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/2006-09
4th May 2006

DIRAC Infrastructure For Distributed Analysis

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 simulation, data processing and distributed user analysis. Using DIRAC, a variety of resources may be integrated, including individual PC's, local batch systems and the LCG grid. We report here on the progress made in extending DIRAC for distributed user analysis on LCG. In this paper we describe the advances in the workload management paradigm for analysis with computing resource reservation by means of Pilot Agents. This approach allows DIRAC to mask any inefficiencies of the underlying Grid from the user thus increasing the effective performance of the distributed computing system. The modular design of DIRAC at every level lends the system intrinsic flexibility. The possible strategy for the evolution of the system will be discussed.

The DIRAC API consolidates new and existing services and provides a transparent and secure way for users to submit jobs to the Grid. Jobs find their input data by interrogating the LCG File Catalogue which the LCG Resource Broker also uses to determine suitable destination sites. While it may be exploited directly by users, it also serves as the interface for the GANGA Grid front-end to perform distributed user analysis for LHCb.

DIRAC has been successfully used to demonstrate distributed data analysis on LCG for LHCb. The system performance results are presented and the experience gained is discussed.

CHEP 2006
13-17 February, Mumbai, India

Contents

1 Introduction

LHCb[1] will generate an unprecedented amount of data when it comes online in 2007. 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[2]. The model adopted by LHCb involves the strong computing facility at CERN which 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)[3], DIRAC[4] can integrate individual PCs or batch systems. Pooling together these resources will revolutionise the way in which data is stored and manipulated.

2 Background & Philosophy

DIRAC was originally created with the following main aims: data production on all resources available to LHCb; providing a means to distribute LHCb data in real time according to the Computing Model and also steering, monitoring and accounting of all LHCb activities on the Grid and other distributed resources.

The DIRAC software architecture 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), DIRAC is lightweight, robust and scalable. This was inspired by the OGSA/OGSI "grid services" concept and the LCG/ARDA RTAG architecture blueprint[5].

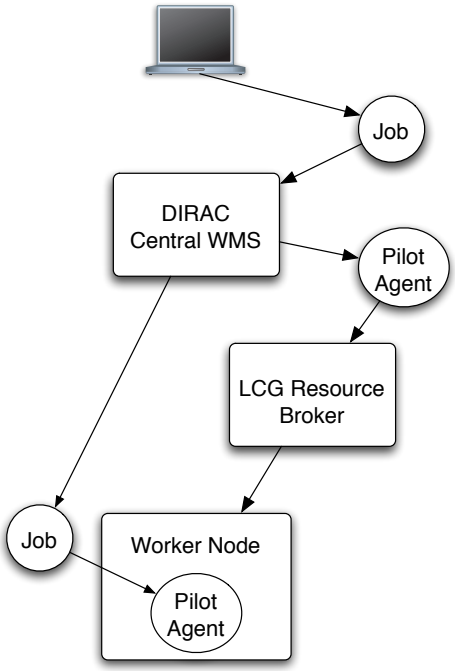

Figure 1: Illustration of the DIRAC Pilot Agent paradigm.

The DIRAC Workload Management System (WMS) realises the PULL scheduling paradigm. Pilot Agents submitted to LCG or DIRAC sites request jobs whenever the corresponding resource is free, this is outlined in Figure 1.

In the context of LHCb, distributed analysis is a batch analysis but with minimised response time. This is not an interactive, parallel analysis system such as PROOF[6] but prioritization and optimization of available resources for LHCb. The aim is to provide a stable platform for analysis on inherently unstable resources and therefore mask the inefficiencies of LCG from the user. Due to the success of the Pilot Agent approach for production jobs it was decided to extend DIRAC to cope with distributed user analysis, the progress made is described here.

3 DIRAC Infrastructure

To outline the DIRAC Infrastructure for distributed analysis let us consider a typical user job with input data. Firstly, the job will be submitted to DIRAC directly or via Ganga[7] using the DIRAC API.

