Using OSG Computing Resources with (iLC)Dirac

A Sailer¹, M Petric¹
¹CERN, 1211 Geneva 23, Switzerland
On behalf of the CLICdp collaboration
E-mail: andre.philippe.sailer@cern.ch, marko.petric@cern.ch

Abstract. CPU cycles for small experiments and projects can be scarce, thus making use of all available resources, whether dedicated or opportunistic, is mandatory. While enabling uniform access to the LCG computing elements (ARC, CREAM), the DIRAC grid interware was not able to use OSG computing elements (GlobusCE, HTCondor-CE) without dedicated support at the grid site through so called 'SiteDirectors', which directly submit to the local batch system. This in turn requires additional dedicated effort for small experiments on the grid site. Adding interfaces to the OSG CEs through the respective grid middleware is therefore allowing accessing them within the DIRAC software without additional site-specific infrastructure. This enables greater use of opportunistic resources for experiments and projects without dedicated clusters or an established computing infrastructure with the DIRAC software. To allow sending jobs to HTCondor-CE and legacy Globus computing elements inside DIRAC the required wrapper classes were developed. Not only is the usage of these types of computing elements now completely transparent for all DIRAC instances, which makes DIRAC a flexible solution for OSG based virtual organisations, but it also allows LCG Grid Sites to move to the HTCondor-CE software, without shutting DIRAC based VOs out of their site. In these proceedings we detail how we interfaced the DIRAC system to the HTCondor-CE and Globus computing elements and explain the encountered obstacles and solutions developed, and how the linear collider community uses resources in the OSG.

1. Introduction
The DIRAC (Distributed Computing with Remote Agent Control) [1] interware allows homogeneous access to heterogeneous resources, from laptops to batchfarms (e.g., LSF, Slurm, Torque) or gridsites using the ARC or CREAM middleware to clouds [2]. It offers a wide range of functionality for distributed computing tasks. These include a workload management system based on pilots [3], a data management system including meta data storage [4], and a transformation system for centralised productions. Initially created for LHCb computing needs, DIRAC has since been adopted by a wide range of users, in and outside of the high energy physics community [5, 6].

The iLCDirac extension of DIRAC was created for linear collider detector studies (CLICdp, ILD, SiD), which are combined in the ILC Virtual Organisation (VO). The iLCDirac extension contains interfaces to the linear collider software, a system to overlay background files, and extensions to run large scale productions for the linear collider detector collaborations [7].

The ILC VO is a merger of a more euro-centric LCG/EGI VO and an Open Science Grid (OSG) VO. Thus resources from both EGI and OSG were in principle available for the ILC users. In the past the access to OSG resources via DIRAC systems was either possible via the
glite-WMS, now no longer supported, or via a SiteDirector installed at the grid site. As the installation of a SiteDirector on-site requires dedicated support, it is not a viable solution for small VOs with the desire to make use of opportunistic resources. Therefore the interfaces to HTCondor-CEs and Globus computing element middlewares had to be implemented as DIRAC classes to make use of all available computing resources.

2. Computing Element Interface in DIRAC
In the DIRAC system, access to middlewares or batch systems is provided via classes inheriting from the ComputingElement base class. The computing element classes are only used for the pilot system. The matching of payload jobs to pilots is independent of the specific computing elements. The computing element classes need to fulfill the following interface:

- **submitJob**: Submit one or many pilots to a given computing element;
- **getJobStatus**: Give the status of an individual pilot on a computing element;
- **getCEStatus**: Get running and pending pilots at a computing element;
- **getJobOutput**: Get pilot output and error file;
- **killJob**: Kill the pilot job;
- **getPilotLoggingInfo**: Get logging information for a pilot job, e.g., submission time, start time, end time.

The submitJob method is used to send a pilot to a computing element. The getJobOutput and getPilotLoggingInfo methods are used to understand the execution of pilots mostly for debugging purposes. The getJobStatus, getCEStatus methods are used to monitor the number of pilots and their status at the computing elements. The number of pending and running pilots on each site is controlled to decide if more pilots should or can be submitted. The monitoring of aborted pilots lets the system assess the performance of grid sites.

