

Subject: CONDOR-G

Author: Massimo Sgaravatto (Massimo.sgaravatto@pd.infn.it)

Partner: INFN Padova

Diffusion:

Information:

1 WHAT IS CONDOR-G

Condor-G is a Personal-Condor enhanced with Globus services.

Condor-G knows how to speak to Globus resources via GRAM, so it can be used to submit jobs from the user personal workstation to remote Globus resources, and have Condor keep track of their progress.

Figure 1 shows the architecture of Condor-G:

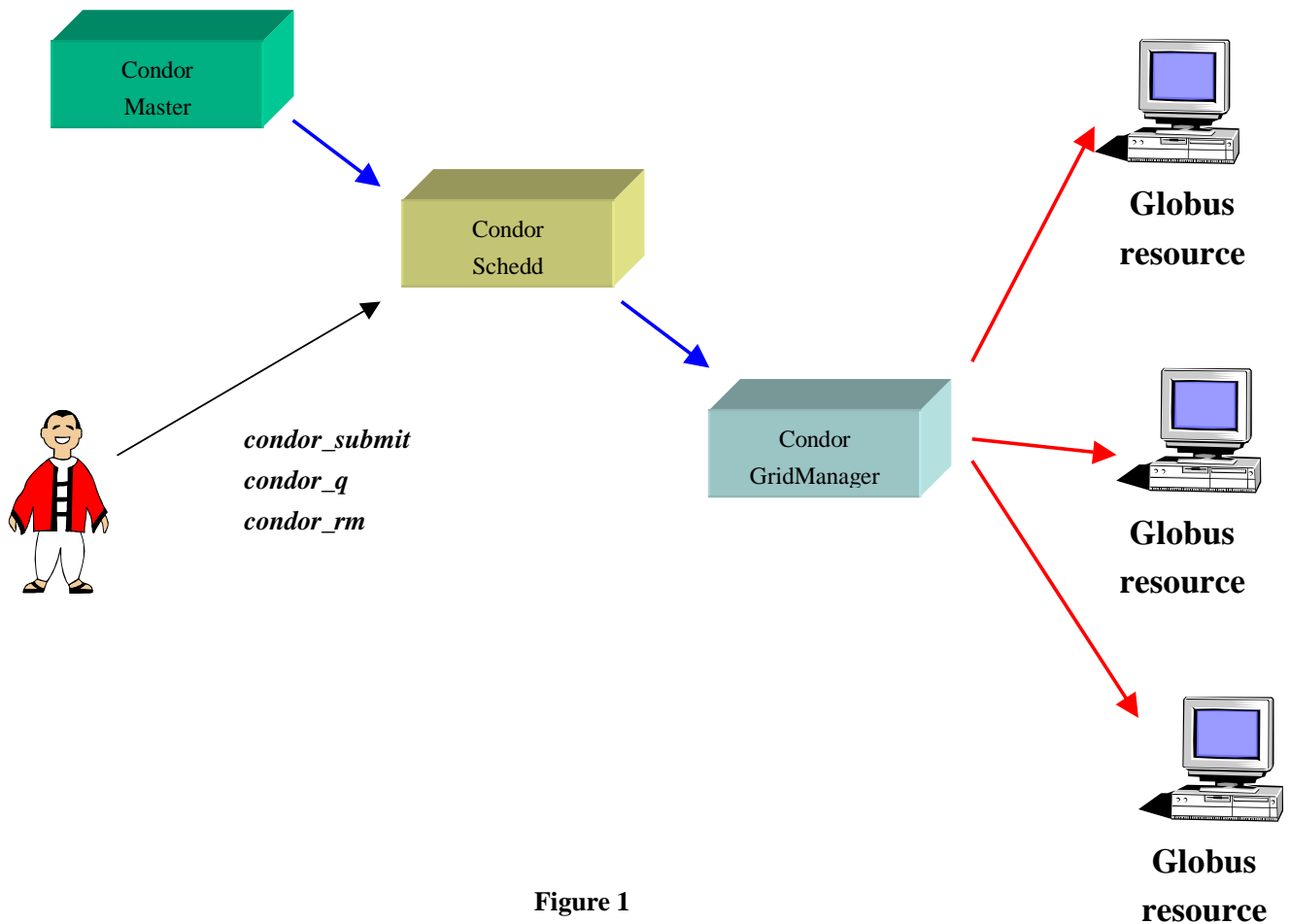


Figure 1

The *Condor Master* is the first process started: it creates the Personal Condor. All it really does is watch out over the other Condor daemons, and starts them up and shuts them down as appropriate.

