UNIVERSITY
of
GLASGOW

Department of Physics & Astronomy
Experimental Particle Physics Group
Kelvin Building, University of Glasgow,
Glasgow, G12 8QQ, Scotland
Telephone: +44 (0)141 339 8855 Fax: +44 (0)141 330 5881

GLAS-PPE/2006-08
4th May 2006

DIRAC Distributed Analysis on the Computing Grid
DIRAC API Specification

Paterson, S.K. [1], Tsaregorodtsev, A.[2]

[1] University of Glasgow, Glasgow, G12 8QQ, Scotland
[2] C.P.P.M, Marseille

Abstract

DIRAC is the LHCb Workload and Data Management system for Monte Carlo production and distributed user analysis. A new API for production and analysis job submission to the DIRAC Workload Management System (WMS) has been developed which ties together new and existing functionality in a consistent way. This note gives a general overview of DIRAC and the process of job submission to the WMS. A quick start guide and examples of typical production and analysis jobs are supplied for potential users of the system. A more technical description of how the API for job submission works is also provided, topics covered are treatment of input data by Logical File Name (LFN), Input / Output sandbox handling and finally the treatment of output data.

LHCb Note 2005-072

Contents

1 Introduction

LHCb[1] will generate an unprecedented amount of data when it comes online in 2007. As well as the real data from the experiment, Monte Carlo (MC) data must also be generated and stored. In fact, it is expected that many times more Monte Carlo-events will be needed than the number of interesting events in the physics channels[2]. The amount of data is so vast that no single institute can cope. LHCb needs to use all available facilities across the entire collaboration in a distributed computing model through the Grid [3] [4].

The model adopted by LHCb involves the MONARC hierarchical system of classifying sites[5]. The strong computing facility at CERN forms the Tier 0 centre, being supported by other facilities distributed across the world. Tier 1 centres service a large region or country and Tier 2 centres do the same on a smaller scale. As well as the resources available for LHCb on the LHC Computing Grid (LCG)[6], DIRAC can integrate individual PCs or batch systems.

Pooling together these resources will revolutionise the way in which data is stored and manipulated. The Grid will ideally allow physicists to perform their analyses without having to worry about issues such as where any required data is stored and which site the job will actually be executed at. The only limitation is that the applications must be able to run at the site and that this site authorizes LHCb to use its resources.

Distributed Infrastructure with Remote Agent Control (DIRAC)[7] is the LHCb Workload and Data Management system which was originally developed for Monte Carlo production. Recent Developments in DIRAC enable both user production and analysis jobs to be run on LCG or DIRAC sites using the same API in a simple way.

The first part of this note, Section 2, is a quick start guide which outlines all that is necessary to begin using DIRAC to submit user production and analysis jobs. This is complemented by Section 5 which presents example jobs. DIRAC is outlined in Section 3 with special emphasis on how the job submission procedure works.

The remainder of this note presents the DIRAC API for job submission to the WMS with a more technical overview of the key components in Section 4. This is a living note and reflects the current state of the DIRAC API, more functionality will be added over time.

2 Quick Start Guide

2.1 Introduction

This is a quick start guide for submission of analysis jobs to LCG using the DIRAC API. A version of this is also available online [8] where the examples can be obtained.

Important:- to follow this tutorial successfully you must have the following:-

- **A valid Grid Certificate registered in the LHCb Virtual Organisation (VO)**
  - If you would like more information on how to configure your Grid Certificate please refer to the LCG Users Registration[9] page.
  - Your Certification Authority (CA) may also be a good point of reference.

For further information, a more detailed outline of the API can be found in Appendix A.

To run any of the projects in the LHCb software it is envisaged that a potential user of the DIRAC API should have the following:-

- A selected application (Gauss, Boole, Brunel, DaVinci)
- User Options File
- User DLL(s) (if necessary)
- List of dataset(s)

Jobs have to be relocatable on the Grid, hence in order to specify which datasets to run on, a list of Logical File Names (LFNs) is required. These can be obtained from the Bookkeeping Database[10].

2.2 Configuring DIRAC

It is important to note that jobs can be submitted from any LCG User Interface with an installation of DIRAC. Here we shall look at both using a shared installation on **lxplus.cern.ch** as well as a private DIRAC installation. In both cases, a valid proxy is required. For the rest of this tutorial we will assume the C-Shell is being used although this should work equally well in others.

On lxplus.cern.ch

Firstly, we need to gain access to the LCG User Interface and make sure the correct version of DIRAC is used. For now, login to **lxplus** and do the following:-

```
> GridEnv
```

This transforms an lxplus node onto an LCG UI although any already functional UI will not require further setup.

```
> DIRACEnv
```

From A Private Installation

Using the public installation on **lxplus** is recommended although it is also possible to submit jobs from a private DIRAC installation.

