LHCb Distributed Data Analysis on the Computing Grid

Stuart Keble Paterson

Department of Physics and Astronomy
University Of Glasgow

Thesis submitted for the degree of
Doctor of Philosophy

September 2006

PAGE NUMBERING AS ORIGINAL

Abstract

LHCb is one of the four Large Hadron Collider (LHC) experiments based at CERN, the European Organisation for Nuclear Research. The LHC experiments will start taking an unprecedented amount of data when they come online in 2007. Since no single institute has the compute resources to handle this data, resources must be pooled to form the Grid. Where the Internet has made it possible to share information stored on computers across the world, Grid computing aims to provide access to computing power and storage capacity on geographically distributed systems. LHCb software applications must work seamlessly on the Grid allowing users to efficiently access distributed compute resources. It is essential to the success of the LHCb experiment that physicists can access data from the detector, stored in many heterogeneous systems, to perform distributed data analysis. This thesis describes the work performed to enable distributed data analysis for the LHCb experiment on the LHC Computing Grid.

Acknowledgements

I would like to thank Prof. David Saxon for the opportunity to work in the Experimental Particle Physics group and PPARC for providing funding for three years. My supervisors, Dr. Paul Soler and Dr. Chris Parkes deserve great thanks for their help, guidance and support during this undertaking. They have also had the unenviable task of reading and re-reading this thesis during the past few months and I am very grateful to both of them for all their efforts.

All of the Glasgow PPE group deserve some mention but I would like to single out David Petrie, Kenny Walaron and Christian Shaw for putting up with me from day one. I would also like to thank Dr. Caitriana Nicholson and Dr. David Cameron for two excellent examples of how to write a thesis in the field of Grid computing.

At CERN, I would like to thank Dr. Massimo Lamanna for allowing me to work with the ARDA group. My time at CERN would not have been the same without the steady contingent of U.K. Ph.D. students and I am grateful to them for their company and friendship. I would also like to thank Mary Elizabeth Shewry and Sue Cannon for their support during my stay in Geneva.

Thanks must go to Dr. Elie Aslanides as well as the E.U. Marie Curie Foundation for the opportunity to work in Marseille. I also owe a tremen-

dous amount to my 'third' supervisor, Dr. Andrei Tsaregorodtsev for his guidance, help and support during my time there. A big thanks must also go to my predecessors Vincent Garonne and Ian Stokes-Rees whose work with Andrei on making DIRAC a successful production system provided me with an excellent starting point. In Marseille, thanks must also go to Andrei and Julien Cogan for transporting me to and from work through prolonged bus strikes. The administrative staff at C.P.P.M. were extremely helpful to me during my time there, thanks especially to Helene Boyer and Fanny Lessous for helping me overcome the language barrier in daily life in France.

All of the current DIRAC team deserve thanks, particularly to Dr. Andrei Tsaregorodtsev, Dr. Philippe Charpentier and Prof. Nick Brook for their advice and guidance. I would also like to mention Dr. Ricardo Graciani, Dr. Joel Closier, Dr. Gennady Kuznetzov, Dr. Roberto Santinelli, Andrew Smith and Adria Casajus who I collaborated closely with. In the Ganga team I would also like to thank Dr. Ulrik Egede, Dr. Karl Harrison, Dr. Andrew Maier and Dr. Jakub Moscicki for their input.

Finally I would like to thank my parents who have been a constant source of encouragement. Without their support and belief in me over the years I am sure that this would not have been possible.

Author's Declaration

This thesis presents work performed from 2003-2006 in the Experimental Particle Physics group in the Department of Physics and Astronomy at the University of Glasgow. The software distribution tests performed with Pacman in Chapter 2 are my own work. Although the current LHCb solution, presented in Section 2.5.4, was developed at CERN the integration of this mechanism into DIRAC was performed by myself. Whilst working in collaboration with the ARDA group at CERN, I performed the first realistic analysis using the EGEE gLite Middleware described in Chapter 3. I contributed a significant fraction of the effort on the design and implementation for the LHCb DIRAC production system to accommodate the distributed user analysis tasks as described in Chapter 4. I was the principal author of the Agent Director and Agent Monitor services as well as the Software Distribution and Threaded Computing Element modules. To enable the distributed analysis tasks, I also modified the following modules: Job; Step; Module; Job Wrapper; Gaudi Application; Client; Matcher; and Data Optimiser. Using this system I also explored several optimisation strategies that are described in Chapter 5, along with my results from tests on LCG. Based on this work, I made several policy decisions on how the Pilot Agent paradigm is utilised in LHCb. My analysis of results from real users submitting distributed data analysis jobs to DIRAC is presented in Chapter 6. To facilitate user job

submission, I was also the principal author of the DIRAC API. This consolidates new and existing functionalities of DIRAC and forms the interface for users, including the Ganga Grid front-end, to submit jobs to LCG for LHCb. All of the results presented in Chapters 5 and 6, are my own work, unless explicitly referenced. The data samples in Chapters 5 and 6 were obtained from DIRAC through a Performance Monitoring service of which I was the author. In addition I have contributed significantly to the establishment and daily running of the system such as: initially setting up and maintaining the DIRAC Analysis Service; installing the first public distributions of DIRAC at CERN; as well as supporting and training users.

Preface

This thesis concerns the development of a framework to support distributed data analysis in the Large Hadron Collider beauty (LHCb) experiment. In Chapter 1, the field of Grid computing will be introduced. This will present definitions of the Grid, an overview of distributed computing, and the potential applications of Grid computing. The LHC Computing Grid (LCG) and the treatment of data in a Grid environment will also be discussed.

The LHCb experiment and its computing model are described in Chapter 2, along with some discussion of software distribution on the Grid. The progress made in the automation of the installation procedure for LHCb software using Pacman is briefly discussed. Chapter 2 will conclude with a discussion of the software distribution mechanism chosen by the experiment.

In Chapter 3, some of the paradigms for distributed analysis in LHCb will be presented with a discussion of approaches used by other experiments. This is followed by an outline of the first realistic physics analysis carried out on the EGEE (Enabling Grids for E-sciencE) gLite framework prototype with DaVinci, the LHCb analysis software.

Distributed Infrastructure with Remote Agent Control (DIRAC) was successfully used during the 2004 Data Challenge for Monte-Carlo production tasks and it was decided to extend DIRAC to accommodate LHCb user activities. The DIRAC system and the work performed to extend the system

to accommodate distributed user analysis tasks on LCG will be described in Chapter 4. The advances in the workload management paradigm for analysis with computing resource reservation (by means of Pilot Agents) will be discussed in Chapter 5. 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. Several workload management optimisation strategies will be presented that demonstrate results which are not possible using the standard LCG Grid middleware.

DIRAC has since been successfully used to demonstrate distributed data analysis on the Grid for LHCb and has since become the default mode of submission for all LHCb Grid jobs. In Chapter 6, the system performance results are presented and the experience gained is discussed. Future directions involving further development of DIRAC for user tasks are also described. Finally, conclusions will be presented in Chapter 7.

Throughout this thesis, frequently mentioned components of the DIRAC Workload Management System will be referred to in *italics* to improve readability.

Contents

Acknowledgements i

Author's Declaration iii

Preface v

List of Figures xi

List of Tables xiv

1 Introduction 1

1.1 What is a Grid? . . . 3
1.1.1 Definitions of a Grid . . . 4
1.1.2 Computing Power on Demand . . . 6
1.1.3 What is e-Science? . . . 8
1.2 Overview of Grid Systems and Distributed Computing . . . 9
1.2.1 A Brief History of Grid Computing . . . 10
1.2.2 Emerging Standards . . . 15
1.2.3 Components of a Typical Grid . . . 18
1.3 Applications of Grid Computing . . . 21
1.3.1 Scientific Grid Projects . . . 21
1.3.2 Commercial Grid Projects . . . 23
1.4 Grid Computing Applied to Particle Physics . . . 23
1.5 The LHC Computing Grid . . . 25
1.5.1 A Brief Overview of LCG . . . 26
1.5.2 LCG Information System . . . 29
1.5.3 LCG Workload Management System . . . 31
1.6 Data on the Grid . . . 33
1.6.1 Treatment of Data in a Grid Environment . . . 33
1.6.2 The LCG File Catalogue . . . 35
1.7 Summary . . . 35

2 LHCb Software Environment and Software Distribution 37
2.1 Introduction to LHCb: Physics Aims and Detector 39
2.1.1 LHCb Physics Aims 39
2.1.2 LHCb Detector 41
2.1.3 From the LHCb Trigger to the Grid 43
2.2 LHCb Software: Gaudi, Gauss, Boole, Brunel and DaVinci . . 45
2.2.1 The Gaudi Framework 46
2.2.2 Data Processing Applications 49
2.3 LHCb Computing Model 51
2.3.1 Logical Dataflow and Workflow Model 51
2.3.2 Computing Model 54
2.4 DIRAC as a Production System 56
2.5 Software Distribution in LHCb 57
2.5.1 Software Distribution Assumptions 58
2.5.2 Virtual Machine (Paratrooper) Concept 59
2.5.3 Automating LHCb Software Distribution Using Pacman 60
2.5.4 Final Implementation: install_project.py 66
2.6 Summary 67

3 Data Analysis in a Distributed Environment 68
3.1 Paradigms for Distributed Analysis 69
3.1.1 Push versus Pull 70
3.1.2 Pilot Agent Paradigm 73
3.2 Requirements for LHCb Distributed Data Analysis 77
3.3 Overlayed Network Concept 79
3.3.1 Agents' Control 80
3.3.2 Use of Agents' Control in LHCb 83
3.4 Other Examples of Distributed Analysis 84
3.4.1 Distributed Analysis in ATLAS 85
3.4.2 CMS Distributed Analysis with BOSS 86
3.4.3 Distributed Analysis in ALICE with AliEn 87
3.4.4 Emerging Trends 87
3.5 Distributed Analysis Using DaVinci In the gLite Framework . 89
3.5.1 Using DaVinci with gLite 89
3.5.2 The gLite Framework 90
3.5.3 $B_S \rightarrow J/\Psi\Phi$ Channel 92
3.5.4 Analysis Using gLite 92
3.5.5 Job Splitting and Use of the gLite Package Manager . . 94
3.5.6 Analysis Results and Experience 95
3.5.7 Evaluation of gLite for Distributed Analysis 98
3.6 Summary 100

4 Distributed Infrastructure with Remote Agent Control 101
4.1 Introduction 102
4.1.1 DIRAC Design Principles 103
4.1.2 Main Components of DIRAC 104
4.1.3 History of DIRAC 107
4.2 DIRAC Implementation: Software Tools 108
4.3 Services Framework 111
4.3.1 Security in DIRAC - DISET 112
4.3.2 Deployment 113
4.3.3 Reliability 115
4.3.4 Example: DIRAC Configuration Service 116
4.4 Agents Framework 117
4.4.1 Site Agents 118
4.4.2 Pilot Agents 119
4.4.3 Example: Transfer Agent 120
4.5 Workload Management 121
4.5.1 DIRAC Job Wrapper 122
4.5.2 Underlying Database 123
4.5.3 WMS Services 124
4.5.4 Instant Messaging in DIRAC 129
4.6 Data Management 129
4.6.1 Storage Element 129
4.6.2 File Catalogue 130
4.6.3 Replica Manager 131
4.7 Information, Monitoring and Accounting 132
4.8 Summary 133

5 DIRAC Workload Management 136
5.1 Introduction 137
5.2 Jobs in DIRAC 138
5.2.1 Creating Complicated Job Workflows for Users 139
5.3 Workflow of Jobs 140
5.3.1 Providing Access to Input Data 145
5.4 Optimisation Strategies 147
5.4.1 Resubmission 148
5.4.2 Filling 149
5.4.3 Multi-Threaded Filling 150
5.4.4 Testing Framework 152
5.4.5 Results and Performance 153
5.4.6 Pre-emption and Future Optimisations 157
5.5 Comparison of Strategies with Previous Simulation 159

5.6 DIRAC, Condor and Condor-G 160
5.6.1 Condor 162
5.6.2 Condor-G 163
5.6.3 Gliding-In 165
5.7 DIRAC Paradigms in Other Experiments 166
5.7.1 GlideCAF 166
5.7.2 Panda 167
5.8 Summary 168

6 DIRAC Analysis Service **171**
6.1 Introduction 172
6.2 Implementation of DIRAC Service 172
6.3 DIRAC API 174
6.3.1 Treatment of Input Data by LFN 174
6.3.2 Input and Output Sandbox Handling 176
6.3.3 Output Data 178
6.3.4 Interface to LCG 180
6.3.5 Generic Gaudi Application Job 181
6.3.6 Interface to GANGA 182
6.4 Performance on LCG 183
6.4.1 Job Start Times 184
6.4.2 Total Job Times 185
6.4.3 Matching Times 187
6.4.4 Job Completion Efficiency 188
6.4.5 Effect of DC06 Activity on Performance 192
6.5 DIRAC Analysis Usage 195
6.5.1 Frequency of Submission 195
6.5.2 Size of User Jobs 196
6.6 User Experience 198
6.7 Maintenance of Service 200
6.7.1 User Training 201
6.8 Outlook 202
6.9 Summary 203

7 Conclusions **205**

A Glossary **208**

B Complicated Workflows with the DIRAC API **215**

C DIRAC Job State Machine **217**

List of Figures

1.1 Overview of the main components involved in the LCG Information System and their interactions 30
1.2 Illustration of the LCG workload management components used during job submission and their interactions 32
1.3 Overview of the treatment of data in a Grid environment . . . 34

2.1 Aerial view of CERN and the surrounding region 38
2.2 The reoptmised LHCb detector 41
2.3 The data flow of the LHCb data processing applications . . . 50
2.4 The LHCb user analysis cycle 54
2.5 The LHCb Computing Model highlighting the distributed, multi-tier regional centre model 55
2.6 LHCb CPU requirements breakdown by processing activity during 2008-2010 56
2.7 LHCb Disk and MSS requirements from 2008 to 2010 57
2.8 Dependency tree diagram up to LCG tools 63
2.9 Dependency tree diagram up to Gaudi 64
2.10 Dependency tree diagram up to data processing applications . 65

3.1 Illustration of the PUSH and PULL scheduling paradigms . . 70
3.2 Illustration of the PUSH model with the use of the Pilot Agent paradigm 73
3.3 Matching times during LHCb DC04 activity 75
3.4 Illustration of the DIRAC Pilot Agent paradigm in use on LCG 76
3.5 Overview of the different kinds of computing resources available to LHCb 80
3.6 Illustration of Agent deployment in an Overlay Network . . . 81
3.7 Illustration of Agent interaction with Services in an Overlay Network 82
3.8 Analysis dataflow in the gLite Framework, from the perspective of the user 93

3.9 Reconstructed J/Ψ mass distribution after applying J/Ψ selection cuts 96
3.10 Reconstructed Φ mass distribution after applying J/Ψ and Φ selection cuts 97
3.11 Reconstructed B_S mass distribution after applying all selection cuts 98
3.12 Plot of the cosine of the transversity distribution $\cos\theta_{tr}$ for all selected B_S events 99

4.1 Overview of the main components of DIRAC 105
4.2 The timeline of the main milestones and developments of DIRAC to date 107
4.3 CPU usage on the machine hosting the DIRAC central services 114
4.4 Overview of the DIRAC Configuration Service 116
4.5 Schematic overview of the DIRAC Transfer Agent as used in the Service Challenge 3 activity 121
4.6 Overview of the Job Management Services in DIRAC Production WMS 124
4.7 Overview of the DIRAC Data Management System 132

5.1 DIRAC Job, Step and Module architecture 138
5.2 Structure of a multi-step job using the DIRAC API 139
5.3 Outline of the DIRAC Workload Management System 142
5.4 DIRAC Workload Management on the Worker Node 144
5.5 Start times by mode for a total of 3000 jobs submitted to DIRAC by 30 users 154
5.6 Mean start times for 10 experiments submitting a total of 3000 jobs to DIRAC from 30 users 155
5.7 Effect of optimising the workload on the level of the VO, versus multiple users 156

6.1 Overview of treatment of input data by Logical File Name in DIRAC 175
6.2 Input / Output Sandbox handling as part of the job submission procedure in DIRAC 177
6.3 Treatment of output data files in DIRAC 179
6.4 A generic DIRAC API script for LHCb Gaudi-based applications 182
6.5 Job start times on the DIRAC Analysis System for a sample of 3000 real user distributed analysis jobs 184
6.6 Total job times for 3000 real user distributed analysis jobs collected over a six month period on the DIRAC Analysis System 186

6.7 Matching times on the DIRAC Analysis System for 3000 real user distributed analysis jobs collected over a six month period 187
6.8 Breakdown of results for 3000 real user distributed analysis jobs collected over a six month period 189
6.9 Number of jobs running on the DIRAC Production system during 2005-2006 193
6.10 Mean start times versus the number of submitted jobs submitted to the DIRAC Analysis System, during a six month period 194
6.11 Number of jobs submitted to the DIRAC Analysis System every two weeks over a six month period 196
6.12 Mean start time versus number of users submitting jobs to the DIRAC Analysis System every two weeks over a six month period 197
6.13 Number of datasets submitted per job for 3000 real user distributed analysis jobs 198
6.14 Results of 500 jobs running over a total of 5000 datasets submitted to the WMS via the DIRAC API using Ganga 199
6.15 Efficiency of 500 jobs running over a total of 5000 datasets submitted to the WMS via the DIRAC API using Ganga . . . 200

C.1 Primary job states in the DIRAC status machine where arrows indicate the possible transitions 217
C.2 Secondary job states in the DIRAC status machine where arrows indicate the possible transitions 218

List of Tables

1.1 Comparison of electrical and computational grids. 7
1.2 A sample of the many '@home' style Internet computing projects and a brief description of their aims. 12

3.1 Comparison of distributed data analysis systems 88

5.1 Number of Pilot Agents sent for each mode of submision . . . 156
5.2 Comparison of the DIRAC WMS components and the ATLAS Panda system equivalents 168

6.1 Effect of the DC06 activity on system performance 193

Chapter 1

Introduction

In his 1983 Ph.D. thesis [1] entitled 'Study of Load Balancing Algorithms for Decentralised Distributed Processing Systems' Miron Livny stated:

> *Since the early days of mankind the primary motivation for the establishment of communities has been the idea that by being part of a group the capabilities of an individual are improved. The great progress in the area of intercomputer communication led to the development of means by which stand-alone processing subsystems can be integrated into multicomputer communities.*

Livny's assertion hints at the advantage of being an active part of a greater whole, an approach that is mirrored in countries and governing structures of the world today. As science has advanced over time, so too has the complexity of problems being encountered by the academic community. Computing has played an increasingly significant role in science over the years and scientific communities have often been the driving force behind significant advances in the field. In particular, the High Energy Physics (HEP) community played a crucial role in the establishment of the internet as it is today through the creation of the World Wide Web by Tim Berners-Lee in 1989.

In the past, organisations would tackle computing problems through the creation of individual supercomputers or large, local clusters of computers. However, this solution is not ultimately scalable since the scope of current and future computing requirements has increased beyond the level where all necessary computing power can be provided at one single location. A new infrastructure, capable of dealing with many distributed resources is required and the solution is to be found in the field of Grid computing [2,3]. Livny's words have special relevance here since the infrastructure for Grid computing involves resource sharing on a large level as well as the establishment of Virtual Organisations (VOs) [4], a new type of collaborative community to utilise these geographically distributed resources.

Again, the HEP community is set to play an important role in the development and demonstration of this new infrastructure. This infrastructure is driven by the demands of the Large Hadron Collider (LHC) near Geneva, Switzerland, set to come online in 2007. In this chapter, the concepts of Grid computing will be presented in Section 1.1 with an introduction to distributed computing and Grid systems given in Section 1.2. Some of the many applications of Grid computing will be mentioned in Section 1.3, with Grid computing applied to particle physics being described in Section 1.4.

The LHC Computing Grid (LCG) project [5], that aims to provide the distributed computing infrastructure for the LHC, will be outlined in Section 1.5. With shared computing resources all over the world, providing seamless access to data, which may be stored in many different locations, becomes of vital importance. The treatment of data on the Grid will be discussed in Section 1.6.

1.1 What is a Grid?

In simple terms a computational Grid can be thought of as a collaborative group of networked computers, communicating via the internet. Whereas the internet provides seamless access to information held on computers all over the world, the Grid aims to provide seamless access to computational power and storage systems distributed across the world.

Many factors introduce complexity to the task of sharing computational power and storage systems across national and institutional boundaries. A non-exhaustive list includes:

- Heterogeneity which exists in computer hardware as well as operating systems;
- Resource discovery as well as providing a fair share of resources for all users;
- Ensuring security and traceability for owners of the Grid infrastructure;
- The political nature of collaborating on a global scale *i.e.* each contributing site could have different policies; and
- Assuring high availability of Grid resources.

While these issues present a formidable challenge, there is great potential for Grid computing to cause a revolution on the same scale as the Internet has in recent times, creating many commercial and everyday uses.

Grid computing is a complex and rapidly developing field and as such many definitions of the term 'Grid' exist. These will be examined in Section 1.1.1. The name given to Grid computing is not without meaning, a well documented analogy exists with an electrical power grid and this is explained

in Section 1.1.2. The term 'e-Science' is often used in the same contexts as that of the Grid and this will be described in Section 1.1.3. The computing trends leading to the field of Grid computing will be explored in Section 1.2, with an in-depth look at the key components of Grids and emerging standards.

1.1.1 Definitions of a Grid

Answering the question 'What is a Grid?' is not as simple as it first appears. In [6], Ian Foster presents a definitive three point checklist defining a Grid as a system that:

1. *Coordinates resources that are not subject to centralised control,*

2. *Uses standard, open, general-purpose protocols and interfaces, and*

3. *Delivers non-trivial qualities of service.*

The first point implies that the computing resources of which the Grid is comprised may have different access policies or rules governing their use. This is symptomatic of the international, collaborative nature of Grid computing where each participating site could have rules that should be followed, *e.g.* a priority for local users. To be a Grid, issues such as security, membership and payment [6] should be resolved as part of the system.

Without standard, open protocols and interfaces, as mentioned in the second point of the checklist, a system could fall into the category of providing specific services to a specific community. In this situation, users from another community wishing to perform different tasks may not have the tools to do so. Providing all potential users with distributed computing power is vital for a Grid system. This implies a common infrastructure should be

in place to facilitate the use of available Grid resources, providing means to address issues such as: authentication; authorisation; resource discovery and resource access [6]. Some of the emerging standards in Grid computing will be presented in Section 1.2.2.

As implied by the third point in the checklist, the components of a Grid system should be used in a coordinated way to provide adequate response times, high throughput of jobs and a quality of service to meet the complex requirements of users. The main benefit of integrating many heterogeneous distributed resources is to create a reliable, resilient system capable of providing computing power on demand.

The commercial potential of Grid computing has led to definitions from companies such as IBM [7] that define the Grid as 'using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet', with further mention to the importance of users' quality of service requirements in [8]. More examples are available from companies such as Sun Microsystems [9] and Microsoft [10].

Another pioneer of the field, Rajkumar Buyya, defines the Grid [11] as:

> *Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.*

All of these definitions are correct and so it is difficult to introduce a universal statement that encompasses all of the present and future uses of Grid technology. Other attempts have been made to define Grids by their functionality or requirements [12, 13]. However, for the remainder of this

thesis a working definition will be used that views a Grid in the context of current global computing infrastructures such as LCG, explored in Section 1.5. An overview of the history of Grid computing as well as the typical components of a Grid system will be described in Section 1.2.

1.1.2 Computing Power on Demand

In 1969, Kleinrock talked about the spread of 'computer utilities' which could 'service individual homes and offices across the country' in the same way as as electric and telephone utilities [14]. An electrical power grid does share similar characteristics to the concept of a computational Grid. For instance, electrical devices can be plugged into sockets, which provide a well-defined quantity of power. The user of the device isn't concerned as to where the power comes from, nor how it is delivered, only that the device receives enough power to complete the task it was plugged in to perform. From the perspective of the user, it is irrelevant whether the power was generated by a coal, nuclear or hydroelectric plant, this heterogeneity is masked by the power grid.

While the power grid analogy bears similarities to the computational Grid, some of the points require clarification. Over forty years ago in [15], several differences were highlighted and more recently they have been further expounded in [16]. Table 1.1 presents a summary of the key differences between electrical and computational grids.

While some of the differences described in Table 1.1 are obvious, several warrant further discussion. Computational Grids must harness not only the processing power of hardware resources such as individual Personal Computers (PCs) and site clusters, but must also deal with more complex resources such as databases. Whereas any device with a plug may draw power from

Parameter	Electrical Power Grid	Computational Grid
Scope	National	Global
Resources	Heterogeneous power stations	Heterogeneous compute resources
Consumers	Heterogeneous devices	Heterogeneous software applications
Network	Transmission lines, underground cables	Internet connects compute resources
Reliability	Sophisticated protection schemes and redundancy exist	Resource availability must not be relied on, failures must be dealt with
Ease of use	Simple: plug and play	Complex: no 'universal adapter' exists

Table 1.1: *Comparison of electrical and computational grids.*

the electrical power grid, there is no 'universal adapter' for Grid computing systems. Many heterogeneous compute resources exist and it must be possible for all to gain access to the Grid. Inter-Grid compatibility must also be assured. Likewise, software applications running on the Grid need an easy way to 'plug in' to computing resources. Another important point is security. On the electrical power grid circuit-breakers and fuses provide protection. In a Grid environment however, providing a secure way for users to run applications on remote resources, which they do not necessarily own, is less clear. The issues regarding security on the Grid will be introduced in subsequent sections.

It is fair to say that the added complexity of computational Grids limits the effectiveness of the power grid analogy. However, the idea of computing power as a utility is certainly appealing and could eventually become a reality. This potential is only beginning to be realised with Grid computing but with a steadily increasing requirement for computational power, fuelled by experiments such as those at the LHC, Grid systems are set to become more

prevalent through necessity.

1.1.3 What is e-Science?

The drift towards ever increasing amounts of computing power is one of the emerging trends in many fields of science today. Some examples of use-cases in different disciplines will be explored in Section 1.3.

The desire to decouple those who manage compute resources from those who utilise them has led to the creation of 'e-Science'. It is possible to specify e-Science as a field which aims to provide the necessary infrastructure to match the increasing computing requirements of the sciences. Since Grid computing aims to provide computational power to all users, regardless of discipline, the two are inextricably linked.

Some of the main factors in the conception of e-Science include:

- Liberation of scientists from the task of maintaining and managing compute resources;
- Provision of vast amounts of computing power across institutional, national and possibly international boundaries;
- Optimisation of the start times and efficiency of computational tasks; and
- Provision of a simple, uniform way to perform task management.

For example, a scientist should not be concerned with how or where their computing tasks run, only that they do run and with the highest possible degree of efficiency. Whereas in the past, scientists would require familiarity with several types of batch systems to run on local site clusters, the use of the Grid enables uniform access to a larger amount of resources on demand.

With many resources shared across the world there is a higher likelihood that computing tasks can arrive at a site with available processing power.

A recent example of an e-Science project is Enabling Grids for E-sciencE (EGEE) which involves '90 institutions in 32 countries world-wide to provide a seamless Grid infrastructure for e-Science that is available for scientists 24 hours-a-day' [17]. The role of EGEE will be explored in the context of LCG in Section 1.5.

1.2 Overview of Grid Systems and Distributed Computing

This section aims to introduce the background to distributed computing, which in turn has led to Grid computing. The complete history has too broad a scope to cover here so only the trends leading to Grid computing will be considered in Section 1.2.1.

Key to the develoment of Grid computing are the emerging standards by which the vision outlined in Section 1.1.1 can be realised. Although there are currently many Grids with different implementations and policies, the drive towards a single global infrastructure requires open standards to ensure compatibility. A description of the emerging Grid standards will be presented in Section 1.2.2.

While many different Grid systems exist today, several common elements are shared between them. An overview of a typical Grid system will be given in Section 1.2.3.

1.2.1 A Brief History of Grid Computing

Networked computers first arose more than forty years ago with the creation of ARPANET (Advanced Research Projects Agency Network) [18]. This was pioneering work which led to the first message being sent over a wide-area network (WAN) in 1969.

During the subsequent decades, disparate local area networks (LANs) were created that fuelled the desire for inter-network communication. Mechanisms were subsequently conceived to facilitate this such as Transmission Control Protocol / Internet Protocol (TCP/IP) [19]. TCP allows the efficient delivery of packets of data that are addressed and forwarded via IP. As a result of the development of Ethernet [20] computers could easily be connected to form a LAN. The adoption of TCP/IP as a network communication standard led to the first steps towards the Internet.

The explosion of the Internet as we know it today also relied on developments such as the Domain Name System (DNS) [21] to resolve the readable names of hosts to their numeric IP addresses. Other examples are HTML (Hyper-Text Markup Language) and Uniform Resource Locators (URLs) underlying the World Wide Web [22].

In the last twenty years, the lowering costs of computing hardware, in parallel with the development of high-bandwidth networking, has led to the shift from building large mainframe supercomputers to clusters of PCs. These are the circumstances from which distributed computing has emerged. The scope of distributed computing includes the utilisation of any types of physically separated compute resources and is too broad to discuss here. However, two innovative developments in the use of networked computers, namely Internet computing and Peer-to-peer (P2P) computing have special relevance to the field of Grid computing and shall be examined below.

Internet Computing

Many millions of computers are connected to the Internet at any one time with a significant percentage having an idle CPU (Central Processing Unit) [23]. Due to the prevalence of individual PCs at home, in businesses as well as in institutions across the world, linking these resources together to form a distributed computing pool is an attractive possibility, not least because this computing power would otherwise go to waste. This is the aim of Internet computing projects which utilise the so-called 'cycle-stealing' paradigm. Cycle stealing is perhaps a misleading term since legitimate Internet computing projects work with the consent of owners. Participation usually involves installing some software which only makes use of the CPU when idle, *e.g.* as a screensaver.

The first mainstream Internet computing project was Entropia [24] in 1997 whose remit included many problems of scientific interest. Amongst other things, Entropia was used to identify the largest known prime number [25]. Perhaps the most successful Internet computing project to date, is based on the Search for Extra-Terrestial Intelligence (SETI) programme. SETI relies on public support to search through collected radio signals to detect intelligent life outside Earth. SETI@home [26, 27] allows members of the public to get involved by donating their idle CPU power with over half-a-million PCs regularly participating.

Due to the widespread success of SETI@home, a plethora of other '@home' style projects have appeared, some of which are described in Table 1.2. Many Internet computing projects, including SETI@home, are actually based on the Berkeley Open Infrastructure for Network Computing (BOINC) [28, 29] software which provides the infrastructure to support remote execution of project tasks.

Project	Description
Folding@home	Runs protein folding simulations to understand diseases such as Alzheimers and Parkinsons [30]
Compute-against-Cancer	Study side effects of chemotherapy, structure and behavior of cancer cells and create better ways to screen new cancer drugs [31]
Fight AIDS@home	Assist fundamental research to discover new drugs, using our growing knowledge of the structural biology of AIDS [32]
Einstein@home	Searches for spinning neutron stars (also called pulsars) using data from gravitational wave detectors [33]
LHC@home	Allows users to participate in the design of the LHC by simulating particles in the accelerator [34]

Table 1.2: *A sample of the many '@home' style Internet computing projects and a brief description of their aims.*

In recent times, Entropia has become a commercial venture and there are now several companies offering the spare cycles of computers across the world for profit, examples include: Parabon [35] and United Devices [36].

While Internet computing obviously has similarities to Grid computing and demonstrates the effectiveness and importance of aggregating compute resources, it is a special case of a more complicated problem. The software used in Internet computing is purpose built and tends to be used for massively parallel problems which can be split into more manageable parts. Internet computing projects alone cannot support the execution of varied applications and access to well defined services that are necessary in a Grid context. For example, the LHC experiments require reliable access to data stored in many different locations. The treatment of data on the Grid will be explored in Section 1.6.

Peer-to-Peer Computing

Today's Internet works using a client-server model where, for example, web servers host webpages which are accessible via browsers acting as clients. This is sometimes called a two-tier architecture since there are two types of nodes: clients and servers. Servers are generally passive components which wait for requests and issue a response once requests have been dealt with. Clients, on the other hand, are active components that make requests and wait for a response.

P2P computing involves each participating node acting as both a client and a server. In this model, each member of the network adds to the capabilities of the whole by providing processing power, bandwidth and storage space. This makes such systems ideal for mutual file sharing *e.g.* audio, video or other digital formats. P2P systems can be classified as centralised or decentralised. In the centralised approach, a central server is employed to keep information about individual peers and often responding to requests for information by consulting a central index.

Examples of centralised P2P systems are BitTorrent [37] and Napster [38] which both faced legal controversy over the sharing of copyrighted material. Since then, other systems have emerged which deploy a decentralised P2P system with encryption to ensure the anonymity of users, examples include Gnutella [39] and Freenet [40].

It is important to note that the Grid does not aim to succeed P2P computing. In fact there may be uses for P2P file sharing technologies in a Grid environment. For example, P2P networks could be used to analyse remote datasets by determining which datasets are the most utilised and distributing them accordingly.

Grid Computing

The precursor to Grid computing was known as 'metacomputing'. This term was introduced around 1990 and represented an effort to pool the resources of multiple supercomputers [41]. With the advent of the Grid in the late 1990's [2], a concept with a broader scope than metacomputing emerged, not only attempting to link supercomputers but many different compute resources to realise the dream of computing power available as a utility. The bridge between metacomputing and Grid computing was made through the Globus Toolkit, described as 'a Metacomputing Infrastructure Toolkit' [42]. The protocols and services of the Globus Toolkit are employed by many, if not all major Grid projects today and shall be discussed in Section 1.2.2.

As Grid projects started to evolve, attempts were made to further classify Grids according to their main function:

- *Computational Grids* for CPU intensive applications;
- *Data Grids* concentrating on the infrastructure to manage large amounts of data *e.g.* storage capacity; and
- *Service Grids* focus on the coordinated, collaborative use of distributed resources.

These classifications are starting to become superfluous today since many of the current Grid systems exhibit the functionality of several categories. LCG is one example, which combines elements of all three, and is described in Section 1.5. The applications of Grid systems shall be explored in Section 1.3 and an introduction to Grid computing applied to particle physics will be given in Section 1.4.

1.2.2 Emerging Standards

The second point of Foster's Grid checklist in Section 1.1.1 highlights the need for open protocols and interfaces. Open standards are essential to ensure compatibility and provide a framework for Grid development. This section will begin by outlining web services, an Internet standard on which some Grid standards are based. The Open Grid Forum (OGF) and the Open Grid Services standards will then be explored, followed by a description of Web Service Resource Framework and the Globus Toolkit.

Web Services

As mentioned in Section 1.1.1, the Internet can be thought of as the 'carrier' for Grid computing and several emerging Grid standards employ web services to ensure machine interoperability over a network. One of the main Internet standards bodies is the World Wide Web Consortium (W3C) [43] headed by Tim Berners-Lee. The W3C is engaged in the task of creating standards for the web and has the official aim 'to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web'.

Web services allow communication between applications that can be running on different platforms and written in different programming languages. This is accomplished via a standard mechanism for all exchanges of data, for example, using eXtensible Markup Language (XML). Another standard is used for providing the means to access Web services, namely WSDL (Web Services Description Language).

Open Grid Forum

The two leading Grid standards organisations: Enterprise Grid Alliance (EGA) and the Global Grid Forum (GGF) recently merged to form the Open Grid Forum (OGF) [44]. By combining the expertise of both EGA and GGF, the idea is that the OGF will form a stronger whole to accelerate progress in defining standards and ensuring their adoption by the Grid community. The OGF should also provide more cohesion between academic, industrial and other communities involved in the field.

Two of the most well known Grid standards, Open Grid Services Architecture (OGSA) [3] and Open Grid Services Infrastructure (OGSI) [45] were created by the GGF and will be explored below.

Open Grid Services

The concept of Grid services came about by trying to unify Web services and Grid technology. OGSA is an architecture by which Grid services are defined. In OGSA, each entity in a Grid environment becomes a service, allowing access between components via a common framework. This includes everything from storage and computing resources to applications and databases. OGSA provides a common way to access many Grid services since standards such as XML are used.

OGSI is a companion standard that formally specifies Grid services in more technical detail. For example, OGSI defines interfaces and protocols for the interaction of Grid services. The use of OGSI ensures interoperability between Grids designed using OGSA.

Web Services Resource Framework

The Web Services Resource Framework (WSRF) [46] was inspired by OGSI and developments in Web services technology. For example, WS-Addressing allows Web services to be accessed in a protocol independent way.

The advantage of a strong coupling between Web and Grid services means that familiar Internet applications could be made to run in a Grid environment more easily. WSRF retained almost all of the functional capabilities present in OGSI, while changing some of the syntax to accommodate such Web service developments. WSRF led to a 'refactoring and evolution' of OGSI [47] and is likely to emerge as a Grid standard.

The Globus Toolkit

In many ways the history of the Globus Toolkit (GT) reflects the evolution of Grid standards and the importance of Web services. For example, GT Version 2 preceded many of the Grid standards described above and provided an implementation of all core Grid services. GT Version 3 used the OGSI implementation and the most recent version, GT4.0, has been developed using the WSRF implementation.

The GT is an open source project developed by the Globus Alliance [48], aiming to provide the necessary infrastructure for building Grid computing systems and applications. The key areas in which the GT is involved include: security, information services, data management and resource management. Some elements of the GT shall be discussed during the overview of components of a typical Grid in Section 1.2.3. The usage of GT components will also feature when describing LCG in Section 1.5.

1.2.3 Components of a Typical Grid

Having established the main concepts of the Grid and some of the emerging standards, this section aims to provide an overview of the main components required in a 'typical' Grid system. Where relevant, common usage of the GT infrastructure will be mentioned.

Security

Security is paramount when establishing a Grid system. Participating organisations, whether academic institutions, companies or governments must at least be able to trace any misuse of resources. A security infrastructure should ensure traceability and provide a robust system to deter those who would seek illegal access to resources. The main requirements of a Grid security infrastructure are mechanisms for authentication, authorisation and data encryption.

The Globus Grid Security Infrastructure (GSI) provides the current basis for Grid security and is based on the use of Grid certificates. These certificates can be thought of as a passport or 'digital identity' which use public key cryptography to identify genuine users. Regional Certification Authorities (CAs) issue certificates to users after a local Registration Authority (RA) validates requests. Certificates use X.509 format which is a cryptographical standard for public key infrastructure (PKI). Each certificate has an accessible public part and a password-protected, private portion which is used whenever a user needs to confirm their identity. For use on the Grid, a user would typically create a proxy-certificate which is valid for a finite time period. GT4.0 provides credential management services such as MyProxyServer [49] to minimise unnecessary human involvement in automated operations. The issue of proxy expiration will be further explored in

Chapter 6.

Information Service

Any Grid system must be able to access information about connected resources. This is important in contexts such as testing the overall configuration of the system, compiling resource usage statistics as well as providing information for users and site administrators.

In Globus, the Grid Index Information Service (GIIS), sometimes referred to as the Monitoring and Discovery Service (MDS) provides information about Grid resources and most importantly, their status. This can provide a means for locating which resources are available and becomes essential when tasks are waiting to be executed.

The LCG information system will be explored in Section 1.5.2.