The *Condor Schedd* keeps track of all of the jobs in the Personal Condor.

condor_submit is used to add jobs to the queue (that is to submit jobs to Globus resources), *condor_q* to check the status of the queue (that is to monitor the status of the jobs), and *condor_rm* to remove jobs from the queue.

The *Condor GridManager* is roughly the equivalent of the Condor shadow in the “standard” Condor environment: it is the interface between the Globus gatekeeper/jobmanager and the Personal Condor running on the user machine. There is one GridManager per user.

2 WHAT’S NEW IN CONDOR-G

The new Condor-G stuff (just released) has deeply changed with respect to the first prototype released some months ago.

Condor-G now uses the Globus GRAM API to “speak” to Globus: it doesn’t use anymore the Globus command line tools.

For what concerning the job status monitoring, Condor-G doesn’t rely anymore only on the polling mechanism used in the previous release (*globus-job-status* commands were issued every few seconds), but, on the other hand, it doesn’t rely only on the Globus callback mechanisms, considered not a reliable mechanism, since it is not possible to assume that there is always an entity (the *Condor GridManager*) ready to receive and manager these messages (for example the Condor-G machine could be down): every five minutes the *Condor GridManager* contacts the remote job manager, to check the status of the job.

If Condor-G loses track of a job (this happens when Condor-G can contact the gatekeeper, but not the job manager) since there’s no way for Condor-G to find out about what happened to that job, Condor-G considers this job as completed with exit status 1, to indicate this abnormal behavior (while for “normal” jobs an exit status 0 is reported). This will be enhanced when the new persistent job manager (developed by the Condor team) and the new *Condor GridManager* able to exploit it, will be released (see section 3).

If even the gatekeeper can’t be contacted, the job is kept in the queue, and Condor-G will try again later.

With this new version of Condor-G, just released, many problems found in the previous prototype release have been addressed, and some new features have been introduced as well:

- The scalability problem in the submitting side has been addressed, since there is not anymore a shadow process running for each submitted job, but now there is a *Condor GridManager* per user.
- The problem concerning wrong information reported in the log file has been addressed; now the log file reports:

- When a job has been submitted
- The IP address of the submitting machine (the machine running Condor-G)
- When the job has started its execution
- The IP name of the gatekeeper machine (that couldn't be the executing machine; for example let's consider as Globus resource a farm composed by many PC's, with a single PC configured as Globus front-end machine: the IP name of the front end machine is reported in the log file, while the job could have been dispatched to a different PC of this farm)
- When the job has completed its execution

It must be pointed out that, unlike normal Condor jobs, the log file for a Condor-G job doesn't report information about CPU usage and I/O activities.

- The *condor_q* command now reports correctly if a job is idle in the queue (waiting for a idle resource), or if it is running
- Using the GASS service it is possible to automatically stage the executable and the standard input file from the submitting machine to the executing machine, and the standard output and error files can be streamed back to the submitting machine. This is an option: the user decides in the condor submit file if the file system of the submitting machine or the executing machine must be considered for these files
- In the condor submit file it is now possible to use the *input*, *output*, *error* parameters to specify the standard input, output, error file names (while with the previous release it was necessary to use the *GlobusRSL* parameter)
- The *Getenv* and *Environment* attribute now work correctly
- With a single *condor_submit* command it is now possible to submit multiple instances of the same executable, specifying different input/output/error/log files (using *\$(Process)*) for the different instances.

It must be stressed that multiple independent jobs are submitted to the Globus resource (that is multiple job managers run in the gatekeeper machine).

3 NEW FEATURES FORESEEN FOR THE NEXT RELEASE

The Condor team is now implementing a new persistent Globus job manager.

With this new job manager, when a job is submitted to a Globus resource, it will be possible to specify in the corresponding RSL file the string:

(save_state=yes)

This means that, when the job is submitted, the contact string of the corresponding job manager and the id of the job in the underlying resource management system (e.g. the LSF job id) will be saved in a file.

