

First evaluation of the Globus GRAM Service

**Massimo Sgaravatto
INFN Padova
massimo.sgaravatto@pd.infn.it**

**Draft version release 1.0.5
20 June 2000**

1	Introduction	3
2	Running jobs	3
	2.1 Usage examples	3
	2.1.1 globus-job-run examples	4
	2.1.2 globus-job-submit examples	4
	2.1.3 globusrun examples	4
	2.2 Problems, bugs and missing functionalities	5
3	Using Condor as underlying resource management system for Globus	7
	3.1 Configuration	7
	3.2 Globus RSL and Condor ClassAds	8
	3.3 Usage examples	10
	3.4 Problems, bugs and missing functionalities	10
	3.5 Interaction with GIS	12
4	Submitting Condor jobs to Globus resources	13
	4.1 Globus universe	13
	4.2 Condor GlideIn	14
5	Using LSF as underlying resource management system for Globus	16
	5.1 Configuration	16
	5.2 Globus RSL and LSF resource specification language	17
	5.3 Usage examples	18
	5.4 Problems, bugs and missing functionalities	19
	5.5 Interaction with GIS	19
6	RSL	20
7	Conclusions and future activities	21
8	References	22

1 Introduction

This note describes the preliminary activities and results related with the evaluation of the Globus GRAM (Globus Resource Allocation Management) service.

Tests have been performed using, as first phase, the fork system call as job manager, and then considering Condor and LSF as underlying resource manager systems: basic and simple tests have been performed, in order to evaluate the capabilities and functionalities of the service.

It has also been evaluated how each local GRAM is able to provide the Grid Information Service (GIS, formerly known as Metacomputing Directory Services, MDS) with information on characteristics and status of the local resources.

Other tests related with the integration between Globus and Condor (how it is possible to submit Condor jobs on Globus resources) have been performed.

These tests have been done considering Linux (Red Hat 6.1) Intel PCs, using Globus release 1.1.2.

The described tests must be considered as preliminary activities: we will keep working on these items in the next months within the work package 1 (“Installation and evaluation of the Globus Toolkit”) [1] of the INFN-GRID project.

2 Running jobs

There are three commands that can be used to run jobs with Globus:

- **globus-job-run**: this command is used to run jobs in the background
- **globus-job-submit**: this command is used to submit jobs to a batch job scheduler
- **globusrun**: this command is used for submitting jobs that are specified using RSL, the Resource Specification Language. It can run jobs either in the foreground or background. `globus-job-run` and `globus-job-submit` are simply shell scripts that use `globusrun` for job submission.

If a job is submitted in foreground, the prompt is returned to the user only when this job has been completed.

In the contrary, when a job is submitted in background, the prompt is returned to the user immediately, and an ID (i.e. <https://lxde16.pd.infn.it:4560/20533/957883354/>) is returned.

With the command:

```
globus-job-status ID
```

it is possible to verify the status of the submitted job (pending, in execution, completed).

2.1 Usage examples

Here we report some usage examples of the Globus GRAM service, considering the layout represented in figure 1.

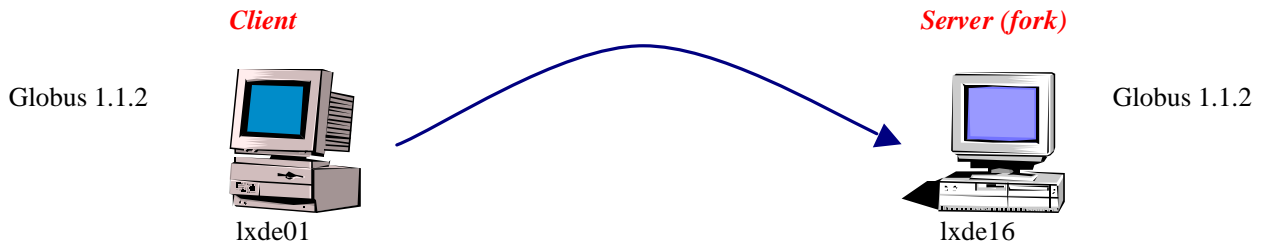


Figure 1

A Linux PC (*lxde01.pd.infn.it*) has been configured as Globus client machine, and an other PC (*lxde16.pd.infn.it*) has been used as Globus server machine, using the fork system call as job manager.

2.1.1 globus-job-run examples

- **lxde01%** globus-job-run lxde16.pd.infn.it/jobmanager-fork -stdout -l /tmp/risultato -count 10 \ -s /tmp/abc/ciao

The executable file (*/tmp/abc/ciao*) is in the file system of the submitting machine (option *-s*).
The output file (*/tmp/risultato*) is created in the file system of the executing machine (option *-l*).

- **lxde01%** globus-job-run lxde16.pd.infn.it/jobmanager-fork -stdout -s /tmp/risultato -np 10 \ -stage /tmp/abc/ciao "first parameter" "second parameter"

The executable file (*/tmp/abc/ciao*) is in the file system of the submitting machine (option *-s*).
"first parameter" and "second parameter" are two arguments for this executable.
The output file (*/tmp/risultato*) is created in the file system of the submitting machine (option *-s*).

2.1.2 globus-job-submit examples

- **lxde01%** globus-job-submit lxde16.pd.infn.it/jobmanager-fork -np 10 -stdout -local /tmp/ris \ /users/noi/sgaravat/ciao "first parameter" "second parameter"

The executable file (*/users/noi/sgaravat/ciao*) is in the file system of the executing machine.
"first parameter" and "second parameter" are two arguments for this executable.
The output file (*/tmp/ris*) is created in the file system of the executing machine.

2.1.3 globusrun examples

- **lxde01%** globusrun -b -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl

file.rsl:

```
&
(executable=https://lxde01.pd.infn.it:2793/tmp/abc/ciao)
(stdout=/tmp/output)
(count=10)
```

The job runs in background (option `-b`).
The executable file is in the file system of the submitting machine.
The output file is created in the file system of the executing machine.
It was first necessary to start the GASS server in the submitting machine (using the command `globus-gass-server`). In this case the command returned: `https://lxde01.pd.infn.it:2793`.