First the DIRAC installation script must be obtained via:-

```
> wget http://lhcb-wdqa.web.cern.ch/lhcb-wdqa/distribution/dirac-install
```

```
> chmod +x dirac-install
```

Whether or not it is possible to obtain access to the LCG UI at the desired installation site, there must be a valid proxy available. DIRAC can then be installed and configured using the following commands from the DIRAC ROOT directory (taken as the level at which DIRAC/ will be):-

```
> dirac-install [-g]

> ./DIRAC/scripts/dirac-setup
```

The -g switch above triggers the installation of the LCG software, this is necessary unless the installation site already has LCG software installed and available. A full list of installation options is available by running:-

```
> dirac-install -help
```

The PYTHONPATH and LHCBPRODROOT variables must then be set to the DIRAC ROOT/DIRAC/python and DIRAC ROOT directories respectively.

2.3 Submitting The Job

From this point onwards,

```
Everything you see in boxes represents the
contents of the script being executed.
```

also note that it is possible to execute API commands via the standard Python prompt.

The next step is to obtain a user grid proxy via:-

```
> grid-proxy-init
```

Now it is possible to submit a job. This is simply done via:-

```
>  python ExampleAnalysis.py
```

The contents of this file can be seen by executing

```
>  more ExampleAnalysis.py
```

```
from DIRAC.Client.Dirac import *
 dirac = Dirac()
 job = Job()
 job.setApplication( 'DaVinci', 'v12r11')
 job.setInputSandbox([ 'Application_DaVinci_v12r11/ DaVinci.opts' ])
 job.setInputData([ '/lhcb/production/DC04/v2/DST/
00000742_00003493_10.dst' ])
 job.setOutputSandbox([ 'DVNtuples.hbook', 'DaVinci_v12r11.log'])
 jobid = dirac.submit(job,verbose=1)
 print "Job ID = ",jobid
```

Application log files are automatically generated by DIRAC using the convention <Application> <Version>.log, as specified for DaVinci in the above example.

After job submission, it is possible to use either the DIRAC web-interface[11] or API commands to monitor the job. <JobID> is the value returned by dirac.submit().

```
>  python Monitor.py <JobId>
```

The contents of this file can be seen by executing

```
> more Monitor.py
```

```
from DIRAC.Client.Dirac import Dirac
import sys
dirac = Dirac()
jobid = sys.argv[1]
dirac.status(jobid)
#dirac.parameters(jobid)
#dirac.loggingInfo(jobid)
```

If you uncomment a couple of lines in this script it is possible to obtain additional information about the job.

2.4 Retrieving The Output

To retrieve the output of the job please execute:-

```
> python Retrieve.py <JobId>
```

The contents of this file can be seen by executing

```
> more Retrieve.py
```

```
from DIRAC.Client.Dirac import Dirac
import sys
dirac = Dirac()
jobid = sys.argv[1]
dirac.getOutput(jobid)
```

This will return the output of the DIRAC job. Please note that LCG output (mostly for debugging purposes) may not be ready but this will be made clear in what is printed to the screen. This is because the monitoring of DIRAC is quicker than LCG and so it may be that the LCG job is still in the running state while the DIRAC job is actually finished.

At this point a <JobID> directory can be found containing the output of the job. If the LCG output was retrieved it is returned to a subdirectory of the <JobID> directory.

3 Overview of DIRAC

DIRAC is the LHCb Grid system for Monte Carlo simulation, data production and distributed analysis. Using DIRAC, a variety of available resources can be integrated, including individual PC's, various batch systems and also the LCG Grid.

The DIRAC software architecture [12] is based on a set of distributed, collaborating services. Designed to have a light implementation, DIRAC is easy to deploy, configure and maintain on a variety of platforms.

Following the paradigm of a Services Oriented Architecture (SOA) [13], DIRAC is lightweight, robust and scalable. This was inspired by the OGSA/OGSI "grid services" concept and the

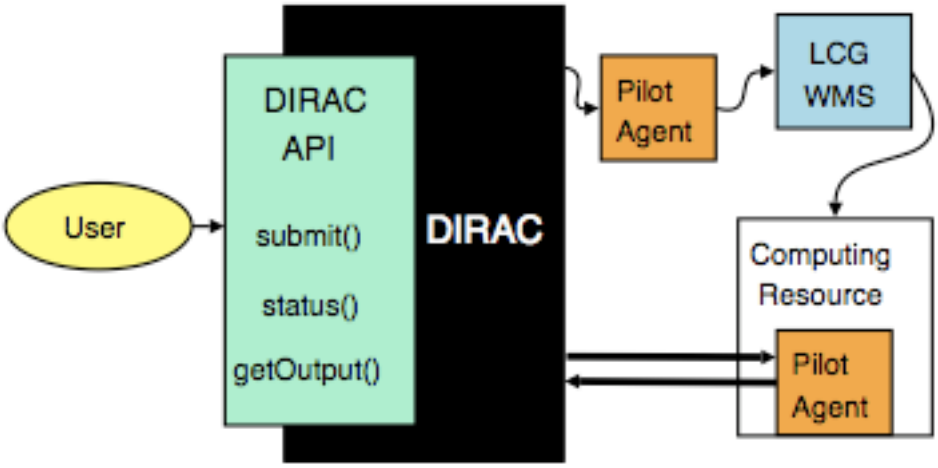

Figure 1: Overview of job submission mechanism in DIRAC from the perspective of the user.