If a job manager crashes, while the corresponding job is still running (this could happen for example if the gatekeeper machine [where the job manager runs] is rebooted, while the actual job has been dispatched to an other node), using the information saved in the log file it is possible to start a new job manager, able to “reattach” itself to the running job.

This can be done issuing a *globusrun* command, specifying in the corresponding RSL file:

(recover=ContactStringOfJobManager)

The new Condor-G software, which will be hopefully released in a few weeks, will be able to exploit this new persistent job manager: if the *Condor GridManager* didn’t receive a “completed” callback message from a job manager, but it can’t contact it, then the problem could be that the job manager crashed, while the job is still running. The new *Condor GridManager* will be able to exploit this new capability, restarting the job manager, and reattaching it to the running job.

It must be pointed out that the new persistent job manager can be used also without Condor-G; it is just necessary to specify the RSL string:

(save_state=yes)

when a job is submitted, and “save” the contact string of the corresponding job manager.

If later it is necessary to start a new job manager for this job, and reattach it to the running “orphan” job, a *globusrun* command must be issued, specifying the RSL attribute:

(recover=ContactStringOfJobManager)

Besides the exploitation of the new persistent job manager, the new Condor-G release will use a new two phase commit submission protocol, that will make sure that when a job is submitted via Condor-G, one and only one job is actually submitted (otherwise problems could arise for example if the acknowledgment from the job manager which specifies that the job has been submitted, is not received by the client).

4 EXAMPLES

Here is an example of a Condor-G submit file, used to submit 10 jobs to a remote LSF cluster, configured as Globus resource:

```
Universe = globus
TransferExecutable=True
Executable = /home/userx/startsim.sh
TransferInput=True
Input=/home/userx/inp.$(Process)
TransferOutput=False
Output = /data/out.$(Process)
TransferError=True
Error = /home/userx/error.$(Process)
Environment = CMSVER=118
Log = /home/userx/log.$(Process)
Arguments=123
GlobusRSL=(queue=cmsprod)
GlobusScheduler = pcmsfarm01.pi.infn.it/jobmanager-lsf
Queue 10
```

Queue 10 means that 10 jobs must be submitted. 10 independent jobs are submitted to the remote Globus resources, and therefore 10 job managers will run in the gatekeeper machine.

Note the use of *\$(Process)* for the standard input/output/error files, and for the log files: the first job will use as standard input the file */home/userx/inp.0*, save its standard output on the file */data/out.0*, write the standard error on */home/userx/error.0*, while the log information will be saved on */home/userx/log.0*.

For the second job the files: */home/userx/inp.1*, */data/out.1*, */home/userx/error.1*, */home/userx/log.1* will be used.

For the last (tenth) jobs the files */home/userx/inp.9*, */data/out.9*, */home/userx/error.9*, */home/userx/log.9* will be considered.

The executable and the standard input files are staged from the submitting machine to the executing machine (specified by the attributes *TransferExecutable* and *TransferInput*, defined as *True*). The standard error files are streamed back to the submitting machine (the attribute *TransferError* is defined equal to *True*), while the standard output files are saved in the file system of the executing machine, and not copied back (this is specified by the attribute *TransferOutput*, defined as *False*).

The log files are created in the submitting machine.

The *Environment* attribute is used to specify a logical variable that must be defined for these jobs, while *Arguments* specify the command line parameters for the executable.

GlobusScheduler is the contact string of the Globus resource, where the jobs must be submitted.

The parameter *GlobusRSL=(queue=cmsprod)* defines the LSF queue that must be considered.

This Condor-G submit file is the argument of the *condor_submit* command, used to actually submit the jobs to the remote Globus resource: the command must be issued after having acquired valid credentials (using the *grid-proxy-init* command).

The command *condor_q* can then be used to monitor the status of the jobs:

```
[sgaravat@lxde01]$ condor_q
```

```
-- Submitter: lxde01.pd.infn.it : <193.205.157.15:33012> : lxde01.pd.infn.it
```

ID	OWNER	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
7.0	sgaravat	3/26 14:08	0+00:00:00	R	0	0.0	startsim.sh 123
7.1	sgaravat	3/26 14:08	0+00:00:00	R	0	0.0	startsim.sh 123
7.2	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.3	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.4	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.5	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.6	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.7	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.8	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123
7.9	sgaravat	3/26 14:08	0+00:00:00	I	0	0.0	startsim.sh 123

10 jobs; 8 idle, 2 running, 0 held

When the jobs complete their execution, they leave the queue.