- **lxde01%** globusrun -b -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl

file.rsl:

```
&
(executable=https://lxde01.pd.infn.it:2793/tmp/abc/ciao)
(stdout=https://lxde01.pd.infn.it:2793/tmp/output)
(count=10)
```

In this case also the output file is created in the file system of the submitting machine.

- **lxde01%** globusrun -s -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl

file.rsl:

```
&
(executable=$(GLOBUSRUN_GASS_URL)/tmp/abc/ciao)
(stdout=/tmp/output)
(count=10)
(arguments= "first parameter" "second parameter")
```

The executable file is stored in the file system of the submitting machine.
The output file is created in the file system of the executing machine.
It is important to use the option `-s`, otherwise the variable `GLOBUS_GASS_URL` is not recognized.

2.2 Problems, bugs and missing functionalities

- It seems not possible to have the executable file in the file system of the submitting machine (option `-s`) with `globus-job-submit`:

```
lxde01% globus-job-submit lxde16.pd.infn.it/jobmanager-fork -np 10 -stdout -l /tmp/ris \
-s /tmp/abc/ciao "first parameter" "second parameter"
```

It doesn't work:

GRAM Job submission failed because the provided RSL string includes variables that could not be identified (error code 39)

The option `-dumprsl` is useful to debug the problem:

```
lxde01% globus-job-submit -dumprsl lxde16.pd.infn.it/jobmanager-fork -np 10 -stdout \
-l /tmp/ris -s /tmp/abc/ciao "first parameter" "second parameter"
```

This is the output:

```
&(executable=$(GLOBUSRUN_GASS_URL) # "/tmp/abc/ciao")
(count=10)
(arguments= "first parameter" "second parameter")
(stdout="/tmp/ris")
(stderr=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stderr anExtraTag)
```

This RSL expression has been saved in a file (*file.rsl*).

```
lxde01% globusrun -b -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl
```

GRAM Job submission failed because the provided RSL string includes variables that could not be identified (error code 39)

```
lxde01% globusrun -s -b -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl
```

ERROR: option -s and -b are exclusive

```
lxde01% globusrun -s -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl
```

It works, but in foreground (not as a batch job).

There is the same behavior with the standard output.

- When the fork system call is used as job manager, if multiple jobs are submitted, and an output file is defined, the output of all these jobs is written in this file: it is not possible to have different output files for the different jobs.

For example:

```
lxde01% globusrun -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl
```

file.rsl:

```
&
(executable=$(HOME)/ciao)
(stdout=/tmp/output)
(count=10)
```

ciao is a simple program that prints “Ciao” in the standard output.
After the execution, in the file */tmp/output* it is written “Ciao” 10 times.

There is the same behavior if the jobs are submitted as batch jobs (again using the fork system call):

```
lxde01% globusrun -b -r lxde16.pd.infn.it/jobmanager-fork -f file.rsl
```

3 Using Condor as underlying resource management system for Globus

3.1 Configuration

As represented in figure 2, a PC (*lxde16.pd.infn.it*) running Globus 1.1.2 and Condor 6.1.12, has been configured as front-end machine with the INFN WAN Condor pool, a pool composed by about 200 machines of different architectures (Digital Unix workstations, HP-UX machines, Linux PCs, Sun Solaris workstations) distributed in the various INFN sites [2]. An other PC (*lxde01.pd.infn.it*), running Globus 1.1.2, has been used as client machine.