3.1 DIRAC API

The DIRAC API consolidates new and existing functionality to provide users with a transparent way to submit jobs to the Grid. Jobs in DIRAC are composed of Steps and Modules as outlined in Figure 2.

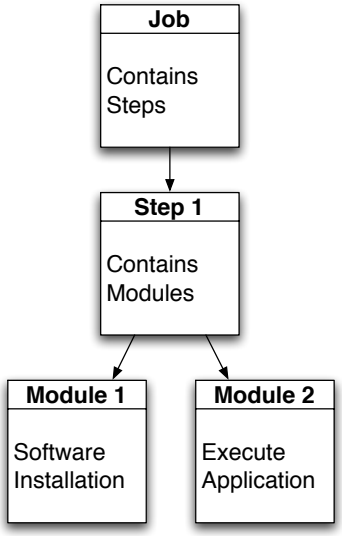

Figure 2: Jobs in DIRAC are composed of Steps which in turn are composed of modules. In principle, any workflow (DAG) can be created using this architecture.

Using these as building-blocks, any topology of Steps can be created but this is transparent from the perspective of the user. Jobs may contain many Steps, each of which may depend on each other in a complicated manner. In this way DIRAC Jobs can be thought of as a Directed Acyclic Graph (DAG). The DIRAC API is principally a scripting language but may also be used from the Python prompt. It provides functionality to securely submit, monitor, retrieve and delete Jobs. Input data is specified by LFN and in principle this is all the user need know when submitting jobs.

3.2 Workload Management System

Once a Job has been submitted to the DIRAC WMS via the DISET[8] Security infrastructure, the *Job Receiver* service assigns a Job ID and saves the Job in the *Job Database* along with

the proxy of the user. During the submission process the *Sandbox* services ensure the upload of any input files to steer the application. Figure 3 shows an outline of the Central WMS services and interactions with LCG components.

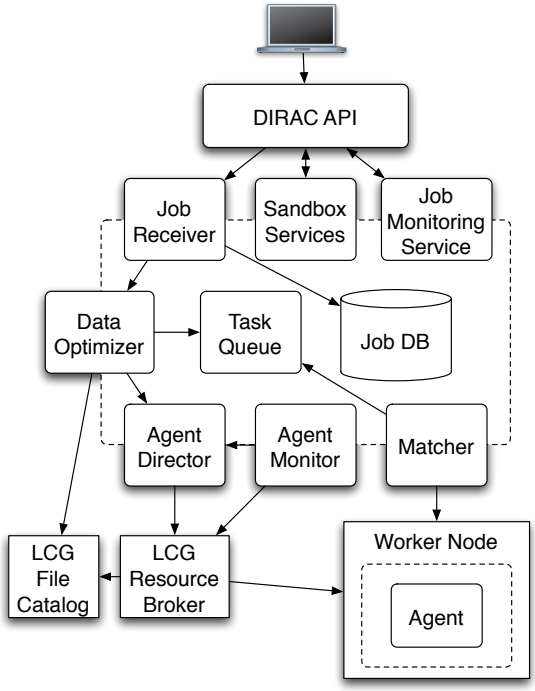

Figure 3: Outline of the DIRAC Workload Management System.

The *Job Receiver* then notifies the *Data Optimizer* which queries the LCG File Catalog (LFC) for input data files to find a suitable Grid Storage Element (SE). The *Data Optimizer* then inserts the Job into a *Task Queue*. At this point the *Agent Director* sends a Pilot Agent to LCG using the requirements of the Job. The *Agent Monitor* checks the Pilot Agent and triggers resubmission as required. Once a Pilot Agent successfully reaches a Worker Node (WN) it installs DIRAC and runs an Agent which requests a particular job from a particular user. The *Matcher* service matches the requirements of jobs (e.g. possible SEs) to the properties of the computing resource presented by the Agent. Since the Agent can in turn put specific requirements on jobs, this is called a 'double match' procedure. Figure 4 outlines the interactions between a DIRAC Agent running on a Worker Node, the WMS Central services and LCG components. Once a job has been delivered to the WN, any software which is not already available locally is installed. Links to any pre-installed software are created local to the job during installation of DIRAC.