LCG/ARDA RTAG architecture blueprint.

The DIRAC Workload Management System (WMS)[14] realises the *PULL* scheduling paradigm [15]. Agents submitted to LCG or DIRAC sites request jobs whenever the corresponding resource is free.

DIRAC is implemented in Python, using XML-RPC protocol for client- service access and Jabber for reliable server-server communication[16]. Using standard components and third party developments as much as possible has enabled DIRAC to remain highly adaptable. The modular design at each level makes adding new functionality relatively simple.

The LCG file catalogue (LFC) [17] is now being used by DIRAC in order to locate the required input data (if any). Note that a job will be eligible to run only if all data files are available on at least one site. It is advisable to include only those files actually needed by the job.

3.1 Workflow in DIRAC

For the purposes of outlining the workflow, DIRAC can temporarily be considered as a "black box". Figure 1 outlines the DIRAC job submission procedure from the perspective of the user.

As a user, the exposed part of DIRAC is the API. Once a job is submitted to the WMS a PilotAgent is submitted to LCG. PilotAgents are submitted using the proxy of the user and contain information about the job they should pick up.

When the PilotAgent arrives at the LCG Worker Node (WN) it proceeds to autonomously install DIRAC and start an Agent. This Agent then queries the WMS for the job and if it has not yet been picked up, the WMS sends the job to the Worker Node. After this, the job is executed and output is sent back to the WMS to be retrieved by the user at a later stage.

If the submitted PilotAgent does not pick up the job after a specified waiting period, further PilotAgents can be submitted which allow for failures on the part of LCG. In the case of a

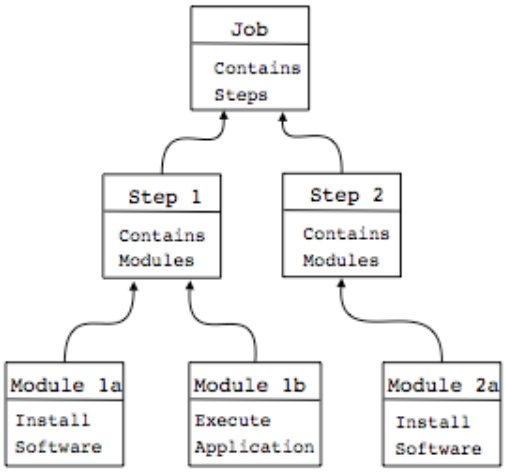

Figure 2: Overview of the relationship between Job(), Step() and Module() classes in DIRAC. Any topology can be described using these components and executed as a DAG.

PilotAgent starting successfully but not finding any jobs waiting in the WMS, the PilotAgent will terminate gracefully thus freeing up the WN for future use.

In fact, after a PilotAgent has finished running (or attempting to run) the job it was sent for, a further request for jobs from the same user with suitable input data requirements is made depending on the amount of CPU time left on the WN. This maximises the resource usage on LCG. If all PilotAgents fail to retrieve a particular job then it may be rescheduled which triggers resubmission of PilotAgents to LCG.

For jobs involving input data there is an extra couple of steps. The JDL of the PilotAgent jobs submitted to LCG includes the specified LFNs. The LCG Resource Broker (RB) uses these to query the LCG File Catalogue (LFC) to ensure that the job ends up at a site which can access this data. Before sending a job to a site, the DIRAC WMS performs a similar resolution to make sure the PilotAgent requesting the job is at a suitable site. Secondly, once the job is at the WN, the data is resolved locally, see Section 4.1.

All data is specified by Logical File Name (LFN), this is because every LFN has a certain number of replicas which have corresponding Physical File Names (PFNs) associated with them. Therefore if there are many replicas of the desired data, the submitted job will be able to run at a larger number of sites.

3.2 Jobs In DIRAC

Jobs in DIRAC are made up of three classes: Job(); Step() and Module(). Figure 2 outlines how objects of these classes are related. The DIRAC API uses these classes to build user jobs but this is transparent from the perspective of the user.

The main purpose of the Job class is to contain Steps. Likewise, the main purpose of the Step class is to contain Modules. Using these three classes as building blocks, any topology of Steps can be created. Jobs may contain many Steps each of which can execute different applications. Steps may depend on each other in a complicated manner so in this way, DIRAC Jobs can be thought of as a Directed Acyclic Graph (DAG). Figure 3 outlines the resulting

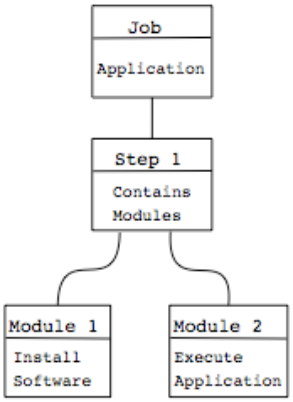

Figure 3: Structure of a typical single application job submitted using the DIRAC API.

structure when executing a single application script such as in Section 2.3.