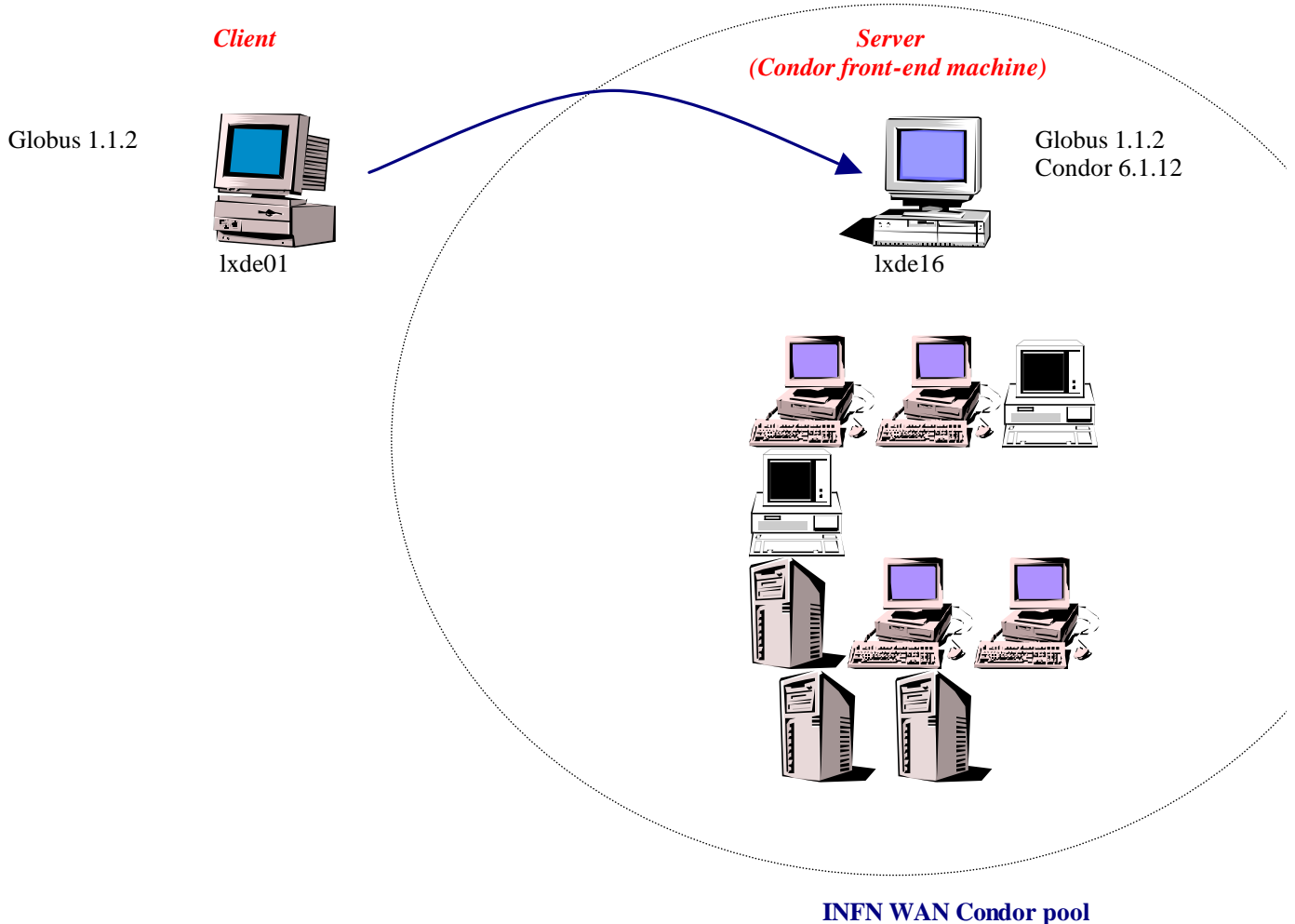


Figure 2

To configure Condor as a job manager for Globus, it has been necessary to follow the instructions reported in the “Globus Toolkit System Administration Manual” [3] (Chapter 5: “Configuring the Globus Toolkit Gatekeeper and Services”). It has also been necessary to define the attributes:

```
-condor-os  
-condor-arch
```

also in the files:

```
<deploy-dir>/etc/globus-services  
</deploy-dir>/etc/globus-daemons.conf
```

and not only in the file:

```
<deploy-dir>/etc/globus-jobmanager-condor.conf
```

3.2 Globus RSL and Condor ClassAds

To evaluate the capabilities of Globus GRAM in conjunction with Condor as underlying resource manager system, it is necessary to analyze how the RSL expressions are translated in the Condor ClassAds, the language used in Condor to describe both the resources offered by the pool, and the resources requested by the submitted jobs.

For this purpose, it is useful to edit the file:

```
<deploy-dir>/libexec/globus-script-condor-submit file
```

and comment out the line near the end that removes the file:

```
#{rm} $condor_file
```

After the submission of a job (using `globusrun`, or `globus-job-run`, or `globus-job-submit`), a file:

```
/tmp/condor_script.description.*
```

is created: this is the actual submission file used by Condor.

The important result of this analysis is that the Globus job manager only cares about a set of common RSL attributes. Any attributes outside of the set are not passed unchanged to the underlying resource management, as described in the Globus documentation: they are simply ignored.

For example it is not possible to use the *requirements* Condor attribute, in order to “force” a job to run on a particular machine (or on a particular set of machines that have specific characteristics).

For example, in the following RSL file it is asked that the job must be executed on a specific machine (*lxde01.pd.infn.it*):

```

&
(executable=pippo.condor)
(requirements='machine == "lxde01.pd.infn.it"')
(stdout=file.out)
(stderr=file.err)
(log=/home/sgaravat/file.log)
(jobtype=condor)
(count=1)
(arguments= 'first' 'second')
(notification=always)
(notify_user=massimo.sgaravatto@pd.infn.it)

```

The result is not the expected one, because the *requirements* attribute is ignored, since it is not a common RSL attribute.

In this example, also the *log*, *notification* and *notify_user* parameters are ignored, for the same reason: they are not common RSL attributes.

This can be a serious problem, in particular when the Condor pool (like the INFN WAN Condor pool) is composed by heterogeneous machines, with different architectures, different characteristics, different performances.

For example it is not possible to consider vanilla jobs (that require a uniform file system and UID domain), without the possibility to force the jobs to run on a “uniform subset” of the Condor pool, composed by machines that share the same file system and UID domain.

As reported by the Globus team, there is some work going on to come up with a good way to allow scheduler specific attributes to be passed down to the various scheduler scripts, but nothing concrete has been decided upon yet.

In the mean time the workaround is to modify the file:

```
<deploy-dir>/libexec/globus-script-condor-submit
```

copying the technique used in the script:

```
<deploy-dir>/libexec/globus-script-loadleveler-submit
```

(it looks for environment variables set in the (*environment=xxx*) attribute).

Therefore, considering the previous example, the RSL file should be:

```

&
(executable=pippo.condor)
(stdout=file.out)
(stderr=file.err)
(jobtype=condor)
(count=1)
(arguments= 'first' 'second')
(environment=(REQUIREMENTS=""machine=="lxde01.pd.infn.it"')(LOG=/home/sgaravat/log)
(NOTIFICATION=always)(NOTIFY_USER=massimo.sgaravatto@pd.infn.it))

```

3.3 Usage examples

Using the described configuration, some tests have been performed, submitting jobs from Globus to the Condor pool, considering both standard Condor jobs (jobs linked with the Condor library, that can profit from the remote I/O and checkpointing features) and vanilla jobs.

- **lxde01%** globusrun -r lxde16.pd.infn.it/jobmanager-condor -f file.rsl

file.rsl:

```
&
(executable=$(HOME)/ciao.condor)
(stdout=/tmp/output)
(jobtype=condor)
(count=10)
```

10 jobs are submitted to the Condor pool, using *lxde16.pd.infn.it* as front-end machine.

ciao.condor is a standard Condor job (linked with the Condor library).

The prompt is returned to the user when the 10 jobs have been completed.

The output file is created in the file system of the front-end machine (*lxde16.pd.infn.it* in this case) even if this machine was not the actual executing machine (an other machine of the Condor pool ran this job).

- **lxde01%** globusrun -b -r lxde16.pd.infn.it/jobmanager-condor -f file.rsl

file.rsl:

```
&
(executable=$(HOME)/ciao.condor)
(stdout=/tmp/output)
(jobtype=condor)
(count=10)
```

As in the previous example, but in this case the jobs are submitted in background: the prompt is returned to the user immediately.

3.4 Problems, bugs and missing functionalities

- Since for vanilla jobs it is necessary to consider a uniform file system and a single UID domain, there are problems submitting this kind of jobs to the INFN WAN Condor pool, an heterogeneous environment composed by machines that don't have a common file system and UID domain:

```
lxde01% globus-job-run lxde16.pd.infn.it/jobmanager-condor -stdout -l /tmp/result -count 10 -s /tmp/abc/ciao
```

With this command, 10 condor vanilla jobs are submitted to the Condor pool, using *lxde16.pd.infn.it* as front-end machine.