5 OPEN PROBLEMS AND ISSUES NOT ADDRESSED BY CONDOR-G

- If Condor-G is not able to submit a job to a Globus resource, the submission is tried again after 5 minutes. This event (and the reason of this job submission failure) is not recorded in the job log file (it is reported only in the *Condor GridManager* log file). Moreover, instead of simply trying the resubmission after 5 minutes, probably the user should be notified as well (with a message reporting the problem). The Condor team is going to address this problem.
- Condor-G is not able to discover if a job “disappears” without any exit job status (for example because of a crash of the executing machine), and the underlying resource management

system is not able to manage this situation and resubmit the job. This is because if a job in the underlying resource management system disappears, the correspondent job manager assumes that the job has been completed (it is not able to understand about the problem), and so a “completed” message is sent back to the grid manager.

- Since it is not possible to get back from Globus the exit status of the job, Condor-G can't tell this exit status. Actually, Condor-G now always tells that the job exited with status 0; as described in section 2, Condor-G reports '1' as exit status only if it loses track of the job.
- It is not easy to use Condor-G capabilities from applications, since there is not a Condor-G API (only command line tools are available).
- It is not possible to be asynchronously notified about job status transitions (e.g. idle → running → completed), with mechanisms such as callbacks.
However we must say that these events are recorded in the job log file, and a specific library, that could be used to “automate” the search of these events in the log file, is already available.
Moreover the standard distribution of Condor comprises a perl module that could be useful to monitor the submitted jobs, but with this module it is possible to monitor just one cluster of jobs at a time
- Condor-G doesn't deal with the expiration of the user proxy (e.g. it is not able to automatically renew it if it is expiring). There is just a parameter in the Condor-G configuration file, that defines a minimum lifetime for the proxy: the *Condor GridManager* will check to make sure that the user proxy has at least this much life left in it before it will submit the job.
- As described in section 2, the job log file reports as executing machine the name of the machine where the job manager is running (not the machine where the job is actually running), and the log file doesn't report information on resource (e.g. CPU, I/O) usage.
- In the log file it is not reported when a running job is suspended, and when it is restarted. This is because Globus can't detect these two job status transitions.
There is the same problem using *condor_q* command.
- It is not possible to move from/to the submitting machine to/from the executing machine other files besides the executable and the standard input/output/error files. The Condor team is planning to introduce this capability in Condor 6.3 for vanilla jobs, and hopefully it will then be included also in Condor-G.

6 CONDOR-G AS USER JOB SUBMISSION SERVICE

I think that Condor-G is a very useful job submission service, much better than the Globus command line tools (*globusrun*, *globus-job-submit*) to submit jobs to Globus resources, and I suggest to use it instead of relying on the “default” Globus command line tools.

Condor-G is not just an other way to submit jobs to Globus resources, “equivalent” to the Globus tools, but it introduces new useful functionalities and features, such as:

- Condor-G maintains a queue of the submitted jobs, and therefore it is possible to keep tracks of their progress.
- This queue is persistent, so for example everything will keep working after a reboot of the submitting machine.
- Condor-G provides logging information, that reports when the job has been submitted, has started, has completed its execution, and where the execution has taken place.
- Condor-G will be able to exploit the new persistent job manager, so it will be able to automatically recover from scenarios where “orphans” jobs are running without job managers taking care of them.
- Condor-G will use a more reliable submission protocol (a two phase commit protocol) than the one provided by default by Globus.

Everything that can be accomplished with the Globus tools can be done with Condor-G, with improved and new functionalities.

The only limitation is for interactive jobs, not managed by Condor-G.