Job Scheduling

When a trusted user wishes to execute an application on a Grid, the process of identifying suitable resources for the job to be executed on is known as job scheduling. Strategies for job scheduling in a Grid environment will be discussed in detail in later chapters. The LCG approach to job scheduling will be mentioned in Section 1.5.3.

The GT does not provide concrete mechanisms to enable job scheduling although several elements such as the MDS, described above, could be used to facilitate this.

Data Management

Data management in a Grid system can cover many aspects concerning storage, retrieval and access to digital media. Tools must also exist in a Grid

environment to facilitate data movement and replication between sites.

In GT4.0, the component which handles data replication operations is known as the Data Replication Service (DRS). In practice, the DRS is rarely used directly, with most Grids opting for a robust solution to data management issues based on file catalogues, see Section 1.6.2. While the DRS offers tools for the discovery and replication of files, file catalogues can provide an internal record of data without explicit dependence on other Grid components such as the information system.

The GT also provides tools for data movement, such as GridFTP (Grid File Transfer Protocol) [50]. GridFTP is built on ordinary FTP but uses GSI for user authentication and authorisation. GridFTP therefore provides a secure, fast and reliable mechanism for data transfer on the Grid.

The treatment of data on the Grid will be discussed in more detail in Section 1.6.

Job Management

Once a job has been scheduled to a particular resource, services are necessary to allow job execution, job monitoring and output retrieval.

In the GT, a component known as the Grid Resource Allocation Manager (GRAM) provides this functionality. In GT4.0, GRAM is available in both Web services and pre-Web services forms to ensure interoperability between systems running different versions of the toolkit.

The provision of services for dealing with many jobs, or a workload, is an important theme of this thesis which will be elaborated upon in later chapters.

1.3 Applications of Grid Computing

In the early stages of the Internet, it is unlikely anyone could have imagined the multitude of applications which are available today. The same will most likely hold true for the applications of Grid computing. This section will describe some of the current applications of Grid computing, focussing on projects which are outside of HEP (since HEP will be the subject of Section 1.4). Grid projects with a scientific stance will be explored in Section 1.3.1, whilst examples of other applications of Grid computing can be found in Section 1.3.2.

1.3.1 Scientific Grid Projects

As mentioned in the introduction to this chapter, the complexity of problems faced in many areas of science are fuelling the need for Grid computing. A selection of projects making use of Grid computing in different disciplines is presented below. All, to some extent, utilise the Globus Toolkit described in Section 1.2.2.

The Virtual Laboratory Project

The Virtual Laboratory Project involves large-scale molecular studies on geographically distributed Grid resources [51]. This helps scientists in the field of molecular biology to screen millions of chemical compounds to determine their potential in the field of drug design. The screening of each compound is expected to take approximately three hours on a standard PC [52].

Earth System Grid

The Earth System Grid (ESG) [53] aims to provide a common grid environment for climate research. The climate models used to simulate changes in our global environment produce tens of Petabytes of data that must be accessible for further analysis.

Grid Enabled Optimisation and Design Search for Engineering

The Grid Enabled Optimisation and Design Search for Engineering (GEODISE) [54] project aims to build a state of the art design tool demonstrator for large-scale distributed simulations involving fluid dynamics. Individual designs, including their optimisations and simulations can approach Terabytes of distributed data and hence require the use of Grid computing technologies.

International Virtual Observatory Alliance

International Virtual Observatory Alliance (IVOA) [55] is a worldwide initiative to create a virtual observatory utilising astronomical archives. This involves the creation of tools, systems and organisational structures accessible for all those taking part. The AstroGrid project [56] forms the UK contribution to the IVOA.

HealthGRID

A Grid for Health, HealthGRID [57], aims to provide a synergy between bioinformatics and medical-informatics. This would allow sharing of resources and collaboration between those studying fields such as genomics, medical imaging and modelling of biological structures. HealthGRID involves both laboratories and commercial companies in a collaborative environment but also offers the provision of computing power and storage capacity at cost.

1.3.2 Commercial Grid Projects

The commercial sector has been heavily involved since the inception of Grid computing. The economic potential of the Internet has continued to be realised in recent times, with a similar level of interest expected for the applications of Grid computing.

In the past few years several companies have begun to offer Grid technologies for business applications such as IBM [7] and Sun Microsystems [9]. Further examples come from Internet computing such as Entropia [24], Parabon [35] and United Devices [36] which offer software to establish Grids within organisations and businesses. Grid consultancy firms have also started to appear, these offer tailored advice on how to apply Grid computing to the immediate needs of individual businesses. Examples include Gridwise Tech [58] and GridSystems [59].

With large scale demonstrations of Grid technology, such as the LCG infrastructure for the LHC in 2007, the commercial aspects of Grid computing are set to grow significantly in the years ahead. An overview of LCG will be presented in Section 1.5.

1.4 Grid Computing Applied to Particle Physics

The HEP community is a major driving force behind the development of the Grid. HEP experiments, such as those at the LHC, will produce data on the Petabyte scale which must be stored and made accessible to physicists for further analysis. Many Grid projects are currently in existence to meet the computing requirements of HEP. Due to the scale of this undertaking, many Grid projects also support other scientific activities. A sample of these are listed below.

- **GridPP** The UK Grid for particle physics GridPP [60] currently links 17 U.K. institutions and is fully functioning. GridPP forms the U.K. contribution to LCG, which is discussed in Section 1.5, and is also part the larger, interdisciplinary EGEE [17] project. A recent proposal has been made to extend the GridPP project and facilitate the exploitation of available Grid resources [61] for use in particle physics. The vision of GridPP is for the Grid to become the primary means of providing compute resources to the U.K. particle physics community. Significant expansion of resources is expected before and during the start of the LHC, reaching an equivalent of 50,000 desktop PCs with over 20 PB of accessible storage capacity by 2012 [61].

- **GriPhyN** The Grid Physics Network (GriPhyN) [62] is based in the U.S. and aims to provide the necessary infrastructure for current experiments in astronomy and particle physics to perform distributed, collaborative analysis of data. Along with the iVDGL and PPDG below, GriPhyN has developed the Virtual Data Toolkit (VDT) to support this task. The VDT includes basic Grid services and tools to support working with distributed datasets in a Grid environment.

- **iVDGL** The International Virtual Data Grid Laboratory (iVDGL) [63] aims to facilitate interdisciplinary experimentation in Grid-enabled, data-intensive scientific computing in a single system. Based in the U.S., the iVDGL also aims to aggregate heterogeneous computing and storage resources in Europe and Asia.

- **PPDG** The Particle Physics Data Grid (PPDG) [64] is involved in the development and deployment of production Grid systems for several experiments in particle physics. PPDG is also working towards the

integration of experiment-specific applications to run in a Grid environment. PPDG is a joint venture between several U.S. laboratories and together with iVDGL and GriPhyN, forms a collaborative trio of U.S. Grid projects for physics.

- **NorduGrid** NorduGrid [65] is a European project which focusses on the development, maintenance and support of Grid middleware known as the Advance Resource Connector (ARC). ARC is freely available and utilises other open source software such as the Globus Toolkit.

- **OSG** Open Science Grid (OSG) [66] is a project built and operated by a consortium of U.S. universities and national laboratories. OSG is comprised of an Integration and a Production Grid. The Integration Grid is used as a testbed for new Grid applications and software whereas the Production Grid provides a stable environment for intensive usage. OSG is principally used for particle physics *e.g.* LHC experiments.

An intentional omission from the above list is LCG, currently the world's current largest Grid, which will be described in the next section.

1.5 The LHC Computing Grid

The main mission of LCG [5] is to build and maintain a data storage and analysis infrastructure for the HEP community that will use the LHC. LCG provides Grid 'middleware' to facilitate this. Grid middleware is the layer of software that provides key services for security, information, data management and access to resources. Therefore, middleware is often thought of as the 'glue' that binds disparate resources together. LCG actively supports Globus and uses the Globus-based VDT, described in Section 1.4 as part of

the project middleware.

LCG is the primary production environment for the EGEE [17] project, which aims to establish a Grid infrastructure for European science. EGEE is leading a worldwide effort to re-engineer existing Grid middleware. For example, LCG-2 middleware on LCG is in the process of being replaced by gLite [67]. The EGEE gLite middleware will be discussed in Chapter 3 with some important components mentioned below

In later chapters topics such as: the paradigms for distributed analysis; different approaches to job scheduling; and possible strategies in order to minimise the start time of jobs will be explored. This discussion depends on an understanding of some LCG components, which will be described below. Whether or not the implementations change due to shifts in the middleware providers, the concepts should remain the same in the future.

The architecture and components of LCG are explained in much greater detail in [68]. The goal here is to present a brief overview of the system, concentrating on the components that play an important role in the treatment of jobs. The LCG Information and Workload Management Systems are two such components and will be outlined in Sections 1.5.2 and 1.5.3 respectively.

1.5.1 A Brief Overview of LCG

This section discusses the key components of LCG, which will be the subject of further discussion later in this thesis. These elements described here include: security mechanisms; the VO membership service; and a description of storage element and computing element interfaces.

Security

The importance of security for Grid systems was discussed in Section 1.2.3. The security infrastructure for LCG must be robust in terms of design and implementation, but also for deployment and operation. Authentication is based on the GSI from Globus, using PKI based on X.509 format digital certificates. Regional CAs act as a trusted third-party that digitally signs the certificate to confirm the binding of the individual identity to the name and the public key [68]. VOMS, described below, is used to incorporate information about the groups and roles of individual users.

Grid proxies are used to access Grid resources with a finite period of validity. When longer-term proxies are needed, MyProxy [49] services can be used to renew the proxy. Sites maintain Certificate Revocation Lists (CRLs) to prevent unauthorised access to Grid resources from expired and compromised user certificates. The Distinguished Name (DN) of a user is a meaningful string which is encoded in all Grid certificates and proxies. This can be used for accounting and further authorisation *e.g.* when accessing storage systems.

Virtual Organisation Membership Service

The Virtual Organisation Membership Service (VOMS) [69, 70] component of gLite allows additional information about users to be incorporated in the proxies which are used to access Grid resources.

This information can include the VO of which the user is a member and also any sub-groups. For instance, in an academic context, sub-groups could include particular research groups or administrators of local systems. VOMS is also used to encode roles and capabilities of users in order to define their access privileges in a Grid environment.

Storage Elements

A Storage Element (SE) is an abstraction of physical storage devices which provide services and interfaces to access them. For example, SEs in LCG provide access to the following: Mass Storage Systems (MSS), including either disk cache or disk cache front-end backed by a tape system; GridFTP service, to provide data transfer in and out of the SE as well as to and from the Grid; and also local Unix, POSIX-like (Portable Operating System Interface) input/output facilities to the local site, providing application access to the data on the SE [68]. SEs also provide a Storage Resource Manager (SRM) [71] interface. SRM allows access to different MSS implementations in a transparent way.

It is important to note that there is not a one-to-one relationship between sites and SEs. In fact, a site may have several associated SEs depending on the available resources located there. SEs also provide access control and traceability based on the use of proxy certificates with a user DN, as well as information about groups and roles provided by VOMS.

Computing Elements

In a similar way to how SEs present an abstraction of storage devices, Computing Elements (CEs) provide an abstraction of compute resources. CEs provide a set of services to enable access to different implementations of local batch systems running on site compute farms. Each site establishes job queues on the local batch system and the CE is used to access them through the Grid.

On LCG, CEs provide the following services and interfaces: mechanisms by which work may be submitted and monitored on local batch systems; and publication of information, including accounting, through the Grid informa-

tion system, which will be described in Section 1.5.2. CEs must also provide authentication and authorisation mechanisms based on the VOMS security model and ensure that user credentials, provided as Grid proxies, are used to create appropriate local mappings by the DN.

Existing CEs on LCG are based on GRAM, part of the Globus Toolkit described in Section 1.2.3, although a new gLite CE is set to replace this. The gLite CE is based on a variant of Condor [72] which will be explored in detail in Chapter 5.

Virtual Data Toolkit

The VDT originally developed by the GriPhyN and iVDGL projects introduced in Section 1.4 is a collection of Grid middleware that can be easily installed and configured. VDT is now used by LCG and the PPDG, with both LCG-2 and gLite middleware components relying on VDT versions of Condor, Globus and MyProxy software. Selected components of gLite such as VOMS are also being added to the VDT.

1.5.2 LCG Information System

The LCG Information System (LCG IS) consists of services that publish and maintain data concerning Grid resources. CEs and SEs publish information, according to the Grid Laboratory Uniform Environment (GLUE) [73] schema, that describes the resources available at a site and their current state. GLUE is an information model for resource discovery and monitoring, which is composed of attributes with a name, multiplicity, type, and description of the content. There are several equivalent implementations of GLUE including an XML representation and LDAP (Lightweight Directory Access Protocol) schema.

The LCG-2 monitoring system is based on Globus such that CEs and SEs each have a local Grid Resource Information Server (GRIS) which sends information to the nearest GIIS. The GIIS then publishes this information using LDAP to a Berkeley Database Information Index (BDII), which adheres to the GLUE information model. The BDII is an LCG implementation of the Globus GIIS based on the Berkeley Database with increased scalability. Figure 1.1 illustrates the LCG IS components and their basic interaction. The GIIS may publish information to several BDIIs which allows, for example, a separate BDII per VO.

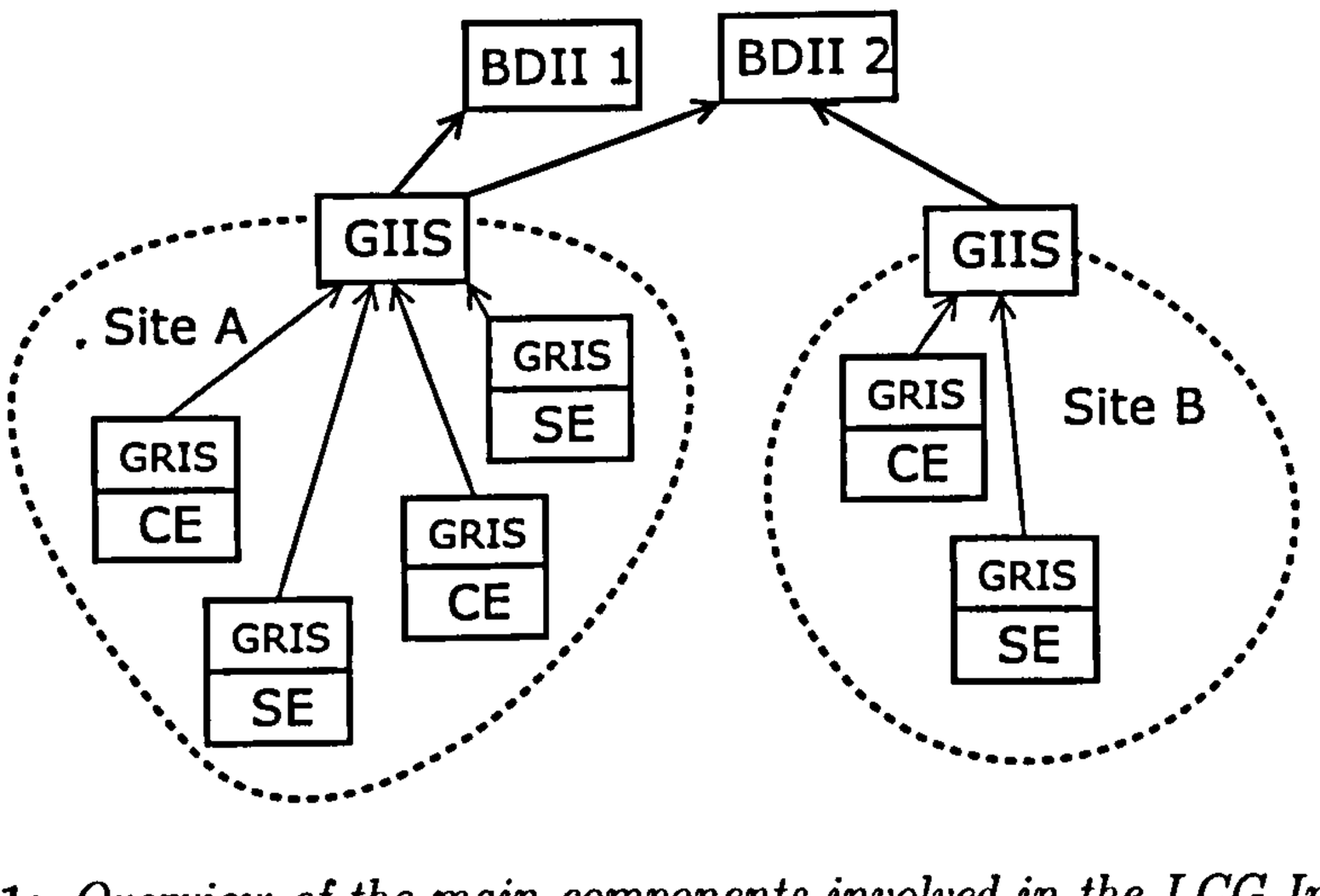

Figure 1.1: *Overview of the main components involved in the LCG Information System and their interactions.*

Both LCG-2 and gLite middlewares rely on the BDII for proper operation. However, the gLite information service implementation is based on a Grid Monitoring Architecture proposed by GGF, called R-GMA (Relational Grid Monitoring Architecture) [74]. R-GMA adopts a consumer/producer model to represent the information infrastructure of the Grid. This works by separating information providers and those which request information with a central registry to mediate communication. R-GMA can use the same infor-

mation providers which populate the BDII and is consequently interoperable with the LCG-2 system.

The BDII is therefore a repository of information on the current state of Grid resources and can be queried by other services such as those in the LCG Workload Management System, described in the next section.

1.5.3 LCG Workload Management System

The Workload Management System in gLite is an evolution of the one in LCG-2. However, the main components are very similar, so the LCG-2 Workload Management System (LCG WMS) will be described here. Both rely on the BDII described in the last section as an information system and the gLite Workload Management System will be interoperable with LCG-2 CEs. This section will present an overview of the main components, omitting technical details where possible. Discussion in future chapters will rely on an understanding of the concepts introduced here.

The LCG WMS provides basic job management facilities such as job submission, job deletion and monitoring and is also responsible for accounting and error reporting. It makes use of Condor and Globus technologies and relies on Globus GSI security. The user interacts with the LCG WMS using a Command Line Interface or APIs (Application Programming Interfaces) with tasks specified by a Job Description Language (JDL) based on Condor Classads [75]. As shown in Figure 1.2, the LCG Resource Broker (RB) [76], accepts and satisfies job management requests from clients in order to submit jobs to a suitable CE and finally to a Worker Node (WN). A WN is a compute resource (*e.g.* node of a batch system) that provides CPU power to process a task. The user may interact with a MyProxy [49] server in order to prevent proxy expiration whilst a job is running.

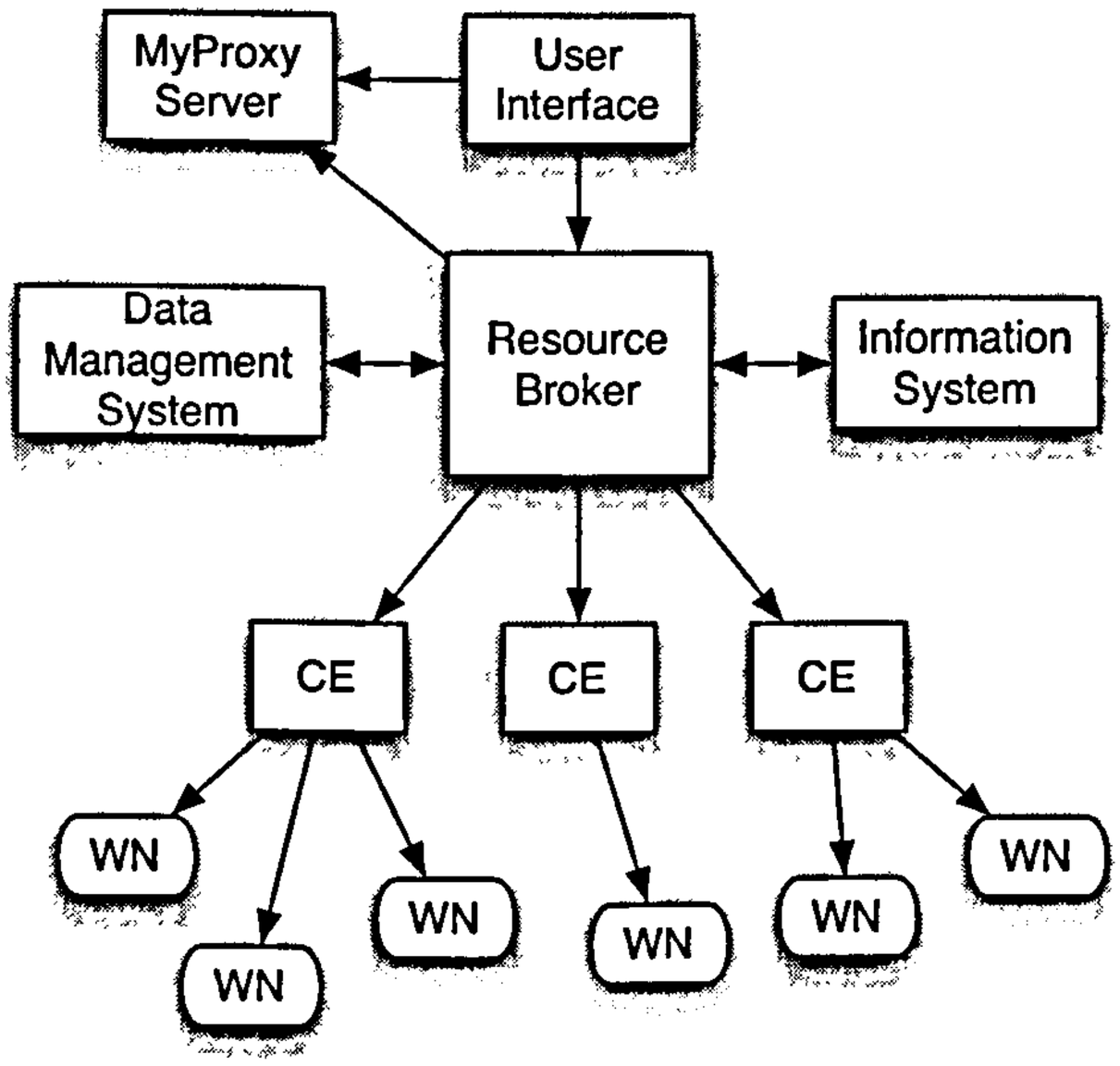

Figure 1.2: *Illustration of the LCG workload management components used during job submission and their interactions.*

The LCG RB schedules jobs based on the Condor [76] centralised scheduling mechanism. This shall be further discussed in Chapters 3 and 5, with the decision made based on a matchmaking process. The RB accesses information about resources through the LCG IS, organised according to the GLUE schema, and published through a BDII. Jobs are then dispatched to appropriate CEs, depending on such factors as: job requirements; availability of resources; and also any policies that are in place on particular sites. Policies may be in place to give priority to local users on site resources or, on the level of the VO, to provide a certain quality of service. For jobs with input data requirements, the data management services of LCG are used to determine suitable SEs and hence CEs for the job. This will be described in the next section.

1.6 Data on the Grid

One of the most difficult challenges for HEP Grid computing is to provide reliable and efficient access to input datasets. It is essential to the success of the LHC experiments that physicists are able to access the data produced by the LHC detectors. Distributed data analysis frameworks aim to provide this means. A mechanism for enabling reliable access to datasets in a Grid environment will be described in Chapter 5. This requires an understanding of how data is treated on the Grid, which will be discussed in Section 1.6.1. The LCG File Catalogue (LFC) plays a crucial role in the handling of distributed data and shall be introduced in Section 1.6.2.

1.6.1 Treatment of Data in a Grid Environment

In order to introduce the treatment of data in a Grid environment some definitions are first required. All data is specified by a meaningful 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. These replicas may be at the same or different Grid sites corresponding to different SEs. Each file may have several LFNs associated with it according to user defined names. This is analogous to the use of SymLinks in a Unix environment.

PFNs are also referred to as Storage URLs (SURLs) since their names are determined by the Grid SE on which the replica exists. In order to access files, Transport URLs (TURLs) are used. TURLs are temporary locators of a replica which include a protocol determining how the files can be accessed and understood by SEs.

Files must be uniquely identifiable on the Grid and the use of Globally

Unique Identifiers (GUID) facilitates this. GUIDs are 128-bit hexadecimally grouped strings which provide a sufficient number of combinations to address all files on the Grid.

This complex machinery is required since files on the Grid may exist in many different geographically distributed storage systems. Figure 1.3 outlines the relationship between LFNs, SURLs, TURLs and GUIDs for a typical file in a Grid environment.

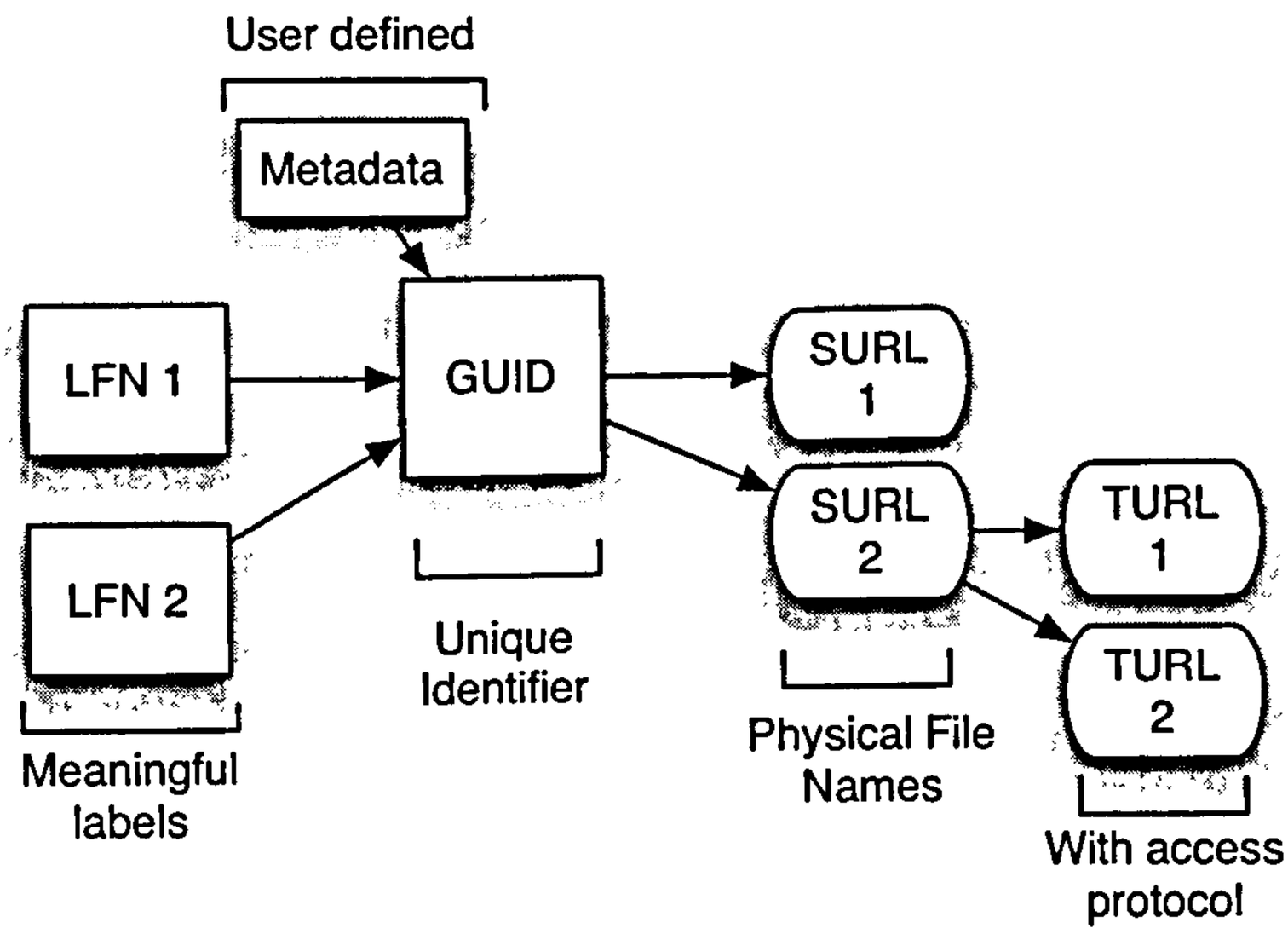

Figure 1.3: *Overview of the treatment of data in a Grid environment. This Figure shows the relationship between LFNs, SURLs, TURLs and GUIDs for a typical file.*

In practice, GUIDs are not user-friendly file names and it is clear why LFNs are preferred. The complicated nature of addressing files in a Grid environment should be masked from the end user as much as possible. Chapters 5 and 6 will outline some of the steps taken to achieve this. In the context of submitting jobs with input data requirements to a traditional batch system, users would have to specify the exact PFNs (SURLs) on which to run. In a

Grid environment, the machinery should restrict users to only ever dealing with LFNs.

1.6.2 The LCG File Catalogue

The LCG File catalogue (LFC) [77] offers a hierarchical view of the logical file name space and will be discussed in more detail in Chapter 4. The LFC provides a LFN to SURL translation using file GUIDs and allows the determination of which site a given file resides. The LFC exposes an API that provides Unix style permissions and POSIX Access Control Lists (ACL) to define ownership. The LCG RB interacts with the LFC through the Data Location Interface (DLI) in order to resolve the suitable SEs for the requirements of jobs. Metadata can be associated with file entries, *e.g.* information about the file defined by the user. The LFC supports Oracle [78] and MySQL [79] databases and can also be interfaced through Python.

In order to perform data management operations on files stored at SEs, LCG has developed Grid File Access Library (GFAL). This is a POSIX-like layer for access to Grid files via their LFN and provides familiar style of calls to open, read, write and close files while interfacing to the LFC.

1.7 Summary

In this chapter, the basic principles of Grid computing have been described and set in the context of experimental particle physics. The concept of the Grid was presented in Section 1.1 along with some definitions and the idea of computing power on demand, as a utility.

The history of distributed computing leading to Grid systems was presented in Section 1.2 along with some discussion of the emerging standards

in the field. Components of a typical Grid system were also mentioned, highlighting the Globus Toolkit components which are commonly utilised.

Some applications of Grid computing were described in Section 1.3 where increasingly prevalent commercial projects were highlighted. Grid computing projects focussing on particle physics were summarised in Section 1.4, demonstrating that HEP is one of the main driving forces behind Grid computing.

An overview of LCG was given in Section 1.5 with emphasis on components which have relevance in the context of distributed data analysis jobs. In Section 1.6, the treatment of data on the Grid was introduced. Accessing datasets in a Grid envrionment is vital to the success of the LHC experiments and the LCG WMS relies on the LFC for this.

In subsequent chapters, this thesis will further explore many of the concepts introduced here. The next chapter will introduce LHCb (Large Hadron Collider beauty) which is one of the four main LHC experiments. LHCb must rely on Grid technologies to successfully store and access data from the detector. In particular, the LHCb application software and computing model will be discussed. Chapter 3 will highlight the paradigms associated with distributed data analysis on the Grid, showing how the LCG resources described in this chapter are utilised. Chapter 4 will introduce the system that has subsequently been adopted by LHCb for performing distributed data analysis on LCG. Optimisation strategies in order to minimise the start time of user analysis jobs will be discussed in Chapter 5. The results of providing an analysis service to real users will be presented in Chapter 6 and overall conclusions will be given in Chapter 7.

Chapter 2

LHCb Software Environment and Software Distribution

The Large Hadron Collider (LHC) is based at CERN [80], the European Organisation for Nuclear Research, in Geneva, Switzerland. The LHC is a proton-proton collider with a 27 km circumference and will be the world's most powerful particle accelerator when it comes fully online. Two proton beams, each carrying bunches of 7 TeV protons, will travel in opposite directions and collide at four points corresponding to the detectors of the four main experiments: ALICE [81]; ATLAS [82]; CMS [83]; and LHCb [84]. Figure 2.1 highlights the four main LHC experiments, CERN, and the surrounding region.

The Standard Model (SM) is the current theory that describes the fundamental properties of matter. Although this has been tested rigorously over the years, many important questions remain unanswered. Examples of potential discoveries within the scope of the LHC include:

- Particles acquire mass via the Higgs mechanism according to the SM. If the mediating particle (the Higgs boson) exists, it should be detectable;

Figure 2.1: *Aerial view of the CERN and the surrounding region. The largest ring is the LHC that has a circumference of 27km. The approximate locations of the ATLAS, ALICE, CMS and LHCb experiments are also highlighted. This figure is modified and reproduced from [80].*

- The existence of supersymmetry, potentially leading to the unification of the four fundamental forces;

- Further exploration of CP violation should place more stringent limits on the SM and lead to a deeper understanding of the matter-antimatter imbalance that exists in the universe today; and

- Observation of the transition to a new state of matter (the quark-gluon plasma).

The LHC experiments will test the SM at a new level of precision and are in an excellent position to explore possible new physics beyond the SM. Of the four main experiments, ATLAS and CMS are 'general purpose' detectors that attempt to encompass as broad a range of physics as possible. Whilst ATLAS and CMS have similar physics goals, their respective designs and implementations are distinct. Of the remaining experiments ALICE is a

dedicated heavy-ion detector, studying strongly interacting matter at high densities, whereas LHCb has been designed to study B-physics.

This chapter has two main aims, the first of which is to introduce the LHCb experiment. This will include a brief discussion of the physics objectives of LHCb and an overview of the detector in Section 2.1. The LHCb software framework and data processing applications are examined in Section 2.2, with the computing model being described in Section 2.3.

Secondly, it is essential that application software is successfully distributed to the WN where the job is executed on the Grid. This chapter will also describe work performed in evaluating software distribution mechanisms for use by LHCb in Section 2.5.

2.1 Introduction to LHCb: Physics Aims and Detector

The LHCb [85,86] experiment is a forward single arm spectrometer that has been designed principally to study CP violation in the b-quark sector at the LHC. The physics aims of LHCb will be explored in Section 2.1.1. This is followed by an overview of the detector in Section 2.1.2. Lastly, the treatment of detector data from the LHCb trigger to the Grid will be explored in Section 2.1.3.

2.1.1 LHCb Physics Aims

The two B-factory experiments, BaBar [87] and Belle [88], were the first to observe CP violation with B-mesons. Other recent and ongoing experiments in the field include CDF and D0 at the Tevatron [89] where the heavier B-

mesons are accessible. CDF and D0 recently made the first measurements of B_S oscillations [90] and also a precision measurement of the B_C mass [91]. LHCb is a second generation experiment which will operate at a centre of mass energy of 14 TeV. With this large energy and high luminosity of the LHC, a large statistics B physics sample will be available to LHCb. This will allow measurements to be made to a higher precision compared with previous experiments. In addition, LHCb will be capable of investigating a larger number of decay channels than previously accessible.

The main physics aim of LHCb is to measure CP violation in a variety of decays of B-mesons to place stringent limits on the consistency of Unitarity Triangles derived from the Cabibbo-Kobayashi-Maskawa (CKM) quark mixing matrix in the SM. With the high level of statistics available to the experiment, it is possible for LHCb to analyse decay modes having small branching ratios. Some examples of the results accessible to LHCb include:

- First measurement of CP violation in the B_S system;
- Precision measurement of the B_S mass and width differences (Δm_s and $\Delta\Gamma_s/\Gamma_s$);
- Observation of rare B decays such as $B_S \rightarrow \mu^+\mu^-$; and
- Precision measurement of the angle γ of the unitarity triangle.

The nominal LHCb luminosity of 10^{32} cm^{-2}s^{-1}, is expected to produce approximately 10^5 B particles per second [86]. However, the B hadrons of interest for CP violation studies all have small (less than 10^{-4}) branching fractions and the bb cross section is two orders of magnitude smaller than the total visible cross section [92]. Moreover, LHCb requires fast track reconstruction with high efficiency in order to distinguish between the B decays

and the plethora of background events arising from the proton-proton collisions. Therefore, one of the biggest challenges for LHCb is to ensure a selective and sophisticated trigger system, which will be described in Section 2.1.3. The design of the LHCb detector is explored in the next section.