Since it is not possible to force the jobs to run on a "uniform" sub-pool (as explained in section 3.2), these jobs didn't run successfully.

- The option `-jobtype` (described in the Globus documentation) is not recognized:

```
lxde01% globus-job-run lxde16.pd.infn.it/jobmanager-condor -stdout -l /tmp/risultato \
-count 10 -jobtype condor -s /tmp/abc/ciao
```

```
lxde01% globus-job-submit lxde16.pd.infn.it/jobmanager-condor -np 10 \
-stdout -local /tmp/ris -jobtype condor /users/noi/sgaravat/Condor/ciao
```

These commands don't work.

As reported by the Globus team, this problem will be solved in a future release of the software. In the meantime, the workaround is to use an extra RSL option: `-x (jobtype=condor)`

- There are different behaviors related with the standard output when the fork system call and Condor are used as job managers.
As reported in section 2.3, when the fork system call is used, if multiple jobs are submitted, and an output file is defined, the output of all these jobs is written in this file.
There is a different behavior for jobs submitted to Condor.

For example:

```
lxde01% globusrun -r lxde16.pd.infn.it/jobmanager-condor -f file.rsl
```

file.rsl:

```
&
(executable=$(HOME)/ciao.condor)
(stdout=/tmp/output)
(jobtype=condor)
(count=10)
```

ciao.condor is a standard Condor job (linked with the Condor library) that just prints "Ciao" in the standard output.

In this case the output file contains just a single "Ciao" (the output of a single job).

There is the same behavior if the jobs are submitted as batch jobs.

- There is a problem related with the passing of parameters:

```
lxde01% globusrun -b -r lxde16.pd.infn.it/jobmanager-condor -f file.rsl
```

file.rsl:

```
&
(executable=$(HOME)/Condor/ciao.condor)
(stdout=/tmp/output)
(stderr=/tmp/error)
(jobtype=condor)
(count=10)
(arguments="first parameter" "second parameter")
```

The system detects 4 arguments ("first", "parameter", "second", "parameter") instead of 2 arguments ("first parameter", "second parameter").

Using the fork system call, the passing of arguments works fine.

3.5 Interaction with GIS

Once the Condor service has been configured in the gatekeeper of the front-end machine of the Condor pool, a new entry, associated to this PC, has automatically been created in the GIS, as represented in figure 3.

```
queue=intel-linux, service=jobmanager-condor, hn=lxde16.pd.infn.it, Ou=Sezione di Padova, o=Istituto Nazionale di
Fisica Nucleare, o=Globus, c=US
◦ createtimestamp: 20000505110753Z
◦ creatorsname: cn=Directory Manager,ou=Sezione di Padova,o=Istituto Nazionale di Fisica
  Nucleare,o=Globus,c=US
◦ dispatchtype: batch
◦ freenodes: 50
◦ jobwait: NULL
◦ lastupdate: Thu Jun 01 13:37:54 GMT 2000
◦ maxcount: 0
◦ maxcputime: 0
◦ maxjobsinqueue: 0
◦ maxrunningjobs: 0
◦ maxsinglememory: 0
◦ maxtime: 0
◦ maxtotalmemory: 0
◦ modifiersname: cn=Directory Manager,ou=Sezione di Padova,o=Istituto Nazionale di Fisica
  Nucleare,o=Globus,c=US
◦ modifytimestamp: 20000601123906Z
◦ objectclass: GlobusTop GlobusQueue
◦ priority: NULL
◦ queue: intel-linux
◦ schedulerspecific: NULL
◦ status: 0
◦ totalnodes: 69
◦ tti: 00:01:00
◦ whenactive: 0
```

Figure 3

totalnodes represents the number of machines of the Condor pool with the same architectures and operating system of the front-end machine (in this case, since *lxde16.pd.infn.it*, the front-end machine, is a Linux Intel PC, this parameter represents the number of Linux Intel PCs belonging to the considered Condor pool). The machines belonging to the Condor pool, with different architectures and operating systems are not taken into account.

freenodes represents the number of unclaimed machines (machines not used by their owners, and not running Condor jobs) of the Condor pool with the same architecture and operating system of the front-end machine (in this case Linux Intel PCs).

The meaning of the parameters *maxcount*, *maxcputime*, *maxjobsinqueue*, *maxrunningjobs*, *maxsinglememory*, *maxtime*, *maxtotalmemory* is not clear: these parameters are always equal to 0, even if there are running jobs.

These information are not enough to describe the characteristics and the status of the Condor pool: reporting only the total number of machines, and the number of idle machines, could be useful only if the Condor pool is composed by an homogeneous set of computers, with the same characteristics and performance. Other information (e.g. the total CPU power) of the pool should be provided to the GIS server. Moreover other information related with the jobs (running and pending jobs) should be published in the GIS.

This and other information are necessary, for example, to implement a Resource Broker.

4 Submitting Condor jobs to Globus resources

There are two ways to run programs on Globus resources, using Condor. The first method is by using the Globus universe (as opposed to the standard or vanilla universe), while the second one involves doing the GlideIn mechanisms.

4.1 Globus universe

Doing so effectively has Condor fire up *globusrun* behind the scenes: the *condor_submit* command is simply "translated" to *globusrun*.

The configuration, represented in figure 4, has been considered.

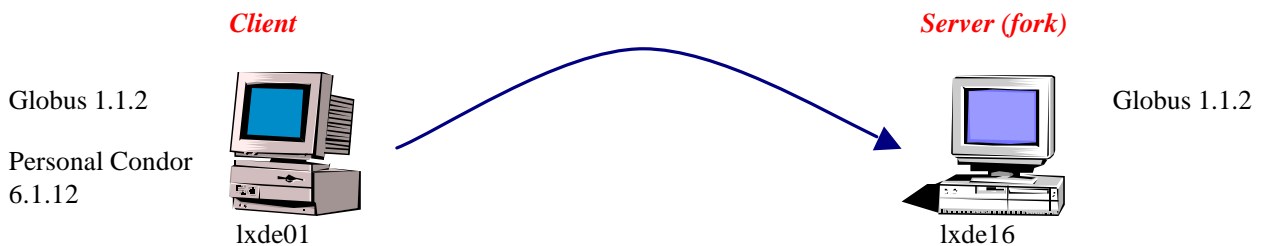


Figure 4

A PC (*lxde01.pd.infn.it*) running Globus 1.1.2 has been configured as Personal Condor, that is a Condor pool composed by a single workstation (Condor release 6.1.12 has been considered). An other PC (*lxde16.pd.infn.it*) running Globus 1.1.2 as been used as a server machine (considering the fork system call as job manager).

