
```









```
wall clock time = 10.003797
```

```
Process 1 of 2 on grid037.ct.infn.it
```



- It is a job that can be decomposed in several steps;
- In every step the job state can be saved in the LB and retrieved later in case of failures;
- The job can start running from a previously saved state instead from the beginning again.



- `Type = "job";`  *Mandatory*
- `JobType = "checkpointable";`  *Mandatory*
- `Executable = "...";`  *Mandatory*
- `JobSteps = "list int | list string";`  *Mandatory*
- `CurrentStep = "int >= 0";`  *Mandatory*
- `Argument = "...";`  *Optional*
- `Requirements = "...";`  *Optional*
- `Rank = "";`  *Optional*

- **JDL Submission:**

**edg-job-submit** -o guidfile jobCheck.jdl

**glite-job-submit** -o guidfile jobCheck.jdl

- **JDL Status:**

**edg-job-status** -i guidfile | **glite-job-status** -i guidfile

- **JDL Output:**

**edg-job-get-output** -i guidfile | **glite-job-output** -i guidfile

- **Get Latest Job State:**

**edg-job-get-chkpt** -o statefile -i guidfile

**glite-job-get-chkpt** -o statefile -i guidfile

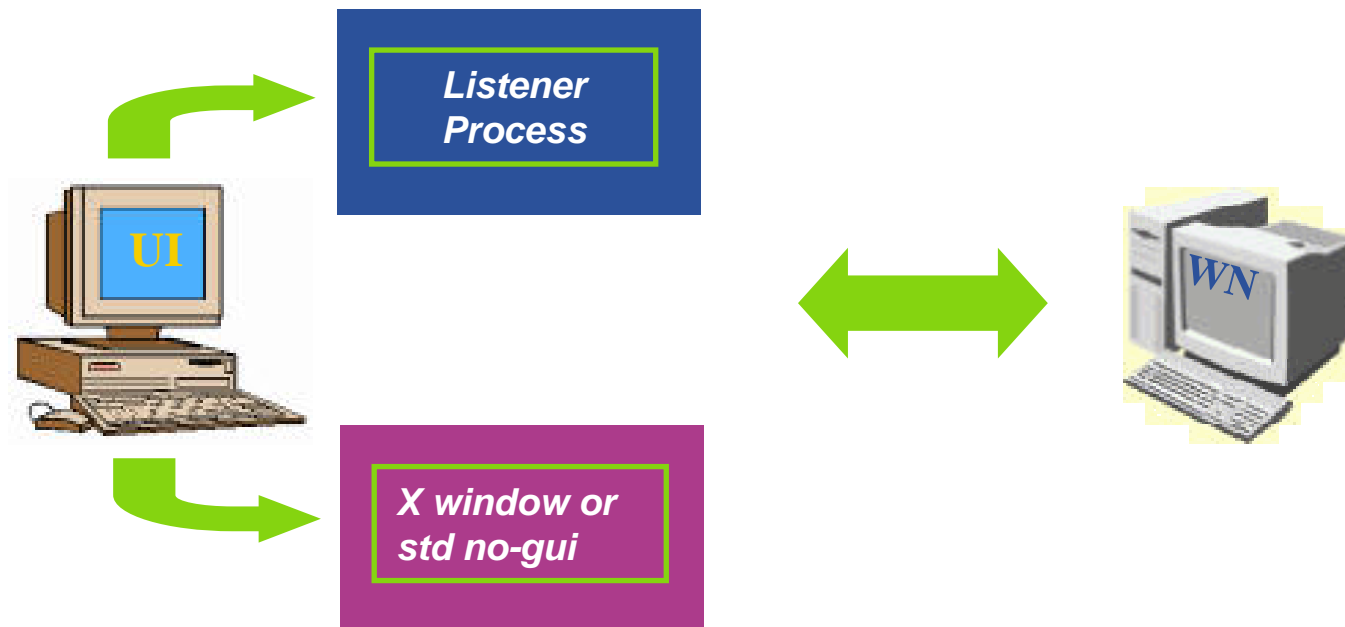
- **Submit a JDL from a state:**

**edg-job-submit** -chkpt statefile -o guidfile jobCheck.jdl

**glite-job-submit** -chkpt statefile -o guidfile jobCheck.jdl

- **See also [options] from -help of the commands.**

- It is a job whose standard streams are forwarded to the submitting client.
- The DISPLAY environment variable has to be set correctly, because an X window is open.



- Type = “job”;
- JobType = “**interactive**”;
- Executable = “...”;
- Argument = “...”;
- ListenerPort = “**int > 0**”;
- OutputSandbox = “”;
- Requirements = “...”;
- Rank = “”;

- ➡ *Mandatory*
- ➡ *Mandatory*
- ➡ *Mandatory*
- ➡ *Optional*
- ➡ *Optional*
- ➡ *Optional*
- ➡ *Mandatory*
- ➡ *Mandatory*

LCG | gLite Commands:

**edg-job-attach** [options] <jobID>

| **glite-job-attach** [options] <jobID>

- **Feature available starting from gLite 1.3 (WMPProxy)**
- **Multiple job submission with a single command**
- **Job executions monitored with a unique job ID (job father ID)**
- **Two ways of performing it :**
  - Submit a **collection** of jdl's
  - Submit repeatedly the same jdl (parametric job)

- `glite-wms-job-submit -d <user> --collection <dir_name>`
- **dir\_name** is a directory containing a collection of jdl files
- Each jdl in the collection is automatically submitted
- Job-father ID is returned
- Jobs retrieval can be performed only when each job is done
- Jobs output are retrieved as a unique directory, with a subdir for each sub-job

- `glite-wms-job-submit -d <user> job.jdl`
- **The jdl must contain these two attribute**  
`JobType = "parametric";`  
`Parameters = <n>; //# of submissions`
- **Job-father ID is returned**
- **Jobs retrieval can be performed only when each job is done**
- **Jobs output are retrieved as a unique directory, with a subdir for each sub-job**



There are some questions ?