2.1.2 LHCb Detector

Figure 2.2 shows a side view of the LHCb detector that has the following main components: the Vertex Locator (VELO), covering the region where

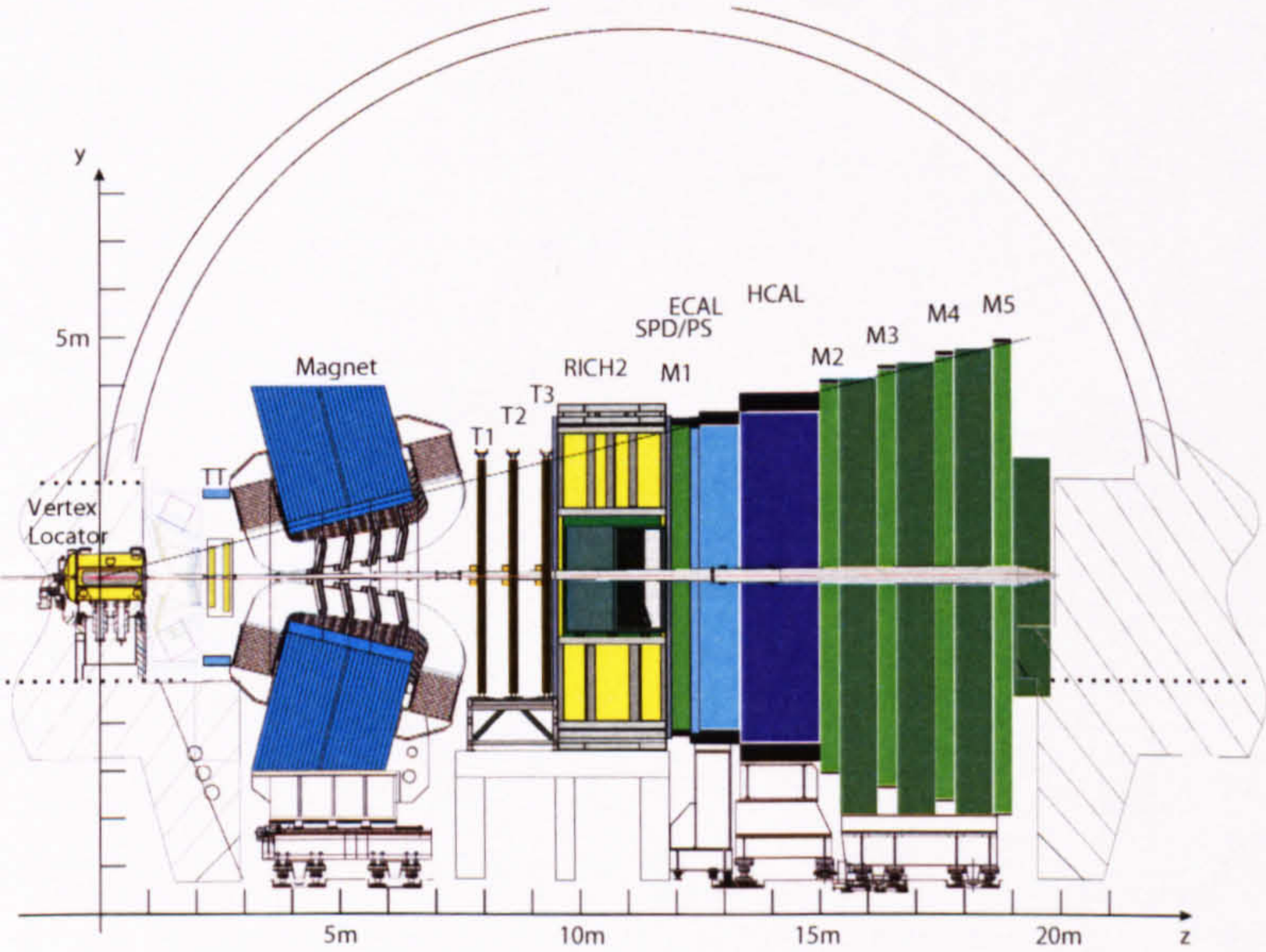

Figure 2.2: *The reoptimised LHCb detector, from [86].*

protons arriving from the left and right will eventually collide; the beam pipe; the dipole magnet; the tracking system (TT, T1-T3); two Ring Imaging Cherenkov detectors (RICH1, RICH2); the calorimeter system (SPD/PS, ECAL and HCAL); and the muon system (M1-M5). The important factors for LHCb include: the ability to reconstruct the B production and decay

vertices; particle identification; and triggering. The main components of the LHCb detector are further explored below:

- **VELO** The VELO is situated around the proton-proton interaction point and is used to identify forward travelling tracks with a high impact parameter and reconstruct primary and secondary vertices. The VELO features a series of silicon stations situated along the direction of the beam and is described in more detail in [93]. The VELO is a principle component of the LHCb tracking system and information from the VELO is also used in the trigger, to reject background decays;

- **Magnet** The LHCb magnet is shown in Figure 2.2. LHCb chose a warm dipole magnet with a field strength of 4 Tm which is described in more detail in [94];

- **TT/T1-T3** The momentum of charged particles can be measured by the amount they are deflected in the magnetic field and the tracking detectors establish this via efficient track reconstruction. Data from the Trigger Tracker (TT), Tracking Stations (T1-T3) and the VELO is used to make the trigger decision [92]. The LHCb Tracking system is composed of a silicon based Inner Tracker and an Outer Tracker made of gas straw tubes [95, 96];

- **RICH1/RICH2** The RICH1 and RICH2 detectors [97] are used for particle identification, both are necessary in order to cover the whole momentum range (between 1 and 100 GeV/c). The RICH detectors use the Cherenkov effect to determine the velocity of charged particles. This is used in combination with the momentum from the Tracking system to determine the mass of particles;

- **ECAL/HCAL** The electromagnetic and hadron calorimeters (ECAL and HCAL) [98] measure the energy and position of charged and neutral particles. The ECAL measures electromagnetic showers of electrons and photons. The HCAL is situated behind the ECAL and measures the hadronic showers of pions, kaons and protons;

- **SPD/PS** Two separate detection layers are placed in front of the ECAL. These are the scintillator pad detector (SPD) and the preshower detector (PS). The SPD and PS are used to determine how the electromagnetic shower from the ECAL evolves longitudinally, relative to the detector; and

- **M1-M5** All detectable particles except for muons are absorbed by the calorimeter system [98]. Therefore a separate system, M1-M5 in Figure 2.2, is used to identify the muons. M1 is is placed before the calorimeters in order to decrease the error associated with particle scattering in the calorimeter when measuring the momentum [99]. The remaining muon detectors, M2-M5, are located behind the calorimeters as shown in Figure 2.2.

The next section will discuss the LHCb trigger system, describing the mechanism by which data is selected and eventually stored on the Grid.

2.1.3 From the LHCb Trigger to the Grid

The LHCb experiment plans to operate at a luminosity of 2×10^{32} cm^{-2}s^{-1}, which is a factor of 50 lower than the design luminosity of the LHC (10^{34} cm^{-2}s^{-1}). As the luminosity increases, multiple proton-proton interactions occur in a single bunch crossing. The background decays must be distinguished from B decay vertices and the luminosity of 2×10^{32} cm^{-2}s^{-1} was chosen in order to

optimise the triggering of the detector. This was based on an optimisation study in [85] that showed only 10% of beam crossings contain more than one hard proton-proton interaction at a luminosity of 2×10^{32} cm^{-2}s^{-1}.

LHCb will operate a two-level trigger [92, 100] that includes: a hardware trigger, Level 0 (L0); followed by a software trigger, the High Level Trigger (HLT). The HLT is run on a dedicated farm of approximately 1800 CPUs. The LHCb Trigger system must reduce the expected 40 MHz LHC beam crossing rate to a value of 2 kHz [100] before moving the data to permanent storage on the Grid.

The L0 trigger will reduce the 40 MHz LHC beam crossing rate to 1 MHz by only selecting events which contain decay particles with a high transverse momentum (p_T) larger than 2 or 3 GeV. This is because b-hadrons will decay to a high energy lepton, hadron or photon due to their large mass. The L0 trigger reconstructs the highest energy hadron, electron and photon clusters in the Calorimeters as well as the two highest p_T muons in the Muon Chambers. This information is fed to the L0 Decision Unit to select events. At this point, events can also be rejected based on global event variables such as track multiplicities and number of interactions. Background decays from other proton-proton interactions occurring within the same beam crossing are called pile-up vertices. Pile-up vertices can significantly reduce trigger efficiency and the L0 trigger plays an important role in reducing this effect. This is performed by a dedicated pile-up veto system consisting of three silicon planes in the backward direction of the VELO.

All the necessary data from the L0 trigger is stored at the 1 MHz output rate so that the HLT algorithm can be processed. The HLT algorithm has access to all the information from the detector. An important element of the HLT is identification of secondary vertices using the VELO and tracking

system. The selection is made after selection cuts for specific final states and confirmation of the L0 decision with greatly increased resolution [92]. The former involves the results of HLT algorithms which determine the decay chain of events and filter them according to specific selections. The HLT reduces the 1 MHz output rate of the L0 trigger to a rate of 2 kHz.

A full reconstruction of events passing the HLT is performed on the CPU farm before sending the data to storage. During data taking, reconstruction and first stripping of the data is expected to take place within a few days of production [100].

Subsequent additional stripping phases and the re-processing of data is expected to last for durations of one and two months respectively for data collected over a year (10^7 seconds) of running. Stripping and re-processing will take place at CERN and national-level computing facilities (Tier-1 centres) available to LHCb. The LHCb Computing Model, described in Section 2.3, will elaborate on these activities.

2.2 LHCb Software: Gaudi, Gauss, Boole, Brunel and DaVinci

The LHCb software is described in detail in the Computing Technical Design Report [100], a brief overview of the key components is given here. An architecture-centric approach was adopted in order to create a resilient software framework capable of withstanding changes in requirements and technology for the lifetime of the experiment. The LHCb software is developed in C++. Section 2.2.1 will describe the Gaudi framework, a general Object-Oriented framework that aims to provide a common infrastructure and environment for the different software applications of the experiment.

The main LHCb data processing applications which encompass the phases from simulation to reconstruction and analysis will be introduced in Section 2.2.2 and are all built within the Gaudi framework.

2.2.1 The Gaudi Framework

The Gaudi architecture [101] was conceived to provide a software framework useable by the entire LHCb collaboration for the simulation, reconstruction and analysis of proton-proton interactions at the LHC. Physicists working on LHCb typically write customised code for simulation, reconstruction and analysis of data. Therefore, the software framework must be flexible enough to support this activity without having all the specific requirements of the user code in advance. In other words, it should be simple for the end user to write any necessary code without having to duplicate functionality already present in the framework because a particular use case wasn't considered. To this end, many components have been identified which have specific functionality and well-defined interfaces. Components interact with each other through their interfaces and together provide all the functionality of the framework.

In the Gaudi framework, software blocks known as 'algorithms' and 'tools' are elements that have well defined input and output data. A clear separation exists between data objects and algorithms. For instance, algorithms and tools are what process the data objects necessary to perform event simulation, reconstruction and analysis. Whereas, data objects are containers of data quantities such as vectors or matrices. This decoupling allows data objects to remain stable over time, and therefore algorithms can be developed independently, at their own pace.

Data flow between algorithms occurs via the Transient Store. By distin-

guishing between transient and persistent representations of data objects, the Gaudi framework shields algorithms and tools from underlying technologies. Algorithms can only see data objects in the transient representation. This means that physics code can withstand changes to the technology employed in the framework to store persistent data objects. For example, a change was recently made from ROOT/IO [102] to POOL [103] without adversely affecting the algorithms. This also means that it is relatively simple to implement and test new technologies to optimise the framework.

There are three types of Transient Store in the Gaudi framework, which correspond to different categories of data with different access patterns during the lifetime of a job [100]. These include:

- **Transient Event Store (TES)** Event data is obtained from real or simulated particle collisions and is handled by the TES on an event by event level

- **Transient Detector Store (TDS)** Detector data that describes the detecting apparatus is handled by the TDS for the duration of many events

- **Transient Histogram Store (THS)** Statistical data derived from processing a set of events is dealt with by the THS at the level of a complete job.

Some of the experiment specific core software components within the Gaudi framework are the LHCb Event Model, the Conditions Database and the Detector Description, these are discussed in turn below.

LHCb Event Model

The LHCb Event Model is defined as the set of classes (and relationships between classes) needed to describe both simulated and real LHCb event data [104]. The Gaudi TES is used to exchange event data inside the event-processing loop. Algorithms simply retrieve their input data from the TES and publish their output data to the TES without needing to know how their input data was produced. This is made possible through the use of a tree structure analogous to a Unix file system.

The same classes in the LHCb Event Model may be used for reconstructed real data and reconstructed simulated data. This is accomplished by restricting relationships between classes to only those adjacent in the data processing sequence. However, it is still possible to perform comparisons between objects which are distant in the processing chain through the use of tables which can be accessed via association code.

Conditions Database

The Conditions Database (ConDB) aims to provide a means to handle information regarding the current running conditions of the LHCb sub-systems which may vary in time. Each condition will have an interval of validity which can be superseded by a newer version. The Gaudi ConDB service provides a framework for users to access conditions data.

LHCb Detector Description

The Gaudi Detector Description Service provides a full description of all detector elements through the use of volumes. Logical volumes represent the shape and composition of an object without reference to its position in space. Conversely, physical volumes include the placement of an object in space and

a top-level volume contains the whole LHCb detector, along with part of the cavern it will be housed in.

Detector elements are stored and accessed via the TDS making use of its hierarchical nature. Logical and physical volumes are used in order to simplify the description of repetitive volumes. Information regarding the material from which the volumes are made is also stored. One of the main users of this service is Geant 4 [105] for the purposes of detector simulation.

2.2.2 Data Processing Applications

Data processing applications are collections of software packages, including algorithms and tools, that are grouped in order to perform a particular task. The data processing applications for LHCb are built within the Gaudi framework, they share and communicate via the LHCb Event Model and make use of the LHCb Detector Description. Each application is a producer and/or consumer of data for the other stages. As shown in Figure 2.3, Gauss handles simulation of events whilst Boole takes the 'hits' generated by Gauss and applies the detector response. The digitization step also includes simulation of the read-out electronics and L0 trigger hardware. The resulting Raw Buffer output has the same format as data coming from the detector.

Brunel is the reconstruction application and takes the Raw Buffer output from Boole, or real data from the detector, as input. This produces either a reduced Data Summary Tape (rDST) for use in production analysis, stripping (see Section 2.3) or a complete DST for use in end-user analysis. In both cases it is possible to output all events or only those corresponding to a particular selection. The reconstruction is completely independent of the Monte Carlo truth information and all access to this data occurs in a dedicated phase which can be switched off when processing real data. This

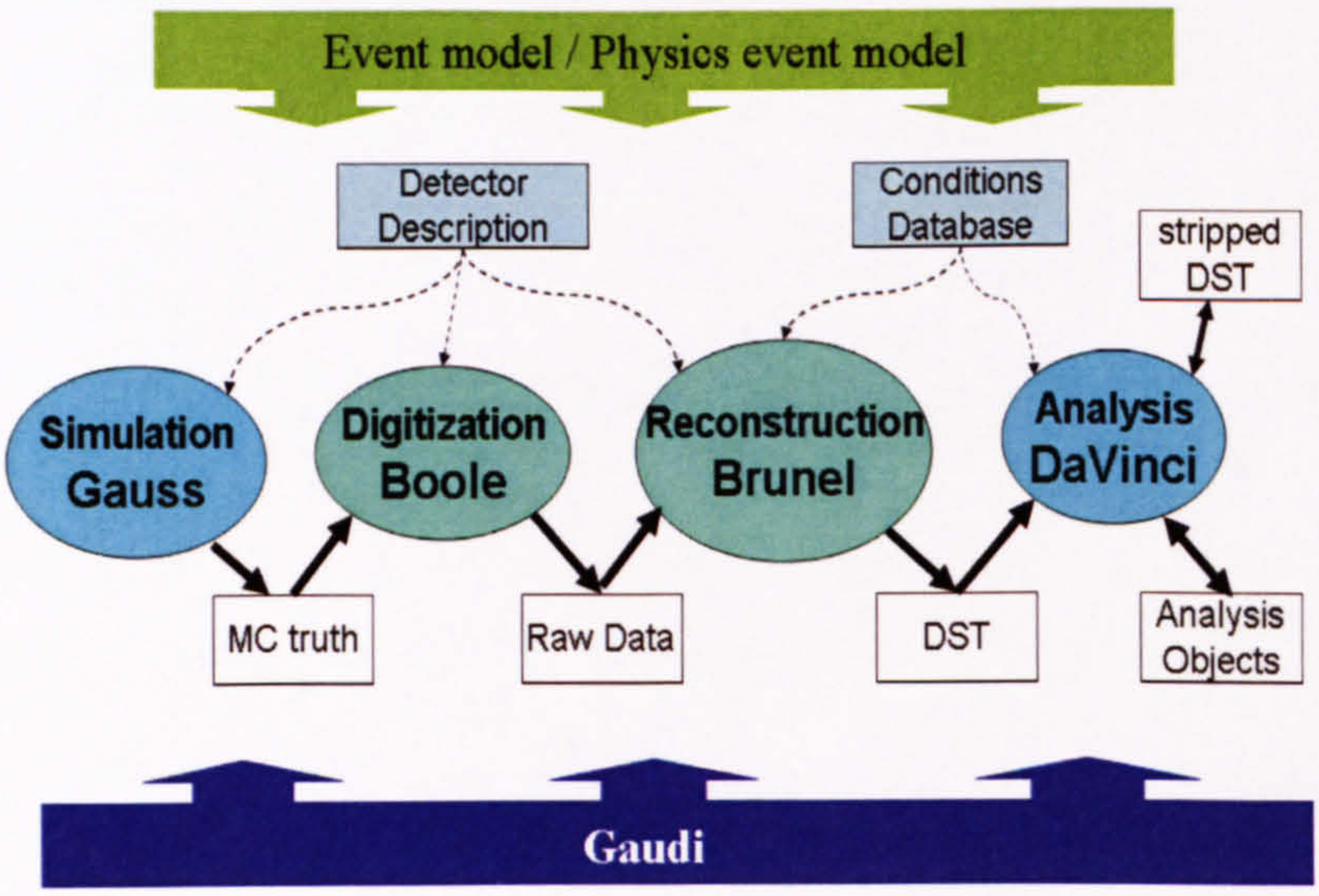

Figure 2.3: *The data flow of the LHCb data processing applications: Gauss; Boole; Brunel; and DaVinci. The Gaudi framework underlies these applications which share and communicate via the LHCb Event Model, from [100].*

guarantees that the same algorithms can be run on both real and simulated data.

DaVinci is the LHCb analysis framework which further processes the DST or rDST output of Brunel to produce Analysis Objects. The output of DaVinci can include: statistical or event data; histograms; and Ntuples (files containing physics objects) that can also be written for further processing.

The data processing applications are 'steered' through job options files. An application manager in the Gaudi framework controls which algorithms are instantiated and when to execute them using the job option files. Typically, inputs from a user are therefore algorithms, in the form of Dynamically Linked Libraries (DLLs), and/or job options files. The Gaudi framework and services discussed in Section 2.2.1, along with the data processing applications above, make up the complete LHCb software system.

2.3 LHCb Computing Model

The LHCb detector will generate approximately 1 PB of data per year when it comes online. 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 [100]. 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 [4, 106].

The model adopted by LHCb involves the MONARC hierarchical system of classifying sites [107]. The 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. The LHC Computing Grid (LCG) project [5] will provide all the distributed computing resources for LHCb.

2.3.1 Logical Dataflow and Workflow Model

The processing of event data occurs in several well defined phases, the terminology and outputs at each step are discussed below.

RAW Data

As mentioned in Section 2.1.3, data is collected and triggering occurs on events of interest. RAW data are transferred to the CERN Tier-0 centre for archiving and further processing. The data not passing the final trigger selection are discarded at this point. The size per event of the RAW data is 25 kB [100].

Simulated Data

RAWmc data sets contain simulated hit information as well as 'truth' information and are produced from a detailed Monte Carlo model of LHCb. The 'truth' information records the physics history of the event which is carried to subsequent steps for use in analysis. Simulated data sets are therefore larger than real raw data (approximately 500 kB/event [100]) but nevertheless have an identical format to that of the real data and are processed using the same reconstruction software.

Reconstruction

Simulated and real RAW data must be reconstructed in order to provide physical quantities. The event reconstruction results in the generation of new data in the form of the Data Summary Tape (DST). During reconstruction, only enough data will be stored to allow the physics pre-selection algorithms to run at a later stage. This is known as a reduced DST (rDST). The DST format has a size of 75 kB/event and this is significantly lower for the rDST, 25 kB/event [100]. After the initial processing of data as described in Section 2.1.3, re-processing of the data is planned to occur once per year after the data taking has finished and then periodically as required. In order to take into account changing detector conditions such as alignment or calibration as well as improvements in algorithms, the reconstruction step will be repeated to regenerate new improved rDST information.

Data Stripping

Each channel of interest for LHCb provides a pre-selection algorithm in order to identify suitable particles. The rDST from the reconstruction phase is analysed in a production-type mode, which selects event streams for further

user analysis.

Those events passing the selection criteria are fully reconstructed to include all the information associated with the event. The RAW data is also added at this point. The output of this phase is the full DST, which contains more information than the rDST and has an approximate size of 100 kB/event.

To provide a quick means to access events of interest, an Event Tag Collection (ETC) is created. This contains a brief summary of the characteristics of each event, as well as results from the pre-selection algorithms. The event tags are stored in files independent of the actual DST files.

User physics analysis is expected to be performed from the output of this phase of data processing, using the full DST plus the RAW data and TAG. Data stripping is expected to be performed four times per year, twice associated with the reconstruction or re-processing of data and twice outside these periods [100].

Analysis

Physicists will run their analysis jobs processing the DST output of the stripping phase. Figure 2.4 outlines the user analysis cycle.

Physicists run on selected DSTs possibly using an ETC along with their own algorithms. Typical outputs for user analysis include histograms, Ntuples, statistical data in the form of a text file, or personal DSTs. Due to the collaboratory nature of particle physics experiments, including LHCb, this data could be shared by individuals in many different countries. Therefore, it is necessary to make the outputs of analysis private whilst allowing the sharing of data within a secure context.

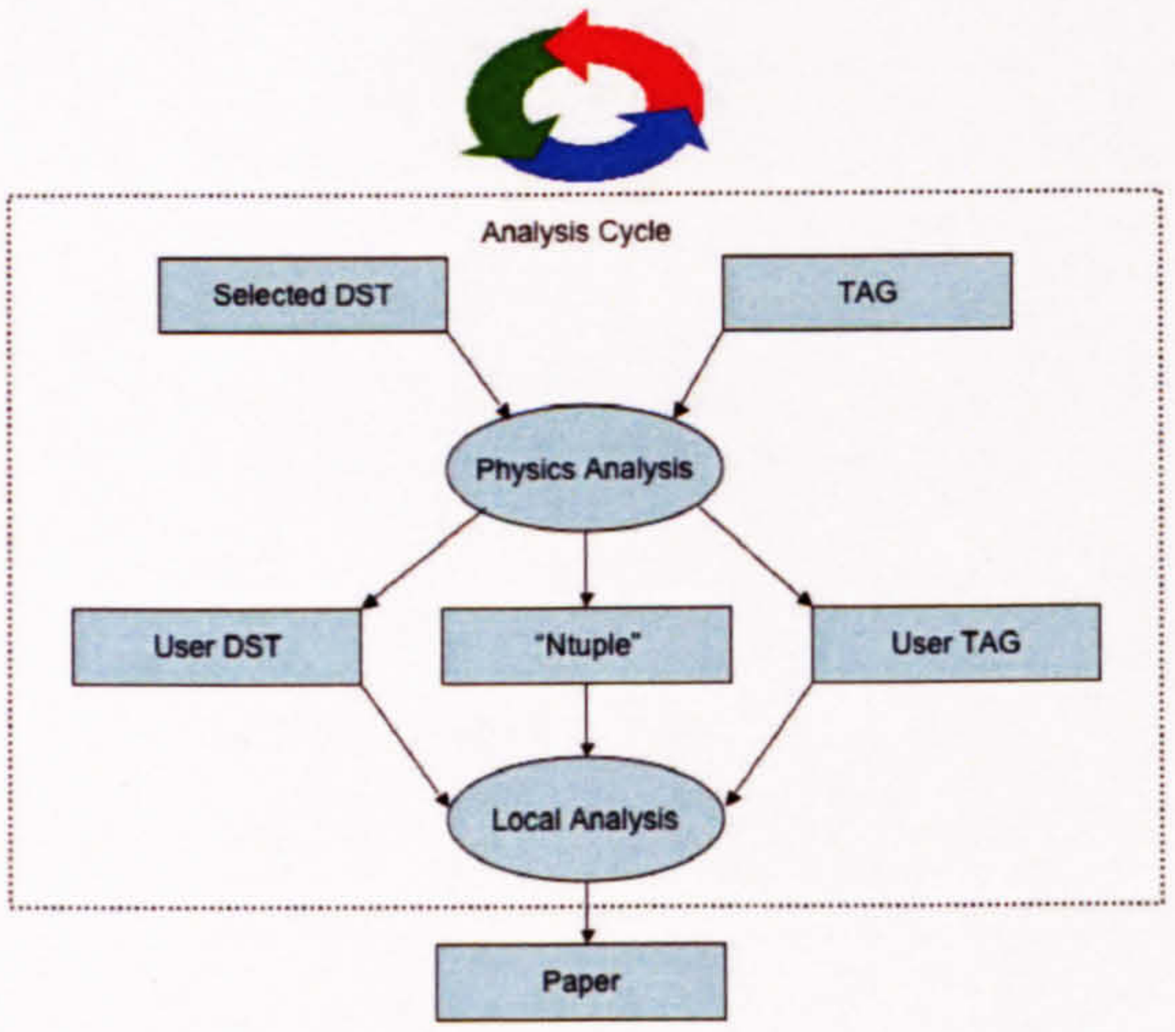

Figure 2.4: *The LHCb user analysis cycle, from [100].*

2.3.2 Computing Model

As shown in Figure 2.5, CERN is the central production centre and also takes the role of a Tier-1 centre. A further six Tier-1 centres have been identified for use by LHCb these include: CNAF (Italy); FZK (Germany); IN2P3 (France); NIKHEF (The Netherlands); PIC (Spain) and RAL (United Kingdom).

In addition there are roughly fourteen Tier-2 centres mostly based at universities throughout Europe. The RAW data from the detector will be stored at CERN with a further copy distributed across the Tier-1 centres [100]. Production of stripped DSTs will occur at these sites and therefore it is envisaged that the majority of the distributed analysis activity will occur there. Tier-2 centres will primarily be Monte Carlo production centres, with the simulated data being transferred to CERN and the Tier-1 sites for storage.

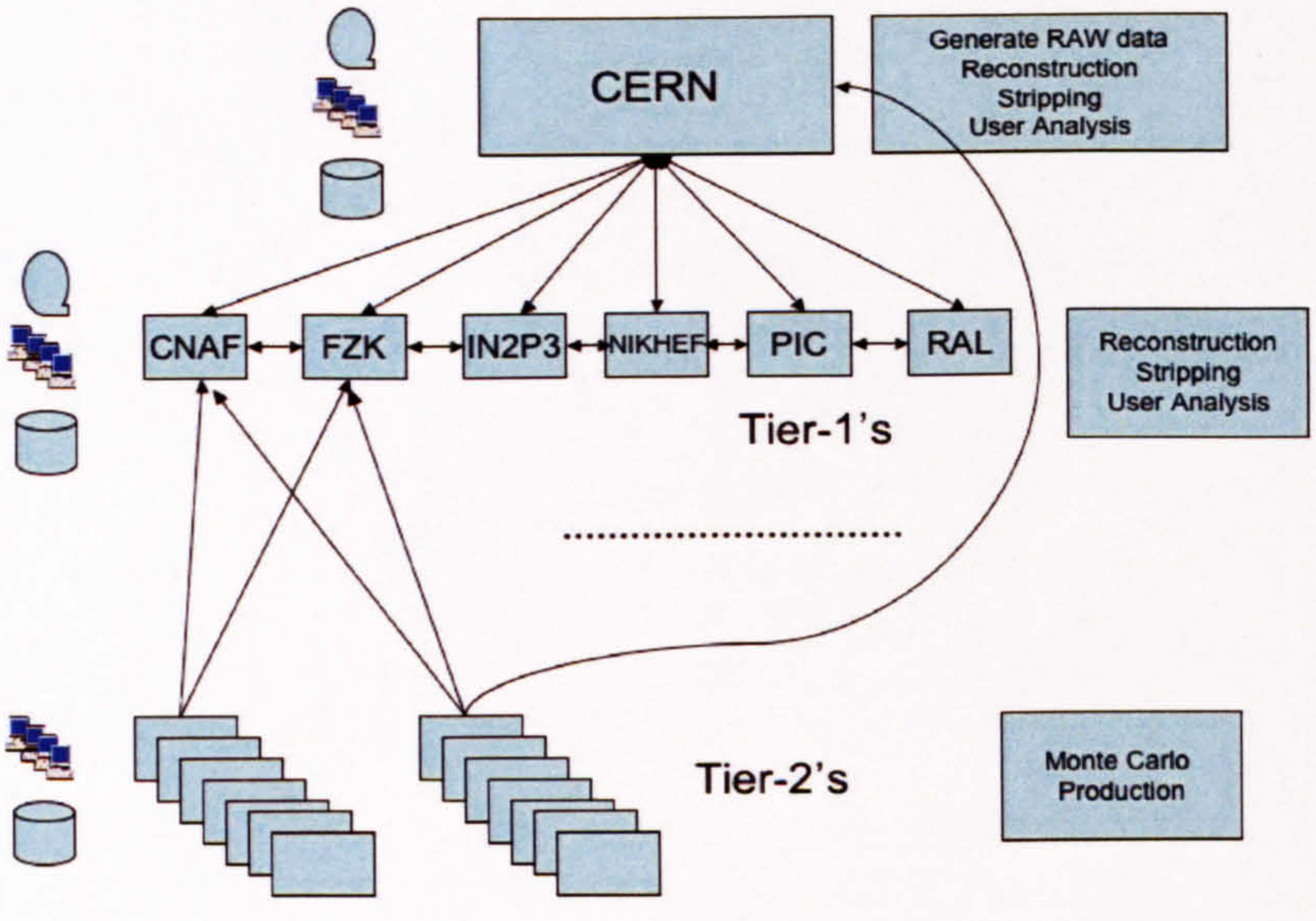

Figure 2.5: *The LHCb Computing Model highlighting the distributed, multi-tier regional centre model, from [100].*

Resource Requirements

LHCb will need to utilise the resources of the Tier-0, Tier-1 and Tier-2 centres in order to meet the required levels of CPU, disk storage and mass storage. There will be a slow ramp-up phase of the LHC during 2007, with 2008 being the first full year of data-taking. The projected CPU and storage requirements[1] for LHCb during 2008-2010 [100] are shown in Figures 2.6 and 2.7. This assumes a year of data taking to be 10^7 seconds at the luminosity of 2×10^{32} cm^{-2}s^{-1}.

As the experiment matures, the CPU requirement increases. It is also interesting to note in Figure 2.6 that whilst the requirements of the Stripping, Full Reconstruction and Monte Carlo activities are relatively stable, the Analysis activity shows a steady increase over the three year period.

[1] 1 kSI2k is approximately equivalent to one single core 3 GHz processor.

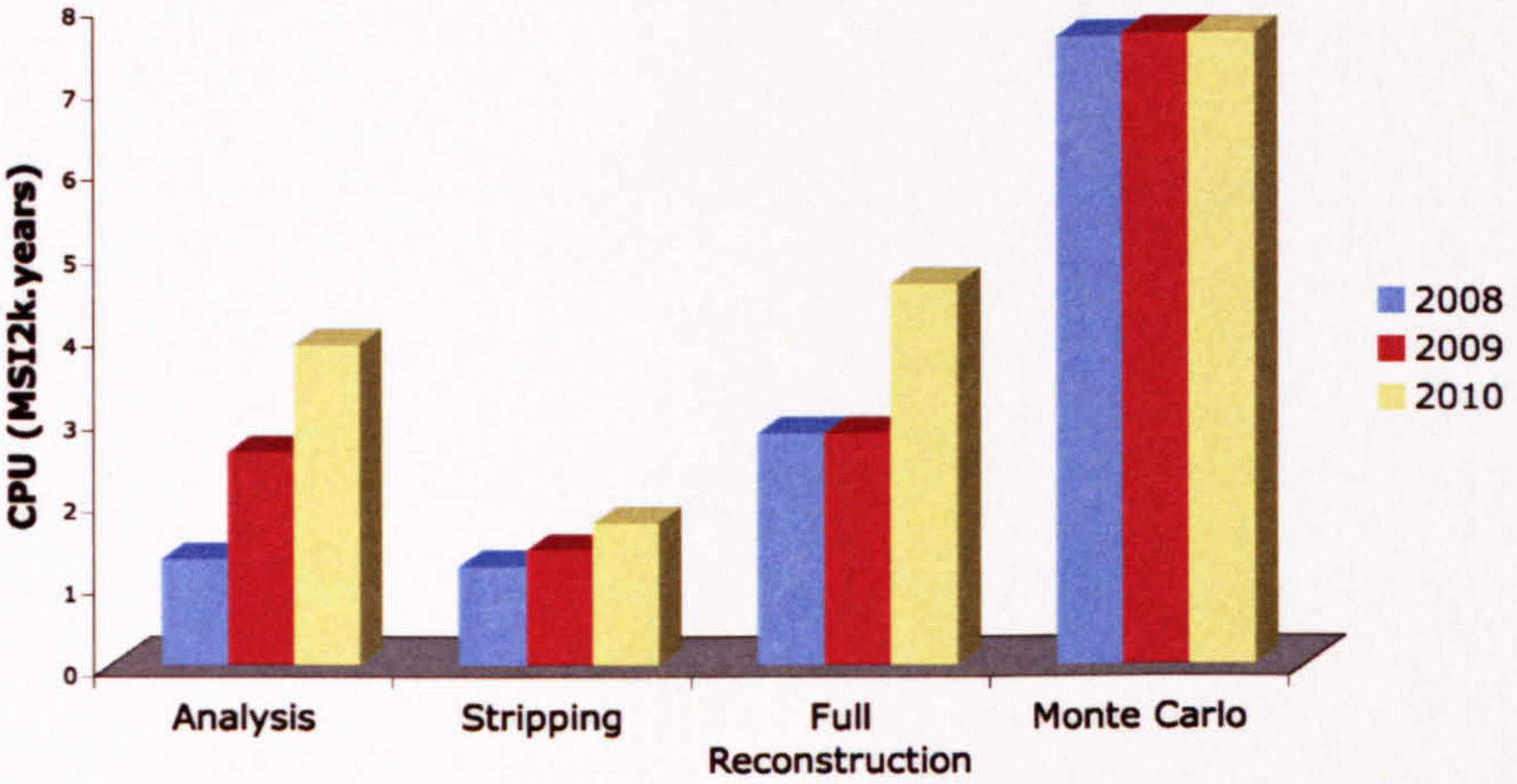

Figure 2.6: *LHCb CPU requirements breakdown by processing activity during 2008-2010. Adapted from [100].*

Therefore, it is essential that a suitably scalable system is in place to deal with these increases.

The LHCb experiment is expected to generate around a Petabyte of data per year. The Disk and Mass Storage System (MSS) requirements, shown in Figure 2.7, reflect this and also the large amount of simulated data required to be stored each year. LHCb must integrate all of the available resources to accomplish the necessary computing tasks. This means everything from individual PCs to computing clusters and the LHC computing Grid.

2.4 DIRAC as a Production System

Distributed Infrastructure with Remote Agent Control (DIRAC) was originally created to provide LHCb with a set of tools for managing production jobs for simulation and reconstruction.

During the Data Challlenge in 2004 (DC04), DIRAC was used to generate

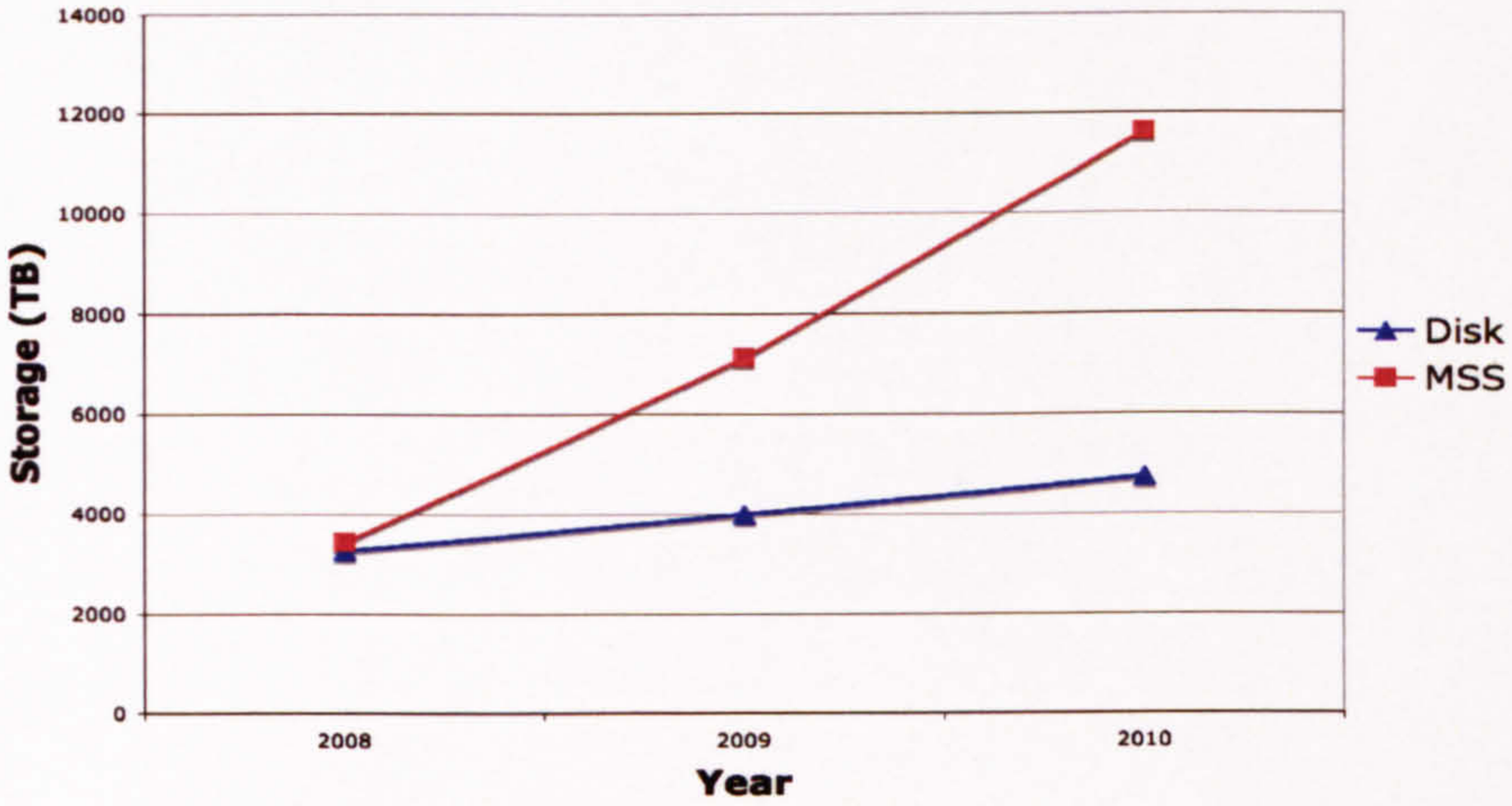

Figure 2.7: *LHCb Disk and MSS requirements from 2008 to 2010. Adapted from [100].*

187 Million events constituting 62 TB of data, which was stored in 5 Tier-1 centres [108]. Subsequent use of DIRAC led to a peak value of 5,500 simultaneous production jobs comfortably running on LCG resources without nearing the limits of the system itself.

Due to the successes of DC04, it was decided that DIRAC would be used as a submission tool to the Grid also for analysis jobs. This decision was based on the stability of the system as well as the efficiency which it delivered for Grid jobs. The DIRAC system and how it was extended for distributed user analysis is the main topic of this thesis and will be described in Chapters 4, 5 and 6.

2.5 Software Distribution in LHCb

Even within the LHCb experiment, several different platforms are in use. There is no guarantee that each university in the collaboration has the same

operating system or hardware. With the advent of the Grid and an abundance of distributed systems available, platform independent installation procedures are a vital element of a working system.

The software distribution mechanism for LHCb must contend with the many different compute systems present on the Grid whilst also minimizing the level of manpower required to maintain the service. Rather than solving the problem once for each type of platform, the approach taken was to find a generic solution for all platforms on LCG. This should also accommodate changes that can occur to hardware and operating systems, without necessarily having to update the software distribution mechanism.

When new releases of LHCb software occur, it is also vital to minimise the human intervention necessary to support them. Ideally, the software should be immediately available to be used on the Grid after a release, in an automated manner.

The installation of software can either be performed from source, or a binary based distribution. If binaries are available, they are normally the optimal choice since no compilation is required. When binaries are not available however, installation from source can be necessary.

This section presents an overview of software distribution on the Grid, with emphasis on LHCb. Software distribution assumptions are discussed in Section 2.5.1 and the Virtual Machine paradigm is described in Section 2.5.2. The work performed in evaluating Pacman for use in LHCb is mentioned in Section 2.5.3 before discussing the final implementation in Section 2.5.4.

2.5.1 Software Distribution Assumptions

Several assumptions can be made for the LHCb software distribution mechanism running on LCG. Firstly, it is assumed that there is a flavour of

Unix/Linux running on the compute resource. Software being developed on one platform is not by any means guaranteed to work on the multitude of different systems across the world. The platform which is running at CERN (currently Scientific Linux 3 with a GCC 3.2.3 compiler) is considered to be the standard. Furthermore, it is assumed that any user DLLs will be compiled only for this platform at this stage. However, other platforms may be supported in the future.

For running on an LCG Worker Node (WN) it is assumed that outbound connectivity exists in order for the software to be installed, e.g. via HyperText Transfer Protocol (HTTP). To reduce overheads, the situation should be avoided where large files, containing the software, are packaged with each job.