Some (10) simple jobs have been submitted from the client machine to the server machine, considering the following simple condor submit file:

```
Universe = globus
Executable = /users/noi/sgaravat/CondorGlobus/ciao
output = /users/noi/sgaravat/CondorGlobus/out.$(Process)
error = /users/noi/sgaravat/CondorGlobus/err.$(Process)
log = /users/noi/sgaravat/CondorGlobus/log.$(Process)
Getenv = True
GlobusScheduler = lxde16.pd.infn.it/jobmanager-fork
queue 10
```

There are some problems with this test, that we are trying to debug with the help of the Condor team: the gatekeeper on the remote machine (*lxde16.pd.infn.it*) is contacted, the jobs seem running, but they are not actually in execution.

As reported by the Condor team, they are currently redesigning this architecture to be more intelligent in dealing with Globus and more user-friendly.

Personal Condor, considering the Globus Universe, could be used in a workload management system to manage the queue of the submitted jobs, to keep track of their status, etc... Moreover it could provide features and capabilities of the Condor system, such as robustness and reliability.

Personal Condor could then be “interfaced” with a Master, smart enough to decide in which Globus resource (typically a PC farm managed by a resource management system such as LSF, PBS, Condor, etc..) the jobs must be submitted taking into account the status and characteristics of these resources (the Grid Information Service should be able to provide these information), in order to optimize the usage of these resources.

4.2 Condor glidein

With GlideIn, what happens is that the Condor daemons (specifically, the *master* and the *startd*) are effectively run on Globus resources. These resources then temporarily (the Condor daemons exit gracefully when no jobs run for a configurable period of time: the default length of time is 20 minutes) become part of the considered Condor pool.

It is possible to run vanilla or standard condor jobs on Globus resources via the GlideIn procedure.

To use this technology it is necessary a perl script (*condor_glidein*) and a special Globus ftp program (which will probably become part of the standard Globus distribution). The setup phase of GlideIn copies the necessary condor binaries from a server in Wisconsin to the machine to be glided in to, so the users don't have to worry about having the Condor binaries for all the architectures of machines that they want to glide in to. Before a user can access the server in Wisconsin, he needs to mail his X509 certificate name to the Condor team so that they can grant access to him (adding the entry in their *grid-mapfile*).

The information necessary to setup the GlideIn mechanisms can be found in the Condor Manual [4].

The GlideIn mechanisms have been evaluated considering the layout represented in figure 5.

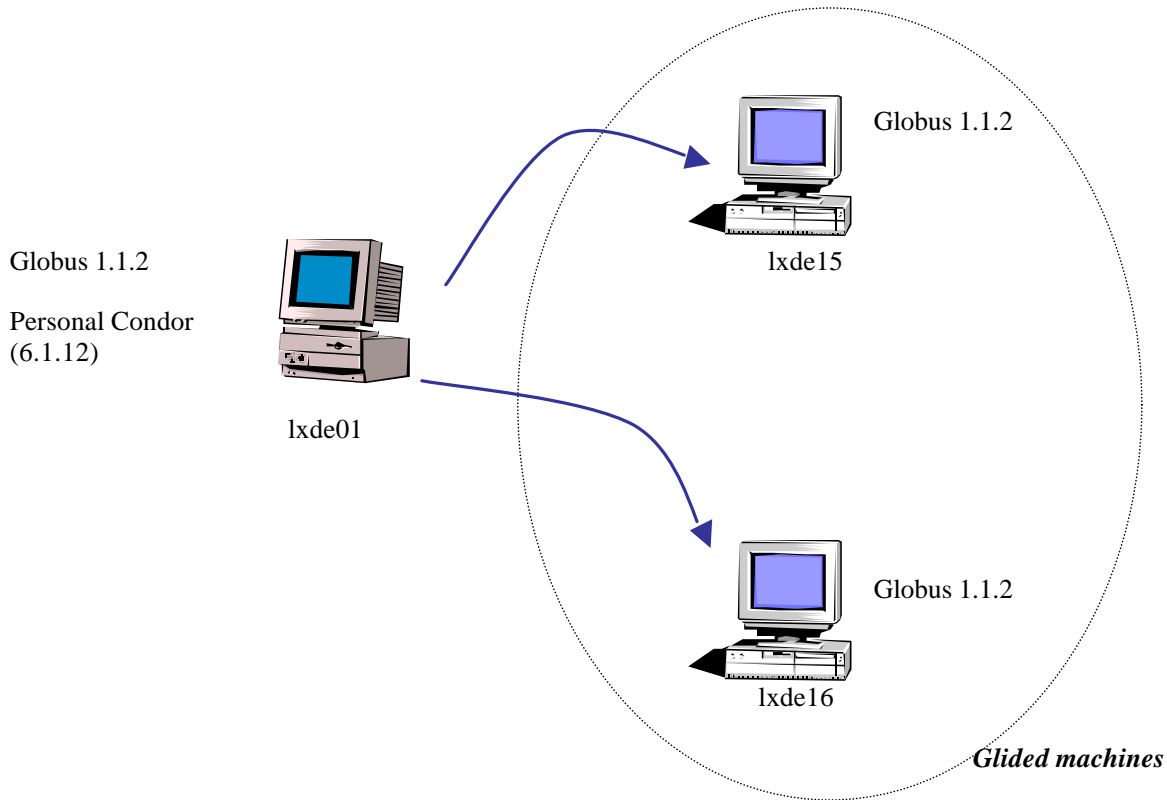


Figure 5

A PC (*lxde01.pd.infn.it*) running Globus 1.1.2 has been configured as Personal Condor (Condor release 6.1.12 has been considered). Other 2 PCs (*lxde15.pd.infn.it* and *lxde16.pd.infn.it*), running Globus 1.1.2 have then been glided in to.