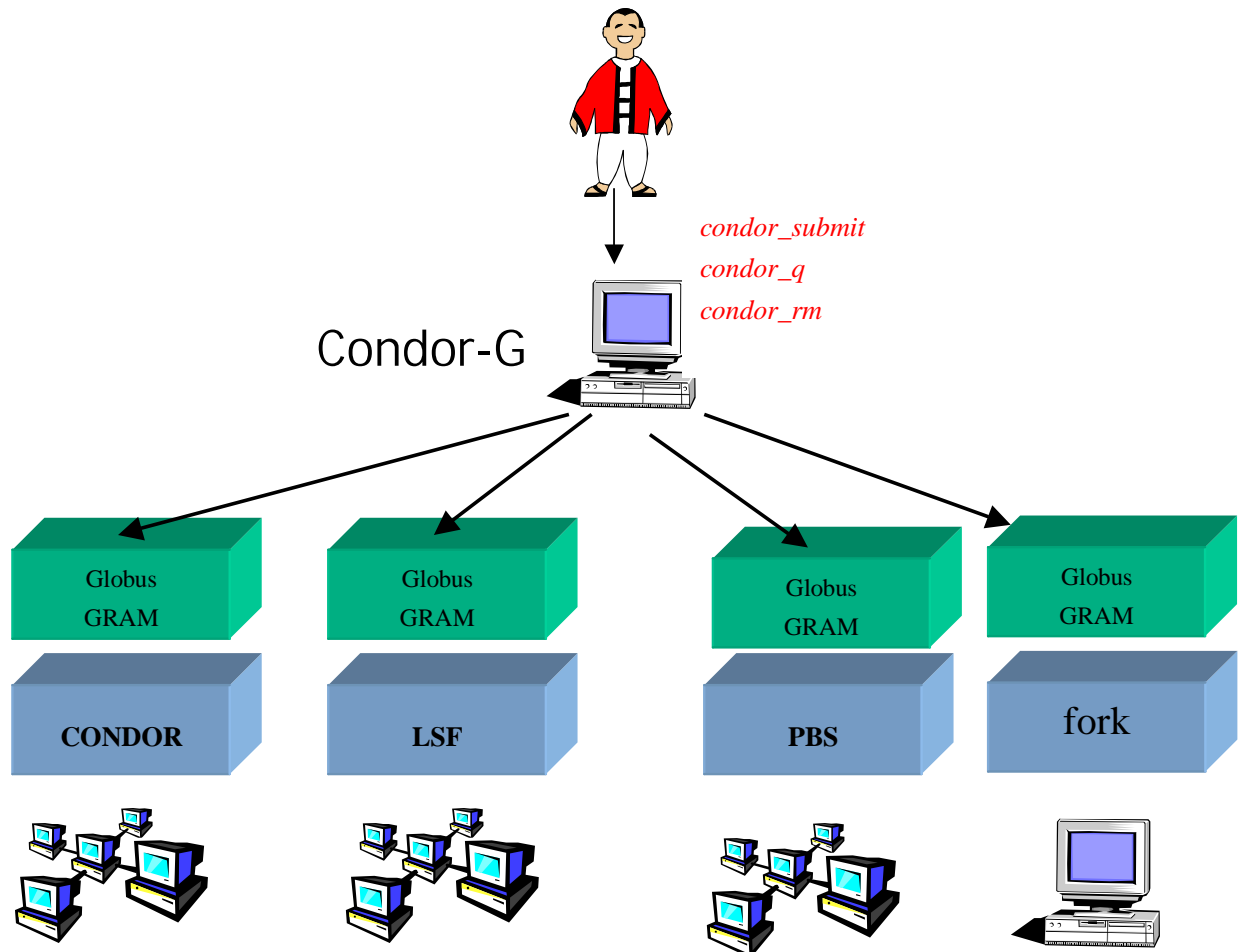


Figure 2: Condor-G as user job submission service

7 CONDOR-G AS JOB SUBMISSION SERVICE FOR THE WORKLOAD MANAGEMENT SYSTEM

The work package 1 of the DataGrid project is implementing a first prototype of a workload management system. A picture describing the building blocks of this system, the functional interactions among the various components, and the dependencies on “external” functionalities is available at <http://www.mi.infn.it/~prelz/grid>.

In short, users submit jobs, specifying the characteristics of these jobs and the required and preferable resources for their execution using a *Job Description Language (JDL)*, which will be based on Condor ClassAds. When a job is submitted, it is “inserted” in a queue of jobs in the resource broker. The resource broker chooses a suitable Globus resource for the job, taking into account the resources required and preferred by the job, and the available Grid resources (the information concerning the characteristics and status of Grid resources is published in the Grid Information Service). The JDL expression and the id of the Globus resource chosen by the resource broker is then passed to a Job Submission Service, responsible to actually submit the job to the chosen Globus resource. This job submission flow is represented in figure 3.