Another factor is whether the computing resource is running with a 32 or 64 bit architecture. Since DIRAC is developed in Python, it has been demonstrated to run successfully on both systems.

2.5.2 Virtual Machine (Paratrooper) Concept

The Virtual Machine concept is perhaps the cleanest solution to deal with compatibility issues on the Grid. Instead of running applications on the native system of a computing resource, this involves running one or more instances of an operating system on the same CPU to create the illusion of many smaller Virtual Machines. For the Grid, the beauty of this approach lies in the fact that users could simply choose their platform when submitting a particular job and always be guaranteed that everthing would work as expected. One example of this paradigm in practice is the Xen [109] project.

The idea of making heterogeneous compute resources homogenous has led to a quasi-Virtual Machine paradigm or 'Paratrooper' approach being used

with DIRAC. This involves shipping compiler libraries along with a self-consistent set of binaries which do not require any special environment. This allows DIRAC to treat many flavours of Unix/Linux systems in a uniform way when invoking software applications. The DIRAC approach works well for LHCb because a flavour of Unix/Linux can be guaranteed on LCG. The only problems which occasionally arise are due to missing or conflicting libraries on some exotic platforms. Overall, this ensures that the LHCb Grid jobs are fully equipped to 'land' on a computing resource and automatically deploy necessary software, like a paratrooper.

2.5.3 Automating LHCb Software Distribution Using Pacman

This section describes the use of Pacman [110] to perform an automated installation of the LHCb software from source [111]. This not only lends confidence to the functionality and reliability of Pacman but provides at least a starting point for those wanting to install from source, *e.g.* on non-supported platforms.

Pacman: A Package Manager

Pacman is a package manager developed by Saul Youssef [110]. It has been programmed in Python, developed on Cygwin, and hence is very portable. There are many advantages to using Pacman for software installation and these are discussed below. A collection of 'tarballs' is kept in a web-visible cache, each tarball contains the files needed for a particular package and each one has a Pacman file associated with it. All of the necessary installation instructions are kept in the Pacman files inside the cache, which is ideally maintained by experts.

Installing Pacman is simply a matter of unpacking a tarball. After a couple of trivial steps the user may then install any of the software packages in the cache. Any dependencies are automatically recognised, resolved and installed for the user and what once was a time consuming operation can be reduced to executing one command. Through the Pacman approach, package installation is configured once by an expert and their knowledge is passed to those who need to perform the installation in a transparent way.

The usual information needed to install and maintain a software package can be summarised as follows:

- Location of software *e.g.* a URL
- Correct release for desired platform, also whether updates or patches are required
- Dependency on other packages
  - Whether required dependents are already installed
- If root access is required
- Exact installation commands for the package
- Any environment variables and paths that must be setup.

All of these issues are dealt with automatically by Pacman. Ideally the end user should never need to execute more than one Pacman command for any software installation. Pacman is a robust package, any problems are easily diagnosed from meaningful error messages. Errors will only affect the package it is installing at the time and any installation progress is saved so that the user can restart on the package where they left off.

Why Pacman?

Pacman is a fully functional software tool, capable of performing the full LHCb package installation from source. While alternatives exist they are generally more limited in scope.

The RPM package manager [112], for example, requires a user to be root in order to install packages, which severely limits the effectiveness of this approach. Another popular package manager is Relink [113]. You do not have to be root to use this and dependencies can be tracked. However, Relink is aimed at system administrators. As such, there is a much more lengthly installation procedure and the user must still go through the installation by hand. Relink is useful for performing an installation once then transferring this information to many other machines but lacks flexibility and ease of use.

While a number of other alternatives to Pacman exist, none of those considered by the author have the same functionality, robustness and ease of use. The installation procedure is trivial and the responsibility for successful package management firmly shifts from the end user to the managers of the Pacman cache. This is ideal for an international collaboration such as LHCb where physicists must currently dedicate time to installing a rapidly evolving collection of software packages.

Advantages Presented by Pacman

Pacman is very simple to install and can even manage itself as a package. This means any updates for Pacman can be performed automatically. Backward compatibility between versions of Pacman is assured and it is very portable. The user does not have to be root and recursive dependencies are automatically dealt with. If an installation fails, only the current package is affected and Pacman stores any progress made. With clearly defined cache

managers, there is one point of contact when things go wrong and the user also has the benefit of an automatically generated index page [111].

Having a regularly maintained Pacman cache for all of the LHCb software and dependents is obviously advantageous for the simple reason that dependencies would very easily be tracked. At present, there seems to be no easy method to do this, especially for institutes outside of CERN.

Summary of the Progress Made

A full installation of the LHCb software using Pacman was performed at Liverpool and ScotGrid at Glasgow. From start to finish the full compilation from source took around 23 hours on both systems.

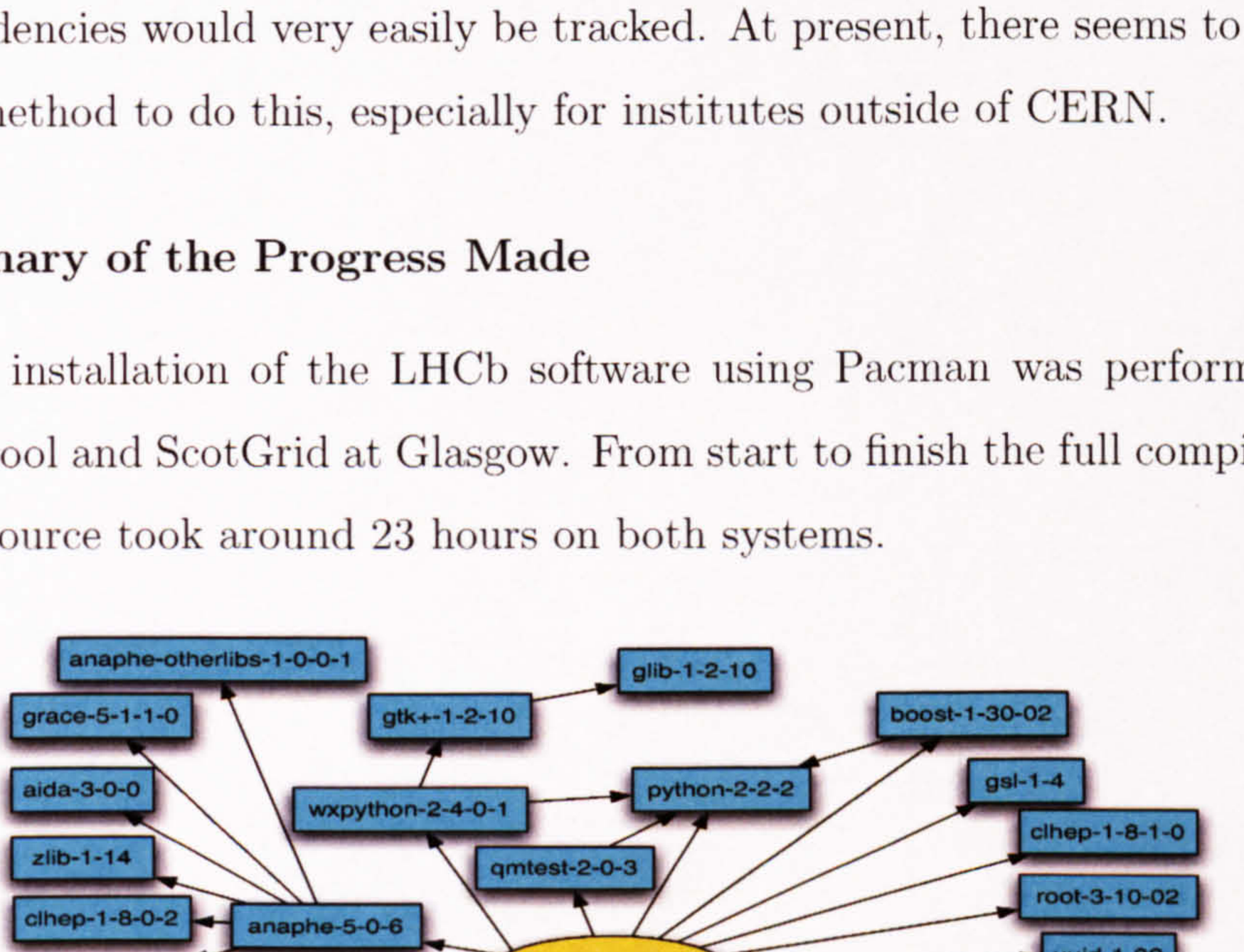

Figure 2.8: *Dependency tree diagram for all Pacman automated packages up to LCG tools.*

The LHCb software rests on the Gaudi framework which in turn, sits on

top of the LCG tools. Tracking dependencies can be challenging and it was found that some of the sixty or so packages which were automated using Pacman, while present in the requirements files, are in fact not necessary for the LHCb software. Figure 2.8 shows the full list of packages automated, up to the level of the LCG tools, and the perceived dependencies between them as taken from the appropriate requirements files. This is inaccurate, but highlights the fact that without appropriate documentation, mistakes can be made.

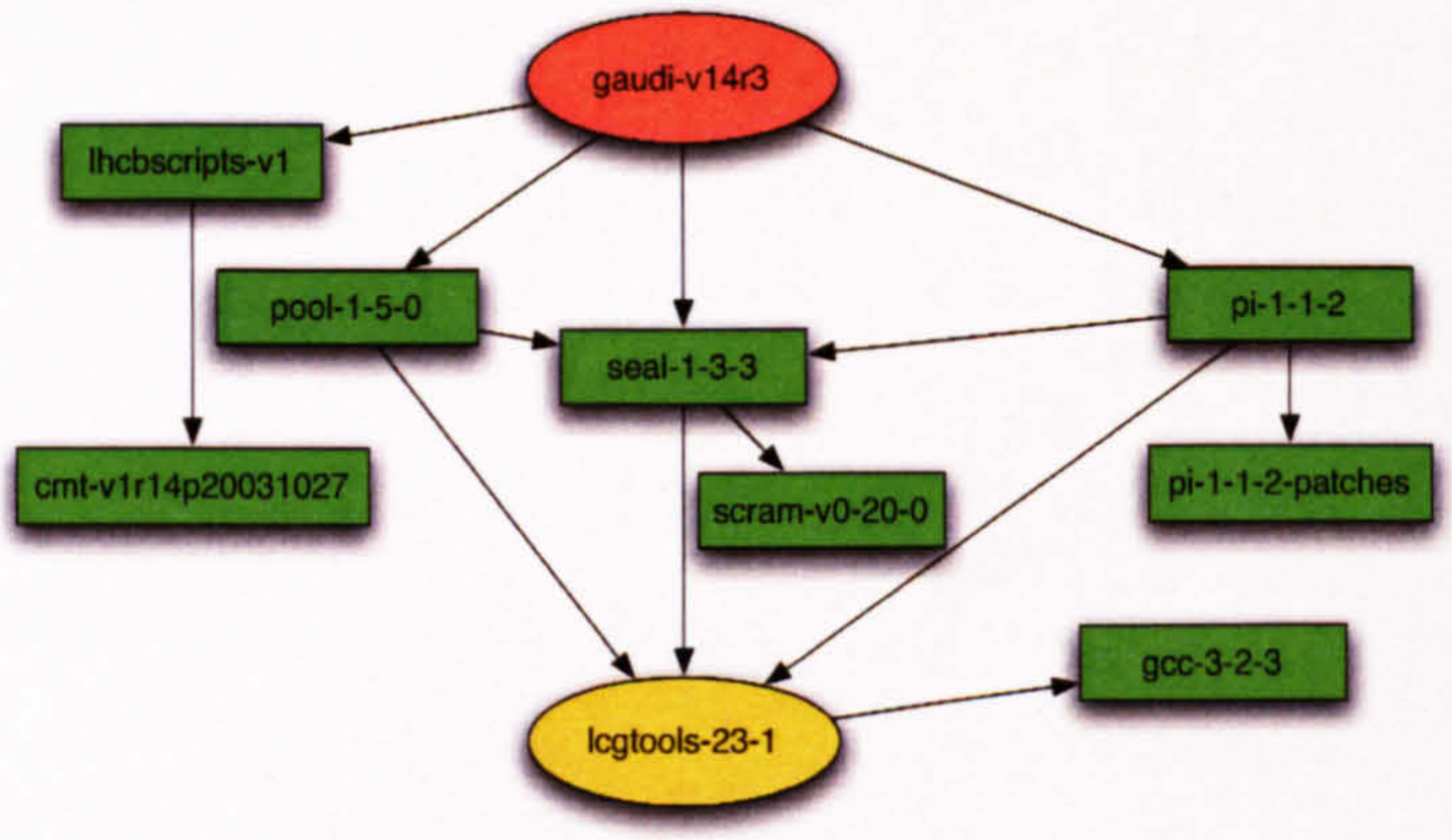

Figure 2.9: *Dependency tree diagram for all Pacman automated packages up to Gaudi.*

The next level of packages includes all those up to Gaudi, see Figure 2.9, and this is folllowed by the LHCb core software along with DaVinci, Boole, Brunel, Gauss and their dependents, in Figure 2.10. The top level package is lhcb-software which allows a full installation to be requested. This can be updated so that when other complete versions of the software exists, the head version is always returned.

Pacman is a versatile package manager, which is continuously evolving to accommodate new features. Installation of the LHCb software which once

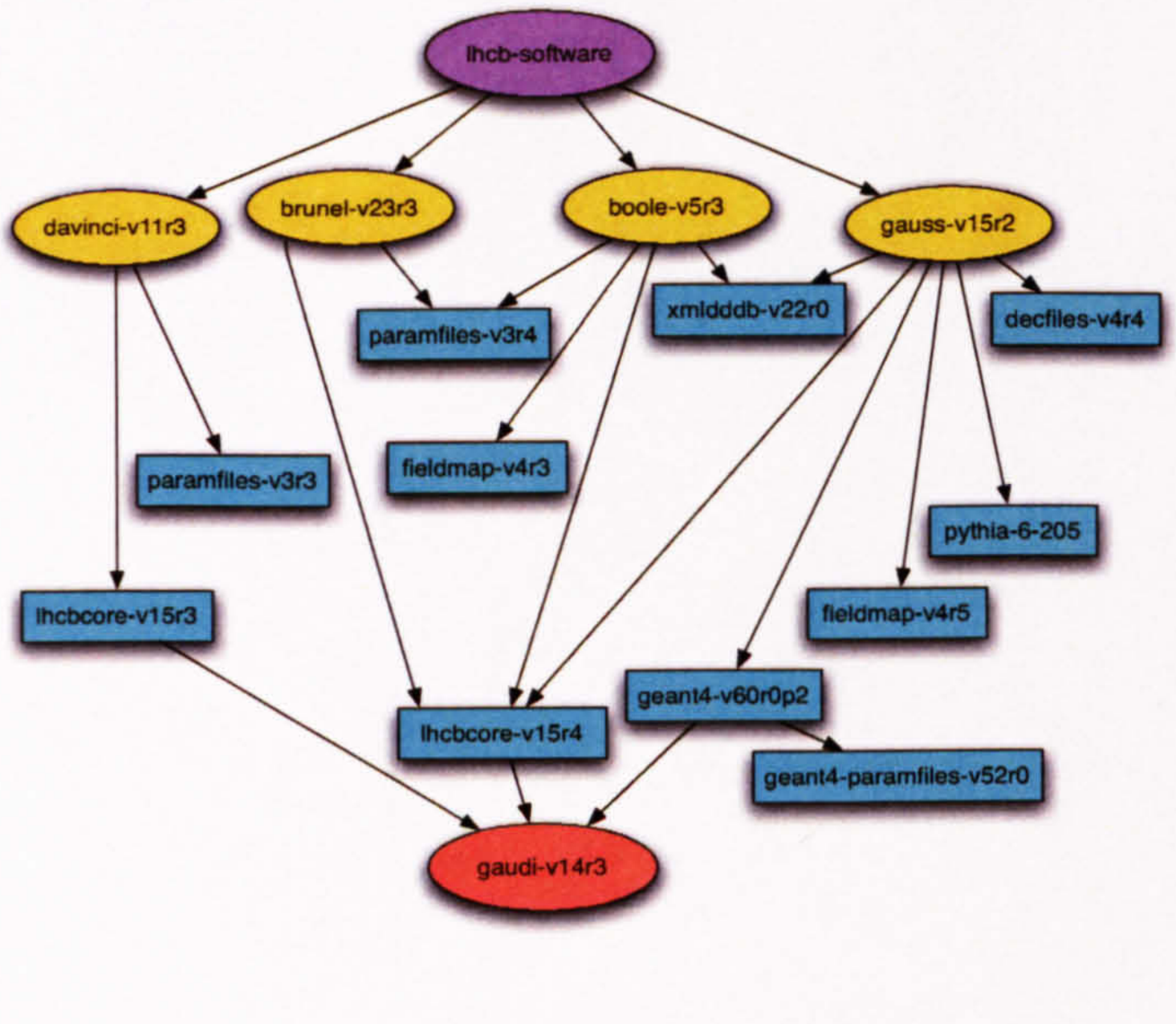

Figure 2.10: *Dependency tree diagram for all Pacman automated packages from Gaudi up to the data processing applications. Note that the explicit dependency of DaVinci, Boole, Brunel and Gauss on Gaudi has been removed for this figure.*

was a very complicated procedure can be reduced to executing a few simple commands.

Use of Pacman in LHCb

Pacman has been shown to be very capable of handling fully automated installations and could easily be used for automating binary installations. Unfortunately, there were some problems with using Pacman for Windows DOS-based installations and so the mechanism described in the next section was preferred.

LHCb software installation using a new release of Pacman is being maintained by a colleague at Glasgow as a service for the LHCb community [114]. A recent LHCb software training course [115] in Cambridge utilised this mechanism to temporarily install simulation software, from source, on sev-

eral PCs.

2.5.4 Final Implementation: install_project.py

As mentioned in the last section, Pacman was not officially adopted by LHCb as a software distribution mechanism. Instead, the preferred solution was the LHCb software distribution tool [116], `install_project.py`.

The first step in extending DIRAC for distributed user analysis was to create a reliable and resilient software distribution mechanism for jobs on the Grid. The LHCb software distribution tool [116], `install_project.py`, was integrated into DIRAC to facilitate this. This exports the CMT [117] structure based at CERN and relies on CMT for application setup and execution. Via DIRAC, `install_project.py` realises the Virtual Machine (Paratrooper) concept in which applications can be invoked in an operating system independent way and significantly reduces the manpower requirements of LHCb.

When executing within DIRAC, software installation is completely transparent for the user. From a user perspective, only the name and version of an application needs to be specified in the job description. All software available in the LHCb Release Area at CERN, is now also available to DIRAC through this software distribution mechanism. This also means that DIRAC can immediately utilise the most recent software without any human intervention.

Since `install_project.py` is developed in Python, it is easily compatible with DIRAC, and the latest version can be retrieved as necessary at the level of a job on the Grid. This is an improvement on the previous mechanism to distribute LHCb software for use by DIRAC, since this required the manual construction of each new software release. The software packaging at CERN can now be relied upon without any intervention from a DIRAC point of

view, and little or no maintenance is required when new releases are made.

2.6 Summary

The LHCb experiment has been briefly described, with special emphasis on the software and computing model. The main LHCb data processing applications: Gauss; Boole; Brunel; and DaVinci, are all based on the Gaudi framework. The Gaudi framework has been created to provide the necessary infrastructure in a way that shields the physics code from the actual implementation technologies.

The tiered architecture of the LHCb Computing Model serves to provide all of the distributed computing needs of the experiment. DIRAC is used to integrate available resources in a consistent way and the extension of the system for the distributed data analysis tasks will be described in later chapters.

Pacman is a versatile package manager that is continuously evolving to provide new features. Installation of the LHCb software, which once was a very complicated procedure, can be reduced to executing a few simple lines. With platform specific commands and complicated interdependencies, the fact that Pacman is capable of managing such a complicated installation as the LHCb software from source is very encouraging.

In the end, the LHCb software distribution tool, `install_project.py`, was chosen to be integrated into DIRAC. This realises the Virtual Machine (Paratrooper) paradigm through DIRAC and allows new releases of LHCb software to become immediately available for use on the Grid.

Chapter 3

Data Analysis in a Distributed Environment

There are many challenges in performing distributed analysis on the Grid. One of the most important is how to deal with geographically distributed, heterogeneous resources in a consistent way. With each site on the Grid potentially having different access policies, different operating systems and different hardware, it is imperative to adopt a system that can deal with these, sometimes subtle, differences in a uniform manner. It is paramount that physics analyses in LHCb should be able to be carried out using the Grid; furthermore, use of the Grid should be transparent for users.

A key issue is how to provide reliable access to the required input data for each job, via the available access protocols. This and other job requirements, such as particular LHCb software versions, can vary on a job by job basis, so the infrastructure must be in place to contend with this. Since physicists will be configuring and submitting their own jobs, there is no obvious way to predict this type of workload. It can therefore be considered chaotic in nature. Nevertheless, analysis jobs are normally of the highest priority with

respect to other computing activities in LHCb and are essential for users to produce physics results and publications.

This chapter will describe the key paradigms for LHCb distributed analysis in Section 3.1, as well as the analysis requirements in Section 3.2. The concept of using an Overlayed Network to aggregate disparate resources will be introduced in Section 3.3. The approaches of the other main LHC experiments to the distributed analysis activity will be discussed in Section 3.4. The first attempt at realistic physics analysis using the gLite Grid framework prototype will be described in Section 3.5, and some of the reasons for finally choosing to extend DIRAC for the LHCb distributed data analysis activity will be highlighted in Section 3.6.

3.1 Paradigms for Distributed Analysis

Some of the key mechanisms which result in high efficiency on the Grid as well as other resources available to LHCb are discussed below. The first is the *PULL* scheduling paradigm, which ensures that jobs are only sent to computing resources after the execution environment has been checked. This is based on an idea first presented by the Condor Project [72], whereby resources are utilised immediately when they become available. This is contrary to the *PUSH* scheduling paradigm, which involves central optimisations, based on global information about the system, to match jobs to resources.

The second is the Pilot Agent Paradigm, which involves sending an Agent instead of a job to a computing resource. This means that failures can occur to the Agent without affecting the job. In practice, the use of these paradigms ensures that jobs are no longer sent to a computing resource with a decision based on possibly incomplete, static information. Jobs are instead requested

by an Agent on a computing resource, in a reliable and efficient way.

3.1.1 Push versus Pull

Job scheduling can be thought of as the process of assigning a particular resource to a particular job. The two main approaches to job scheduling can be referred to as *PULL* and *PUSH*. Whether referring to the Grid or a batch system, similar components are involved. In general, we can consider resources to be a heterogeneous set of clusters that belong to a local area network (LAN).

Each cluster may have its own access policy and could place stringent limits on the amount of resources to provide to each user or Virtual Organisation (VO). To schedule jobs in this context, decisions are usually made with an overall, global picture of the system, for example, the situation where one site

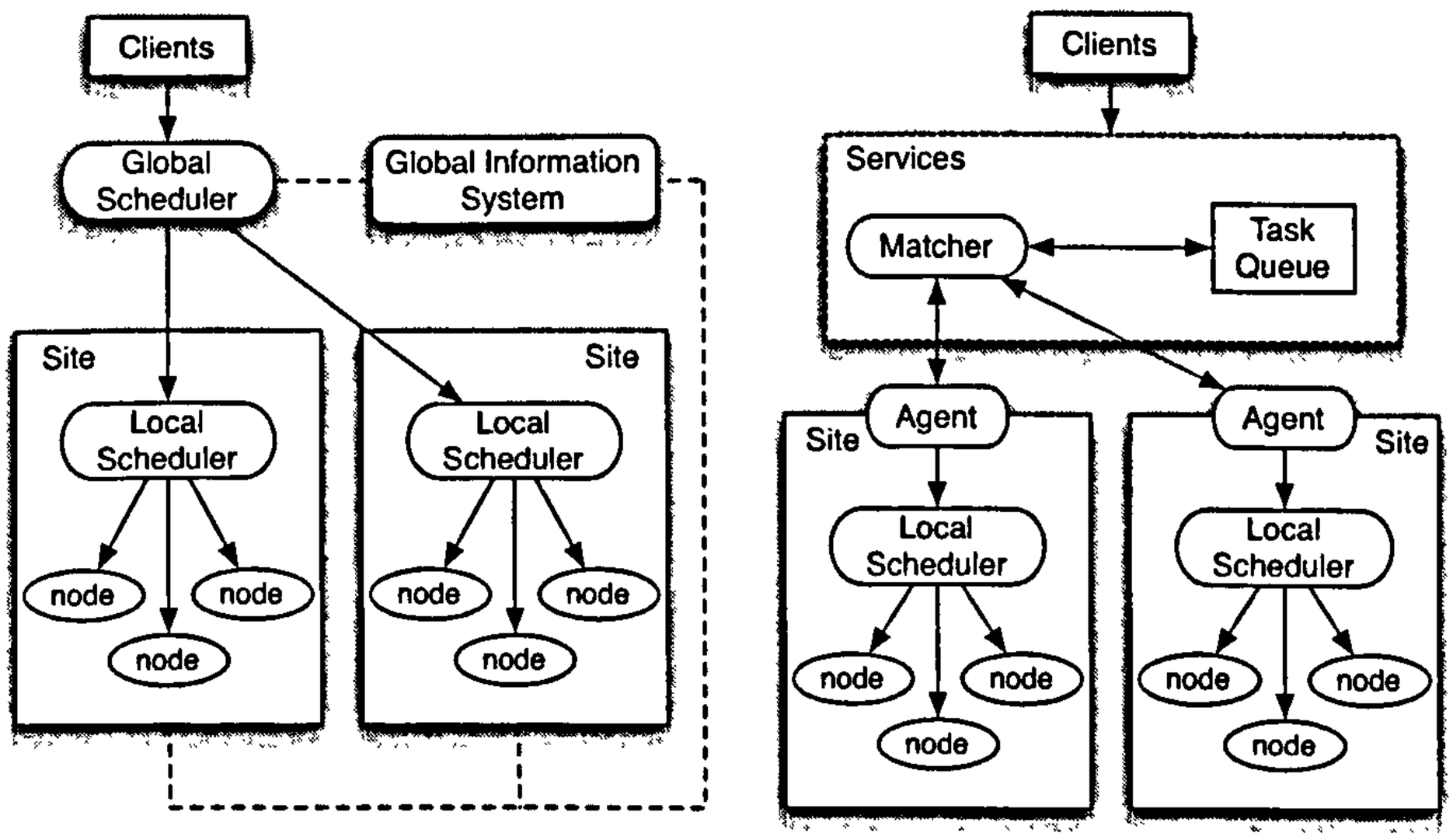

Figure 3.1: *Illustration of the PUSH model in (a) and the PULL model in (b). Solid lines reflect the task flow.*

is saturated and further jobs are sent there should be avoided. This would clearly compromise performance. Another reason for this is the need to control individual sites. If for whatever reason, one site becomes unavailable, it is necessary to prevent jobs being sent there.

Figure 3.1 (a) illustrates the *PUSH* scheduling paradigm. In this approach, clients submit jobs to a Global Scheduler that makes a decision about which site to send, or schedule, the job. This decision is based on information from a Global Information System that continuously monitors all resources and reports on their current state. At any one time, in a global system with shared resources, the availability of these resources can fluctuate considerably. In the context of the Grid, site resources can be shared amongst many independent VOs but also local users, which can have a higher priority. Since the monitoring information is gathered centrally, it can be considered as static information about a dynamic system, often being out of date as soon as it is sent.

The problems associated with a *PUSH* based architecture are mainly due to the incomplete picture of the system and the stability of the information system. A Global Scheduler, such as a Resource Broker (RB) on LCG, must resolve many complicated parameters to determine the best location for a particular job. These parameters are used to build up a picture of the state of many resources and can lead to complicated scheduling calculations. This becomes even more difficult when trying to implement prioritisation of jobs and quotas because this would place an additional load on the Global Scheduler. There is also the question of system stability, if the Global Information System in the *PUSH* model were to fail, this would cause the whole system to fail.

The *PULL* scheduling paradigm solves many of the problems associated

with the *PUSH* system by design, and is shown in Figure 3.1 (b). In this model, clients interact via services, which provide specific tasks associated with the management of jobs. Not all services are depicted in Figure 3.1 (b), which is just illustrating the concept. By placing agents close to resources, jobs can be requested from a service and delivered to a free resource whenever it is available. Agents only request jobs when the resources are free so there is no need for complicated scheduling algorithms to be performed. Another advantage of this approach is that jobs can be stored in a central task queue before being delivered to resources, allowing prioritisation policies to be applied. The *PULL* approach allows access to all Grid resources in a similar way to a batch system with a single task queue. Since the problem of handling priorities and fair shares in a batch system has been solved for a long time, this experience can be applied in a new context. The use of remote Agents to determine the location of available resources is a considerable improvement since they always have an up to date view of the sites to which they are deployed.

A simulated study of *PULL* versus *PUSH* was performed in [118] where DIRAC, which realises the *PULL* scheduling paradigm, was compared to a centralised scheduling approach. The results showed that for an ideal system, there is a slight improvement on job scheduling via the *PUSH* model. Unfortunately in practice, the 'ideal' system is often unrealistic and it was found that the system cannot adapt to common failures such as: network problems; unavailability of services or power cuts. This is further discussed in Section 5.5. Keeping a global view of a system that is continuously in flux becomes more and more problematic and, with a dependence on a global information system, the *PUSH* approach often does not scale well. On the other hand, the *PULL* approach adapts well to changes in the system and

does not depend on any centrally collected information about resources by design.

3.1.2 Pilot Agent Paradigm

The Pilot Agent paradigm works in a complementary fashion to the *PULL* scheduling mechanism. Consider Figure 3.1 (a) which highlights the *PUSH* approach. Here, jobs are sent and scheduled to a particular resource based on the static information from the global information system.

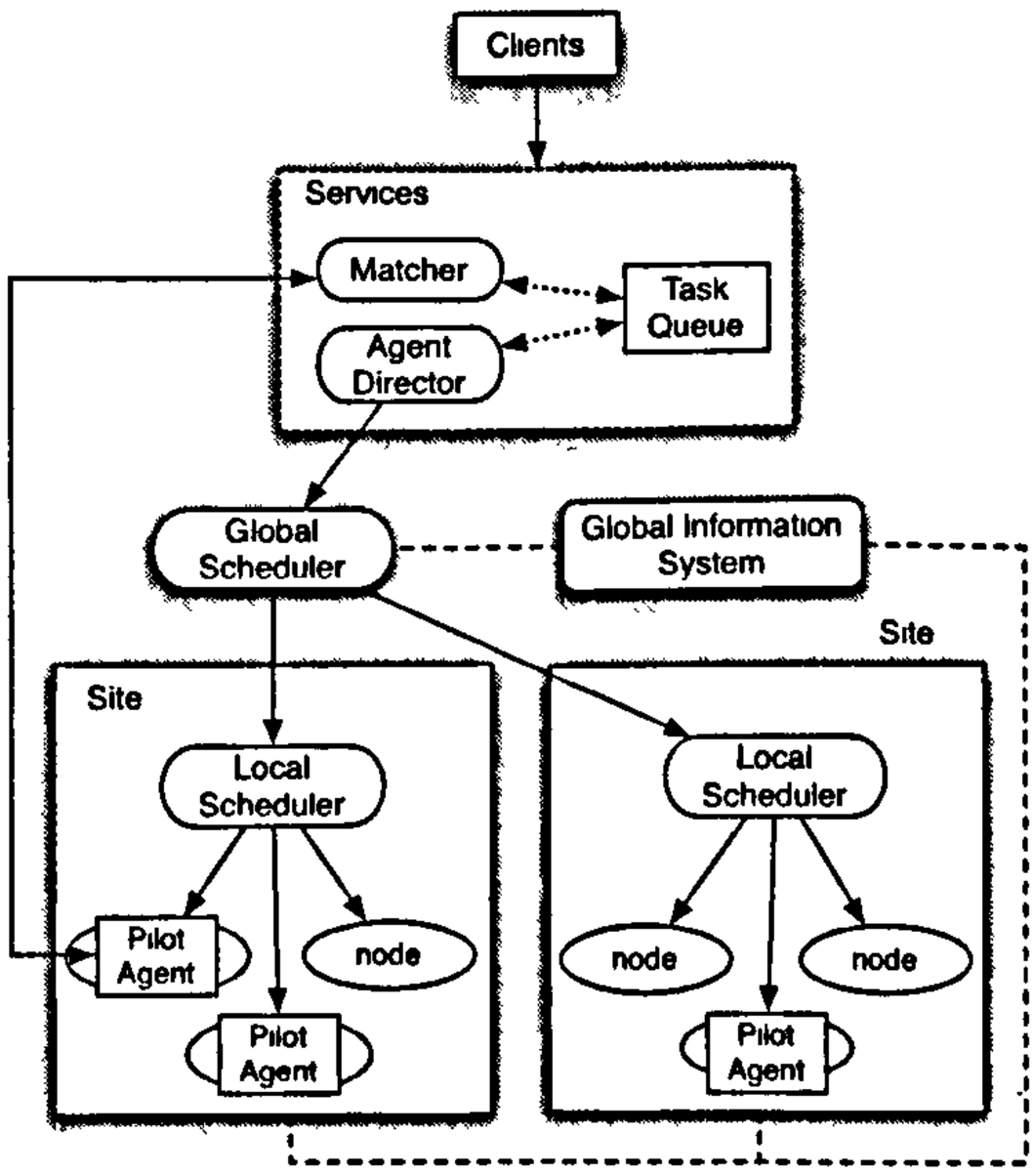

Figure 3.2: *Illustration of the PUSH model with the use of the Pilot Agent paradigm. This effectively transforms a PUSH scheduling system into a PULL scheduling system. With a central task queue, the implementation of policies and quotas becomes possible.*

Instead of submitting jobs to the Global Scheduler, it is possible to submit Agents with exactly the same requirements. The typical requirements of a job could include a specified CPU time or particular input datasets of interest.

Agents are sent with these requirements as well as any policies that concern the entire Virtual Organisation. One example of this could be information about particular sites that should be banned. In this way, the Agent can be scheduled to a particular computing resource whilst holding the job in a central task queue. If necessary, multiple Agents may be sent for the same job in case of failures.

LHCb has access to one computing Grid, LCG, which operates using the *PUSH* paradigm. Figure 3.2 illustrates how the use of Pilot Agents can transform a *PUSH* scheduling system into a *PULL* scheduling system. Through the use of Pilot Agents on LCG, there is no explicit dependence of the services on the Global Information system. Since the Pilot Agents pass through the standard job scheduling mechanism of LCG it means there is an extra degree of safety in their arrival at a particular Worker Node without assuming it will be guaranteed to run successfully. In effect, this is a zero-trust approach that, whilst ensuring a high efficiency, carries some overheads. For instance, it is possible that Pilot Agents can be delayed and will not pick up any jobs when they eventually start. This could create an unnecessary load on the LCG Resource Brokers and is a potential drawback that will be further explored in Chapter 5. Also, the time taken for the Matcher service to assign a job to an Agent has to be taken into account.

Results from using DIRAC, which realises the *PULL* scheduling paradigm, can be used to clarify the last point. Figure 3.3 from [119] shows results for the DIRAC Matching times from the LHCb Data Challenge in 2004 (DC04). The average matching time for Monte-Carlo Production jobs using this approach was 0.42 seconds over almost 60,000 jobs. This is an encouraging result because the production jobs can typically run for 24 hours on a computing resource and the matching takes a negligible amount of time in com-

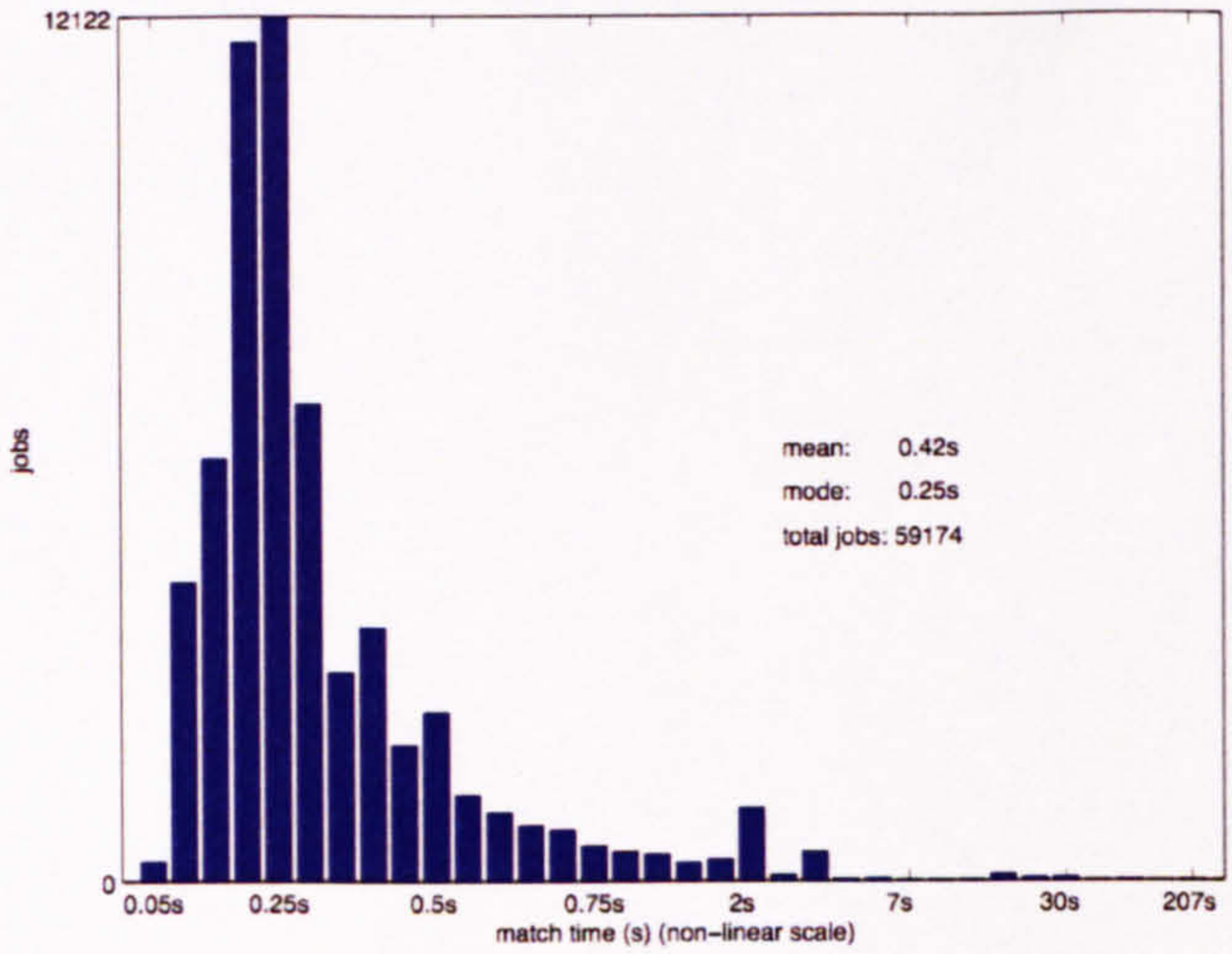

Figure 3.3: *Matching times during LHCb DC04 activity for Monte Carlo Production jobs running on LCG, from [119].*

parison. Furthermore, it is worth noting that many thousands of jobs were queued and thousands of jobs were also running concurrently during DC04. This lends confidence to the system and also to the *PULL* approach. At this stage, however, it still remains to be proved that the same method extends to distributed analysis jobs, with more demanding requirements and chaotic usage patterns. This will be clarified in Chapter 6.