In the beginning the Condor pool was composed by a single PC (*lxde01.pd.infn.it*):

lxde01% condor_status

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
<i>lxde01.pd.inf</i>	LINUX	INTEL	Unclaimed	Idle	0.090	251	0+00:00:08

Machines Owner Claimed Unclaimed Matched Preempting

INTEL/LINUX	1	0	0	1	0	0
Total	1	0	0	1	0	0

Then, using condor_glidein, *lxde16.pd.infn.it* has been joined to the Condor pool:

lxde01% condor_glidein lxde16.pd.infn.it

This procedure checks if the user has permission to use the remote node as a resource (therefore it is necessary to have a valid proxy and be mapped to a valid login account) and then checks if the Condor configuration files and executables are correctly located in the remote machine (*lxde16.pd.infn.it*); if not, they are downloaded from a server in Wisconsin.

The Condor executables are then run, and the machine joins the pool:

lxde01% condor_status

Name	OpSys	Arch	State	Activity	LoadAv	Mem	ActvtyTime
<i>lxde01.pd.inf</i>	LINUX	INTEL	Unclaimed	Idle	0.040	251	0+00:05:08
23658@ <i>lxde16</i> .	LINUX	INTEL	Unclaimed	Idle	0.020	251	0+00:03:04

Machines Owner Claimed Unclaimed Matched Preempting

INTEL/LINUX	2	0	0	2	0	0
Total	2	0	0	2	0	0

Then also *lxde15.pd.infn.it* has been joined to the Condor pool:

lxde01% condor_glidein lxde15.pd.infn.it

lxde01% condor_status

<i>Name</i>	<i>OpSys</i>	<i>Arch</i>	<i>State</i>	<i>Activity</i>	<i>LoadAv</i>	<i>Mem</i>	<i>ActvtyTime</i>
<i>lxde01.pd.inf</i>	<i>LINUX</i>	<i>INTEL</i>	<i>Unclaimed</i>	<i>Idle</i>	<i>0.100</i>	<i>251</i>	<i>0+00:10:08</i>
<i>25781@lxde15</i>	<i>LINUX</i>	<i>INTEL</i>	<i>Unclaimed</i>	<i>Idle</i>	<i>0.100</i>	<i>251</i>	<i>0+00:00:04</i>
<i>23658@lxde16</i>	<i>LINUX</i>	<i>INTEL</i>	<i>Unclaimed</i>	<i>Idle</i>	<i>0.050</i>	<i>251</i>	<i>0+00:07:45</i>

Machines Owner Claimed Unclaimed Matched Preempting

<i>INTEL/LINUX</i>	<i>3</i>	<i>0</i>	<i>0</i>	<i>3</i>	<i>0</i>	<i>0</i>
<i>Total</i>	<i>3</i>	<i>0</i>	<i>0</i>	<i>3</i>	<i>0</i>	<i>0</i>

Considering this Condor pool composed by 3 PCs, it has been possible to submit jobs. Both standard Condor jobs and vanilla jobs have been successfully submitted and executed.

Even in this case the feeling is that the current form of this technology is not too much "useful". The only advantage seems to be that it is not necessary to install Condor on the remote resource (but it is necessary to install Globus !).

As reported by the Condor team, they are currently working on making things more automatic: the goal is to just give a list of Globus resources, and Condor will have to "GlideIn" to them when there are idle jobs.

5 Using LSF as underlying resource management system for Globus

5.1 Configuration

In our test, as represented in figure 6, a PC (*lxde15.pd.infn.it*) running Globus 1.1.2 and LSF 4.0 has been configured as front-end machine to a small LSF test cluster, composed by other 2 Linux PCs (*lxde02.pd.infn.it* and *lxde03.pd.infn.it*). A queue (*mqueue*) has been configured to submit jobs on this LSF cluster.

An other PC (*lxde01.pd.infn.it*), running Globus 1.1.2, has been used as client machine.