The Agent dynamically creates a *Job Wrapper* using information local to the WN, which is then executed. The *Job Wrapper* downloads the Input Sandbox of the job and provides access to the input data. The LFNs are resolved into 'best replica' PFNs for the execution site and a POOL[9] XML Slice is automatically generated for the available protocols. Currently any protocols supported by POOL can be used, although in the absence of these data is brought local to the job before execution.

The Job application is then invoked in a child process and a *Watchdog* process is started in parallel to the application which provides 'heart-beats' for the *Job Monitoring Service*. This also collects accounting information such as CPU and memory consumption. If the application ceases consuming CPU, the job can be marked as stalled. The *Job Wrapper* notifies the *Job Monitoring Service* of the changes in the job state. After the Job has finished, the *Job Wrapper*

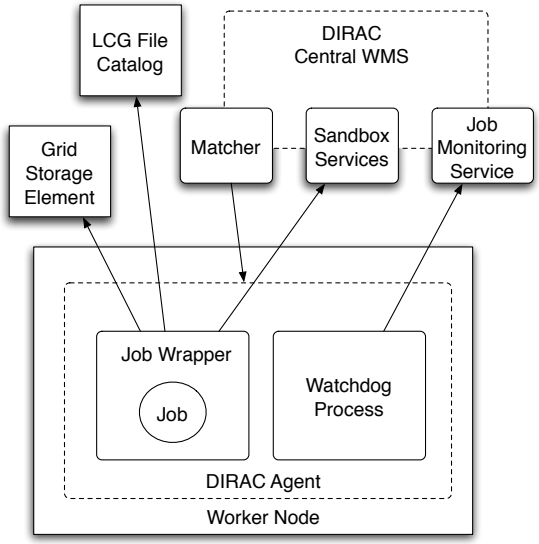

Figure 4: DIRAC Workload Management on the Worker Node.

handles the upload of the Output Sandbox using the *Sandbox* services and storing the output in the *Job Database*. Any Output Data will be uploaded to a predefined SE at this point. Once a DIRAC Agent has finished, the Pilot Agent terminates gracefully thus freeing the LCG resource. At all stages, the *Job Monitoring Service* is used as an interface to update the Job information.

3.3 Pilot Agent Strategy

There are several ways to use the DIRAC infrastructure but the end goal is to minimise the start time of user analysis jobs. Firstly, the *Agent Director* and *Agent Monitor* services may be used to define a policy on how Pilot Agents are submitted. Secondly, the choice of DIRAC Agent can be made which affects how jobs are picked up from the WN. Therefore, it is possible to define modes of submission 'tuned' for the needs of specific jobs:

- **'Resubmission'** Mode: LCG submission with monitoring of the LCG failures, multiple Pilot Agents may be sent if necessary
- **'Filling'** Mode: Multiple Pilot Agents may be sent which request several jobs from the same user, only requesting a new job once the current one has finished
- **'Multi-Threaded Filling'** Mode: Same as Filling Mode above except two jobs can be run in parallel on the WN

It is important to note that this is a DIRAC optimization and is not possible with standard LCG tools. Consider a typical LHCb Monte Carlo (MC) Production job which will execute for approximately one day. The start time for this job is not a priority although getting it started is still an issue, thus the 'Resubmission' mode would be sufficient. For user analysis jobs, however, the 'Filling' and 'Multi-Threaded' modes become useful in minimizing the start times on LCG whilst maximising the use of resources. This can be effective since analysis jobs are heavily I/O bound and hence less CPU intensive than e.g. MC Production jobs.

3.4 Implementation Details

DIRAC is implemented in Python, using XML-RPC protocol for client-service access and Jabber for reliable service-service communication. A MySQL database is used for maintaining all information for services and jobs. The client-service communications are secured using the DISET framework which is conformant with the standard GSI infrastructure. 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.