The need to have a central task queue for the implementation of priorities and quotas, along with the advantages posed by this approach for aggregating resources, led to the idea of a Workload Management System (WMS), which can be considered as a collection of services. Figure 3.4 demonstrates how the approach from Figure 3.2 can be generalised for use on LCG. Recent extensions to the DIRAC WMS, which facilitate this approach, are described in Chapter 4. Clients now submit jobs to the DIRAC WMS, which submits Pilot Agents on demand in the form of LCG jobs. Multiple Pilot Agents may be sent in case of failures.

When Pilot Agents arrive at a computing resource or Worker Node, they then proceed to request a job from the WMS. At this point, the job is delivered only after the execution environment has been checked. This ensures that the job has arrived at a reliable resource and has a very good chance to complete successfully.

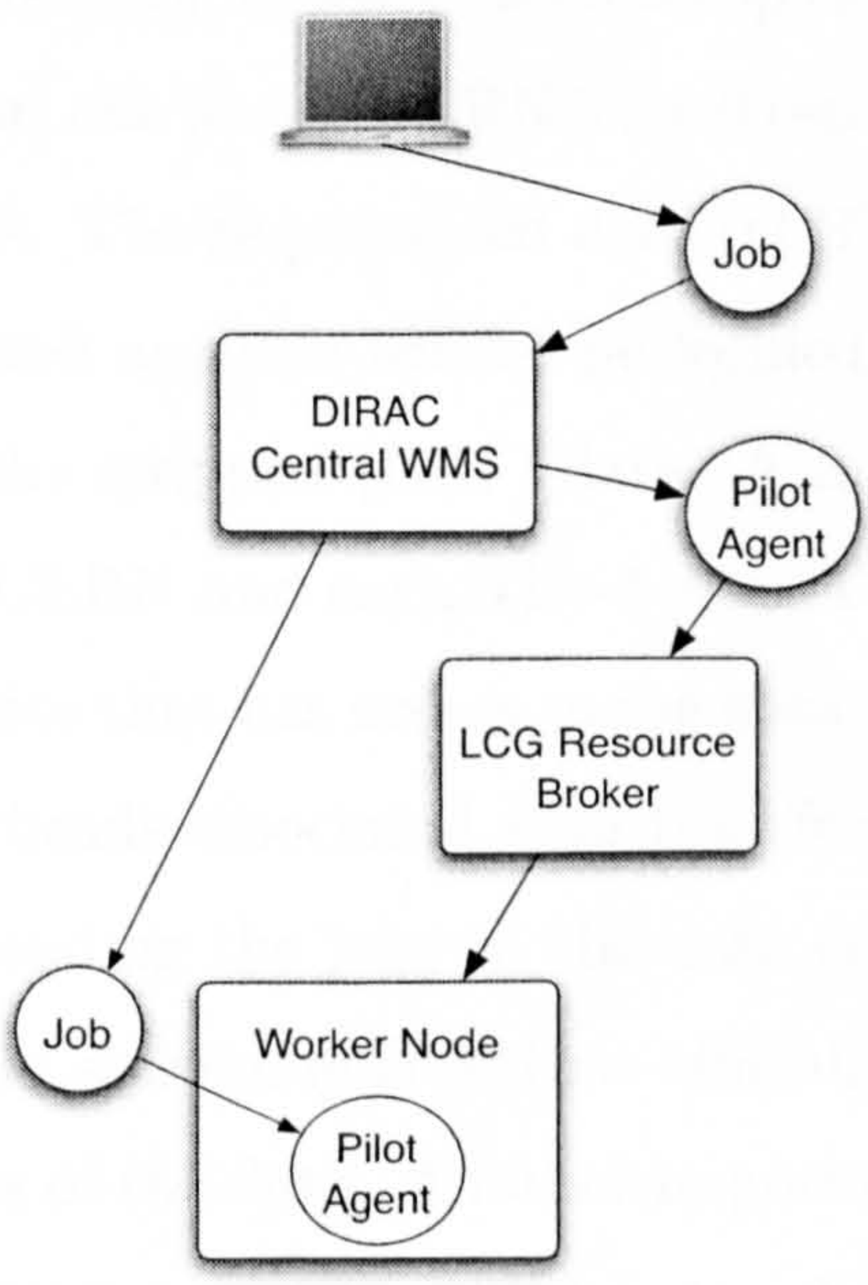

Figure 3.4: *Illustration of the DIRAC Pilot Agent paradigm in use on LCG. This is a generalisation of Figure 3.2 where clients interact with DIRAC Workload Management System services and Pilot Agents are submitted to the LCG Resource Broker.*

Further uses of Agent Control in LHCb will be discussed in the context of LCG in Section 3.3.2. Since Agents are being sent, this allows for further optimisations. See Chapter 5 for how this concept was utilised and extended for distributed analysis using the DIRAC infrastructure.

3.2 Requirements for LHCb Distributed Data Analysis

As mentioned in the introduction to this chapter, analysis jobs are chaotic in nature yet are often of the highest priority in the context of LHCb. In the LHCb Computing Model [100], at least two full copies of the RAW data from the detector will be kept, one at the CERN Tier-0 centre and one distributed amongst the Tier-1 sites. The re-processed data (rDST) will be stored in the same fashion. Distributed analysis will be performed mainly at CERN and the Tier-1 sites using the stripped (DST) data. A full copy of the stripped data will be stored at CERN and each Tier-1 site. Therefore, it is assumed that jobs are sent to a site that has access to the data it requires. This serves to reduce network overheads associated with transferring data for each job. The policy of always sending the jobs to the data ensures a certain degree of reliability on the Grid since only sites that officially provide resources for LHCb will have replicas of the data. Another important point is that all the DST output of the stripping activity will be stored and made available on disk at CERN as well as the Tier-1 sites [100]. This effectively eliminates problems associated with efficiently retrieving small amounts of data from Mass Storage Systems, something which they were not designed for. Ideally, some redundancy should be in place to account for situations where, *e.g.* not all data is available at one site. The infrastructure for distributed analysis should be able to deal with these kind of situations dynamically. Currently the system cannot transfer data automatically to satisfy the requirements of jobs, however, work is ongoing to facilitate this.

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 [120], but a prioritisation and optimisation of available resources for LHCb. The aim is to provide a stable platform for analysis on inherently unstable resources and therefore mask any inefficiencies of LCG and Grid hardware from the user.

Jobs running on the Worker Nodes need to access services in order to run successfully. The LCG File Catalogue (LFC) [77] must be contacted to obtain information about the local replicas of any required input datasets. It is most efficient to do this from the Worker Node since, in line with the paradigm of Grid computing, there is no advanced knowledge about which site the job will run at. Indeed, this transparency is something the infrastructure of Grid computing should provide. Access must also be possible to the LHCb Conditions Database to provide information about the current running conditions of the LHCb sub-systems, which may vary in time. The jobs must also be able to contact the central workload management services in order to provide, for example, monitoring information. It is therefore necessary that computing resources on the Grid provide outbound connectivity for LHCb jobs. This is still secure because the services being accessed are well defined and Agents autonomously request them. Inbound connectivity is not necessary however, since Agents do not provide services outside the site where they are located.

In the LHCb Computing model [100], it is assumed that 140 physicists will perform analysis, each submitting 2 jobs per week which will process $\sim 10^6$ events per job, increasing to $\sim 10^7$ events for larger samples. These jobs can be split into smaller 'chunks' in order to be run in parallel. This reduces the time in which results can be returned, but there are overheads in terms of gathering the output of each sub-job in a useful way. The splitting of larger jobs is something that places more of a burden on the Grid computing

infrastructure since it means more jobs must be scheduled, monitored and their outputs retrieved. A sufficiently scalable system should be put in place in order to contend with this demand and allow submission of these jobs in an efficient manner.

The output of analysis jobs can be an Ntuple-like object or 'private' stripped DSTs which will be analysed further on resources local to the physicist. The estimated storage requirements for analysis are $\sim$ 200 TB in 2008 [100], which is expected to grow linearly in the early years of data taking. Therefore, the infrastructure for distributed analysis should also be able to cope with efficient storage and retrieval of user output data.

3.3 Overlayed Network Concept

In order for all computing resources to be utilised to their fullest potential, disparate resources must become aggregated in some way. Figure 3.5 illustrates the typical resources available to a Virtual Organisation such as LHCb. These include: individual PCs; site clusters; and the Grid. Although LHCb is only able to access LCG, the Grid will be discussed in a general sense here.

The resources displayed in Figure 3.5 are generally composed of many different operating systems and hardware, but must be pooled together to form a consistent set of resources in a transparent way, from the perspective of the user. The question here is how to get these seemingly very different resources to work together in a seamless manner? A possible solution is through the use of Agents. Agents are intelligent pieces of code designed to work in line with the *PULL* scheduling paradigm to facilitate job submission and execution.

Through the use of Agents, it is possible to create an Overlay Network,

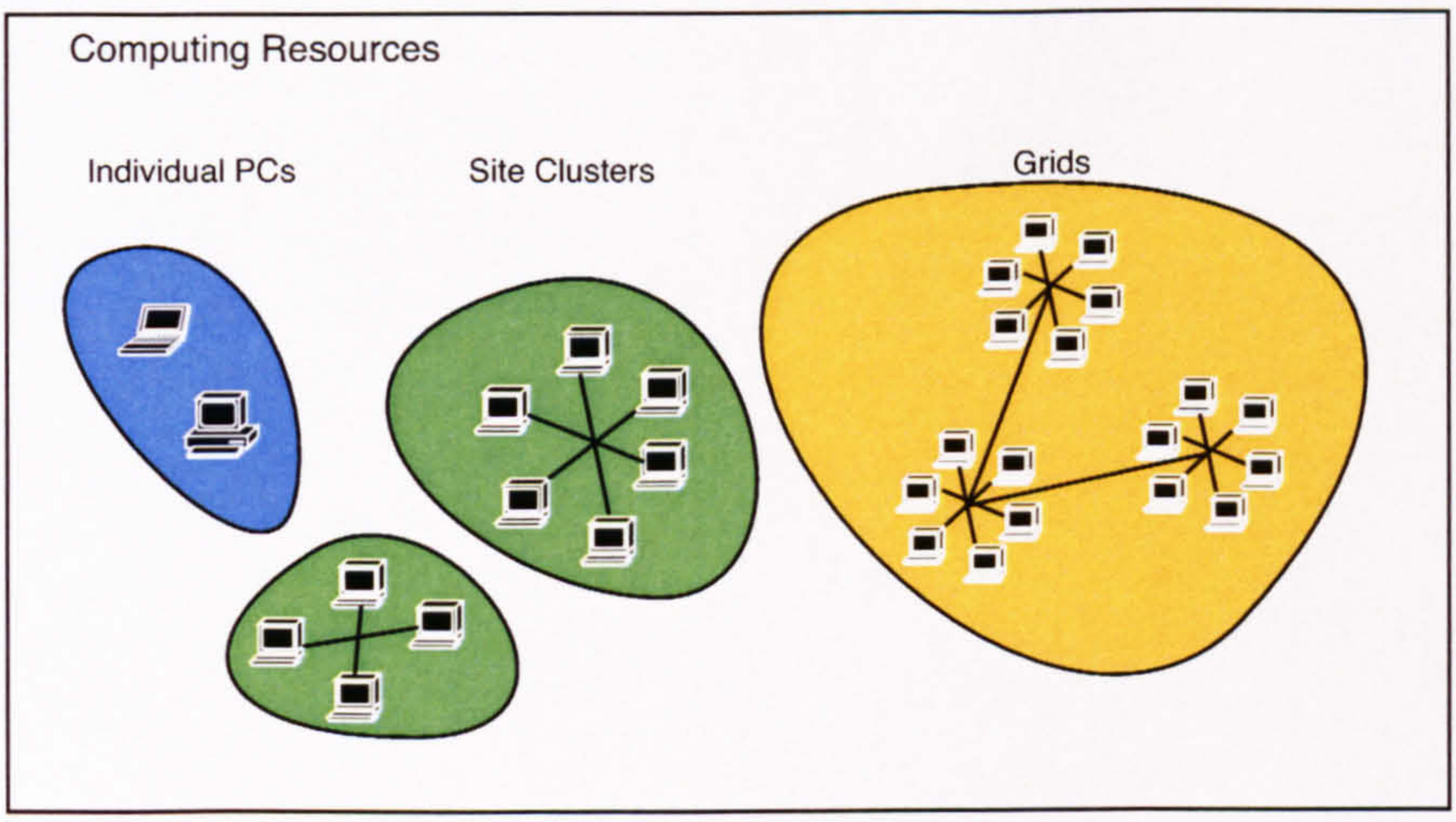

Figure 3.5: *Overview of the different kinds of computing resources available to LHCb.*

making inherently heterogeneous resources homogenous, so that any resource captured by a successfully deployed Agent is almost certain to be able to run the jobs of LHCb. An Overlay Network can be simply thought of as a layer on top of computing resources, which masks the complexity associated with pooling them together. Once established, this layer can then be used very efficiently for the computing needs it was created for, since the Agents autonomously take control of the resource they are sent to on behalf of the user. Agents interact via central Services which deal with all common tasks, including interaction with users.

3.3.1 Agents' Control as a Means of Implementing an Overlayed Network

Through the deployment of Agents close to the available resources, see Figure 3.5, a layer of Agents is formed. This is highlighted in Figure 3.6, where the Agents layer serves to mask the underlying diversity of the layer beneath.

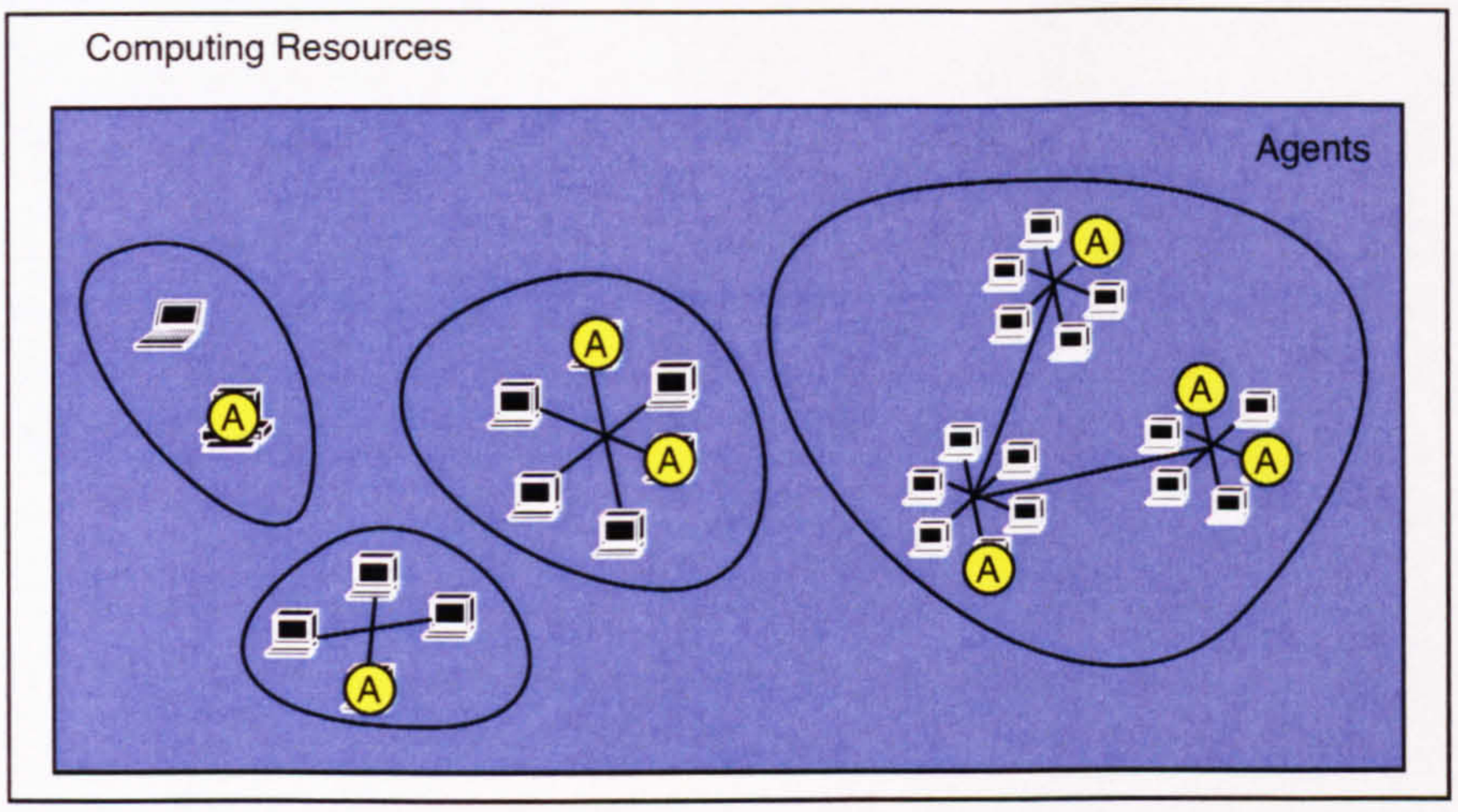

Figure 3.6: *Through opportunistic deployment to computing resources, an Overlay Network of Agents is formed. This overlay network masks the underlying diversity of the layer beneath.*

Although the Agents exist on different computing resources, being deployed through different means, they all become providers of resources in a similar manner and can interact with Services in the same way. For instance, on individual PCs, DIRAC Agents may be started 'by hand' or manually, as a script. On the Grid, Pilot Agents may be automatically sent to resources on behalf of a user, using the paradigms described in Section 3.1, and start a DIRAC Agent which acts autonomously. In both these cases, Agents start at a particular resource and, if possible, will request jobs in a similar way. This layer of Agents therefore masks the underlying diversity of the disparate resources beneath and presents a homogenous set of resources to the Services.

Figure 3.7 illustrates the use of Services to manage the activity of the Agents. This is the final layer in the creation of an Overlay Network. By this point, all Agents can be considered equal in the sense that they provide resources for a particular user or job. The resources may differ considerably

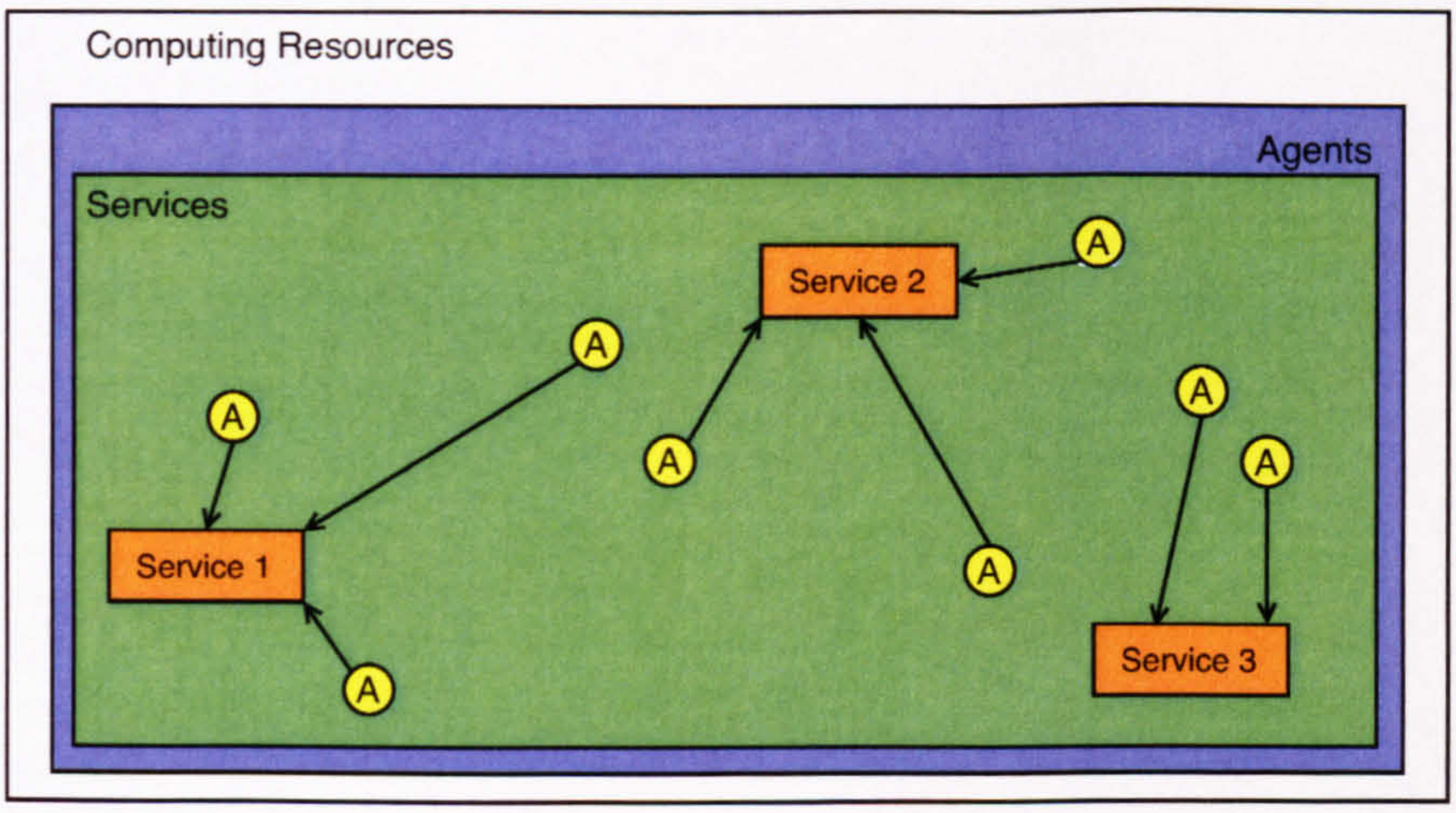

Figure 3.7: *Users communicate with services to execute their tasks, the Overlay Network transforms seemingly heterogeneous resources into a uniform, homogenous group of resources.*

in terms of, for example, CPU power or geographic location but Services can interact with them all in the same way. When Agents have reached the point where a resource has been successfully captured, they can interact with Services in order to pick up and run jobs that have requirements satisfied by the resource hidden beneath.

The key point of the Overlay Network Concept is to maximise the use of the resources, once they have been successfully obtained. Through Agents, resources effectively go through a 'screening' process to ensure that there is a very good chance of success when a job is eventually delivered there. The Overlay Network is a dynamic entity since resource availability can vary considerably over time. Furthermore, on the Grid there is normally a finite period of time allowed once a resource has been captured. However, by ensuring the availability of the Services, individual Agents can come and go without adversely affecting the whole system.

There may also be further requirements placed on the Agents to ensure that the resource is ready to receive a job. For example, by installing any required software at the level of the Agent, the job can be saved from any installation failures. These requirements can vary depending on the Virtual Organisation in question. For example, a Bio-Medical VO may want to test a secure connection to a remote database before allowing a job to execute at a particular site. The specific ways in which LHCb makes use of Agents' control shall be discussed below.

3.3.2 Use of Agents' Control in LHCb

As mentioned in Section 3.1.2 and highlighted in Figure 3.4, LHCb uses DIRAC to send Agents to the Grid. Through the effective use of the Pilot Agent Paradigm, the *PUSH* architecture of LCG can be transformed into a *PULL* system which brings a greater efficiency for LHCb jobs.

The use of Agents to create an Overlay Network, as described in Section 3.3, results in a homogeneous view of heterogeneous resources. This serves to reduce human intervention required to manage LHCb jobs and means that heterogeneous resources can be dealt with in a uniform way.

LHCb also uses Agents' control on LCG in order to place further requirements on acquired resources before user jobs are executed. As mentioned in 3.3.1, it is possible for the software installation step to be delegated to Agents. Therefore, Agents can receive jobs and install any software outside the scope of the job itself. Should a failure occur, this also means there is some redundancy in place since the Agent can fail with a meaningful error and the job can be rescheduled. The software installation itself can take two routes. If no software is available at a particular site, the DIRAC software mechanism is employed, see Section 2.5.4. Alternatively, if the LHCb software has been

pre-installed by an administrator, the Agent can simply set up this software for immediate use. The advantage of the latter is an improved start-up time and greatly reduced overheads for the site. A full binary distribution of the LHCb software is approximately 1Gb in size: this storage requirement can place additional load on the Grid for two main reasons. Firstly, the time it takes for the download of binaries to each computing resource is time spent occupying a resource without actually executing the job, and should be minimised. Secondly, when dealing with thousands of jobs, each having to install software independently, it can become problematic for the sites to clean up after each job and provide enough storage space to satisfy the running jobs.

Another use of Agents' control is to place requirements on jobs from the resource, to be balanced with the requirements of a particular job, in a two way 'double-matching' mechanism. Jobs generally have some form of requirements that should be satisfied by the resource before scheduling to a site can be allowed. The double-matching mechanism means that, not only does the job place requirements on the resource, in addition, the Agent can be used to place requirements from the resource on the job. One example of this would be requiring that jobs come from a particular user. This is an important aspect of Workload Management and the implications will be further explored in Chapter 5.

3.4 Other Examples of Distributed Analysis

Having described the key paradigms for distributed analysis for LHCb in Section 3.1, the aim of this section is to highlight and briefly discuss the main approaches taken by the other main experiments at CERN, namely, ATLAS, CMS and ALICE. The four main experiments in the era of the LHC

will each generate amounts of data on the scale of Petabytes. Therefore, all the experiments must overcome the difficulties of running jobs on the Grid, as well as other available resources, in a consistent way.

Distributed data analysis systems can be broadly classified into two main groups, submission systems and front-end analysis systems. The distinction here is that submission systems can be viewed as launch vehicles, seeking to provide uniform access to many resources in an optimal way. On the other hand, front-end analysis systems normally concentrate on local tools, *e.g.* Graphical User Interfaces (GUIs), for users to configure and manage jobs. The former will mainly be looked at in this section although many submission systems, including DIRAC, also tend to offer some of the functionality pertaining to front-end analysis systems.

A detailed comparison of DIRAC with regard to other systems will be given in later chapters. In this section, a conceptual overview of how the other experiments intend to enable distributed analysis will be given, bypassing some of the more technical details. Several approaches are in place for each of the larger experiments, so what will be discussed here cannot be considered exhaustive. In the next section there will be a more detailed evaluation of distributed analysis using the gLite framework prototype.

3.4.1 Distributed Analysis in ATLAS

DIAL and Panda are two of the submission systems in place for ATLAS, which will briefly be discussed in turn.

Distributed Interactive Analysis of Large datasets (DIAL)

Users interact with DIAL [121] through a user analysis framework. At present, the only supported framework is ROOT [102] and the aims of DIAL

are to extend this to allow submission to batch systems and the Grid in a seamless way. The main component of DIAL is the Scheduler and its interface may be thought of as a high-level job definition language.

Jobs in DIAL consist of an application specification, task and dataset. A task in this context is how to configure the specified application. The DIAL Scheduler can be thought of as a WMS which either runs a job directly, passes it to another Scheduler or splits it by input data. In the latter case, jobs are created for each sub-job and the Scheduler will concatenate the results. Each job produces a result and the result of the original submission is available to the user. A binding in Python exists for DIAL [122] although it is mostly written in C++. DIAL realises the *PUSH* scheduling paradigm.

Production and Distributed Analysis System (PanDA)

The Panda system [123] began in August 2005 and has been inspired by DIRAC. Panda has a very similar architecture to DIRAC and will be discussed in more detail in Chapter 5. Panda is also developed in Python and adopts the *PULL* paradigm for job scheduling, including the use of Pilot Agents. In the same way as DIRAC, Panda began as a production system and is currently being extended for the distributed analysis activity.

3.4.2 CMS Distributed Analysis with BOSS

Batch Object Submission System (BOSS) [124] is a tool for batch job submission, real time monitoring and bookkeeping. BOSS is interfaced to many schedulers both local and Grid, to provide seamless access to resources.

BOSS realises the *PUSH* paradigm through the use of schedulers such as PBS, LSF or the LCG Resource Broker. BOSS provides logging and monitoring information and allows complicated job flows of multiple applications

chained together. The ability to manage jobs with an arbitrary scheduler means that Grid and non-Grid resources can be accessed in a consistent manner. BOSS was successfully used in CMS Monte Carlo productions before deciding to extend it for the user analysis activity.

3.4.3 Distributed Analysis in ALICE with AliEn

Alice Environment (AliEn) was envisaged to provide the ALICE user community a transparent access to computing resources distributed worldwide through a single interface [125].

The AliEn WMS is based on the *PULL* approach and is developed in Perl. AliEn uses the concept of a central task queue and uses central services to manage all the tasks. Computing Elements are defined as 'remote queues' which can send tasks to a single machine, a cluster of computers or a computing Grid. These 'remote queues' can be thought of as Agents.

Input and output associated with any job are registered in the AliEn File Catalogue. This is a virtual file system in which logical names are assigned to files [126], with a semantics similar to the Unix file system.

AliEn and its architecture has been taken as one of the fundamental components on which to build the Enabling Grids for E-Sciences in Europe (EGEE) Grid Middleware, this will be discussed in Section 3.5.2.

3.4.4 Emerging Trends

To summarise the main features of the systems described above, Table 3.1, outlines the main trends. It is interesting to note that several of the systems have adopted the *PULL* scheduling paradigm.

Of the systems considered, it also appears that Python is a popular choice. The main reason for this is that Python is an interpreted language, designed

for rapid application development and deployment. With no dependence on specific compilers, the use of Python lends the system an implicit degree of platform independence.

Experiment	System	Scheduling	Agents Control	Implementation
ATLAS	DIAL	*PUSH*	None	C++, Python
ATLAS	Panda	*PULL*	Pilot Agents	Python
CMS	BOSS	*PUSH*	None	C++,Python
ALICE	AliEn	*PULL*	Remote Queues	Perl
LHCb	DIRAC	*PULL*	Pilot Agents	Python

Table 3.1: *Comparison of distributed data analysis systems.*

Excluding DIAL, the remaining systems have been used for Monte Carlo Production activities. The process of extending these systems by building on previous successes lends confidence through prior experience.

Currently DIRAC and Panda make use of the LHCb Pilot Agent paradigm. This is also being investigated by the other experiments due to the higher efficiency demonstrated with jobs on the Grid. This will be further discussed in Chapter 5.

Through the inception of A Realisation of Distributed Analysis (ARDA) [127] project for the LHC, prototypes of distributed analysis systems have been introduced for the main LHC experiments. In the next section, the EGEE gLite Framework is evaluated for the LHCb distributed analysis activity.

3.5 Distributed Analysis Using DaVinci In the gLite Framework

This section describes work carried out between September 2004 and February 2005 to perform the first realistic physics analysis using the gLite Grid framework [128] prototype. Firstly, an overview of how DaVinci [129] was integrated with gLite will be given in Section 3.5.1. This is followed by an overview of the gLite framework [67] in Section 3.5.2. Next, the example analysis ($B_S \rightarrow J/\Psi\Phi$) carried out with gLite and DaVinci is described in Sections 3.5.3 to 3.5.6.

The gLite prototype is a reduced version of the EGEE Grid Middleware [130,131]. This follows a Service Oriented Architecture and utilises the AliEn [132] file catalogue. DaVinci was introduced to the gLite framework and subsequently a physics analysis on the $B_S \rightarrow J/\Psi\Phi$ channel [133, 134] was carried out.

Using the gLite package manager, analysis jobs were submitted to exploit available Grid resources and test the framework. This required some additional effort but did lead to a successful use of the system. An evaluation of the gLite Framework for LHCb distributed analysis will be given in Section 3.5.7.

3.5.1 Using DaVinci with gLite

DaVinci is the analysis program of LHCb, which is based on the Gaudi Framework [135] and LHCb core packages [136]. Programmed in C++, DaVinci is a collection of distinct packages that are managed using CMT [117]. By using binary releases of the software, currently released as package tarballs, the dependency on CMT can be removed.

In this way, DaVinci depends on five distinct packages, which include: Gaudi; LHCb Software [136]; FieldMap [137]; ParamFiles [138], and XmlDDDB [139]. The typical DaVinci user will generally only need to modify the DaVinci package itself. This procedure is simplified through the use of options files which steer DaVinci. As such, any additional user-specific libraries may be included using only the options files and the dependent packages may effectively be ignored from the perspective of the user.

3.5.2 The gLite Framework

The first instance of the gLite Framework was the gLite prototype [140] which uses the AliEn file catalogue.

As stated in [141], the gLite prototype was designed to accommodate an iterative sequence of user interactions in an analysis context. After a review of existing projects, AliEn [132] was chosen on the basis of showing the most complete distributed analysis functionality. A re-factoring of AliEn and other services into ARDA led to the creation of the gLite prototype.

The gLite Middleware prototype consists of the following core services [142]:

- File catalogue
- Authentication module
- Task queue
- Meta-data catalogue
- Package manager
- Grid access service.

To access gLite, users must first have a valid X.509 Grid certificate registered in a supported VO such as LHCb. This allows the user to become part of a well defined group, sharing resources on the Grid.

An interactive shell is provided for users in order to access Grid services. As described in [143], this shell is implemented as a client within which the user can issue commands similar to those in a standard Unix shell. The file catalogue is organised in a hierarchical way, which is similar to a file system. This has advantages because familiar commands such as `ls` and `rm` may be used in a transparent way for the user. This masks, for example, the relationship between Logical File Names (LFNs) and Physical File Names (PFNs). Files may be added to the catalogue by either specifying a URL or by adding a reference to an already existing file in an accessible storage element.

Around seventy commands are available in the gLite shell. In principle, these provide all the functionality necessary to successfully submit user jobs and retrieve output. However, for the user, this is still very far from what a standard Unix shell provides and the system can feel rather restrictive.

Jobs may be submitted from the gLite shell using a Job Description Language (JDL) file that specifies an executable. For successful submission both the JDL file and the executable must be accessible via the filesystem on the Grid. In practice, the user must manually insert the JDL file and executable file for each specific job into the file catalogue themselves.

In order to retrieve output from gLite to the local file system users must execute a command that brings a copy of the closest PFN to a temporary directory. From this the user can copy the file to their local directory.

The first release of gLite, Release 1, lost some of the functionality of the prototype. The gLite management taskforce decided upon this course of

action during the 4th ARDA Workshop 'The LCG ARDA prototype' (March 2005) in order to focus on key services. As a result, for example, the package manager and Grid access service developed in the AliEn framework were removed from Release 1.

3.5.3 $B_S \rightarrow J/\Psi\Phi$ Channel

The LHCb experiment will investigate asymmetries in the decay of B and $\bar{B}$ mesons, in order to understand the mechanism of CP violation in the quark sector. The $B_S \rightarrow J/\Psi\Phi$ channel will have an annual reconstructed signal yield of 100,000 events and is sensitive to new physics effects. This made it an ideal candidate for performing a typical analysis using DaVinci in the gLite Framework. The 100K events refers to the $J/\Psi \rightarrow \mu\mu$ decay (as does the following analysis), there is an additional 20K $J/\Psi \rightarrow ee$ events in the sample that were not considered. After the initial $B_S \rightarrow J/\Psi\Phi$ decay, the Φ subsequently decays into two K mesons ($\Phi \rightarrow KK$).

The final state of the $B_S \rightarrow J/\Psi\Phi$ decay, consisting of two vector mesons, implies that there are three contributions to the decay [133]. The angular analysis is greatly simplified by considering the transversity basis [144] where it is possible to disentangle the two CP even and one CP odd contributions through the transversity angle (θ_{tr}).

The angular distribution allows the extraction of the CKM triangles property $\delta\gamma$, which could signal new physics if a large enough value is detected.

3.5.4 Analysis Using gLite

The $B_S \rightarrow J/\Psi\Phi$ channel was chosen to provide a typical, generic base on which to test the gLite framework. For the purposes of this analysis, DaVinci v12r3 was deployed using binary tarballs of all dependants.

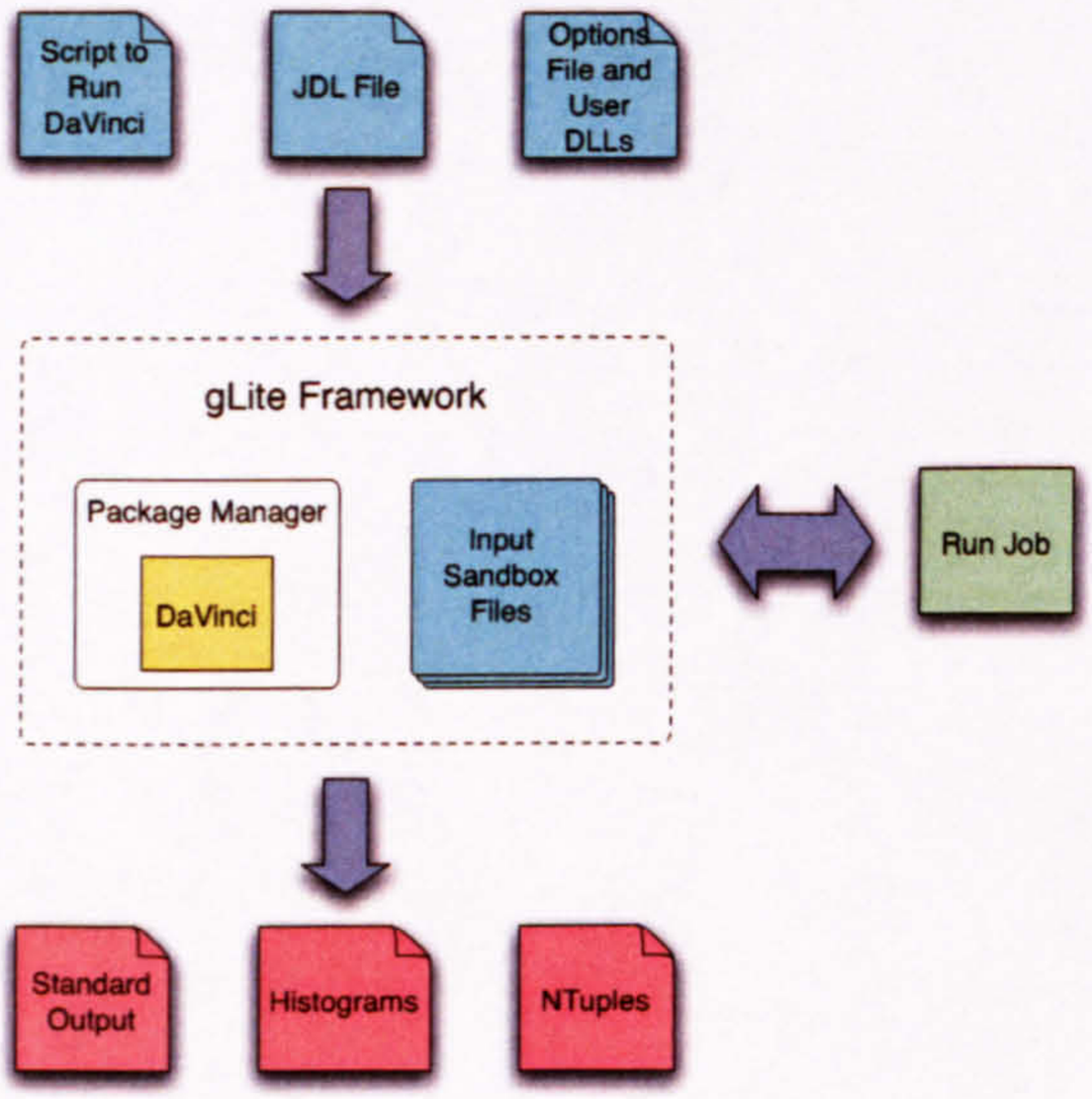

Figure 3.8: *Dataflow during an analysis job using DaVinci through gLite from a user's perspective. The user provides the files at the top and after adding them to the catalogue, gLite will return the output.*

Figure 3.8 highlights the analysis dataflow, from the perspective of the user. A typical analysis using DaVinci involves the creation of user-specific options files and algorithms as well as a large number of standard options files for configuration purposes. These serve as input to the gLite Framework, along with a script to run DaVinci and a JDL file to control job submission. The latter two files are quite generic and could easily be standardised for other LHCb users.