4 API for Job Submission to DIRAC WMS

This section aims to outline the more technical underpinning to the key functionality of the DIRAC API. The full command listing of the API is available in Appendix A.

4.1 Treatment of Input Data by LFN

When a job is submitted to DIRAC with input data specified the inner workings of DIRAC ensure it arrives at a site which can access this data. Once the job is on the Worker Node (WN) there is still some work to be done. Since all data is specified by LFN, it is necessary to create a Pool XML Slice in order for the PFNs of the datasets to be resolved locally. This is actually performed by the DIRAC Job Wrapper which actually executes the user application.

Figure 4, outlines the procedure for doing this. Firstly, the DIRAC Job Wrapper on the WN queries the LFC for the Globally Unique Identifier (GUID) and PFN of a given LFN as specified by the user. Once the LFC returns these values, the file is added to the POOL XML File Catalogue using the GUID, LFN and PFN. After this, the POOL XML Slice is returned. The final step is to append this XML Slice to the options file of the user. This is done automatically before the application starts to execute.

It is important to note that all this is completely transparent for the user. All the user needs to be concerned with is the list of LFNs to be included in the job. DIRAC takes care of sending the job to sites which have access to the data and the Job Wrapper resolves the LFNs local to the site which executes the job. Since LFNs can have multiple PFNs, once data is replicated across the Grid, multiple sites may be available for any particular job.

4.2 Input and Output Sandbox Handling

When a user runs an application on the Grid it may well be the case that small files i.e. less than ten Megabytes in size, are required for the purpose of steering. These files are collectively referred to as the Input Sandbox. Likewise, the term Output Sandbox refers to similarly small

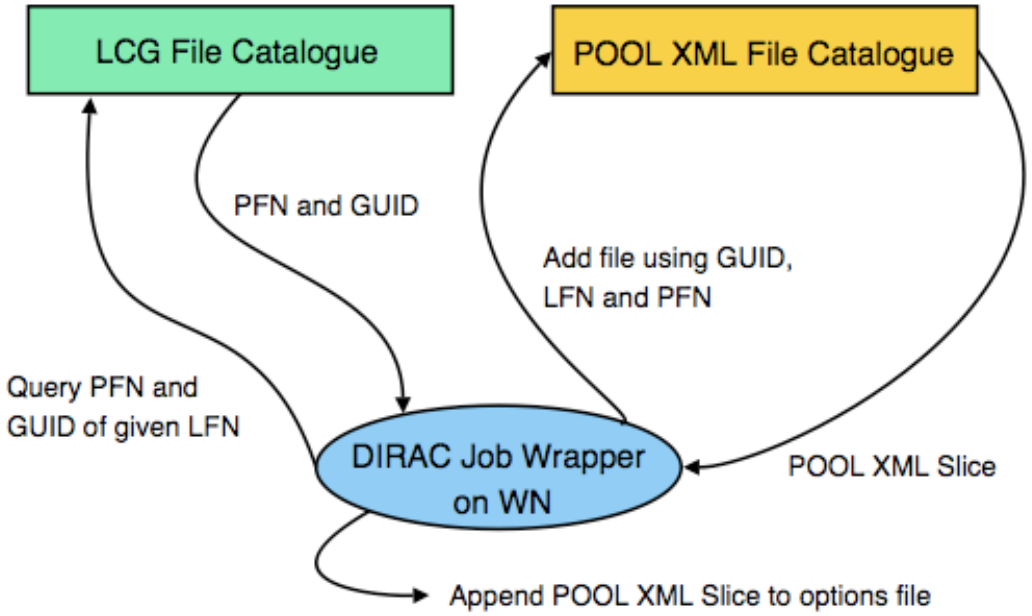

Figure 4: Treatment of Input Data by Logical File Name (LFN) in DIRAC.

output files of a job, e.g. application log files, which do not require permanent storage on the Grid.

Whatever the user specifies in the InputSandbox() and OutputSandbox() of the job is treated as outlined in Figure 5. Firstly, when a job is submitted to the DIRAC WMS, the inputsandbox service receives the Input Sandbox from the User Local area. This is stored in a MySQL database until requested by an Agent.

Once an Agent has successfully requested a job from the WMS, a request is made to the inputsandbox service resulting in a transfer to the computer resource (or WN) so that job execution can commence.

After a job is finished, whatever is specified in the OutputSandbox is transferred to the WMS and stored by the outputsandbox service in the MySQL database. When a user issues the getOutput() command, a request is sent to the outputsandbox service and the output is returned to the users local area.