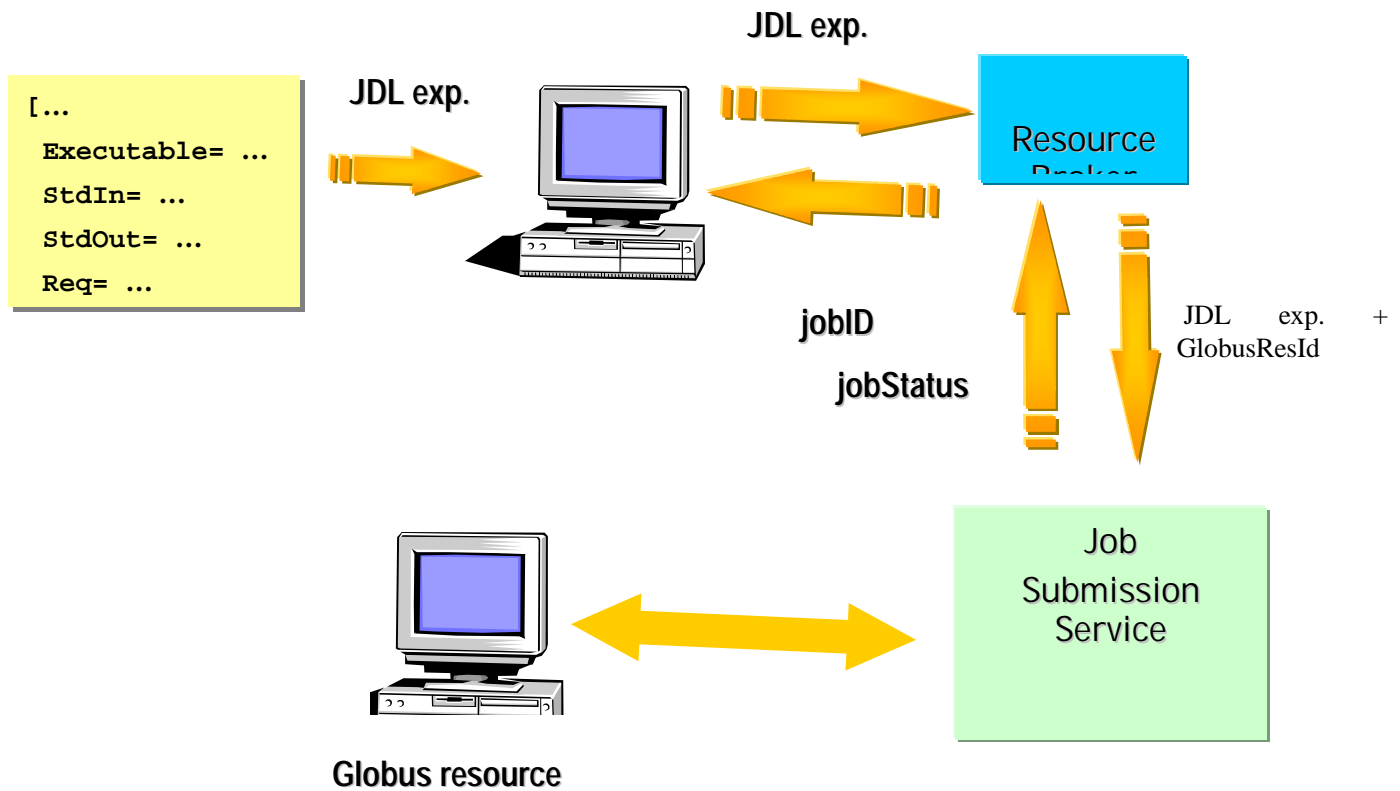


Figure 3: Job submission

While using Condor-G as a user job submission service, to submit jobs to specific Globus resources, instead of relying on the Globus command line tools, seems a very reasonable approach, it must be understood if using Condor-G as job submission service in this workload management system prototype, as initially planned, is still a viable solution, or it is better to rely on the Globus GRAM client API, since:

- as described in section 5, Condor-G is not able to understand if a job “disappeared” without an exit status (for example because of a crash of the executing machine and the underlying resource management system is not able to manage this problem [resubmitting the job], or for example when the fork system call is used as Globus jobmanager, and the executing machine crashes).