To use DaVinci in the gLite Framework it is necessary to condense all of the options files. This is most easily achieved using JOE, the Job Options Editor [145].

There are two options available at this point, one could use a single tarball of all relevant software or could utilise the gLite package manager described in Section 3.5.5. This choice only impacts on the script to run DaVinci and the

JDL file. Ideally, it is recommended that an LHCb administrator or super-user would insert several versions of DaVinci into the gLite file catalogue using the package manager. This would allow all potential LHCb users to make use of a particular version without having to insert it themselves.

Once this is decided and all files are added to the file catalogue, job submission is possible. gLite handles everything from the point at which the user submits the job. One can observe the job status using the shell-like behaviour inherent to the gLite prototype and then gather the output as desired. For a typical analysis using DaVinci, output comes in three forms, histograms, ntuples and the standard output from DaVinci.

3.5.5 Job Splitting and Use of the gLite Package Manager

Job splitting is possible in the gLite Framework [146]. Inside the job description (JDL file) it is possible to specify a flag to enable splitting and this results in a master job being created after submission. From this point, any operations made on the master job will also affect the sub-jobs so, for example, killing the master job would result in the termination of all sub-jobs.

Job splitting was applied to DaVinci jobs, although some problems did arise. In the event of sub-job failure it is necessary to resubmit either the failed sub-jobs independently or the original job again. At this point, there is no mechanism in place to merge sub-jobs after completion, therefore using this method of job submission was found to be overly user-intensive at this time.

Instead of sending one large, manually constructed tarball of all the software necessary to run DaVinci jobs it is preferable to take the individual tarballs of dependent packages from the LHCb release area and insert them

into the gLite file catalogue. DaVinci in this sense directly depends on Gaudi, XmlDDDB, ParamFiles, FieldMap and the LHCb packages.

Inside the gLite prototype it is possible to insert tarballs as packages with each having specific setup commands specified by the user. In this sense it was possible to create the proper environment for the software to run with each package being installed independently in different locations. The structure of the packages is taken from that of CERN, so the tarballs can be inserted directly from the LHCb release area.

By having an LHCb administrator to set up several versions of DaVinci, a typical user would not need to be concerned with where the packages are installed and how they are set up. There is little change to the environment between versions of DaVinci so the mechanism in place is quite scalable. Subsequent to the work presented here, the gLite package manager has evolved to become more streamlined [147].

3.5.6 Analysis Results and Experience

The results presented here are based on a selection of B_S events run on DaVinci v12r3 in the gLite Framework using a 100,000 event sample. Figures 3.9, 3.10 and 3.11 show the reconstructed J/Ψ, ϕ and B_S respectively. The overall selection efficiency of the B_S was 8.2%.

As mentioned in Section 3.5.3, the angular analysis is greatly simplified by considering the transversity basis. In this basis, the x-axis is defined as the direction of the Φ in the J/Ψ rest frame. The z-axis is perpendicular to the $\Phi \rightarrow KK$ decay plane and the transversity angle (θ_{tr}) is defined as the polar angle of the positive lepton in the J/Ψ rest frame [144].

The transversity distribution in Figure 3.12 shows a very good correlation with the plot on page 12 of [134], which was obtained using a fast parame-

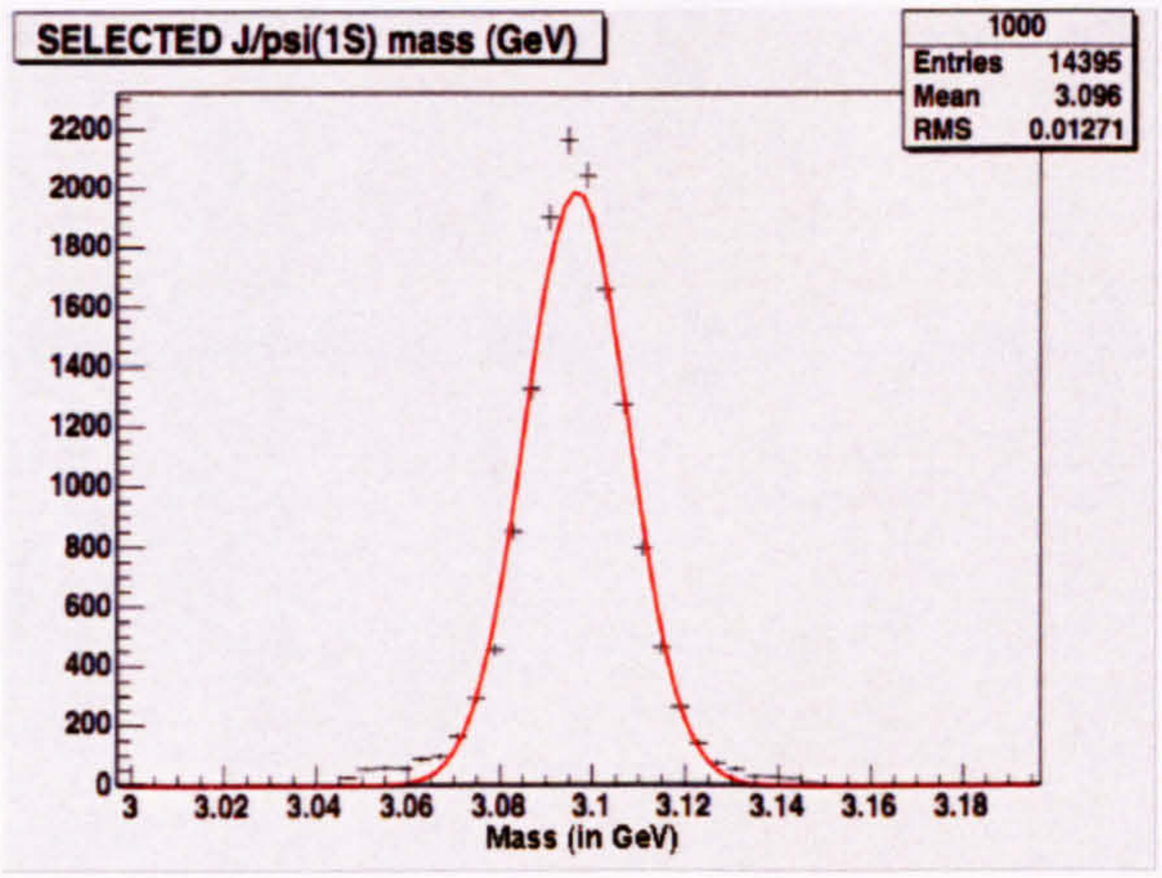

Figure 3.9: *Reconstructed J/Ψ mass distribution (in GeV) after applying J/Ψ selection cuts, run over 100,000 events using DaVinci through the gLite Framework.*

terised 'toy' Monte Carlo experiment. The resulting distribution shows what one would expect from the admixture of helicity states but some investigation into the event generator is required in order to determine that all states are being accounted for.

The gLite Middleware prototype was very much in its infancy when carrying out this analysis. Unfortunately the system could be down for a period of days or even weeks at a time due to many factors. The infrastructure was prone to hanging and often needed to be rebooted. Getting real estimates of system performance and efficiency was also hampered by issues of reliability with individual commands and job submission.

An attempt at robustness tests was made (submission of fifty 25,000 event jobs were sent every day over several days) but unfortunately the system would either execute some or all of the jobs sent or none at all due to stability issues. This was further compounded by the fact that new Worker Nodes were being frequently added, and these did not always behave as was initially

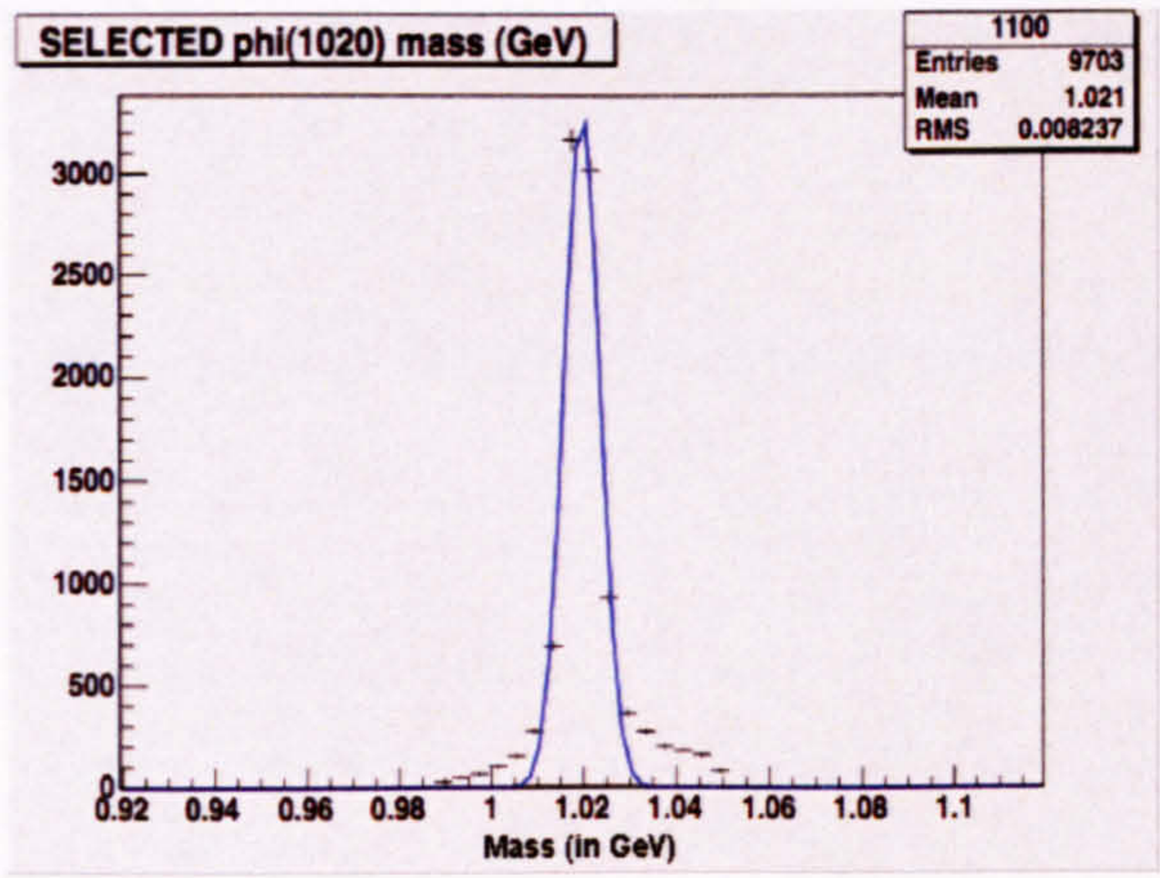

Figure 3.10: *Reconstructed Φ mass distribution (in GeV) after applying J/Ψ and Φ selection cuts, run over 100,000 events using DaVinci through the gLite Framework.*

expected.

CASTOR [148] access was also a problem. Originally all available datasets were picked up without any issues but towards the end of this work the system failed inexplicably and the cause of this was not determined. Another issue was with the Worker Nodes only having 20Gb disks. Unfortunately this made it impossible to run over large numbers of datasets directly. This effectively forces a user to either split their jobs themselves or via the system (see Section 3.5.5).

Using the gLite prototype to perform user analysis required significant additional effort from the user when compared to the use of standard batch systems. Unfortunately it often seemed to be unclear whether the user was at fault or the system itself. There is much scope for improvement however and when the system was working, results for the analysis were obtained.

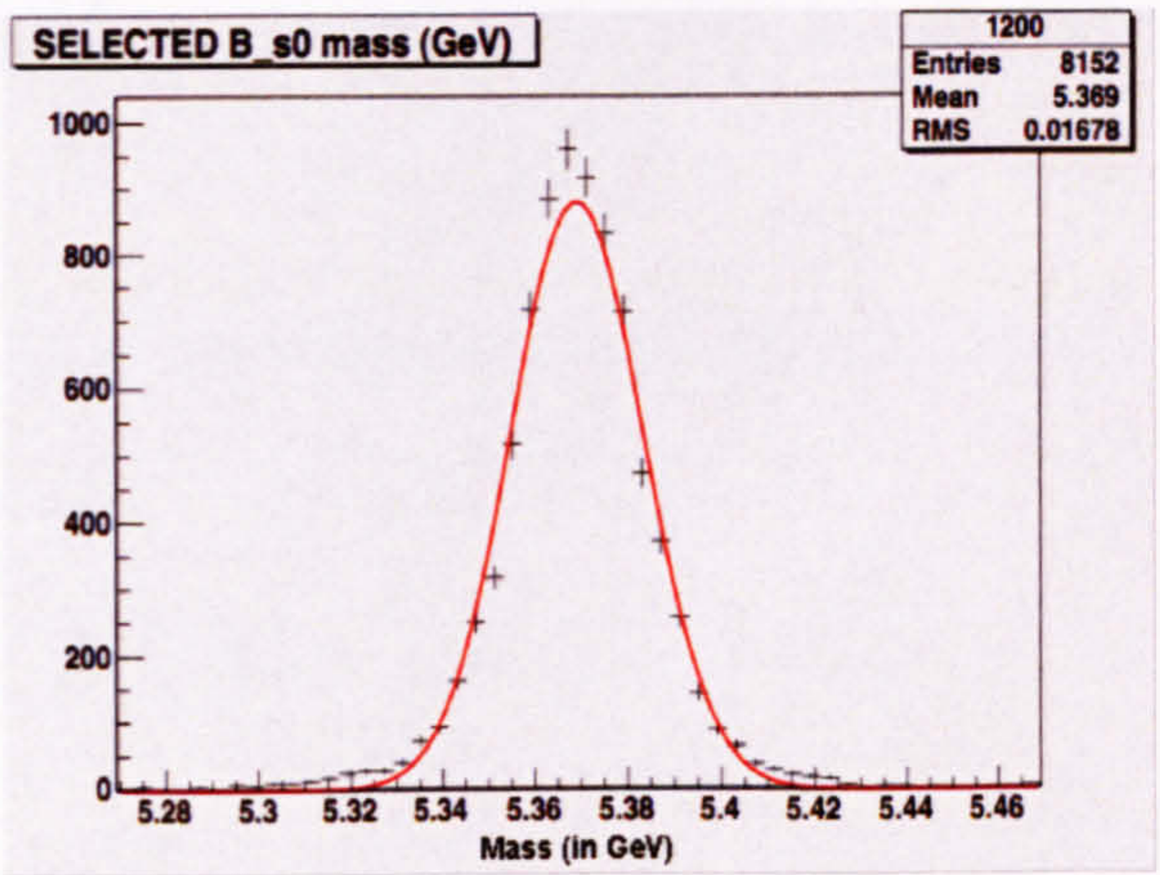

Figure 3.11: *Reconstructed B_S mass distribution (in GeV) after applying all selection cuts, run over 100,000 events using DaVinci through the gLite Framework. The selection efficiency for this was 8.2%.*

3.5.7 Evaluation of gLite for Distributed Analysis

The gLite prototype is a reduced version of the Grid middleware. This infrastructure was tested by carrying out a physics analysis using the LHCb DaVinci software on the Grid. The importance of this was two-fold. Firstly, the tests were used to determine where improvements could be made to the framework. Secondly, the utilisation of Grid resources becomes increasingly important as the start of the LHCb experiment approaches and it has been necessary for new mechanisms of analysis to be explored.

Overall, analysis is possible using DaVinci in the gLite Framework. When the system works it can be relatively painless to use after a familiarity with the system has been established. However, since the system is experimental there were some reliability issues and teething problems. Nonetheless, large jobs were successfully executed using the gLite prototype and this led to the exploitation of Grid resources.

There was initially no direct Agents' control for the gLite Framework and

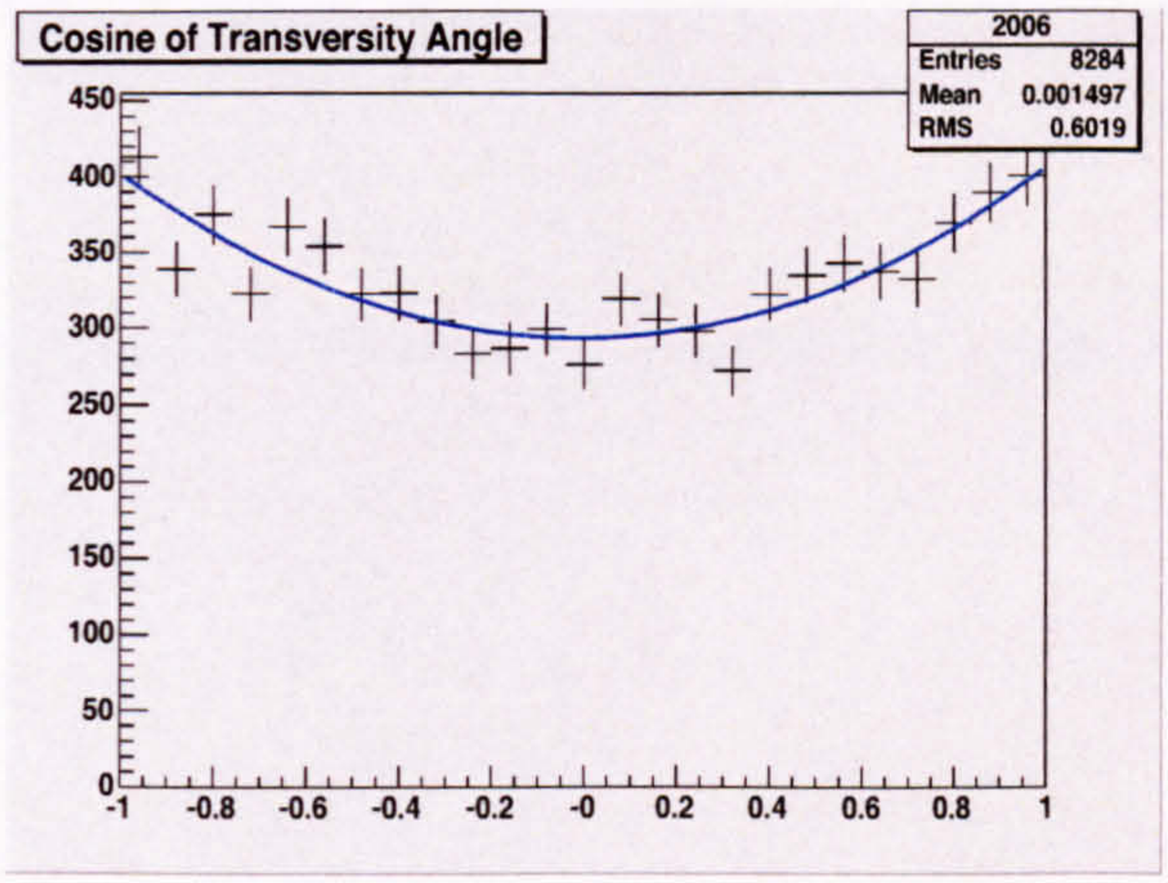

Figure 3.12: *Plot of the cosine of the transversity distribution* $\cos\theta_{tr}$ *for all selected* B_S *events. The distribution shows the projection of all contributions including CP odd and CP even and shows a good correlation with the plot on page 12 of [134] which was obtained using a fast parameterised 'toy' MC experiment.*

although a *PULL* model was envisaged as part of the gLite WMS, the system was initially based on a *PUSH* mechanism, which raises the possibility of scalability problems.

The limitations encountered when performing this analysis led to the decision to extend DIRAC for distributed analysis rather than use the gLite Framework. Since DIRAC proved to be a success for production tasks during DC04, and features were being removed from the gLite first release candidate, as mentioned in Section 3.5.2, it was felt that the already established tools provided by DIRAC should be developed. This decision also meant LHCb could use the same system for production and analysis for all available resources, including those outside of the Grid. It also allows LHCb to remain in direct control of all Grid activity with no dependence on external software providers.

3.6 Summary

This chapter began by introducing the key paradigms for LHCb distributed data analysis. The first of these was the use of *PULL* instead of *PUSH* scheduling and the second was the use of Pilot Agents. It was shown that the Pilot Agent paradigm can be used to facilitate the *PULL* approach, through a *PUSH* system, as in Figure 3.2.

The requirements for LHCb data analysis were discussed in Section 3.2. In the context of LHCb, distributed analysis is a batch analysis but with minimised response time. This involves prioritisation and optimisation of available resources for LHCb. The overall aim is to provide a stable platform for analysis on inherently unstable resources and therefore mask any inefficiencies of LCG from the user. Using the Overlay Network paradigm described in Section 3.3, it is possible to achieve this.

Other examples of distributed analysis were discussed in Section 3.4. Several trends were highlighted such as the use of Python to gain a degree of platform independence and the use of the *PULL* scheduling paradigm. The other experiments are also starting to adopt the Pilot Agent paradigm, which was first realised through DIRAC. One example would be ATLAS with Panda.

In Section 3.5, gLite was evaluated for distributed analysis for LHCb. Although the prototype was being investigated, reliability issues and impressions of general ease of use led instead to the decision to extend the LHCb Production system, DIRAC, for user analysis.

DIRAC makes use of *PULL* scheduling through the Pilot Agent paradigm to increase the efficiency of LHCb Grid jobs. This is accomplished through the Overlay Network of Agents, interacting via Services, which together make up the Workload Management System. The DIRAC system will be described in the next chapter.

Chapter 4

Distributed Infrastructure with Remote Agent Control - DIRAC

This chapter will describe the LHCb distributed workload management system known as Distributed Infrastructure with Remote Agent Control (DIRAC). Section 4.1.1 begins with a discussion of the design principles and philosophy of conception. The main components of DIRAC are resources, services and agents, which are key to realising the paradigms introduced in the previous chapter. The interactions of these components are discussed in Section 4.1.2. A brief history of the DIRAC project will be given in Section 4.1.3, and the software tools chosen to implement the system will be described in Section 4.2.

The Services Framework will be discussed in Section 4.3. This will convey how the software tools are used to securely deploy services in a reliable way. Section 4.4 will describe the Agents Framework, focussing on the two main types of Agent present in DIRAC and how they are utilised. A description

of the Workload and Data Management components of DIRAC is given in Sections 4.5 and 4.6. This is followed by an overview of the Information, Monitoring and Accounting systems in Section 4.7.

4.1 Introduction

DIRAC is the LHCb Workload and Data Management system for Monte Carlo simulation, data processing and distributed user analysis. The present goals and scope of the DIRAC [149] project are to provide the LHCb Collaboration with the following:

- A robust platform to run data productions on all the resources available to LHCb including individual PCs, site clusters and Grids;
- A means to distribute LHCb data as soon as it becomes available, according to the Computing Model [100];
- A well controlled environment to efficiently run user analysis jobs on the Grid; and
- Efficient steering, monitoring and accounting of all the LHCb activities on the Grid and other distributed resources.

These goals have evolved over time. In fact, when DIRAC first started it was with a rather reduced scope and this will be discussed in Section 4.1.3. The Pilot Agent paradigm outlined in the last chapter allows DIRAC to realise the *PULL* scheduling approach on LCG, as described in Section 3.1.2. Through the Overlay Network concept, where Agents interact through Services, the underlying diversity of the heterogeneous resources of the Grid can be hidden from users. Both of these paradigms have been very influential

on how DIRAC has been implemented and are naturally part of the system by design.

4.1.1 DIRAC Design Principles

Following the paradigm of a Service Oriented Architecture (SOA), DIRAC is lightweight, robust and scalable. This was inspired by the LCG/ARDA RTAG architecture blueprint [150] and also the 'Grid services' concept. The latter was introduced through an architecture by which Grid services are defined, Open Grid Services Architecture (OGSA) [3], as well as the Open Grid Services Infrastructure (OGSI) [45] which is a standard to formally specify Grid services in more technical detail.

Although DIRAC has been developed for the LHCb VO which will only be using one Grid (LCG), the system has been designed to be independent of the Grid being used as well as the VO using it. In order to establish some of the design principles of DIRAC, two assumptions are made about applications running on a Grid Worker Node (WN). Firstly, it is assumed that no root privileges exist on the remote site and secondly, that none of the machines are dedicated for LHCb use only. This means that the Grid resources are not assumed to be owned or used exclusively by LHCb. Therefore, one of the key design principles is to ensure a light implementation which is easy to deploy on various platforms. Also, this should be non-intrusive since machines are not necessarily administered for LHCb use alone.

One of the paradigms of the Grid is that users may submit jobs and not be concerned with where these jobs execute: only that they do execute. For this reason, the system must be easy to configure, maintain and operate. The main goal is to minimise human intervention so that DIRAC can run autonomously once installed and configured. Furthermore, it is important

to ensure DIRAC can run in a platform independent way. To facilitate a high degree of efficiency, the platform independence of the system should be demonstrated for the various Linux flavours running on the Grid.

The use of standard components and open-source, third party developments is encouraged where possible. This ensures the system can sustain a high level of adaptability. Therefore, a modular design at each level of DIRAC has been adopted, which lends the system intrinsic flexibility. This simplifies the process of adding new functionality since new modules can be 'plugged in' as required.

4.1.2 Main Components of DIRAC

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. Figure 4.1 outlines the relationship between resources, services, agents and clients which form the main components of DIRAC. These will be briefly discussed in turn below.

Clients

At this stage clients can simply be considered as submitters of jobs or requests. Clients include the Bookkeeping Query Webpage [151], which requests information about datasets and their replicas on the Grid. For distributed analysis and user production jobs, clients interact with the central services via the DIRAC Application Programming Interface (API). This will be further discussed in Chapter 6.

For LHCb production tasks, the Production Console is used. This provides a general framework for the construction and management of produc-

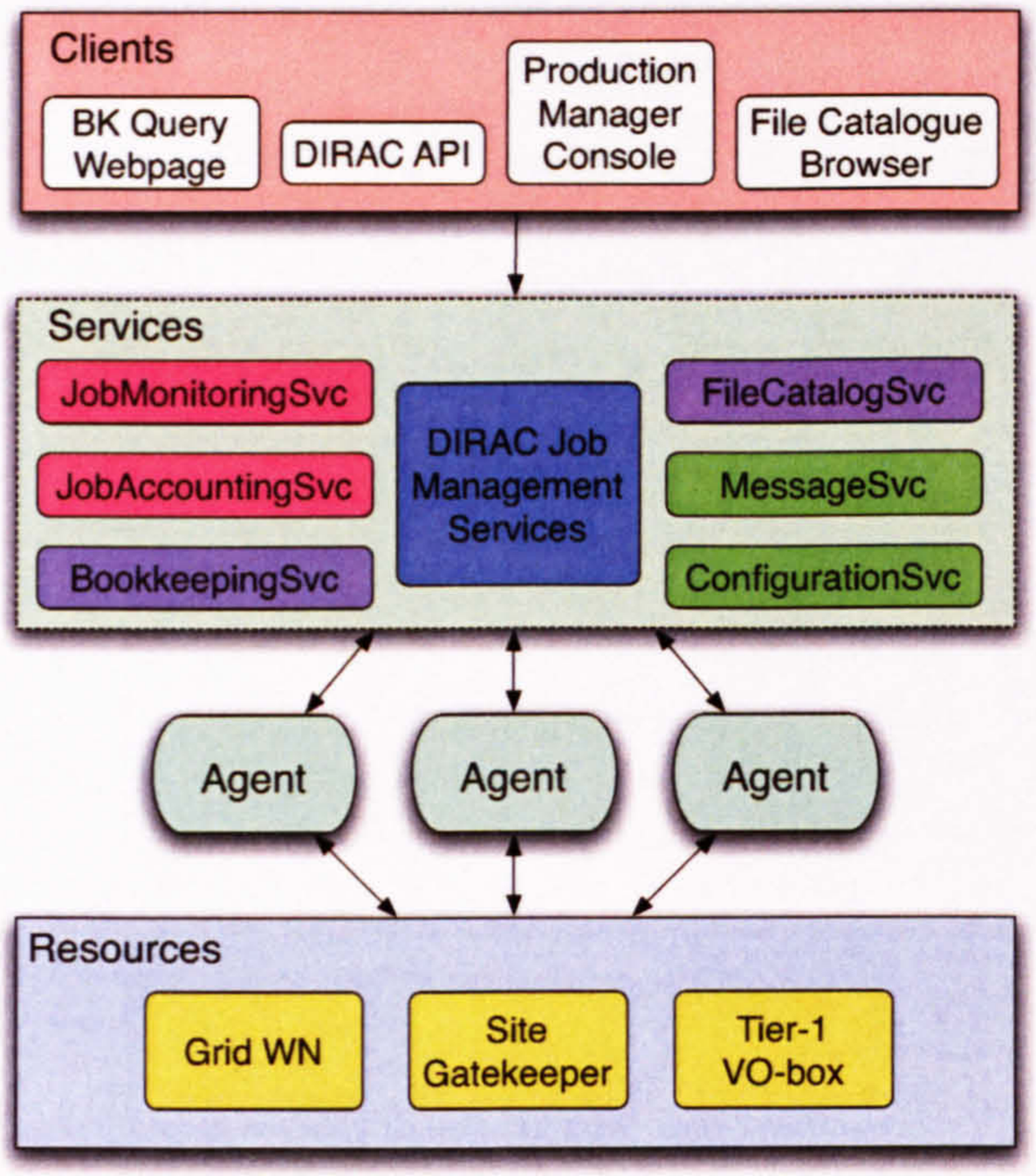

Figure 4.1: *Overview of the main components of DIRAC: Resources; Services; Agents and Clients and how these components interact.*

tion tasks and provides a GUI for users [152]. There is also a File Catalogue Browser which makes use of the Data Management components of DIRAC which will be described in Section 4.6.

Services

The Services highlighted in Figure 4.1 accept requests from Clients and Agents. The DIRAC Job Management Services will be described individually in Section 4.5. They perform vital operations for production and distributed analysis jobs, such as: uploading any necessary files for application steering; and checking any requested input data is available.

The Configuration Service provides necessary site dependent information

for Agents and will be described in Section 4.3.4. The Job Monitoring Service keeps track of changes in job status. Similarly, the Bookkeeping Service will log selected results to provide a history about jobs in case of failure. The role of the Job Accounting Service is to provide statistics, in an automated way, about success rates and the locations where DIRAC jobs are running. The Message Service currently utilises Jabber and is outlined in Section 4.5.4. The File Catalogue Service is used, for example, when outputs must be placed in permanent storage, this will be discussed in Section 4.6.

Agents

Agents are deployed close to resources and form an Overlay Network as described in Chapter 3. On LCG, Pilot Agents are deployed to Worker Nodes via the Resource Broker, whereas on individual PCs and site clusters this is done 'by hand'. The use of non-Grid resources was more prevalent in the early stages of the DIRAC project, which will be highlighted in Section 4.1.3.

Resources

As mentioned in the last chapter, DIRAC can integrate resources such as Individual PC's, site clusters and Grids. This is reflected in Figure 4.1, with the only difference from the perspective of Services being how the Agents are deployed in each context.

4.1.3 History of DIRAC: Evolution from Production to Analysis System

The DIRAC project started in September 2002, Figure 4.2 illustrates the major milestones and developments since then. The first production of MC simulation events using DIRAC was demonstrated in the autumn of 2002. During 2003, DIRAC was first used for MC simulation event production in the first Physics Data Challenge (PDC1). Over a two month period of continuous running during PDC1, DIRAC was used to generate 40 million physics events, corresponding to about 9 Terabytes of reconstructed data [153]. For this, DIRAC made use of the DataGRID [154], which was the predecessor of EGEE, as well as institutional batch systems running DIRAC in a non-Grid environment (hereafter referred to as DIRAC sites).

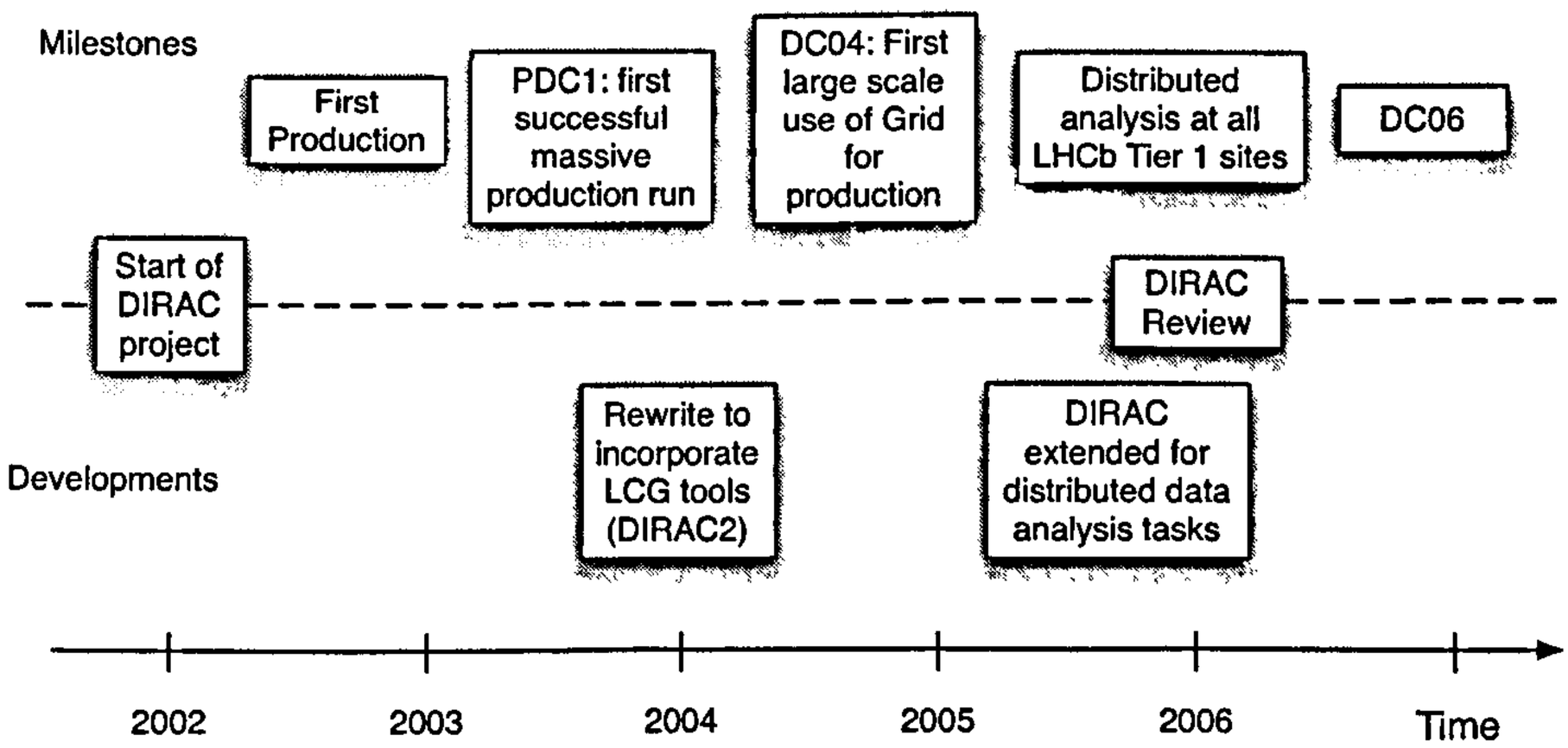

Figure 4.2: *The timeline of the main milestones and developments of DIRAC to date. The project started in September 2002 and is presently being used for the LHCb DC06 activity.*

A complete rewrite of DIRAC was undertaken for the 2004 Data Challenge (DC04) in order to incorporate LCG resources that were available at

the time [149]. This resulted in the second version of the project, DIRAC2. Some of the results of DC04 have already been shown in Section 2.4. This was the first large scale use of the Grid for LHCb data production.

After the successful experiences with DIRAC in DC04, the decision was made to extend the functionality of DIRAC to also include distributed analysis tasks in 2005. This resulted in LHCb successfully meeting an LHCC milestone to perform distributed data analysis at all LHCb Tier-1 sites. The work performed to extend DIRAC will be described below as well as in subsequent chapters. In November 2005, a review of DIRAC was undertaken. This has resulted in many useful recommendations for the organisation and structuring of the project [149].

One of the key themes throughout the history of DIRAC is the increasing use of LCG resources. During DC04, there were still several DIRAC production sites in use. However, the primary mode of submission for LHCb production and analysis jobs is now via LCG.

DIRAC is now the LHCb Workload and Data Management system for Monte Carlo simulation, data processing and distributed user analysis for LHCb, and is actively being used for the 2006 Data Challenge. Many of the software tools used to implement DIRAC have been consistently used throughout the project and this will be discussed in the next section.

4.2 DIRAC Implementation: Software Tools

This section provides an overview of the specific software tools used to implement DIRAC and the motivation for selecting them.

Implementation Language

DIRAC is implemented in Python. Python was selected as it has the following key advantages over other common options such as C++, Java and Perl. Firstly, Python is an interpreted language. This provides a degree of platform independence which, for example, C++ does not exhibit. Java is an interpreted language, although it is arguably easier to program in Python. Unlike Perl, Python is a very readable language that facilitates a fast development cycle when working in a group of developers. The speed of Python has not been an issue for DIRAC thus far, and so it has been unnecessary to rewrite any of the code to increase performance.

Remote Procedure Calls

Efficiently performing Remote Procedure Calls (RPCs) in a distributed environment is essential. This is the way in which clients can interact with services on the Grid. Due to the heterogeneous nature of the computing systems, from hardware to operating system, a standard for communication between clients and services needs to be established. The two choices are XML Remote Procedure Call (XML-RPC) protocol [155] or Simple Object Access Protocol (SOAP) [156].

XML-RPC was chosen for use in DIRAC over SOAP. This was essentially due to its simplicity and lightweight nature. Using HTTP (Hyper-Text Transfer Protocol) for transport and XML for encoding, XML-RPC stores information in key-value pairs which is very simple to implement and maintain. SOAP, on the other hand, is designed for the transport of complicated (user defined) data types, which involves overheads due to the extra information about what is being sent. It was felt that DIRAC did not require the heavier machinery of SOAP and thus far, XML-RPC has been sufficient.

The XML-RPC protocol is available as a standard Python library and with this simple, lightweight approach comes speed.