4 Performance & Experience

A study of the various DIRAC modes of submission was performed using short analysis jobs. Since DIRAC has been proven to cope with long Production jobs, this serves to test the other extreme. Measuring performance on the Grid is not an exact science, therefore to tackle the general Grid 'weather' the following precautions were taken. Firstly, jobs were submitted at the pace of the Resource Broker and job start times were measured relative to the submission time to DIRAC. Secondly, to ensure similar conditions, thirty users were submitting jobs in turn with each user submitting a different mode. The results presented here are ten distinct experiments

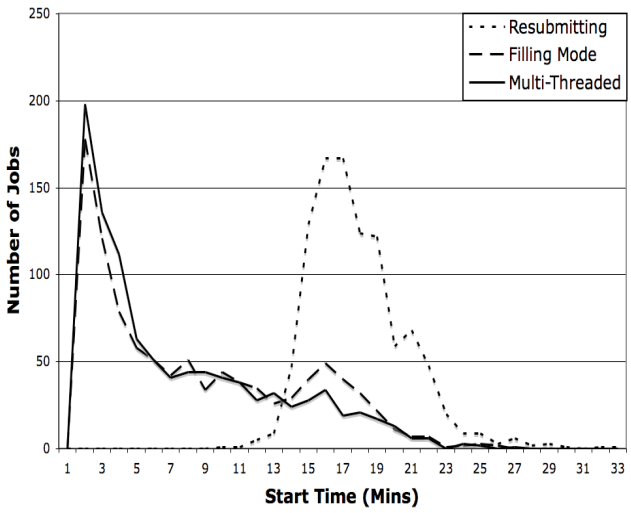

Figure 5: Start times by mode for a total of 3000 jobs submitted to DIRAC by 30 users.

of three users submitting one hundred jobs for each mode with three thousand jobs submitted in total. Figure 5 shows the distribution of job start times for each mode of submission. This shows a considerable improvement for the Filling and Multi-Threaded modes when compared to the peak for Resubmission which is the LCG benchmark result. The first LCG job to start occurs at the nine minute region whereas many jobs for the other two modes have already started. This highlights the power of maximising the responsiveness of the system through the Filling and Multi-Threaded modes. The tails in the Filling and Multi-Threaded distributions are due to the initial jobs at the start of the experiment that need first to reserve an LCG resource. These tails normally diminish at the steady mode of operation. It is important to note that all three thousand jobs completed successfully so the real goal is to minimise the start times.

Figure 6 shows the mean start times by experiment for the three thousand jobs. This shows a clear improvement for the Filling and Multi-Threaded modes and demonstrates reproducability

of the results. These results show that even when LCG is performing well, there is a significant

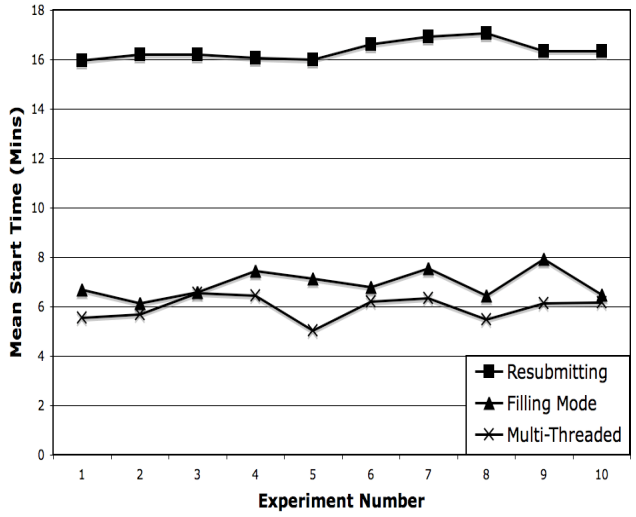

Figure 6: Mean start times for 10 experiments submitting a total of 3000 jobs to DIRAC from 30 users.