Question: do we really have to manage this scenario? Can't we assume that we are using a local resource management system able to manage these events? For example Condor provides this functionality: if a job “disappears” without any exit status, it is resubmitted. LSF provides this functionality as well (called “Automatic Job Rerun”). What about PBS and the other local resource management systems that we have to consider? Or, in general: should be up to the Grid Fabric to manage this issue?

- we are not going to exploit the “persistent queue of jobs” of Condor, necessary to keep tracks of the progress of jobs, since it came out that a persistent queue of submitted jobs will have to be implemented in our broker (a job is inserted in this queue when this job is submitted by the user, and before the broker has chosen the resource where this job should run).

Here are (some of) the pros and cons I see in using Condor-G as job submission service for our workload management system, as we planned:

Pros in using Condor-G:

- It will be possible to exploit the new reliable two-phase commit submission protocol
- It will be possible to automatically exploit the new persistent job manager
However it must be stressed that this new job manager could be used also without using Condor-G; “triggering” the creation of a new job manager that reattaches itself to a running job could be implemented without using Condor-G, since we are going to manage a queue of jobs in the broker (and in this queue for each job the contact string of the corresponding job manager could be saved), and we can therefore understand when we have lost track of a job
- The logging information, recorded in a persistent log file.
A library, that should ease the search of job status changes recorded in this log file, is also available

Question: who is going to use this log file? How?

Cons in using Condor-G:

- As described in section 5, the integration of Condor-G with applications (e.g. our broker) is not trivial, since an API, such as the one provided by the Globus toolkit, used to submit, cancel, monitor jobs, is not available
- As described in section 5, a callback mechanism, such as the one provided by the Globus GRAM client API, is not available, and therefore it is not possible to be asynchronously notified about job status transitions
- Condor-G is not an open source software, and it is not clear at the moment if we will be allowed to use it in the future, considering the policies of the DataGrid project

For what concerning the callback mechanisms, useful to be notified about job status changes, it is not good to rely only on callbacks to detect job status transitions: this is not a reliable mechanism, since in general it is not possible to assume that there is an entity on the other side always ready to receive and “manage” these messages. A mixed callbacks/”polling the status of the job” approach seems a better approach, and this “polling the status of the job” could be easily implemented for example through the *globus_gram_client_job_status* function, provided by the Globus GRAM client API.

8 OTHER POSSIBLE SCENARIOS

An other possible scenario, proposed by the Condor team, could be trying to “join” our broker with Condor-G (which doesn’t have any brokering functionality), in order to avoid rewriting from scratch mechanisms of Condor-G that are already in place and whose implementation could be not trivial (e.g. implementing and managing a persistent queue of submitted jobs, and keeping track of their progress).

However the possible interfaces between our broker and Condor-G have not been defined and are not clear at all.

As first step, in order to implement quite soon a workable, usable prototype system, building on what exists, and implementing just the missing functionalities, the Condor people proposed the system represented in figure 4.

Figure 4 **

Therefore Condor-G is the user interface to submit jobs (and therefore ClassAds are the Job Description Language in all effects).

What is different wrt to the current Condor-G implementation is that in the new “customized” Condor-G submit file the user doesn’t have to define the *GlobusScheduler* parameter, to specify the Globus resource where this job must be submitted.

The job is inserted in the Condor-G queue, with state 'HOLD'.

Then the broker must:

- pick up the jobs in the Condor-G queues with state 'HOLD'
- choose suitable resources for these job, finding a match between the required/preferable resources (specified in the ClassAds defined in the Condor-G submit file) and the available Grid resources (published in the Grid Information Service)
- update the *GlobusScheduler* ClassAd attribute.

so then Condor-G can actually submit the job to the specified Globus resource.

This system works if there is a broker, tied to Condor-G, in each submitting machine.

If we don't want to consider a broker for each submitting machine, and on the contrary we want to implement a central/community broker, the problem is that this broker must be able to find and contact the Condor-G job queues on the various user desktops, to find the jobs with status 'HOLD') that must be scheduled.