Security

The client-service communications are secured using the DIRAC Secure Transport (DISET) framework [157] which is conformant with the standard Grid Security Infrastructure (GSI) [158]. This will be described in more detail in Section 4.3.1. The key elements are the use of XML-RPC transport over a Secure Socket Layer (SSL) tunnel, with authentication being performed via X.509 certificates and grid-proxies.

Third-party Components

Job scheduling in DIRAC is achieved through the *PULL* scheduling paradigm via a matchmaking service. This will be discussed in Section 4.5 but makes use of Condor Classified Advertisements (ClassAds) [75]. These are structures which contain descriptions of the characteristics of the sender, used to determine whether a particular resource is suitable for a job.

Another third-party component integrated into DIRAC is the Jabber [159] instant messaging system which is used for reliable service-service communication. Its potential use for providing job interactivity will be briefly described in the context of the Agents framework in Section 4.5.4.

A MySQL database is used for maintaining all information for services and jobs. MySQL [79] is a free, fast and reliable open source relational database which is used in DIRAC to store information about jobs such as: logging information; input / output sandboxes and task queues. The use of the DIRAC MySQL database will be described in the context of WMS services in Section 4.5. MySQL was chosen instead of more powerful com-

mercial alternatives, such as ORACLE [78], since the performance has been sufficient so far.

The CERN CVS repository [160] is being used to maintain the code. The code is structured in sub-directories broken down by their component family. The distribution of DIRAC is made via a tarball (*i.e.* a `.tar.gz` file) which contains the whole code base. Due to recommendations made in [149], the packaging of the project has been changed so that only the necessary code is deployable in the different contexts of use *e.g.* separate client and WMS. The DIRAC distribution also includes some basic LCG software such as a GridFTP client and LFC client. This is bundled in a Linux flavour-independent way for use on sites that do not provide these tools by default.

The LCG file catalogue (LFC) [77] is now being used by DIRAC and is queried as part of the job submission procedure. The decision to use the LFC was based on experience with other file catalogues and this will be described in Section 4.6.2.

The *runit* [161] set of tools has been used to enhance the reliability of the services framework, which will be further discussed in Section 4.3.3.

4.3 Services Framework

Services in DIRAC are permanently running, passive components, which respond to incoming requests from clients. Therefore services, unlike DIRAC Agents, need inbound connectivity. This section presents an overview of the services framework with consideration to three main topics: security; deployment and reliability.

DIRAC implements a client-server architecture which exposes server methods via the XML-RPC protocol. In order to protect the system from misuse,

such as Denial of Service attacks and unauthorised access, it is imperative that services are actively designed to combat these problems. The public interfaces of DIRAC services have to be able to check the validity of all input parameters and also provide access control for exposed methods, *e.g.* by limiting the number of concurrent threads processing requests [149].

The deployment of DIRAC services aims to provide redundancy and reliability. Currently, some of the central Services are running on stable servers at CERN, Barcelona and Marseille. These are administered by respective site managers. An overview of the DIRAC Configuration Service will be given in Section 4.3.4 to illustrate these points.

4.3.1 Security in DIRAC - DISET

DIRAC Secure Transport (DISET) [157] is the security mechanism for DIRAC. This is based on the use of X.509 digital certificates and Grid proxies, both of which are signed by a trusted Certification Authority (CA). DISET is an extension of HTTP over SSL (HTTPS), which provides an enhanced, secure XML-RPC client that is useable in the same way as the native Python XML-RPC client. DIRAC clients only need access to a valid Grid proxy, CA public keys and the Certificate Revocation List (CRL) in order to establish a secure connection to services.

The process of making a secure connection has three main steps as outlined in [157]:

- Authentication;
- Authorisation; and
- Logging.

Authentication involves not only the client being recognised by the server but also the server being recognised by the client. This communication is encrypted through SSL after successful identification, and the user is identified through the Distinguished Name (DN) present in the certificate or proxy.

To perform authorisation of the client a query is made to the server via XML-RPC. The decision is made based on the server configuration with authorisation rules based on user groups and roles, restricting access based on the identity of the user. The groups and roles for users are defined within the DIRAC Configuration Service, which is described in Section 4.3.4, and are mapped to those present in proxies from the Virtual Organisation Membership Service (VOMS) [70]. The VOMS project [69] aims to provide information about the operations a user is allowed to perform within the context of their VO as well as their group and role.

4.3.2 Deployment

DIRAC services interact with three main components: clients, via the user interface; running jobs; and Agents. The DIRAC services accept incoming connections from these components and are either deployed centrally or running at VO-boxes, which will be described below. The central deployment of services is accomplished on LHCb managed, LXGATE-class machines at CERN. More information on the specific instances and deployment of DIRAC will be given in Chapter 6.

The architecture of DIRAC allows the deployment of different services on different machines. Where necessary, communication is possible via the Jabber Instant Messaging service, discussed in Section 4.5.4. Load balancing can therefore be accomplished by deploying services to different machines as necessary. To date, however, it has been sufficient to deploy an instance of

the DIRAC central services to one machine without overloading it during operations.

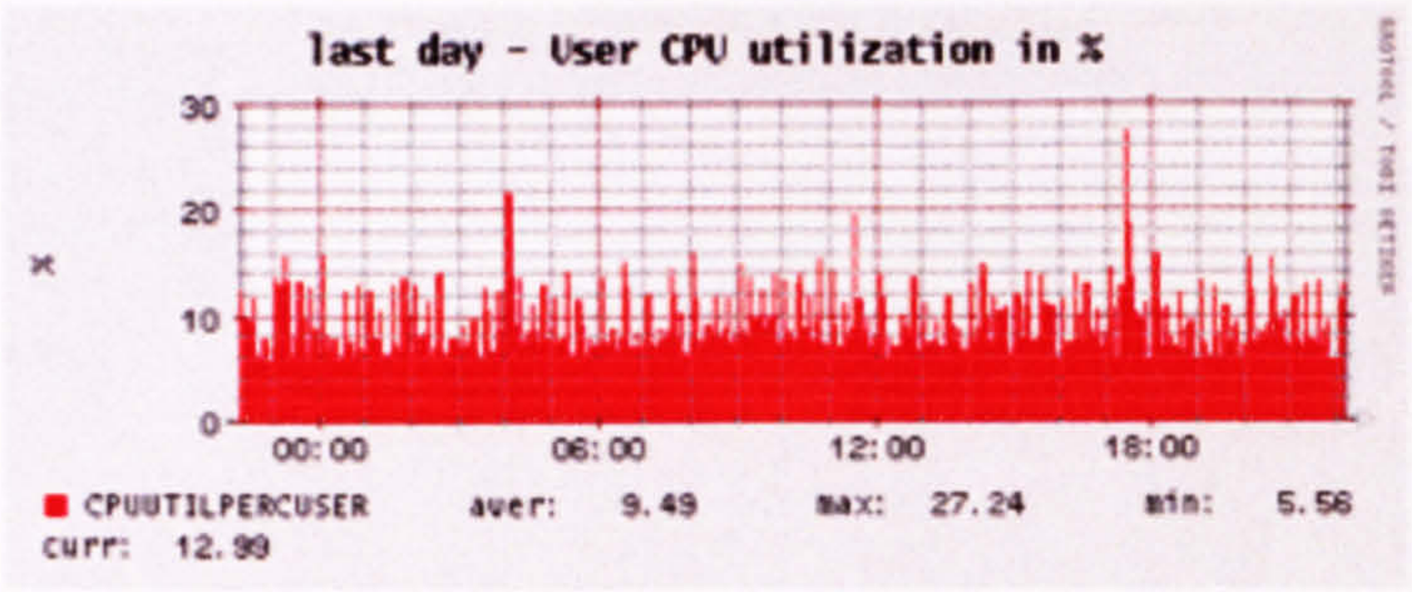

Figure 4.3: *CPU usage on the machine hosting the DIRAC central services in a 24 hour period during the RTTC production in May / June 2005, from [162].*

To illustrate this, Figure 4.3 from [162] shows the CPU usage on the LXGATE machine where the DIRAC central services were deployed during the LHCb Real Time Trigger Challenge (RTTC) production in May / June 2005. During this period, over 5000 simultaneous jobs were running, only limited by the available LCG resources. This exhibits a far from critical load on the server. An evaluation of the system for anticipated future requirements of distributed data analysis jobs will be explored in Chapter 6.

VO-box Services

As mentioned above, DIRAC services are either deployed centrally or to VO-boxes. A VO-box is a dedicated host at a Tier-1 or Tier-2 centre, which can run critical LHCb VO services for the purposes of providing redundancy and efficiency at the site. VO-boxes also provide load-balancing, whereby Tier-2 and Tier-3 sites may access their local Tier-1 VO-box instead of relying purely on central LHCb services.

Each experiment has its own requirements and specification for VO-boxes,

but for LHCb [163], these perform tasks such as retrying failed operations on Grid WNs. One example is data transfer operations at the end of production jobs. Transferring files to Grid SEs can be accomplished via the nearest VO-box even if the central services are down. In fact, by delegating all data moving operations to Agents deployed on VO-boxes, WNs can be freed ahead of time, thus increasing the throughput of sites.

4.3.3 Reliability

Power cuts or system reboots have the potential to interrupt DIRAC services, it is important to recover from these type of events in an automated way. The reliability of DIRAC central services is ensured through the use of *runit* [161]. The services themselves run in user space and *runit* provides a 'watchdog' process in order to restart services in case of failure or system reboot. The use of *runit* does require root access, at least for the installation and configuration of the DIRAC services. *runit* also offers several time-stamped logs which automatically track progress. These rotate in order to provide as much of the recent logging information as possible. This eases the process of monitoring and controlling the DIRAC central services.

For the developers of DIRAC, the use of *runit* means that the process of creating a service also involves the provision of a short script detailing any special setup instructions. These *runit* scripts are normally trivial to write and constitute a negligible overhead on programmer time.

For extra redundancy and load balancing, the critical central services can have mirror services. However to date, the times during which the DIRAC services are unavailable have usually coincided with periods where LCG services have also been affected. The potential strategies and benefits of mirroring the DIRAC WMS will be described in Chapter 6.

4.3.4 Example: DIRAC Configuration Service

To illustrate the principles of the service framework in practice, the DIRAC Configuration Service (CS) [164] will be discussed here. The CS is an integral part of the Information System for DIRAC and provides configuration information for various system components such as Services, Agents and Jobs via XML-RPC.

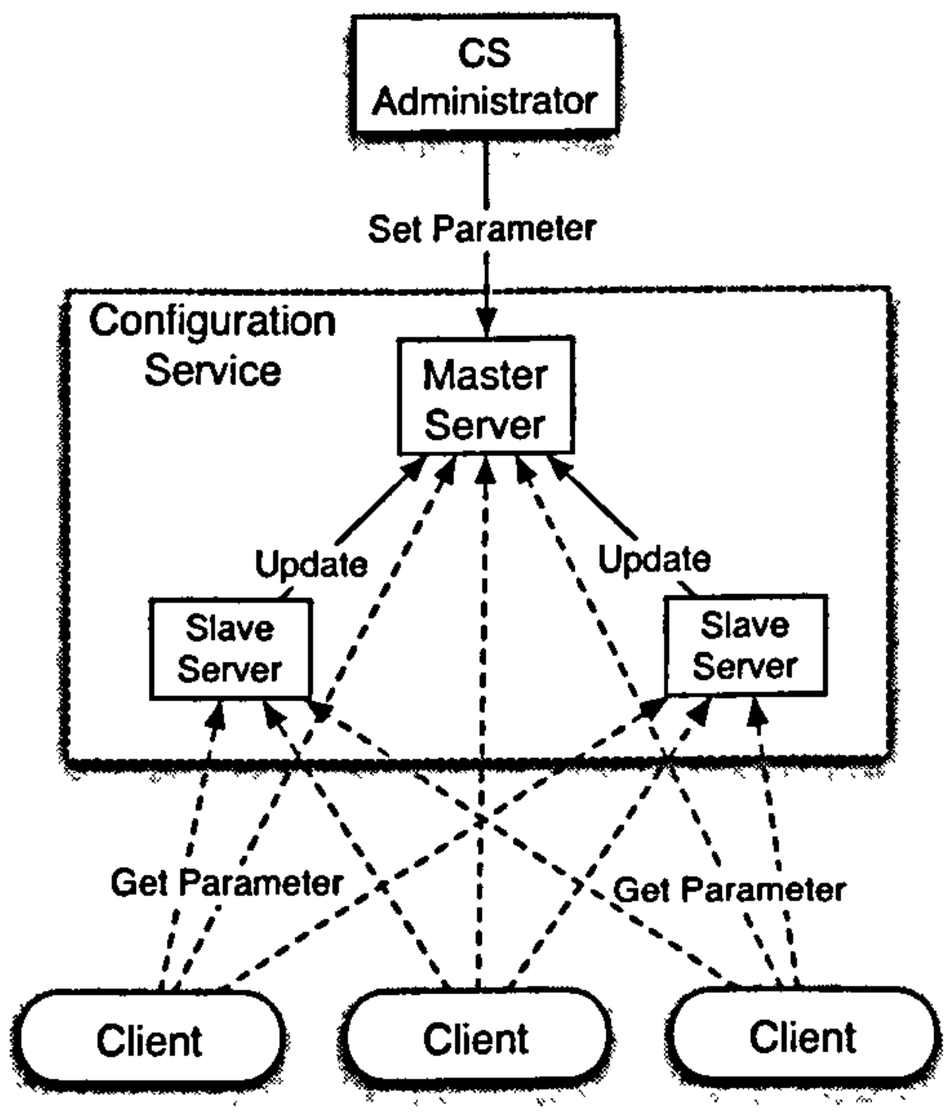

Figure 4.4: *Overview of the DIRAC Configuration Service. This has a hierarchical structure where the Master Server updates the slaves on request. Clients can access any of the servers to receive consistent configuration information.*

A hierarchical structure was chosen for the CS, which is reflected in Figure 4.4. The main components here are the Master Server, Slave Servers and Clients. In order to access all necessary configuration information, Clients only need to have the URL of a CS server. There can be many geographically distributed Slave Servers to provide redundancy and load balancing.

The Master Server keeps all configuration data organised in sections con-

taining options with their values in the form of Microsoft Windows *.ini* files. The information from the Master Server is published to all Slave Servers, which are automatically notified whenever a change takes place. The Slave Servers cannot change any configuration data themselves, in fact, changes can only be made local to the Master Server. When the data changes, the Slave Servers update their local copy and the same is true for Clients, which exhibit the same behaviour. The Clients have a list of possible servers to connect to, with each being tried in turn in the event of a failure.

The DIRAC CS uses a DISET command line interface to secure the system against unauthorised changes. The CS is currently deployed with servers running at CERN, Marseille and Barcelona, providing 100% availability to DIRAC components. The CS servers are also deployed with a watchdog to restart in case of failures.

4.4 Agents Framework

Services provide the means for Agents to communicate and perform tasks. This is part of the Overlay Network concept described in Section 3.3. DIRAC Agents are lightweight components which are easy to deploy, with Services being passive components. Agents bring the whole system to life by sending requests. For this reason Agents need outbound connectivity, but only to well defined URLs. As an example, one such URL could be to the CS servers as described in Section 4.3.4. This is secure by nature and eliminates potential problems with firewalls. Agents are running in user space and do not require any special privileges on sites. Since they are written in Python, only the interpreter is required for deployment.

In keeping with the modular nature of DIRAC, Agents can be thought

of as containers of pluggable modules. These can be put to use in a custom way, with several Agents running on the same site using a different set of modules. The configuration of the Agent determines which modules are used. The DIRAC Data management tools are based on 'plugging in' a module to perform a particular function, such as data transfer operations, with the configured DIRAC Agent running on sites, see Section 4.4.3.

Agents make use of the DIRAC Computing Element to mask the heterogeneity of computing resources. This will be described in Section 4.4.1. It allows Agents to form an Overlay Network and provides a consistent way to execute jobs and interact with services.

There are two types of Agent in use in DIRAC, differing only in their configuration and deployment. Firstly, Section 4.4.1 will describe Site Agents, which are typically used outside of the Grid. Secondly, Pilot Agents will be discussed in Section 4.4.2. Pilot Agents are submitted automatically to LCG via the Resource Broker, as introduced in Section 3.1.2.

4.4.1 Site Agents

Site Agents, can be used outside of the Grid on individual PCs and clusters, but also on VO-boxes. By obtaining a tarball of the DIRAC distribution, it is possible to run an Agent via a script in user space on a site. The Agent can run on individual machines or on a site gatekeeper host to provide access to a site cluster. Site Agents in DIRAC are deployed and updated via human intervention and run as daemon processes.

As mentioned above, Site Agents can be used for job steering on a local cluster. Site Agents have further uses such as: data management tasks on a local Storage Element, discussed in Section 4.4.3; or for the bookkeeping of jobs, where Site Agents can provide logging and accounting information to

track progress.

A possible future direction is the use of Site Agents to set up a temporary DIRAC site on individual PCs, with the consent of the owner, to provide extra resources to LHCb whenever the PC is not in use. This would realise a cycle-stealing paradigm, similar to SETI@Home [27] and BOINC [29]. Since there is also ongoing work to port DIRAC to Windows, it could become a useful way to secure additional resources for the experiment.

Computing Elements

In order for Site Agents to cope with many heterogeneous computing resources, they are equipped with many different Computing Element (CE) interfaces, all of which provide a standard API for job submission and monitoring. This presents an abstract view of a batch system, having a local scheduler and queues, where the CE is a head-node managing a cluster of WNs.

DIRAC currently provides CE interfaces for the following systems: LSF; PBS; NQS; BQS; Sun Grid Engine; Condor; Globus; LCG and stand-alone systems [165].

4.4.2 Pilot Agents

Pilot Agents run on Grid WNs and are submitted by the DIRAC WMS using the credentials of the user. They reserve the resource for the immediate use, requesting jobs from the WMS. Pilot Agents steer job execution as well as operations needing to be performed after the job has finished such as uploading of data to a Grid SE.

Resource reservation through the use of Pilot Agents creates the Overlay Network described in Section 3.3, that masks the heterogeneity of the under-

lying resources from the users of the system. Moreover, since Pilot Agents are sent on behalf of the user, the door is opened to further optimisations on the level of that user. In the past, the main purpose of submission systems such as DIRAC was purely to deploy jobs to the Grid in as quick a manner as possible. Now, however, it becomes important to optimise the use of the resources once they have been captured by Pilot Agents. The possibility of running further jobs for users on captured resources shall be explored in Chapter 5.

The use of Pilot Agents also means that the DIRAC *Task Queue* is the only waiting queue in the system. This allows the LHCb VO to impose prioritisation policies in one place, something that the presently available LCG tools cannot provide. Since distributed analysis tasks generally have a higher priority than production tasks, this is an important way to ensure a minimised start-up time for these jobs.

4.4.3 Example: Transfer Agent

This section discusses a specific example to illustrate the use of the Agents framework. The DIRAC Transfer Agent [166] was used during the Service Challenge 3 (SC3) activity in 2006 to integrate the DIRAC Data Management Services to the gLite File Transfer Service (FTS).

This is illustrated in Figure 4.5, where the Request Database is populated with transfer or replication requests. These requests can be made from a Data Manager directly or via the DIRAC WMS from jobs. This will be discussed along with the Data Management components in Section 4.6.

The Transfer Agent is deployed at the LHCb Tier-1 centers using *runit*, see Section 4.3.3, and runs autonomously once configured. It periodically checks the validity of requests and subsequently passes them to the FTS ser-

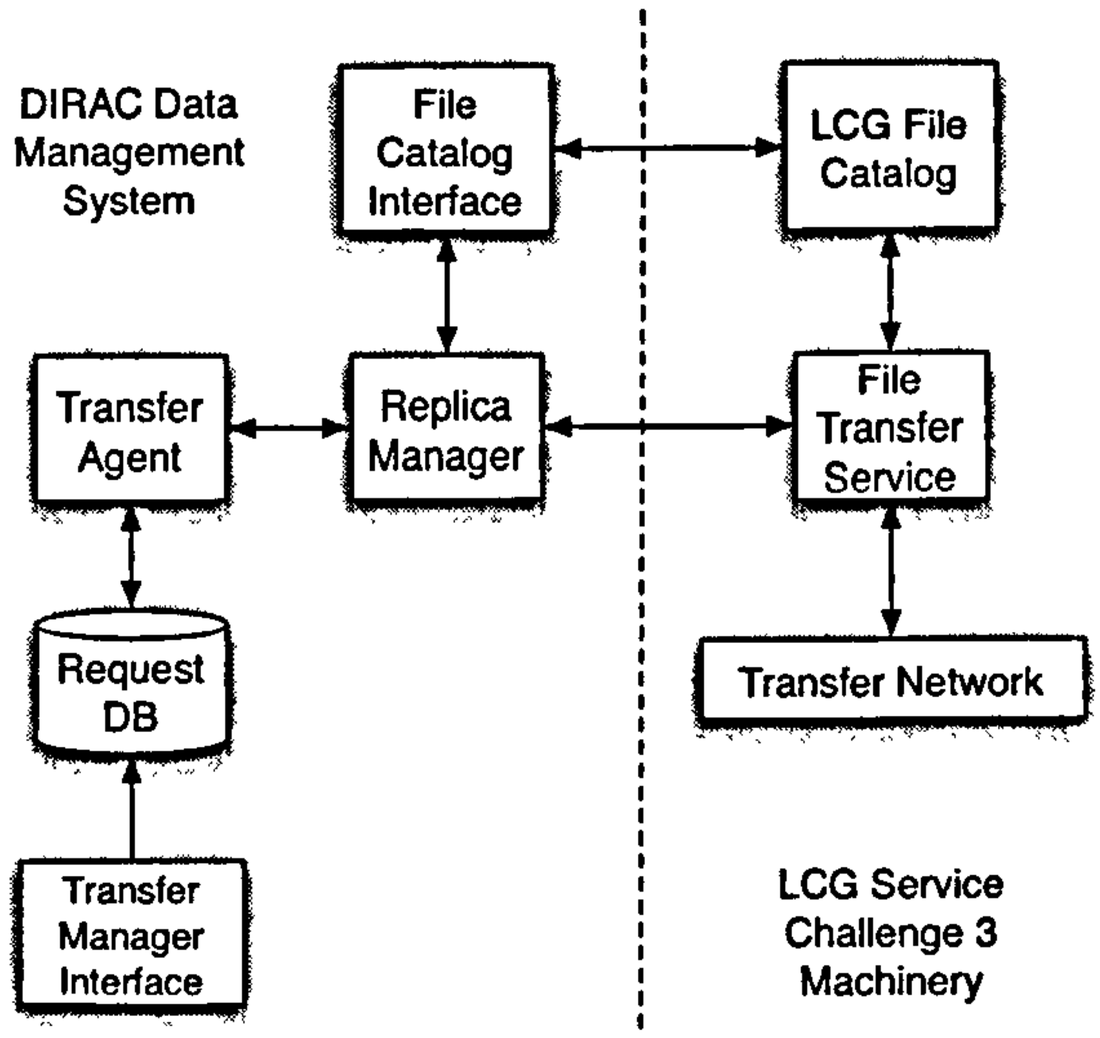

Figure 4.5: *Schematic overview of the DIRAC Transfer Agent and integration with FTS as used in the Service Challenge 3 activity, adapted from [166].*

vice. The infrastructure for this was developed by adding some new methods to interface to FTS and to deal with bulk operations. The existing components are still employed to use third party transfer in case of FTS channel unavailability and for retries in the case of transfer failures. When a transfer has been successful, the new replicas are entered into the file catalogue (LFC).

4.5 Workload Management

As illustrated in Figure 4.1, the services in DIRAC comprise of the Job Management Services as well as several other key elements such as the DIRAC CS and the *Job Monitoring Service*. The focus of this section shall be on the components of the Job Management Services. In the next chapter, job workflow and possible workload management optimisations will be considered.

The DIRAC WMS realises the *PULL* scheduling paradigm whereby Agents

are requesting jobs whenever the corresponding resource is free. Agents steer job execution on sites and jobs report their state and environment to the central *Job Monitoring Service* for the purposes of logging. Job Agents running on sites and on Grid worker nodes create tailored *Job Wrappers*, which are dynamically generated from templates, by providing job as well as site specific data. The DIRAC *Job Wrapper* plays a crucial role in the WMS and will be discussed in Section 4.5.1.

The DIRAC WMS is composed of a set of central services along with Pilot Agents and *Job Wrappers*. Job scheduling occurs late with respect to submission to DIRAC. This is because when scheduling occurs, the job goes to a site or WN for immediate execution. Scheduling is achieved through the *Matcher* service using Condor ClassAds [75].

Several components had to be extended, or newly introduced, to transform the DIRAC production WMS to handle the increasing requirements of LHCb distributed data analysis jobs. Underlying several of the main services is the DIRAC MySQL Job Database (JobDB), which will be described in Section 4.5.2. The key WMS services as well as the more recent developments will be discussed in Section 4.5.3.

Jabber is used in DIRAC for communication between some of the WMS components. This will be explored in Section 4.5.4, along with the possibility of providing interactivity with running jobs.

4.5.1 DIRAC Job Wrapper

The functions of the DIRAC *Job Wrapper* will be discussed in detail in subsequent chapters and is a key component of the DIRAC WMS. It performs many tasks associated with the management of jobs such as:

- Transfer of input files to steer applications

- Invoking the job application
- Providing access to any requested input data files
- Collecting information regarding the job execution environment as well as resource consumption parameters. These are passed along to the *Job Monitoring Service*
- Transfer of small output files via DIRAC
- Transfer of large output data files to Grid Storage Elements.

The *Job Wrapper* also runs as a 'watchdog' process, in parallel to the job, providing 'heart-beats' for the *Job Monitoring Service.*

If, for whatever reason, these jobs stop sending heart-beats, it is assumed a problem has occurred and the job is marked as 'stalled'.

4.5.2 Underlying Database

Underlying several WMS services described below is a MySQL database. In DIRAC, this is not accessed directly but through the *Job Database* (JobDB) class. This is a consistent API that makes the use of MySQL commands transparent in order to mask the underlying technology. In this way, changes could be made to the database without requiring significant changes to the DIRAC code.

The database contains full information about all the jobs such as the job description and status. It is also used to store primary job parameters, which are those common to all jobs, as well as any extra job parameters specific to individual jobs. The access to commonly used primary parameters is optimised through the JobDB class.

The database could eventually migrate to a real SE but it is not clear at this point how much the performance would be compromised. For safety, the database is regularly backed-up, so the WMS can be completely restored on the same or another machine. This is presently done 'by hand' and could be automated in the future. Automation would be particularly important in order to 'mirror' the WMS for extra redundancy.

4.5.3 WMS Services

In order to cope with the increasing requirements of LHCb distributed data analysis jobs, the DIRAC Production WMS was extended.

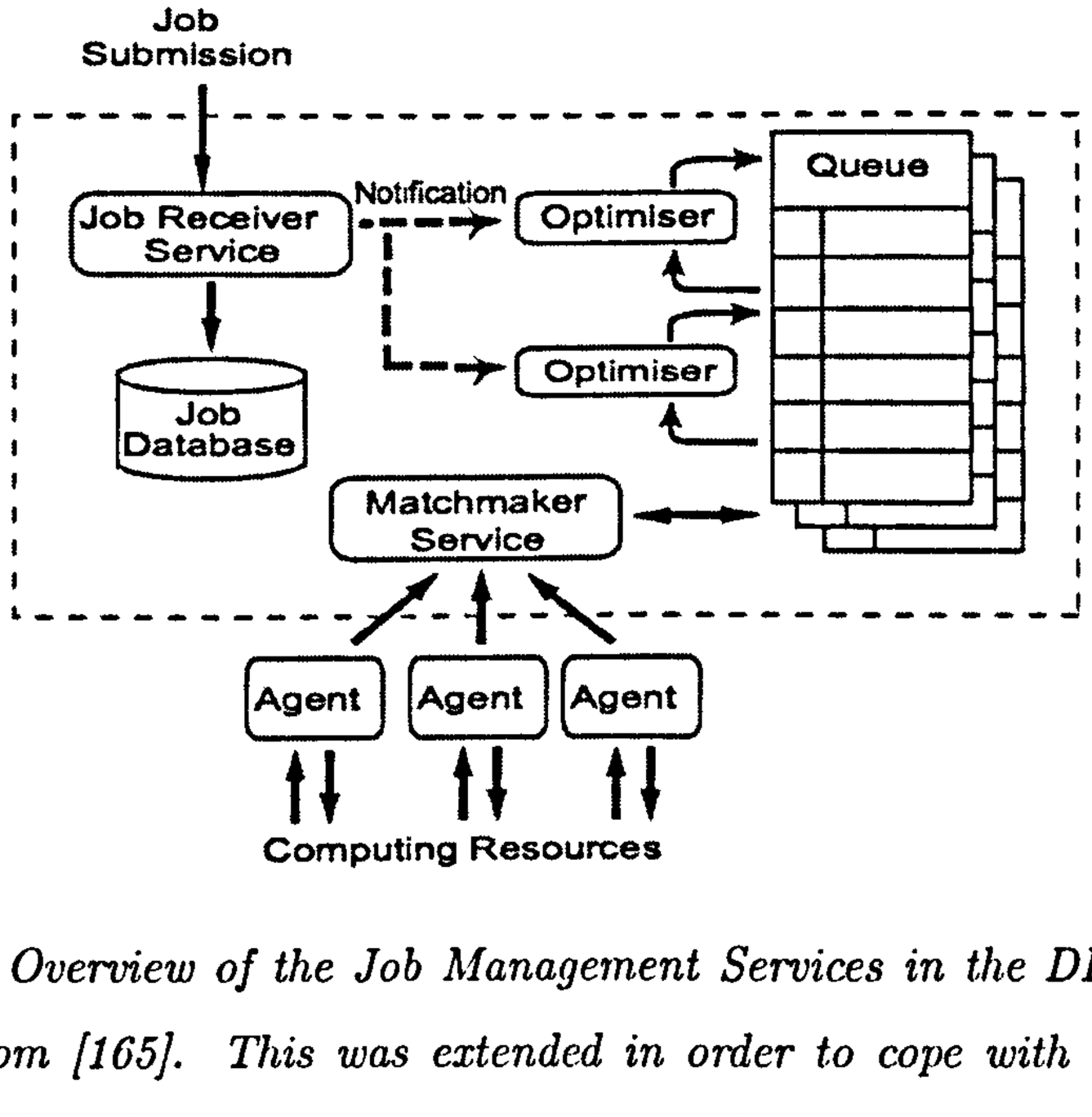

Figure 4.6: *Overview of the Job Management Services in the DIRAC Production WMS from [165]. This was extended in order to cope with the increasing requirements of LHCb distributed data analysis jobs. The current WMS is shown in Figure 5.3.*

Figure 4.6 from [165], highlights the Job Management Services in the DIRAC Production WMS before the extensions to support distributed data

analysis. The infrastructure in Figure 4.6 was deployed for a generic user, namely the Production Manager, on behalf of LHCb. This section will present an overview of components in the DIRAC WMS as well as the recent developments made to support the distributed data analysis activity. The services are secured via DISET as described in Section 4.3.1.

Job Receiver

The DIRAC *Job Receiver* assigns the Job ID, saves the job in the *Job Database* and also uploads and saves the proxy of the user. An *Optimiser* is notified in order to proceed to submission, depending on requirements of job. This communication takes place via Jabber and will be explored in Section 4.5.4.

For the distributed analysis tasks, the most significant change in this service is the introduction of security (via DISET). This allowed explicit use of Grid proxies for authentication as well as further use in the Workload Managment process.

Job Database

The *Job Database* interface is a thin layer on top of a set of SQL statements. This interface also performs high-level operations such as adding jobs, removing jobs and bulk queries (*e.g.* for job monitoring). When jobs have been successfully added, the JobDB changes their status to acknowledge this, allowing further services to begin their tasks.

The JobDB was recently optimised to cope with the high demand for job monitoring, data retrieval, and also to contend with the expected increase in the number of users performing distributed data analysis.

Optimisers

The purpose of *Optimisers* in DIRAC are to allocate jobs to queues, sorting them according to their requirements. The JobDB is used to retrieve the requirements of the jobs.

The *Optimiser FIFO* (First In First Out) handles jobs without any input data requirements, such as production jobs, and inserts them into a *Task Queue* according to the order in which they were submitted.

For jobs with input data requirements, such as distributed analysis or stripping jobs, the *Data Optimiser* checks the availability of all input data files in the file catalogue. This can result in a meaningful failure if not all the data is present at a site. If successful, the job is inserted into a *Task Queue* along with a list of possible Storage Elements for job execution. DIRAC Storage Elements will be described in Section 4.6.1. There is currently no prioritisation policy in place for the submission of LHCb jobs, this will be further discussed in Chapter 6.

Developments to the *Data Optimiser* were made to allow the use of the LFC, see Section 4.6.2, in a secure way via a server certificate running on the host machine. In cooperation with the LFC developers, optimisations were made for bulk requests.

Task Queues

There are many *Task Queues* in DIRAC. In fact, there is one per set of job requirements. This serves to drastically reduce the matching time for jobs with similar requirements and has been demonstrated to be very effective for production jobs, as described in Section 3.1.2.

Of course, too many queues can cause scheduling problems and this becomes important for distributed analysis jobs. A hierarchical organisation of

queues with respect to requirements was adopted to improve the matching.

Matcher

The *Matcher* service receives requests from Agents, checks available jobs in the *Task Queues* and makes a decision based on matching the job requirements with those presented by the Agent. This works via the double matching mechanism, introduced in Section 3.3.1, which was put in place for analysis jobs with many varied requirements. In Chapter 6, the results of these developments will be explored with real user jobs.

The *Matcher* only responds to sites in a mask that contains the list of allowed sites. The mask is managed by an administrator and makes it possible to temporarily ban problematic sites whilst also serving as a security feature. To ensure that jobs are only picked up once, the *Matcher* has a 'semaphore' mechanism in place when scheduling jobs to sites.

After the *Matcher* has scheduled a job, the status is updated and information about the site is logged. The job is deleted from the *Task Queue* and sent to the resource.

Sandbox Services

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 the applications. These files are collectively referred to as the input sandbox. Likewise, the term output sandbox refers to similarly small output files of a job, *e.g.* application log files, which do not require permanent storage on the Grid.

At present, the DIRAC MySQL database is used for storing the input and output sandboxes. To date this has been very fast and efficient with no

problems observed for small files. Larger output files are sent to permanent Grid Storage. Policy decisions had to be made in the context of user jobs, *e.g.* what to do when a specified output file is too large to be returned via the *Sandbox* services. This is discussed in Chapter 6.

Agent Director and Agent Monitor

In Figure 4.6, the submission of Pilot Agents took place via an automated 'cron-job'. This was configured, initiated and maintained manually. In order to minimise human intervention for distributed analysis jobs and to speed up the submission to the Grid, the *Agent Director* and *Agent Monitor* services were created. These have been implemented specifically for LCG, although they have been designed to be easily adaptable to other Grids.

The *Agent Director* is an API for Pilot Agent submission to LCG. Pilot Agents are sent as LCG jobs which first install DIRAC and then run an Agent. The *Agent Director* uses the proxy of the user for submission to LCG and can submit Pilot Agents for each job in the *Task Queue.*

The *Agent Monitor* is used to keep track of Pilot Agents submitted by the *Agent Director.* Using a configurable time interval, the *Agent Monitor* checks the status of the Pilot Agents and flags the jobs for the *Agent Director* to submit further Pilot Agents as necessary. This is useful for preventing unnecessary delays to jobs, such as when Pilot Agents become stuck in long batch queues. The *Agent Monitor* is also essential for ensuring the resubmission of Pilot Agents in case of failures.

The central deployment of Pilot Agents on demand from the DIRAC WMS has interesting repercussions for the distributed analysis jobs. This will be discussed in Chapter 5. The *Agent Director* and *Agent Monitor* are now the default mode of submission to LCG for all DIRAC jobs.

4.5.4 Instant Messaging in DIRAC

Jabber has been successfully demonstrated for communication between DIRAC services [159]. However, it's use has been limited to only one case thus far. This is where the *Job Receiver* uses Jabber to notify the *Optimisers* when a new job arrives. Based on recommendations in [149], this may be dropped in favour of an XML-RPC messaging system. This alternative would require some development but could naturally include the DISET security infrastructure.

Job interactivity, allowing job 'spying' and remote job killing is an attractive prospect which has been demonstrated using Jabber [159]. Unfortunately, however, this is pending until a secure Jabber connection or alternative messaging system becomes available.

4.6 Data Management

The advent of the Grid has led to a necessary revolution in the treatment of data, as described in Chapter 1. The Data Management System in DIRAC consist of the following main components: Storage Element; File Catalogue; and Replica Manager. These will be discussed in turn below.

4.6.1 Storage Element

The DIRAC Storage Element is an abstraction of the plethora of storage resources available to the system. The aim is to determine which protocols are available on a particular resource and ensure these protocols are used in an efficient manner.

To this end, the DIRAC Storage Element uses a description in the Configuration Service, defined in Section 4.3.4, to obtain the list of available

protocols at a given site. Subsequent use of the named protocols relies on plug-in modules, which represent various mechanisms of data access. The list of available plug-ins includes: FILE, RFIO, FTP, SFTP, HTTP, BBFTP, SRM and XMLRPC [166]. It also provides functionality similar to SRM for protocol (TURL) resolution. This is important for providing access to specified input data which will be discussed in Chapter 5.

4.6.2 File Catalogue

The file catalogue plays a vital role in the DIRAC system: for production jobs it is essential to store data in an efficient and easily accessible way; for distributed analysis jobs, it is necessary to efficiently access this data without knowing in advance where the job is running.

When DIRAC was first developed as a production system there was no obvious implementation available and so the system was designed to cope with multiple file catalogues being used in a transparent way. By creating a generic File Catalogue Client API for all File Catalogue services, each file catalogue can be used interchangeably.

During the history of DIRAC, the following File Catalogues were incorporated [167]:

- LHCb Bookkeeping File Catalogue [151]
- AliEn File Catalog [132]
- LFC [77].

Out of the three catalogues, the LFC has been retained as the main catalogue for LHCb. After exploring the LHCb Bookkeeping File Catalogue it was decided that not all the necessary features were available, one example being support for a hierarchical structure of entries. Nevertheless, the

Bookkeeping Catalogue is still in use during production for redundancy and reliability.

The AliEn File Catalogue was used during the DC04 activity and was proven to work in a production environment. The AliEn shell was used to create a binding in Python for DIRAC. AliEn was originally used to explore use cases not provided by the Bookkeeping Catalogue but has since been retired and replaced by the LFC.

The LFC was chosen since it provides all of the functionality necessary for LHCb, after optimisation in close collaboration with the developers. A Python binding to the LFC is shipped with the LCG middleware and this is used to implement the API for DIRAC. It is planned to have one global instance of the LFC catalogue with several read-only mirrors for redundancy and load balancing.

4.6.3 Replica Manager

The DIRAC Replica Manager (RM) implements methods for the manipulation of files on the Grid such as, `get()`, `copy()`, `replicate()` and `register()`. The DIRAC CS provides a list of active File Catalogues and these are used for any requested operations. The RM will always choose the 'best' replica, meaning the closest available replica at the moment of access, using the preferred protocol. All operations performed on the data are logged to provide a record for debugging.

The DIRAC RM has been used with all of the file catalogues described in Section 4.6.2 and provides interfaces to all of the data management clients. Figure 4.7 illustrates how the DIRAC Data Management components interact.