An important point to note is that the Input / Output Sandbox mechanism is only meant for small files. A hard limit of 10Mb is placed on these to avoid congestion of the system. In the case of the InputSandbox, a job will not successfully submit to DIRAC if the limit is exceeded. For the OutputSandbox, a check is performed on the size of the files and if the limit is reached, the largest files are automatically registered in the same way as OutputData (see Section 4.3).

4.3 Output Data

When a user specifies output data for a job using the API, DIRAC submits and executes the job as normal although there is one extra step during the job finalisation.

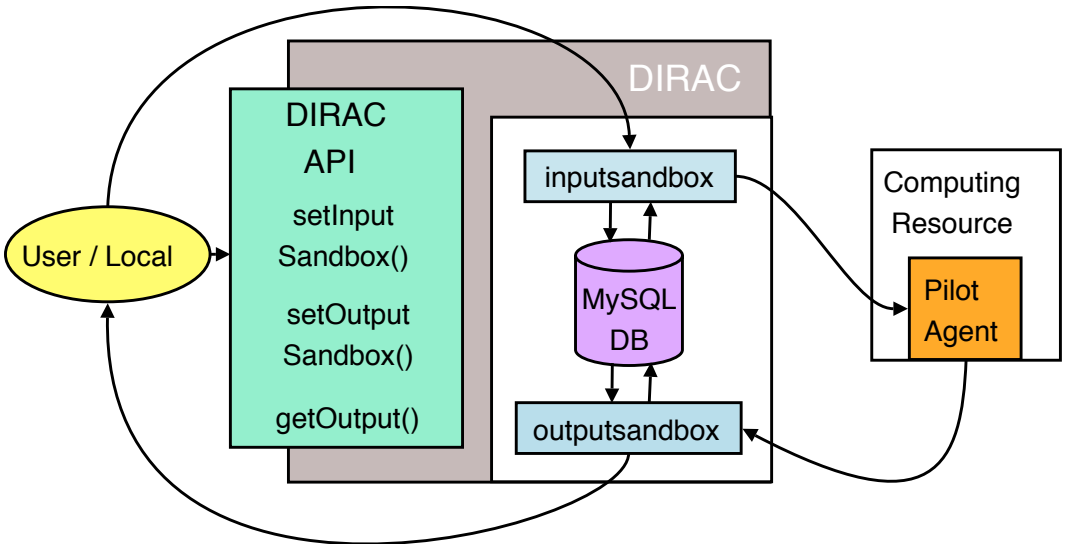

Figure 5: Input / Output Sandbox handling as part of the job submission procedure in DIRAC. All data transfers in this mechanism are performed using the XML-RPC Protocol.

If the files specified in the setOutputData() command are found after the job has finished, they are automatically transferred to Grid storage and registered in the LFC. It is possible for the user to specify which output Storage Element the output data for a particular job will be sent to (some examples are CERN_Castor, CNAF_Castor and PIC_Castor).

The standard convention used for registering the files is as follows:-

`/lhcb/user/<INITIAL>/<USER>/<JOBID>/<FILE>`

This can be overridden by specifying a full LFN path (see Appendix A) although it is important to note that `/lhcb/user/<INITIAL>/<USER>/` is mandatory and automatically prepended to all LFNs. In this way, users can submit a job to the WMS which generates output data which is stored and registered in the LFC. The output data from successful jobs can then be used by subsequent user jobs as input data.

Due to the data being stored in the LFC, the user need not concern themselves over where exactly this data is. The only important thing for the user is to note the LFN which can be used for file retrieval or replication. Therefore users need never know the PFNs of data files.

4.4 Interface to LCG

LCG has an efficiency of around 60% from our previous experience although this figure is higher for Tier 1 sites. Although this may further improve in the future, as part of DIRAC job submission multiple PilotAgents may be sent per job as necessary on behalf of the user. This still means each job is picked up and executed only once but this mechanism makes it much less likely that the job will fail. In practice, submission of extra PilotAgents occurs only in case of failure or excessive waiting and there is a maximum number allowed per job which ensures LCG is not flooded by PilotAgents.

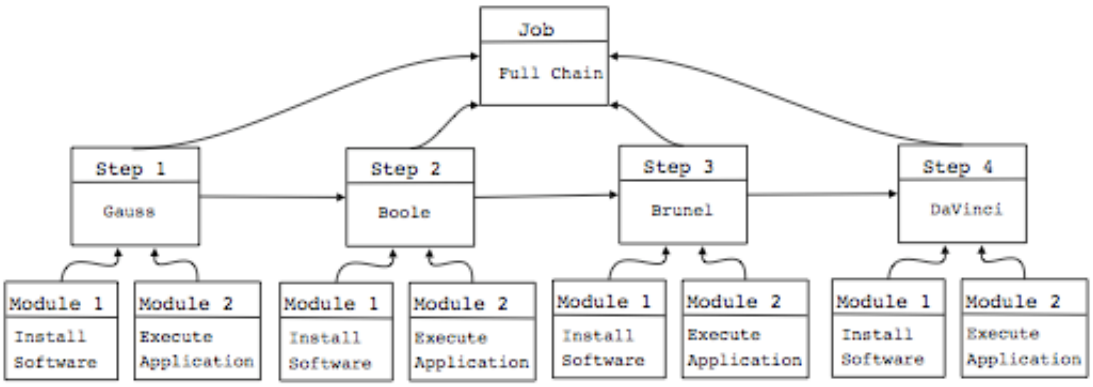

Figure 6: Structure of a multi step job to run Gauss, Boole, Brunel and DaVinci applications using the DIRAC API.