improvement on the results. Another important point is that fewer Pilot Agents need to be sent for the Filling and Multi-Threaded modes and so the load on LCG can be reduced. Comparing the number of Pilot Agents sent versus the number of jobs executed we see a factor three for the Filling and a factor of five for the Multi-Threaded mode in our experiments. These factors depend on the amount of the available resources and on the Job characteristics. It is important to note that no special queues are required to be defined on LCG as are usually required to cope with high priority tasks. The described experiments were performed using thirty distinct users. Optimizing the workload can only currently be performed at the level of the user to satisfy the LCG security rules, therefore the results presented in Figures 5 and 6 reflect the optimization on a one hundred job basis. We can conclude that optimization on this scale is effective but not as powerful as optimization on the level of the Virtual Organisation (VO) could be.

Figure 7 outlines the effect of optimizing the workload on the level of the VO versus multiple users. In this experiment two thousand jobs were submitted simultaneously, to ensure the same

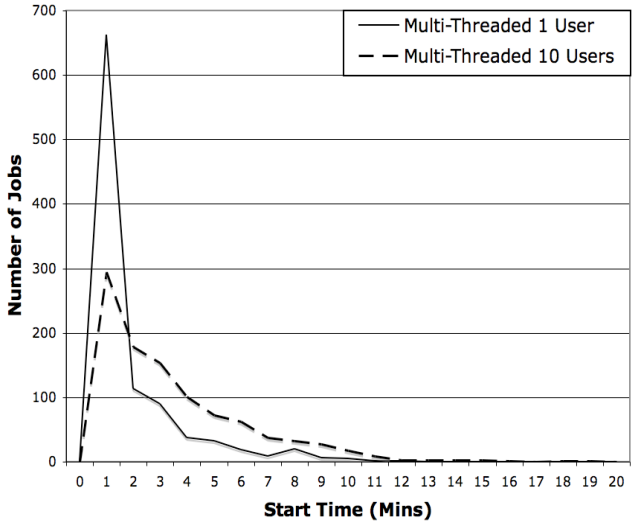

Figure 7: Comparison of the effect of number of users on the start time of jobs.

conditions, in Multi-Threaded mode. Half of the jobs were from a single user and the remainder

were from ten distinct users. A clear improvement of the efficiency is observed in the first case.

5 Conclusion

The use of the Pilot Agent Paradigm for LHCb jobs has resulted in a very high overall efficiency for LCG jobs. Recent tests[10] measure this at 95% with the remaining 5% due to inconsistencies in the LFC. Extending the production system to cope with user analysis jobs has been demonstrated to be effective and this open the door to further optimizations not possible with LCG tools. By testing the performance of the system with short analysis jobs it is evident that a significant improvement on the job start times could be obtained by facilitating optimization of the workload on the level of the VO rather than the individual users. The DIRAC infrastructure for supporting distributed analysis activitites in LHCb is in place. Real users are starting to use and more importantly benefit from the system.

6 Acknowledgements

The authors would like to recognise the Marie Curie Foundation for the fellowship which made this research possible, as well as PPARC and IN2P3. We would also like to thank the members of the DIRAC team.

References

[1] S. Amato et al., LHCb Technical proposal, CERN/LHCC98-4.

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

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

[4] A. Tsaregorodtsev et al., DIRAC, the LHCb Data Production and Distributed Analysis system, CHEP 2006, Mumbai, India.

[5] T. Wenaus et al., Architecture Blueprint RTAG report, CERN-LCGAPP-2002-09.

[6] G. Ganis et al., PROOF - The Parallel ROOT Facility, CHEP 2006, Mumbai, India.

[7] U. Egede et al., GANGA - A GRID User Interface, CHEP 2006, Mumbai, India.

[8] Casajus Ramo, A., Graciani Diaz R., DIRAC Security Infrastructure, CHEP 2006, Mumbai, India.

[9] POOL Project, http://lcgapp.cern.ch/project/persist/.

[10] U. Egede et al., Experience with distributed analysis in LHCb, CHEP 2006, Mumbai, India.