For this purpose the Condor collector could be used, as illustrated in figure 5: it can contact the various Condor Schedd, and find the jobs that must be scheduled.

This looks just like the "classic" Condor architecture, where the *Condor Negotiator* has been replaced by our broker!

The Condor team is willing to implement the required modifications of Condor-G if we are interested in this approach.

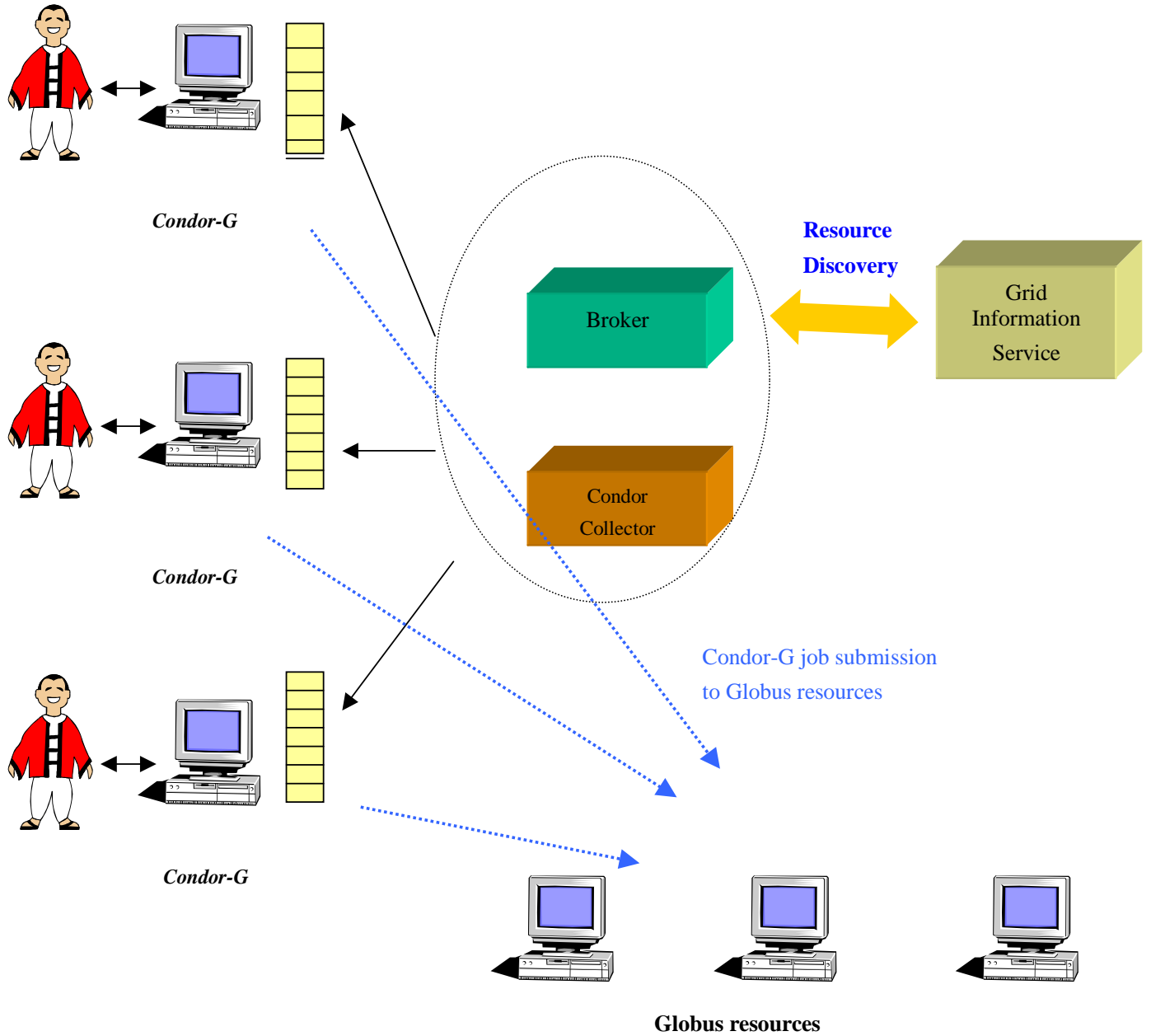


Figure 5