3. HTCondor-CE in DIRAC
The HTCondor-CE middleware can be contacted via python interfaces or the command line. The current implementation of this class is using the command line commands. The access via these commands might not be as efficient or as powerful as the python interface, but on the other hand, by using the commands, debugging of the HTCondor-CE interface can be completely separated from DIRAC by simply running the commands outside the DIRAC system. A large part of the functionality of HTCondor, for example the workload management and matching of payloads, is done by the DIRAC system itself, which means that HTCondor is only used to submit the pilot jobs.

There are two ways to use the HTCondor-CEs in DIRAC: one can either run the HTCondor scheduler daemon locally on the DIRAC server or submit to the remote schedulers at the GRID sites. In the interface there are only small differences in the submission file and the options passed to the HTCondor commands. The benefit of the second approach is that no daemons need to be running on the DIRAC side, however the access to the pilot output files is not implemented in the DIRAC system at the moment.

3.1. Submission and Configuration
The submitJob method for the HTCondor-CE class is implemented by calling the command `condor_submit -terse subFile.jdl` together with a submission file. The output from the commands is a range of condor cluster and process IDs that is stored in DIRAC to later monitor the pilot status. The command has additional arguments if the remote schedulers are used and becomes `condor_submit -terse -pool %(ceName)s:9619 -remote %(ceName)s subFile.jdl`. 
Figure 1. HTCondor-CE submission file template used in DIRAC

Table 1. Assumed DIRAC pilot statuses and their equivalents from the HTCondor-CE and GlobusCE job status. For HTCondor-CE the numeric value is given in parentheses as well.

| DIRAC Status | HTCondor-CE Status | GlobusCE Status |
|--------------|--------------------|----------------|
| Scheduled    | Idle(1)            | Pending, Unsubmitted |
| Waiting      | Running(2)         | Running, Active, Stage_In, Stage_Out |
| Running      | Done(4)            | Done |
| Done         | Failed             | Failed, Suspended |
| Failed       | Aborted            | Removed(3) |
| Aborted      | Killed             | Held(5) |
| Killed       |                    | Cancelled |

The submission file template is shown in Figure 1. The placeholders (e.g., %(ceName)s) in the file are replaced at run-time. Most of the content of the submission file is HTCondor boilerplate. The executable for the job submitted to HTCondor is the pilot executable (DIRAC_XXXXXPilot.py), which is bundled with a proxy and created dynamically for each submission cluster. The submission file also sets an environment variable HTCONDOR_JOBID at run-time of the pilot that is used to identify which pilot a payload job is running under. The extraLine placeholder allows further configuration of the job submission, for example to set different periodic_remove commands for different sites, or request different number of CPUs. When the remote schedulers are used the targeted universe is vanilla instead of grid and the options ShouldTransferFiles and WhenToTransferOutput are omitted.

3.2. Monitoring
The getJobStatus method is implement via the condor_q and condor_history commands. These commands are used to obtain the status of pilots from HTCondor, which is then translated into the DIRAC equivalents of the status. Table 1 shows the HTCondor-CE job statuses and their DIRAC equivalent. The killJob method is implemented via the condor_rm command.

The getJobOutput method returns the content of the pilot stdout and stderr files to DIRAC. This method is only fully implemented for the case where DIRAC is using local HTCondor daemons. These files are automatically downloaded by HTCondor and passed into DIRAC on demand. The getPilotLoggingInfo method passes the HTCondor log to DIRAC.
3.3 Problems and Open Issues
Because the pilots and payloads software do not support check-pointing yet, if a pilot job is held it has to be killed. In case HTCondor is holding one of the pilots it will be killed by DIRAC. As the pilot jobs are cheap no attempt is made to recover a failed state. In fact it may be counterproductive if an aborted pilot is restarted in the HTCondor system as the restarted pilot might not be granted the same resources as when it was originally started. This re-started pilot will then most certainly not finish any payload successfully. For this purpose a periodic_remove statement is added via the extraLine to the HTCondor submission file. It was observed that a simple statement like NumSystemHolds > 0 was not preventing restarts of pilot jobs at some sites. Therefore a more complicated statement was needed which also checks that the JobStatus is not going back to the IDLE status (with the numeric value of 1) after it was matched to a resource in the HTCondor-CE system:

\[
\text{periodic}\_\text{remove} = (\ \text{NumSystemHolds} > 0) \ || \\
(\ \text{JobStatus} == 1 \ && \\
(\ \text{NumJobStarts} > 0) \ || \\
(\ \text{NumJobMatches} != \text{UNDEFINED} \ && \\
\text{NumJobMatches} > 0)))
\]

The getCEStatus method is currently implemented completely inside the DIRAC system. To obtain the number of submitted, pending, or running pilots DIRAC’s pilot database is queried for these numbers.