The AgentDirector and AgentMonitor services running in the WMS submit and keep track of PilotAgents. Submission works by constructing the LCG JDL file required by LCG jobs automatically. In doing this, each PilotAgent is sent as a normal job to LCG containing information about the user and the DIRAC job which is to be executed. As outlined in Section 3.1, Figure 1 the LCG job makes its way to a computing resource (WN) where the Agent is started and the job is then requested from the WMS. To keep track of the submitted PilotAgents, the AgentMonitor service intermittently checks their status and triggers resubmission as required.

4.5 Multi Step Jobs

As outlined in Section 3.2, DIRAC jobs are mainly composed of three classes: Job(); Step() and Module(). There are two ways to execute jobs through the DIRAC API. Firstly, for individual application usage it is possible to fully describe a job using the Job() class. Examples of these can be found for DaVinci and generic Gaudi jobs in Section 5.

If, however, a user requires to execute a chain of applications, this is accomplished via the Step() level API (see Appendix A for the full listing). In this way, a task can be broken down into steps on the application level with each step possibly depending on the output of previous steps. The structure of a typical multi-step job for a Gauss, Boole, Brunel and DaVinci chain is shown in Figure 6. An example DIRAC API script for user production can be found in Section 5.3.

5 Example Jobs

5.1 DaVinci Analysis Job

```
from DIRAC.Client.Dirac import *

dirac = Dirac()

job = Job()
```

These lines import the DIRAC API module and create instances of the Dirac() and Job() classes.

```
job.setApplication('DaVinci', 'v12r11')
```

This is a mandatory line, you must choose an application to execute.

```
job.setInputSandbox([ 'Application_DaVinci_v12r11/DaVinci.opts','lib'])
```

To provide user options files and DLLs. If DLLs are to be used, make a lib/ directory and place them inside.

```
job.setInputData(['/lhcb/user/paterson/2239/BrunelOutputData.dst',

  '/lhcb/production/DC04/v2/DST/00000742_00003494_10.dst'])
```

Input data is entered by LFN. Use the LHCb Bookkeeping Database[10] to select these. Private LFNs can also be used.

```
job.setOutputSandbox(['DaVinci_v12r11.log'])
```

The application log file, DaVinci_v12r11.log is created automatically by DIRAC. This is simply constructed from <AppName>_<AppVersion>.log

```
job.setOutputData(['DVNtuples.hbook'])
```

To specify output files to be transferred and registered in Grid storage.

```
jobid = dirac.submit(job,verbose=1)
```

Submits the job with additional information printed to the screen.

```
print "Job ID = ",jobid
```

5.2 Generic Gaudi Application Job

```
from DIRAC.Client.Dirac import *

dirac = Dirac()

job = Job()
```

These lines import the DIRAC API module and create instances of the Dirac() and Job() classes.

```
job.setApplication('<Application>', '<Version>')
```

This is a mandatory line, you must choose an application to execute. In principle, anything available in the LHCb software repository[18] is available.

```
job.setInputSandbox([ '<Full Path to Options File>'])
```

To provide user options files, root can be taken from the directory in which the script is executed.

```
 job.setInputData(['<LFN>'])
```

To specify input data where needed (e.g. for Boole, Brunel, DaVinci). A new job can immediately make use of the output data from previous jobs by using the LFN where it is stored.

```
job.setOutputSandbox(['<Application>_<Version>.log'])
```

The application log file is created automatically by DIRAC. Other small (i.e. less than 10Mb) output files may also be specified here.

```
job.setOutputData(['<File>'])
```

Files larger than 10Mb can be placed in Grid storage. Output data will automatically be stored in an LFN of the form /lhcb/user/<INITIAL>/<USER>/<JOBID>/<FILE>, see Section 4.3.

```
jobid = dirac.submit(job,verbose=1)
```

Submits the job with additional information printed to the screen.

```
print "Job ID = ",jobid
```