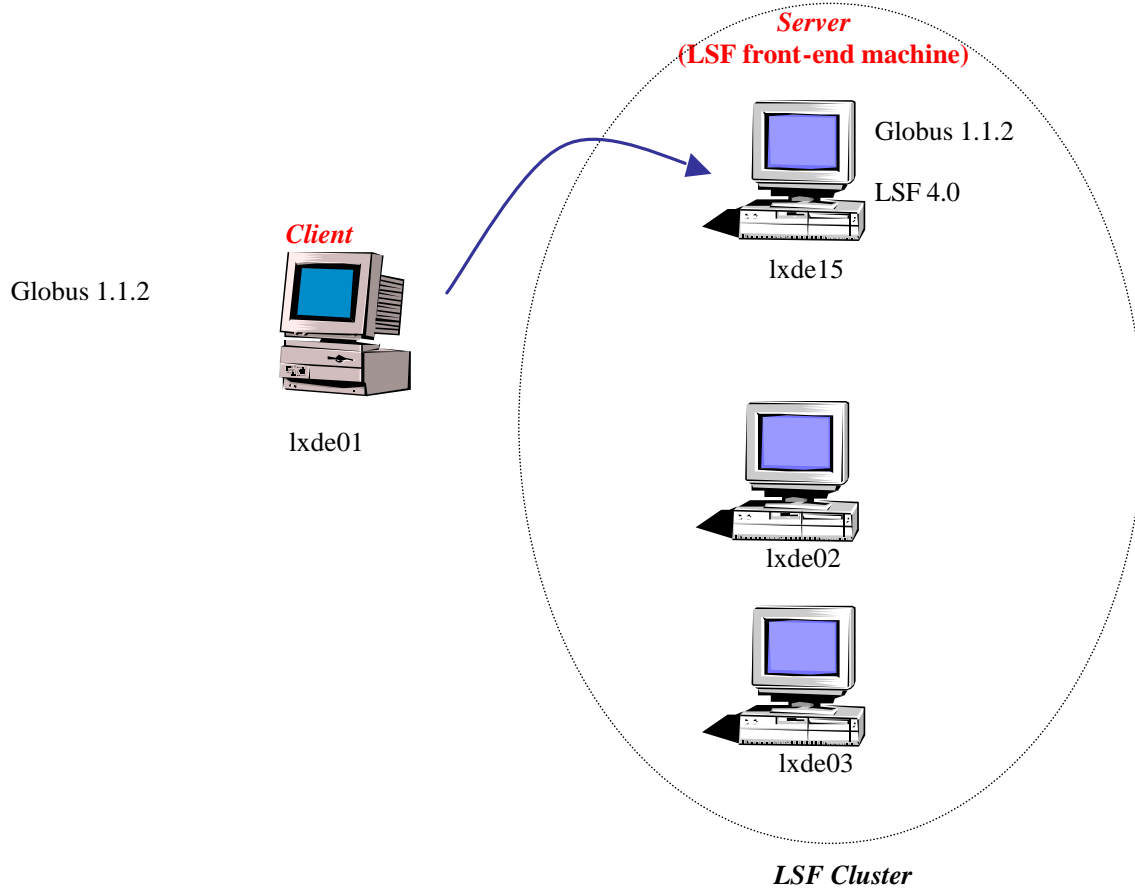


Figure 6

To configure LSF as a job manager for Globus, it has been necessary just to follow the instructions reported in the “Globus Toolkit System Administration Manual” (Chapter 5, “Configuring the Globus Toolkit Gatekeeper and Services”).

5.2 Globus RSL and LSF resource specification language

To analyze how the RSL expressions are translated in the resource specification language used by LSF, it is useful to edit the file:

```
<deploy-dir>/libexec/globus-script-lsf-submit
```

and comment out the line near the end that removes the file:

```
}${rm} $LSF_JOB_SCRIPT
```

After the submission of a job (using globusrun, or globus-job-run, or globus-job-submit), a file:

*/tmp/lsf_job_script.**

is created: this file contains the *bsub* LSF command used to submit the job.

Even for LSF, only the set of common RSL attributes is considered, and therefore it is not possible to use some options of the *bsub* command to require some specific operations. For example, it is not possible to use the option *-m*, to submit a job to specific hosts, or the option *-R*, for specific resource requirements. In any case these limitations are not so serious as for Condor: users usually just define the queue where to submit their jobs, without additional specific requirements, since usually for LSF all the requirements and policies are defined in the queues.

5.3 Usage examples

- **lxde01%** globus-job-run lxde15.pd.infn.it/jobmanager-lsf -q mqueue -stdout -l /tmp/risultato \ -s /tmp/abc/ciao one two

With this command a job (*ciao*, that takes two arguments: *one* and *two*) is run in the queue *mqueue* of the LSF cluster, using *lxde15.pd.infn.it* as front-end machine.

The executable file is stored in the file system of the submitting machine (option *-s*).

The output file is created in the file system of the executing machine (option *-l*).

The prompt is returned to the user when the job has been completed.

- **lxde01%** globusrun -b -r lxde15.pd.infn.it/jobmanager-lsf -f file.rsl

file.rsl:

```
&
(executable=$(HOME)/Condor/ciao)
(stdout=/tmp/output1)
(stderr=/tmp/error1)
(queue=mqueue)
```

As before, with this command a job is submitted to the LSF queue *mqueue*, using *lxde15.pd.infn.it* as front-end machine.

The executable file is stored in the file system of the executing machine.

The output file is created in the file system of the executing machine.

The job is submitted in background: the prompt is returned to the user immediately.

- **lxde01%** globus-job-submit lxde15.pd.infn.it/jobmanager-lsf -q mqueue \ -stdout -l /tmp/risultato -l /users/noi/sgaravat/LsfGlobus/ciao one two

This command submits one job (*ciao*, that takes *one* and *two* as arguments) to the LSF queue *mqueue*, using *lxde15.pd.infn.it* as front-end machine.

The executable file is stored in the file system of the executing machine.

The output file is created in the file system of the executing machine.

Note that, as reported in section 2.3, with the command *globus-job-submit* it is not possible to consider an executable file stored in the submitted machine, or create an output file in the file system of the submitting machine (with the option *-s*).

5.4 Problems, bugs and missing functionalities

- Using LSF as underlying resource management system for Globus, it is not possible to submit multiple instances of the same job.

For example the following command:

```
lxde01% globus-job-run lxde15.pd.infn.it/jobmanager-lsf -q mqueue -count 2 \  
-stdout -l /tmp/risultato -s /tmp/abc/ciao one two
```

doesn't submit 2 instances of job *ciao*, as expected: the option *-count* is translated in the LSF option *-n*, used to specify the number of processors for parallel jobs.

- As for Condor, there are some problems with the passing of parameters:

```
lxde01% globusrun -b -r lxde15.pd.infn.it/jobmanager-lsf -f file.rsl
```

file.rsl:

```
&  
(executable=$(HOME)/LsfGlobus/ciao)  
(stdout=/tmp/output)  
(stderr=/tmp/error)  
(queue=mqueue)  
(arguments="first parameter" "second parameter")
```

The system detects 4 arguments (“first”, “parameter”, “second”, “parameter”) instead of 2 arguments (“first parameter”, “second parameter”).

5.5 Interaction with GIS

Once the LSF job manager has been configured on the front-end machine, in the GIS a new entry is created for each queue of the LSF cluster.

The figure 7 represents one of these entries, associated to a LSF queue.

```

queue=mqueue, service=jobmanager-lsf, hn=lxde15.pd.infn.it, Ou=Sezione di Padova, o=Istituto Nazionale di Fisica
Nucleare, o=Globus, c=US
  ◊ createtimestamp: 20000518060413Z
  ◊ creatorsname: cn=Directory Manager,ou=Sezione di Padova,o=Istituto Nazionale di Fisica
  Nucleare,o=Globus,c=US
  ◊ dispatchtype: batch
  ◊ freenodes: 0
  ◊ jobwait: NULL
  ◊ lastupdate: Thu Jun 01 13:54:16 GMT 2000
  ◊ maxcount: 0
  ◊ maxcputime: 0
  ◊ maxjobsinqueue: 0
  ◊ maxrunningjobs: 2
  ◊ maxsinglememory: 0
  ◊ maxtime: 0
  ◊ maxtotalmemory: 0
  ◊ modifiersname: cn=Directory Manager,ou=Sezione di Padova,o=Istituto Nazionale di Fisica
  Nucleare,o=Globus,c=US
  ◊ modifytimestamp: 20000601125527Z
  ◊ objectclass: GlobusTop GlobusQueue
  ◊ priority: 30
  ◊ queue: mqueue
  ◊ schedulerspecific: NULL
  ◊ status: Open:Active
  ◊ totalnodes: 3
  ◊ ttl: 00:01:00
  ◊ whenactive: 0

```

Figure 7

totalnodes represents the number of machines of the LSF cluster associated to this queue.

priority and *status* represent the priority and the status of this LSF queue.

maxrunningjobs represents the number of running jobs in this queue.