4. GlobusCE in DIRAC
Similarly to the implementation of the HTCondor-CE class, the link from DIRAC to GlobusCEs is implemented via commands.

4.1 Submission
The submitJob method for the Globus computing element class submits the DIRAC pilot executable (DIRAC_XXXXXPilot.py)

\[
globus\text{-}job\text{-}submit \%\text{(ceName)s/%(queueName)s -s DIRAC_XXXXXPilot.py}
\]

the ceName and queueName are replaced by their respective value for a given grid site, and the returned job identifier is stored in the DIRAC system for later use.

4.2 Monitoring
The getJobStatus method is implemented via the globus-job-status command and the GlobusCE job status is translated into a DIRAC pilot status as described in Table 1. As for the HTCondor-CE class the status of a CE is obtained from DIRAC’s pilot database. The killJob method is implemented with the globus-job-clean command.

The getJobOutput method returns the content of the pilot stdout and stderr files to DIRAC. The method is implemented with the globus-job-get-output command. There is no functionality equivalent to the getPilotLoggingInfo method in the GlobusCE system.

5. Usage of OSG Resources in (iLC)DIRAC
Since 2015 the OSG resources are seamlessly integrated into the iLCDirac system via the GlobusCE and HTCondor-CE computing element classes in DIRAC. As shown in Figure 2, from May 2015 until October 2016 about 12% of the CPU time used by the ILC VO were provided by resources which are part of the OSG. The largest provider of resources were the Fermilab and PNNL Grid sites, but in total 12 grid sites were available for use by the ILC
Figure 2. Cumulative CPU time used via the iLCDirac instance separated by Grid.

Figure 3. Cumulative CPU time on the individual OSG grid sites available to the ILC VO.

VO (see Figure 3). Apart from the tweaking of the periodic_remove statement no issues were encountered. The OSG developers were also very helpful by adding support for the ilc.desy.de cvmfs repository to the OSG software stack.

Most of the grid sites are running the HTCondor-CE grid middleware. The number of GlobusCEs is in decline since the HTCondor-CE software became available. In fact there is only one site left that supports the ILC VO and runs a computing element based on the Globus middleware.

6. Summary and Conclusions
The design of the DIRAC framework allowed for an easy integration of the OSG computing resources not only for the iLCDirac extension, but for all users of DIRAC instances. Other users of the HTCondor-CE class include the LHCb VO which is using it to access the HTCondor-CE computing element at CERN and the SNO+VO is using some OSG resources via the UK GridPP DIRAC instance [6]. This also allows sites currently running ARC or CREAM middlewares to switch to the HTCondor-CE software without locking DIRAC users out of their resources.

Acknowledgements
Thanks to Iain Steers (CERN) for his invaluable support during the initial development of the HTCondor-CE DIRAC class, the PNNL staff for the help in implementing the GlobusCE interface, and the HTCondor team and the DIRAC developers for their advice and support.

References
[1] Tsaregorodtsev A et al. 2008 J. Phys. Conf. Ser. 119 062048
[2] Fifield T, Carmona A, Casajs A, Graciani R and Sevior M 2011 J. Phys. Conf. Ser. 331 062009
[3] Stagni F, Tsaregorodtsev A, McNab A and Luzzi C 2015 J. Phys.: Conf. Ser. 664 062061
[4] Tsaregorodtsev A and Poss S 2012 J. Phys. Conf. Ser. 396 032108
[5] Stagni F, Sailer A, Tsaregorodtsev A, Arrabito L, Hara T and Zhang X 2016 J. Phys. Conf. Ser. These Proceedings
[6] Bauer D and Fayer S 2016 J. Phys. Conf. Ser. These Proceedings
[7] Grefe C, Poss S, Sailer A and Tsaregorodtsev A 2013 J. Phys. Conf. Ser. 513 032077