5.3 User Production Job Using the Step-Level API

```
from DIRAC.Client.Dirac import *
dirac = Dirac()
job = Job()
step = Step()
step.setApplication('Gauss','v19r4')
step.setInputSandbox(['Application_Gauss_v19r4/Gauss.opts'])
step.setOutputSandbox(['Gauss_v19r4.log','GaussHistos.hbook'])
step.setOutputData(['Gauss.sim'])
step.setOption("""
ApplicationMgr.EvtMax = 20;
GiGa.PrintG4Particles = 0;
ApplicationMgr.OutStream += { "GaussTape" };
GaussTape.Output = "DATAFILE='PFN:Gauss.sim' TYP='POOL_ROOTTREE' OPT='REC'";
PoolDbCacheSvc.Catalog = { "xmlcatalog_file:NewCatalog.xml" };
""")
job.addStep(step)
step2 = Step()
step2.setApplication('Boole','v8r4')
step2.setInputSandbox(['Application_Boole_v8r4/Boole.opts'])
step2.setOutputSandbox(['Boole_v8r4.log','Boole.root'])
step2.setOutputData(['Boole.digi'])
step2.setOption("""
ApplicationMgr.OutStream += { "DigiWriter" };
ApplicationMgr.EvtMax = -1;
HistogramPersistencySvc.OutputFile = "Boole.root";
PoolDbCacheSvc.Catalog = { "xmlcatalog_file:NewCatalog.xml" };
EventSelector.Input = {"DATAFILE='PFN:Gauss.sim' TYP='POOL_ROOT' OPT='READ'"};
DigiWriter.Output = "DATAFILE='PFN:Boole.digi' TYP='POOL_ROOTTREE' OPT='REC'";
""")
job.addStep(step2)
step3 = Step()
step3.setApplication('Brunel','v26r3')
step3.setInputSandbox(['Application_Brunel_v26r3/Brunel.opts'])
step3.setOutputSandbox(['Brunel_v26r3.log','Brunel.root'])
step3.setOutputData(['Brunel.dst'])
step3.setOption("""
ApplicationMgr.OutStream += { "DstWriter" };
ApplicationMgr.EvtMax = -1;
HistogramPersistencySvc.OutputFile = "Brunel.root";
EventSelector.Input = {"DATAFILE='PFN:Boole.digi' TYP='POOL_ROOT' OPT='READ'"};
PoolDbCacheSvc.Catalog = { "xmlcatalog_file:NewCatalog.xml" };
DstWriter.Output = "DATAFILE='PFN:Brunel.dst' TYP='POOL_ROOTTREE' OPT='REC'";
""")
job.addStep(step3)
jobid = dirac.submit(job,verbose=1)
print "Job ID = ",jobid
```

6 Outlook

The core functionality required for distributed analysis and user production is now in place. Supporting use cases such as Event Tag Collections still requires some thought however. Some important features in the pipeline are bulk job submission and support for job-splitting. These will be developed in line with input and output sandbox services being moved to more permanent storage. The use of MyProxyServer to automatically request the extension of user proxies is also envisaged.

The overall goal is to provide a robust, efficient system which is as transparent as possible. The reliability issues of the Grid should be hidden from users as much as possible.

7 Conclusions

The DIRAC Grid Middleware is the Monte Carlo production and distributed user analysis system for LHCb. A new API for production and analysis job submission to the DIRAC Workload Management System (WMS) has been presented which ties together new and existing functionality in a consistent way.

Overall, running user analysis and production jobs on local PCs, DIRAC sites and also LCG is possible through the DIRAC API in simple, understandable Python code. The system presented here has been used to demonstrate distributed analysis at all Tier 1 sites and more work will be done to achieve the most comfortable environment possible for users.

References

[1] S.Amato et al., LHCb Technical proposal, CERN/LHCC 98-4.

[2] LHCb Computing TDR, LHCb TDR 11, CERN/LHCC 2005-019.

[3] Foster,I.,Kesselman,C.,Tuecke,S., The Anatomy Of The Grid, Intl J. Supercomputer Applications, 2001.

[4] Foster,I.,Kesselman,C., The Grid 2e, 2nd Edition, Blueprint for a New Computing Infrastructure, ISBN 1558609334, 2003.

[5] Models of Networked Analysis at Regional Centres for LHC Experiments (MONARC), Phase 2 Report, CERN/LCB 2000-001.

[6] LHC Computing Grid Project (LCG), http://lcg.web.cern.ch/LCG/.

[7] DIRAC Grid Middleware, http://dirac.cern.ch/.

[8] Dirac Tutorial,http://www.cern.ch/paterson/diracLCG.html.

[9] LCG Users Registration, http://lcg.web.cern.ch/lcg/users/registration/registration.html.

[10] LHCb Bookkeeping Database, http://lhcbdata.home.cern.ch/lhcbdata/bkk/.

[11] LHCb DIRAC Monitoring (test), http://fpegaes1.usc.es/dmon/DIRACtest/joblist.html.

[12] Andrei Tsaregorodsev et al., DIRAC Distributed Implementation with Remote Agent Control, In Proceedings of Computing in High Energy and Nuclear Physics (CHEP), April 2003.

[13] Andrei Tsaregorodsev et al., DIRAC The Distributed MC Production and Analysis for LHCb, In Proceedings of Computing in High Energy and Nuclear Physics (CHEP), October 2004.