The meaning of the parameters *freenodes*, *maxcount*, *maxcputime*, *maxjobsinqueue*, *maxsinglememory*, *maxtime*, *maxtotalmemory* is not clear: these parameters are always equal to 0, even if there are running jobs.

Even in this case this information is not enough to describe the characteristics and the status of the LSF cluster: for example the number of machines running jobs is not published. For LSF, unlike Condor, the information regarding the number of running jobs is reported.

In any case, as for Condor, other important information should be reported in the GIS server, in order to be able to use them for example to implement a resource broker.

6 RSL

The Globus Resource Specification Language (RSL) should provide a common interchange language to describe resources. The RSL provides the skeletal syntax used to compose complicated resource descriptions, and the various resource management components introduce specific <attribute, value> pairings into this common structure. Each attribute in a resource description serves as a parameter to control the behavior of one or more components in the resource management system.

A set of common RSL attributes is defined:

- **arguments:** the command line arguments for the executable
- **count:** the number of executions of the executable
- **directory:** specifies the default directory for the job manager
- **executable:** the name of the executable file to run

- **environment**: the environment variables that will be defined for the executable
- **jobType**: specifies how the job manager should start the job (single, multiple, mpi, condor)
- **maxTime**: the maximum walltime or cputime for a single execution of the executable
- **maxWallTime**: explicitly sets the maximum walltime for a single execution of the executable
- **maxCpuTime**: explicitly sets the maximum cputime for a single execution of the executable
- **gramMyjob**: specifies how the GRAM myjob interface will behave in the started processes
- **stdin**: the name of the file to be used as standard input
- **stdout**: the name of the file to store the standard output from the job
- **stderr**: the name of the file to store the standard error
- **queue**: targets the job to a queue name as defined by the scheduler at the defined resource
- **project**: targets the job to be allocated to a project account at the defined resource
- **dryRun**: used for debugging
- **maxMemory**: specifies the maximum amount of memory required for this job
- **minMemory**: specifies the minimum amount of memory required for this job

Therefore the RSL aims to be a uniform language to specify resources, between different environments and different resource management systems: this functionality is very important for a workload management system for a GRID environment.

The RSL syntax model (where the core element is the relation, and it is possible to define compound requests) seems suitable to define even complicated resource specification expressions. Unfortunately, as described later (in particular in section 3.2) the common set of RSL attributes is sometimes too “poor” to define the resources required for a specific job, in particular when the GRAM service uses an underlying resource management system. It should be useful if the attributes that are not in the common were passed unchanged to the underlying resource management system, and not simply ignored.

Moreover, the language should be more flexible and extensible: the administrator of a resource should be able to define new attributes describing particular (static or dynamic) characteristics of the considered resource, and the users should be allowed to take into account these new attributes, in their resource specification expressions. (for example this functionality is implemented in the Condor ClassAds architecture).

It should also be investigated if it is feasible and suitable using the same language to describe the offered resources and the requested resources, as for Condor.

7 Conclusions and future activities

In this note the preliminary activities related with the evaluation of the Globus GRAM service has been presented.

Simple tests have been performed, submitting jobs on remote resources, considering as first phase only the fork system call as job manager.

As second step the functionalities of the GRAM service have been tested considering Condor and LSF as underlying resource management systems.

The Globus RSL has been evaluated as well. This language aims to be a uniform language to describe resources between different resource management systems (a very important functionality for a workload management system that must manage resources spread across different domains). While the RSL syntax model seems suitable to define even complicated resource specification expressions, the common set of RSL attributes is often insufficient to define the resources required by a specific job. A missing functionality is that the attributes not belonging to this set are not passed through unchanged. This can be a serious problem in particular when the GRAM service relies upon a heterogeneous Condor pool (like the INFN WAN Condor pool), composed by machines of different architectures, different characteristics, without a common file system and UID domain. The problems are not so serious using LSF as underlying

resource management system, since usually in this case all the policies and requirements are defined in the queues.

The interaction between the GRAM and the GIS (Grid Information Service) has been analyzed, considering both Condor and LSF: the local GRAMs are able to provide the GIS with some basic information about the characteristics and status of the local resources, but it is necessary to improve these functionalities, publishing other important information, required for example to implement a resource broker.

Tests have also been performed on the possibility to submit Condor jobs to Globus resources, considering both the Globus universe and the GlideIn mechanisms: the ongoing activities related with these items are quite promising, but now only prototypes of these technologies have been implemented.

The described tests are preliminary activities: it will be necessary to keep working on these items in the next months.

For example it will be necessary to evaluate the GRAM service in conjunction with other resource management systems (e.g. PBS).

Some tests will also be performed considering applications that use directly the GRAM service, using the provided API.

An other important foreseen activity is the use of the GRAM service in production environments, considering real HEP applications (e.g. the HLT activities).

Work will be performed in an incremental fashion. As first step the GRAM service will be used simply as a uniform interface between different resource management systems (in particular local uniform Condor pools, and LSF clusters) that manage farms of PCs. As second step, the GRAM service will be used to submit jobs on remote (located in different sites) farms. Then a high throughput broker, able to optimize the usage of the computational resources, must be integrated in the Globus resource management architecture. The activities related with the integration between Globus and Condor will continue, for example to understand if Personal Condor can be configured as a high throughput broker for Globus resources.

New releases of the Globus software (starting with Globus 1.1.3) will be evaluated, to evaluated if they can provide new functionalities useful for our needs.

7 References

- [1] <http://www.pd.infn.it/~sgaravat/INFN-GRID/Globus>
- [2] <http://www.infn.it/condor>
- [3] http://www.globus.org/toolkit/documentation/sag_1_1_2_fnl.pdf
- [4] <http://www.cs.wisc.edu/condor/manual/v6.1>