[14] Vincent Garonne et al., DIRAC: Workload Management System, In Proceedings of Computing in High Energy and Nuclear Physics (CHEP), October 2004.

[15] Vincent Garonne et al., Evaluation of Meta-scheduler Architectures and Task Assignment Policies for high Throughput Computing, The 4th International Symposium on Parallel and Distributed Computing, University of Lille1, France, July 2005.

[16] Stokes-Rees, I. et al., Grid Information and Monitoring System using XML-RPC and Instant Messaging for DIRAC, In Proceedings of Computing in High Energy and Nuclear Physics (CHEP), September 2004.

[17] LCG File Catalogue(LFC),http://lcg.web.cern.ch/LCG/.

[18] LHCb Distribution Package Repository, http://lhcbproject.web.cern.ch/lhcbproject/dist/distribution.h

A Appendix: Outline of DIRAC API

This is the full command listing of the DIRAC API, for convenience it is split into commands affecting the Job(), Step() and Dirac() classes.

Name	Arguments	Comments
job=Job()	None	Creates instance of Job() class
job.setApplication()	('DaVinci','v12r11')	Same for Boole, Brunel and Gauss. Specifies application and version to Job.
job.setInputData()	(['LFN1','LFN2',...])	Should a user need to specify any input datasets to *e.g.* DaVinci.
job.setInputSandbox()	(['OptionsFile.opts', 'Path/to/lib'])	For a user to provide input files directly. Paths to the options file and (if required) lib directory of the DLLs are specified here. Default is local directory
job.setOption()	("""Options file syntax;""")	So user can provide extra options for the end of the file *e.g.* to change the number of events processed
job.setOutputSandbox()	(['File1','File2',...])	If user wants to specify the output file names, typically .root .hbook etc.
job.setOutputData()	(['File1','File2',...], OutputSE=None)	For specifying output data to be registered in grid storage (CERN_Castor by default)
job.addPackage()	('Name','Version')	For specifying an additional package outwith a standard distribution of the software
job.setDestination()	('DIRAC.SITE.XX')	To specify a destination DIRAC site for a job (mostly for debugging)
job.addStep()	(step)	For adding a user defined application step to a job
job.setCPUTime()	(int)	To specify the LCG TimeRef parameter in seconds. If not specified a default value will be imposed by the Agent Director.
job.setType()	('test')	Default value is 'user'. Set to e.g. 'test' for private DIRAC agents to pick up the jobs.

Name	Arguments	Comments
step=Step()	None	Creates instance of Step() class
step.setApplication()	('DaVinci','v12r11')	Same for Boole, Brunel and Gauss. Specifies application and version to Step.
step.setInputData()	(['LFN1','LFN2',...])	Should a user need to specify any input datasets to *e.g.* DaVinci.
step.setInputSandbox()	(['OptionsFile.opts', 'Path/to/lib'])	For a user to provide input files directly. Paths to the options file and (if required) lib directory of the DLLs are specified here. Default is local directory
step.setOption()	("""Options file syntax;""")	So user can provide extra options for the end of the file *e.g.* to change the number of events processed
step.setOutputSandbox()	(['File1','File2',...])	If user wants to specify the output file names, typically .root .hbook etc.
step.setOutputData()	(['File1','File2',...], OutputSE=None)	For specifying output data to be registered in grid storage (CERN_Castor by default)
step.addPackage()	('Name','Version')	For specifying an additional package outwith a standard distribution of the software

Name	Arguments	Comments
dirac=Dirac()	None	Creates instance of Dirac() class
dirac.setConfiguration()	('FullPathToIni')	To facilitate a particular local configuration through a user specified .ini file
dirac.parameters()	(jobID)	Returns parameters of job
dirac.loggingInfo()	(jobID)	Returns logging information of job
dirac.reschedule()	(jobID)	Reschedules a job already submitted to the DIRAC WMS
dirac.status()	(jobID)	Returns current job status in DIRAC
dirac.bulkStatus()	([jobID1,jobID2,...])	Returns dictionary of status of jobs.
dirac.getOutput()	(jobID,directory='')	Retrieves Output Sandbox as well as any LCG job output if applicable
dirac.submit()	(job,verbose=1)	Submits job to the DIRAC WMS and also sends two PilotAgents to LCG
dirac.killJob()	(jobID)	Kills job, deletes job from DIRAC WMS
dirac.getOutputData()	(['LFN1','LFN2'],pfn=None) ('LFN',pfn=None)	Retrieves LFNs locally. Can also specify a path to store the output.
dirac.getOutputLFN()	(jobid)	Simply returns the main part of the LFN using the DIRAC convention.
dirac.replicate()	(['LFN1','LFN2'],destinationSE) ('LFN',destinationSE)	Replicates files to destination SE and registers LFNs in LFC.
dirac.remove()	('LFN')	Removes LFN from LFC and deletes all replicas.
dirac.lfc()	None	To browse the LCG File Catalogue using the File Catalog Client CLI.