The Transfer Agent, shown in Figure 4.5, is one possible data manage-

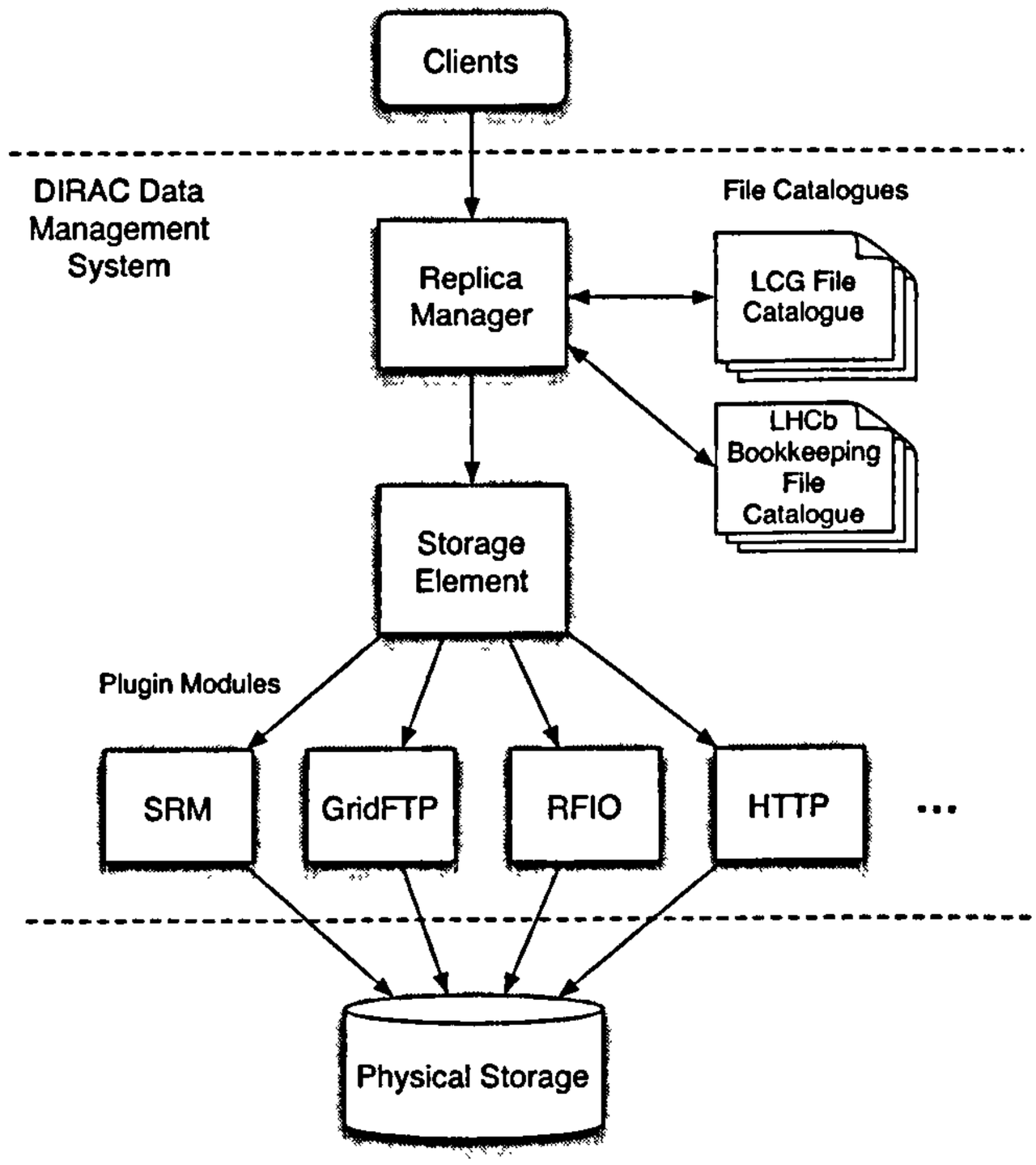

Figure 4.7: *Overview of the DIRAC Data Management System highlighting how the main components (Storage Element, File Catalogue and Replica Manager) interact.*

ment client. Another is the WMS, in the form of the *Job Wrapper*, which uses data management components to provide access to specified input data. This will be explored in Section 5.3.1.

4.7 Information, Monitoring and Accounting

The DIRAC CS was introduced in Section 4.3.4 and forms the basis of the information system for DIRAC. It is reliable due to servers running in several locations and is 100% available. The DIRAC CS is used in many contexts such as: delivering site specific information to the Job Agents on WNs; pro-

viding the RM with the list of available File Catalogues; and providing the Storage Element with the list of available protocols.

The *Job Monitoring Service* is used at all stages of the lifetime of DIRAC jobs to update status information and is one of the most solicited DIRAC services. This changes job states in the JobDB directly and also updates logging information in order to provide a complete history for each job. There are two entry points to the *Job Monitoring Service*. The first is secure, for writing, and the second is used for reading. Clients such as users through the DIRAC API or WMS services such as the *Agent Monitor* interact with the *Job Monitoring Service*. Information from the *Job Monitoring Service* is also used to construct the DIRAC Monitoring Web Interface [168].

After the completion of each DIRAC job, a report is sent to the Accounting Service. This receives accounting information for each job and automatically generates reports based on criteria such as: specific productions, having a unique identifier; or different user groups, as defined in VOMS. Reports can also be generated for a specified time period or a particular site. A visual representation of these reports is published on a dedicated web page, in a similar way as the DIRAC Monitoring Web Interface [168].

4.8 Summary

The purpose of this chapter was to introduce the DIRAC system. This began with an overview of the history of DIRAC and the principles of design in Section 4.1. This highlighted the main components of DIRAC (Clients, Services, Resources and Agents). The implementation and software tools used in DIRAC were introduced in Section 4.2.

The Services Framework was described in Section 4.3 which introduced

DISET, the DIRAC security mechanism. Services in DIRAC are designed to be easily deployable, reliable and secure. As an example, the Configuration Service was discussed in Section 4.3.4.

In Section 4.4, the Agents Framework was explored. This introduced the two main types of Agent: Site Agents and Pilot Agents, differing due to their methods of deployment. Site Agents have many possible DIRAC CEs to cope with the many different batch systems in use. Pilot Agents are used on the Grid and run jobs local to the Agent rather than managing a cluster of nodes.

An overview of the WMS was given in Section 4.5, which described the evolution and introduction of services to cope with the increasing requirements of LHCb distributed data analysis tasks. The DIRAC system was previously used almost exclusively by the Production Manager for production tasks but has now been made secure and capable of supporting many users.

The Data Management Service was described in Section 4.6, highlighting how the Replica Manager provides seamless access to multiple file catalogues, with the LFC being retained as the main catalogue for LHCb. The SE in DIRAC was shown to provide access to physical storage devices through several protocol plugin modules.

The Information, Monitoring and Accounting systems were discussed in Section 4.7, which elaborated on how the CS, *Job Monitoring Service* and Accounting Service are used throughout the lifetime of jobs in DIRAC. The CS provides information enabling jobs to access other services, for example, determining the protocols supported by a local storage element.

The next chapter explores how the DIRAC infrastructure can be used most effectively for LHCb distributed data analysis, focussing on the possible

workload management optimisation strategies. DIRAC will also be compared to other systems, such as Condor and Condor-G, which share many of the principles on which it is based.

Chapter 5

DIRAC Workload Management

Having introduced the DIRAC system in the last chapter, this chapter will explore possible workload management optimisation strategies, which result in a high efficiency for LHCb user jobs. The DIRAC infrastructure for distributed analysis has been developed based on a successful production system. Exposing the functionality of DIRAC to enable the construction of user jobs is discussed in Section 5.2.

The workflow of DIRAC jobs will be described in Section 5.3, explaining the role of each WMS component during the lifetime of user jobs, from submission to completion. This will include a detailed description of how DIRAC provides access to input datasets, which is essential to the success of distributed analysis jobs.

Section 5.4 will present several optimisation strategies with the DIRAC infrastructure. Since similar advances cannot be made with the available LCG tools, these are DIRAC optimisations. Following this is a description of how these strategies can be applied to maximise the usage of resources for LHCb. Results are presented from implementing the strategies on LCG in Section 5.4.5, with a comparison to a recent simulation study in Section 5.5.

The Condor and Condor-G systems provide similar functionality to DIRAC and a detailed comparison will be made in Section 5.6. A description of how implementations of DIRAC paradigms are used in other CERN experiments will be given in Section 5.7, with special emphasis on the ATLAS Panda system.

5.1 Introduction

The DIRAC software architecture is based on a set of distributed, collaborating services, as described in the last chapter. Designed to have a light implementation, DIRAC is easy to deploy, configure and maintain on a variety of platforms. Using the software distribution mechanism introduced in Section 2.5.4, DIRAC can run LHCb jobs on all available LCG resources.

In Chapter 3, the paradigms for distributed analysis were explored. Through the use of the *PULL* scheduling paradigm and the creation of an Overlay Network of Agents, the DIRAC WMS provides the infrastructure to submit and run jobs on the Grid in a seamless way. Pilot Agents submitted to LCG request jobs whenever the corresponding resource is free. The WMS ensures that not only the requirements of the jobs are satisfied, but also the requirements of the resource in a 'double matching' mechanism.

Users submit jobs via the DIRAC API which will be described in the next chapter. In the next section, the constituent parts and structure of jobs in DIRAC are discussed. As a production system, the job framework in DIRAC is capable of building very complicated workflows. However, user jobs do not require the same level of complexity. For example, user jobs do not have to report to the bookkeeping database, although this is essential for production jobs, which require results to be centrally managed.

5.2 Jobs in DIRAC

The DIRAC API provides the interface for users to submit jobs to DIRAC. The specifics of this will be left to the next chapter and here the focus is on the functionality which the DIRAC API encapsulates.

Jobs in DIRAC are composed of three classes: `Job()`; `Step()` and `Module()`. Figure 5.1 illustrates how objects of these classes are related.

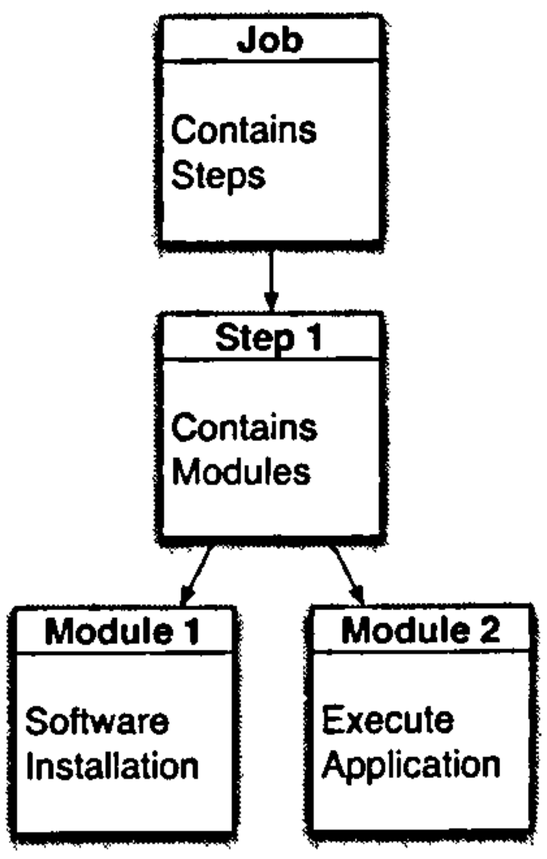

Figure 5.1: *Jobs in DIRAC are composed of Steps which in turn are composed of Modules. In principle, any workflow (DAG) can be created using this architecture.*

Jobs can be thought of abstractly as a set of complex operations. In DIRAC, the main purpose of the Job class is to contain Steps. A Step is defined as the smallest unit that can be executed to produce output files, assuming the necessary input files are available. Likewise, the main purpose of the Step class is to contain Modules.

Modules are smaller operations that can be tailored to perform a desired function. In Figure 5.1, for example, two Modules form the Job. The first Module installs any required software and the second executes the desired application. Modules are reusable components that can be linked with each other. Therefore, if the software installation module in Figure 5.1 were to

fail, running the next Module to execute the application can be prevented.

Jobs may contain many Steps, each of which can execute different applications. Steps may depend on each other in a complicated manner and are composed of Modules. Using these three classes as building blocks, any topology of Steps can be created. Therefore, DIRAC Jobs can be thought of as a Directed Acyclic Graph (DAG). The next section will explore the construction of more complicated job workflows.

5.2.1 Creating Complicated Job Workflows for Users

The Production Console [152] uses the DIRAC Job, Step and Module infrastructure in order to create the complex workflows for production, reprocessing and stripping jobs. Workflows are created locally, converted to an XML job description and sent to the WMS. Each workflow is made up of Steps, which can be connected to each other via input and output files.

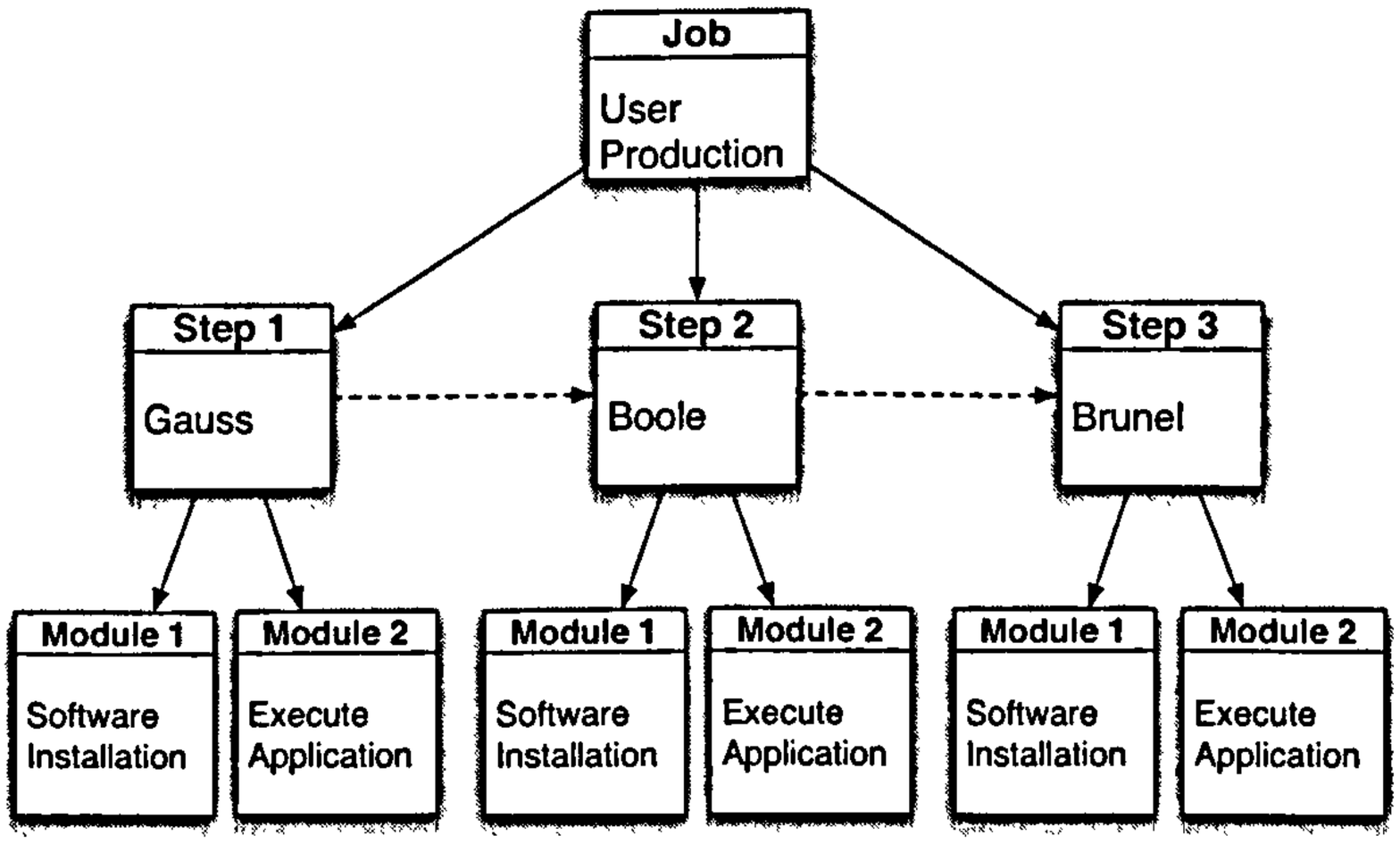

Figure 5.2: *Structure of a multi-step job to run Gauss, Boole and Brunel using the DIRAC API. Dashed lines indicate the processing chain.*

The DIRAC API exposes similar functionality to the Production Console,

but is tailored for user jobs. In order for users to create their own workflows, it was decided to open the functionality of DIRAC Steps. This allows users to perform more complicated distributed analysis tasks, or private production jobs.

By exposing functionality at the Step-level, user jobs can be created to almost any specification. Figure 5.2 illustrates the structure of a typical user production job. The DIRAC API provides users with custom Modules to facilitate the construction of the workflow shown in Figure 5.2 (where dashed lines indicating the processing chain). This example involves three steps: Monte Carlo Simulation using Gauss; digitisation with Boole; and finally reconstruction with Brunel. Appendix B contains a script that generates this structure. The DIRAC API will be discussed in more detail in Chapter 6.

5.3 Workflow of Jobs

This section will describe the workflow of DIRAC jobs submitted via the DIRAC API. There are two cases to consider: firstly, a typical user analysis job with specified input data; and secondly, a user production job having no input data requirement. This is an important distinction because the LHCb Computing Model [100], discussed in Section 2.3, involves sending jobs to the data without explicitly choosing the site in advance. This upholds the paradigms of the Grid, introduced in Chapter 1, in which the main priority for a user is that a particular job is successfully run, without needing to know where the job has run. Distributed analysis jobs have a higher priority with respect to other tasks, so it is imperative to minimise the start time of these jobs.

User jobs are submitted to the WMS through the DIRAC API securely. This is via the DISET [157] security infrastructure, described in the last chapter. The *Job Receiver* service assigns a Job ID and saves the Job in the *Job Database* along with the proxy of the user. At this point, if an existing proxy with a longer lifetime is present in the system, this is retained. Otherwise, the new proxy is saved and made available to existing jobs in the system from the same user. Therefore, any previously submitted jobs in the waiting state with an expired proxy are now able to run. The implications of proxy expiration and possible strategies for proxy renewal will be discussed in Chapter 6.

During the submission process, the *Sandbox* services ensure the upload of any input files to steer the application. Figure 5.3 shows the DIRAC central WMS services and interactions with LCG components. The *Job Receiver* then notifies an *Optimiser.*

For jobs that do not have any input data requirement, this is the *Optimiser FIFO.* The same optimiser is used for production jobs, and corresponds to a First In First Out policy. The *Optimiser FIFO* then inserts the Job into a *Task Queue*, see Section 4.5.3.

For jobs with specified input data, the *Data Optimiser* is notified and this proceeds to query the LFC for specified input data files. If not all the files are available, this is the first possible point of job failure. Reasons for this include:

- Files have been specified incorrectly;
- There is a problem with one or more replicas of specified files in the catalogue; and
- Files are specified correctly, but not present in the LFC.

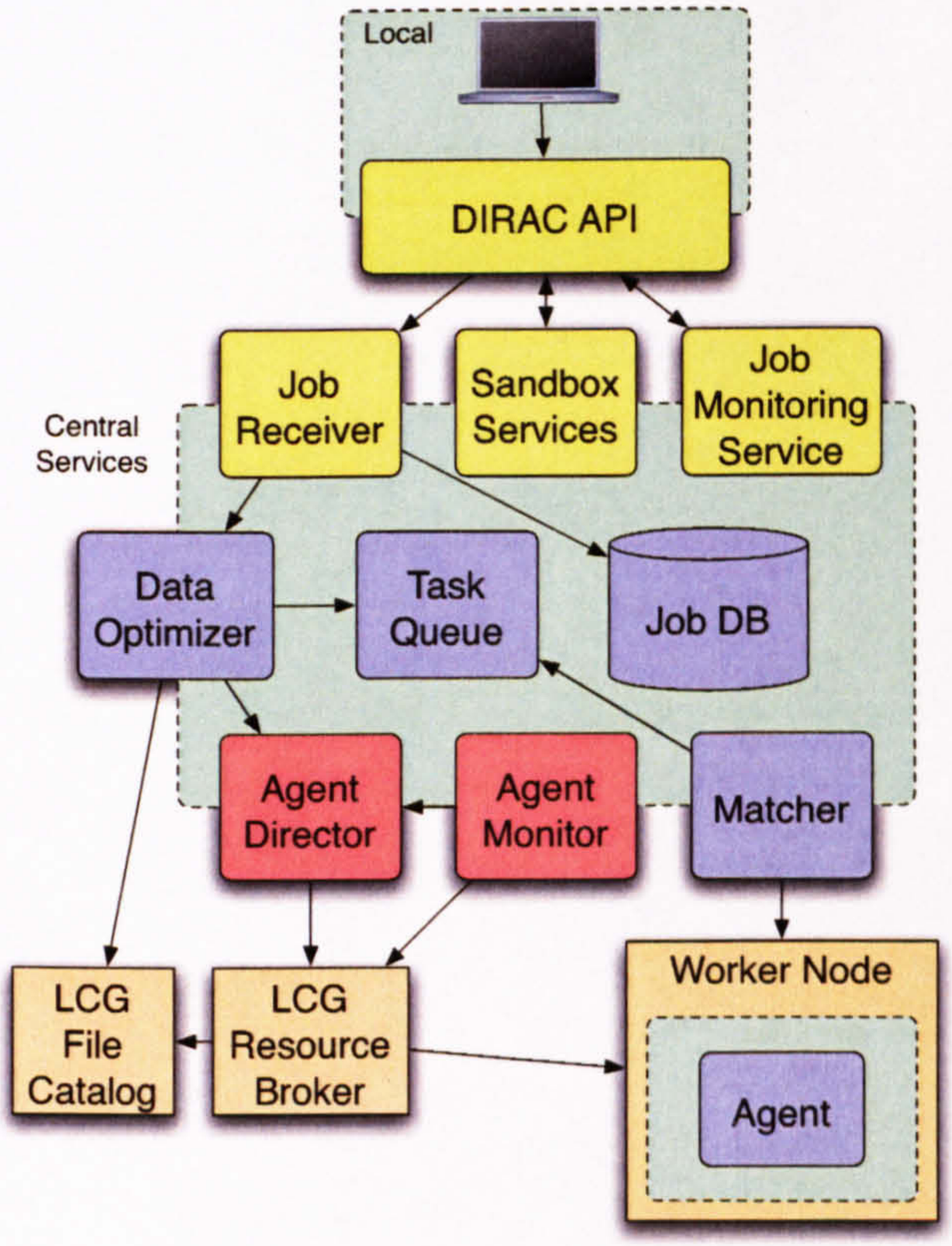

Figure 5.3: *Outline of the DIRAC Workload Management System including the extensions made for LHCb distributed data analysis tasks.*

The LHCb Data Manager is currently required to add the missing files, and correct problematic entries, in order for affected jobs to proceed. This type of failure accounts for some of the user experience described in Chapter 6.

If all the files are available, the result of querying the LFC is a list of one or more suitable Grid Storage Elements (SEs). This is entered into the requirements of the job to be used for matchmaking. The *Data Optimiser* will then insert the Job into a *Task Queue*.

Once a job enters the *Task Queue*, it is ready to be scheduled from the perspective of the WMS. At this point the *Agent Director* sends a Pilot Agent

to LCG using the requirements of the job. The *Agent Monitor* checks the status of the Pilot Agent, which amounts to monitoring a standard LCG job, and triggers resubmission as required.

When a Pilot Agent successfully reaches a Worker Node (WN) it installs DIRAC and runs an Agent, which requests a job from a particular user. In fact, the Agent starts the Job Agent Module, which performs the job request. The *Matcher* service matches the requirements of jobs, such as possible SEs, to the properties of the computing resource presented by the Agent. Since the Agent can also put specific requirements on jobs, this is a 'double match' procedure. Figure 5.4 illustrates 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 as described in Section 2.5.4. Links to any available pre-installed software are created local to the job during the installation of DIRAC, see Section 3.3.2.

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 via the *Sandbox* service, and provides access to the input data. The LFNs are resolved into 'best replica' PFNs (SURLs) for the execution site, see Section 5.3.1.

The job application is then invoked in a child process and a *Watchdog* process is started in parallel to the application. The *Watchdog* process 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. At all stages, the *Job Monitoring Service* is used as an interface to update the Job

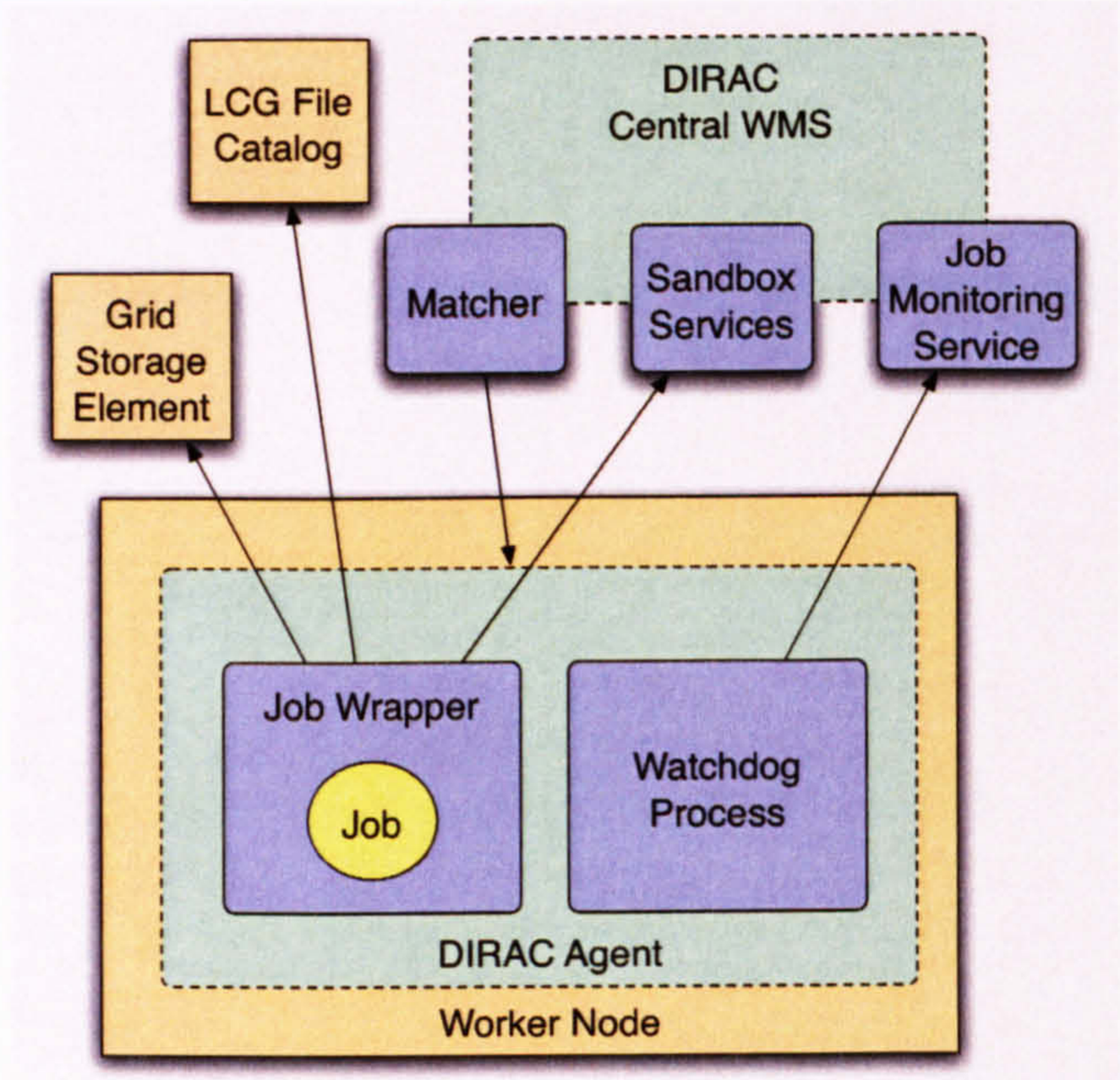

Figure 5.4: *DIRAC Workload Management on the Worker Node.*

information.

After the Job has finished, the *Job Wrapper* also handles the upload of the output sandbox using the *Sandbox* service. The sandbox is stored in the *Job Database*. Any specified output data will be uploaded to a predefined SE at this point. The *Job Wrapper* may also receive messages through Jabber, that has been demonstrated in [159], and discussed in Section 4.5.4. Once the DIRAC Agent has finished, the Pilot Agent terminates gracefully, thus freeing the LCG resource.

If a Pilot Agent does not successfully retrieve a job after a request to the WMS, it again terminates gracefully to release the LCG resource. This reduces the overhead associated with the use of Pilot Agents, which will be further examined in Section 5.4. The reasons for a Pilot Agent not receiving a job include: not having any waiting jobs in the WMS to pick up; and

scheduling failures, where the requirements of the resource do not match those of the job.

The next section will describe how the *Job Wrapper* resolves input data on the WN.

5.3.1 Providing Access to Input Data

For distributed data analysis jobs, it is essential that reliable access to input datasets is possible on all available Grid sites. As discussed in Section 2.3.2, LHCb distributed analysis jobs will mainly be run at the Tier-1 sites, with data being available via disk. This does not exclude the possibility that other sites, *e.g.* Tier-2 centres, maintain replicas of data that would also be accessible for LHCb distributed analysis jobs.

The Tier-1 centres available to LHCb collectively encompass heterogeneous resources with different mechanisms for accessing data. The two most common access protocols are RFIO (Remote File Input/Output) [148], and DCAP (Data Link Switching Client Access Protocol) [169]. In order to realise the paradigm of the Grid, both must be utilised in a seamless way without prior knowledge of where a job will run.

The first approach taken for distributed analysis jobs was to download all datasets local to the job on the Grid WN, using the *Job Wrapper*. The application would be executed when all datasets were available. Whilst being a reliable means to provide access to input data, this was also impractical. Increased network overheads and limited space on the Grid WNs caused failures with this approach. The method currently in place for accessing data stems from the desire to have a working system whilst also being able to incorporate any new LCG developments in a simple manner.

In the *Job Wrapper*, the first step is to determine the local SEs for the site

at which it is running. This information is obtained from the DIRAC CS. It is important to note that there can be more than one SE for any particular site. The local SEs are used to find the 'best' replica for each requested input file. This corresponds to a replica that is accessible through supported protocols (currently RFIO or DCAP). SRM is the standard interface to storage for LHCb [100], and the LFC is queried for replicas at local SEs. This returns a list of SURLs.

At this point, the DIRAC SE class is used to automatically generate a global TURL for each SURL. This is a temporary solution until the LCG tools provide other reliable means to determine the TURLs. However, the current mechanism works well for Gaudi-based applications and has resulted in successful use of RFIO and DCAP to access files. Currently, any protocols supported by POOL (Pool Of persistent Objects for LHC) [170] can be used, although in the absence of these any affected datasets are brought local to the job before execution.

An attempt was made to use '`lcg-gt`' (a component of the LCG Grid middleware which utilises SRM) to stage specified input data files, with the resulting TURLs being used directly. This has been put on hold since the returned TURLs did not work inside the applications. Also, the functionality to 'pin/unpin' files was not available, this would ensure the persistency of data files on disk. Therefore, if the returned TURLs did work, it would be impossible to determine their period of validity.

The last step is to generate a POOL XML slice for the Gaudi applications to be able to resolve the input datasets. The POOL XML Catalog is implemented in DIRAC with a subset of the standard File Catalog Client API described in Section 4.6.2. The *Job Wrapper* on the WN uses the Globally Unique Identifier (GUID) from the LFC and global TURL for each dataset

to construct the slice. This is subsequently added to the POOL XML File Catalogue using the GUID, LFN and PFN. After this, the POOL XML slice is exported as an XML file. The final step is to append the XML slice to the options file of the user. This is done automatically before the application starts to execute.

With the infrastructure described above, it is possible to run LHCb distributed data analysis jobs on the Grid. Another mechanism may be adopted in the near future, which will be mentioned in Chapter 6. The possible optimisation strategies for using the DIRAC system will be explored in the next section.

5.4 Optimisation Strategies

There are several ways to use the DIRAC infrastructure but the end goal is to minimise the start time of user analysis jobs whilst ensuring a high efficiency. The optimisation strategies explored in this section stem from two developments. Firstly, the *Agent Director* and *Agent Monitor* services may be used to define a policy on how Pilot Agents are submitted. Secondly, the DIRAC Agent can be chosen to affect the mechanism by which jobs are picked up from the WMS. As a result, it is possible to define modes of submission 'tuned' for the needs of specific jobs:

- Resubmission;
- Filling; and
- Multi-Threaded Filling.

These shall be explored individually below. It is important to note that these are DIRAC optimisations and not possible with the standard LCG tools.

LHCb has several different 'types' of jobs, with each having different priorities and requirements. For example, a standard production job typically lasts for one day, whereas an analysis job could be relatively short, *e.g.* under one hour. Since production jobs take significantly longer, they also capture a compute resource for longer, reducing the amount of available resources for LHCb. Therefore, part of the ethos behind the DIRAC optimisations described below is to maximise the usage of a resource, once it has been captured.

Whilst these modes of submission would be most effective through the optimisation of all available LHCb jobs, to avoid any violation of LCG security rules, the following optimisations must currently be performed at the level of the user. In terms of workload management it will be shown that this can be restrictive, see Section 5.4.5.

5.4.1 Resubmission

Resubmission mode means that the *Agent Director* is deployed to submit one Pilot Agent that is tracked through the *Agent Monitor*. If a failure to this Pilot Agent occurs, indicated by the LCG IS, the *Agent Monitor* will trigger the submission of an additional Pilot Agent through the *Agent Director*.

This minimises the risk of jobs failing to start but does not optimise the start time to a significant degree. For example, there is still a risk that the single Pilot Agent originally submitted could enter a site batch queue and wait for many hours.

After the Pilot Agent reaches the WN, it first installs DIRAC. After the local execution environment has been checked, a request is made to the WMS to retrieve a job. For Resubmission mode, one request is made after which the Pilot Agent will terminate gracefully.

As mentioned above, a typical LHCb Monte Carlo Production job lasts for approximately one day. The start time for this type of job is not as high a priority when compared to other activities such as distributed analysis. However, it is important that the job does start. In this case, the Resubmission mode is sufficient and has the advantage of not placing load on the Grid by activating the submission of extra Pilot Agents unnecessarily.

5.4.2 Filling

The Filling mode is similar to Resubmission since the *Agent Director* is deployed to submit one Pilot Agent, which is tracked through the *Agent Monitor*. If a failure occurs, the *Agent Monitor* will trigger resubmission of an agent through the *Agent Director*. The difference with submission in the Filling mode is that multiple Pilot Agents can be sent up to a configurable maximum. Jobs remain waiting in the WMS for a configurable time period, before triggering the resubmission. After the maximum number of Pilot Agents has been reached, no additional Agents are sent. However, the configurable maximum value does not include Pilot Agents which fail. Resubmission is triggered for those Agents.

Filling mode allows Agents to request several jobs from the same user, only requesting a new job once the current one has finished. In this regime, the Agent 'fills' the computing slot allocated by LCG and therefore maximises the resource usage. The Filling Mode can be implemented in two ways: the DIRAC Pilot Agent can be configured to run several 'one-shot' Agents, with each making one request to the WMS; or one Agent can be started that continues to make requests at regular intervals. The former will be explored here since it is less intrusive, *i.e.* resources are only held when a job is running. The latter is normally used for Site Agents, which generally

represent resources outside of the Grid with high availability.

With additional Pilot Agents sent, there is a higher chance that jobs will start promptly. For example, Pilot Agents can be scheduled to different sites and avoid batch queues. The additional Agents may also pick up subsequent user jobs in Filling mode. In this case, subsequent jobs could potentially run without the submission of any Agents.

The advantage of sending multiple Pilot Agents is that more requests to the WMS are made when resources have been captured. Each request means that a waiting job can be delivered for immediate execution on the WN. This can significantly minimise the start time of jobs.

Filling mode is most useful for high priority tasks, such as distributed data analysis jobs, and has less relevance for production jobs which have a lower priority and can fill up an allocated slot themselves. It was decided to make Filling mode the default mode of operation for the DIRAC Analysis system, which will be described in Chapter 6.

5.4.3 Multi-Threaded Filling

When a DIRAC Agent is started, there is a choice of possible Computing Elements to choose from, which determine the behaviour of the Agent, as described in Section 4.4. For example, an 'InProcess' Agent will request one job at a time on a particular site. A new 'Threaded' Computing Element was created that allows multiple jobs to run simultaneously on a computing resource. When a job arrives at a site, the Threaded Agent checks how many jobs are currently running. If this is less than the defined maximum, the Threaded Agent starts executing the job in a new thread. When the maximum is reached, no more jobs are requested until a running thread finishes.

The Multi-Threaded Filling mode represents exploratory work based on the assumption that, whereas production jobs are CPU intensive, distributed analysis jobs are Input/Output (I/O) bound and therefore do not utilise the full power of the CPU at all times during execution.

In a similar way to the Filling mode, Multi-Threaded Filling involves submission of multiple Agents using the *Agent Director*, with jobs waiting in the WMS for a configurable time period before the *Agent Monitor* triggers resubmission. The Multi-Threaded Filling mode also allows Agents to request several jobs from the same user but the difference here is that multiple jobs can run at the same time on one WN. To allow for the many different types of resources on the Grid and constraints such as available memory, it was decided to limit this number to two jobs running in parallel on the WN.

Multi-Threaded Filling is principally useful for the high priority distributed analysis activity and serves to greatly reduce the start time of jobs. In this regime, every Pilot Agent successfully reaching a WN requests up to two jobs to run in parallel. Requests are made at regular intervals if only one job is picked up initially. This also fills the computing slot by running several 'one-shot' Threaded Agents in a similar way to the Filling mode.

The real benefit of using a Threaded Agent is that the available resource can be used extensively. Due to the unstable nature of running jobs on the Grid, it is imperative to maximise the usage of a resource once it has been obtained. The Pilot Agent mechanism allows effective resource discovery and ensures any requested data is accessible for any particular job. The next section will describe the assumptions and precautions taken for testing the optimisation strategies introduced above on LCG.

5.4.4 Testing Framework

Measuring performance on the Grid is not an exact science, many external factors can affect how jobs run. Some examples include:

- Load on the Grid, *e.g.* other experiment activities such as concurrent production phases;
- Site availability, *e.g.* 'draining' occurs before maintenance operations, this prevents new jobs being accepted and reduces the total number of available nodes;
- Site configuration problems resulting in job failures;
- High load on the Resource Brokers resulting in significant time lag when submitting jobs;
- Time of submission, *e.g.* the response is slower during peak periods of load on the system; and
- Events affecting critical resources, *e.g.* power cuts and network outages.

Therefore, to tackle the general Grid 'weather', the following precautions were taken for the following performance study. Firstly, jobs were submitted at the pace of the Resource Broker so that waiting times were not artificially skewed, and job start times were measured relative to the submission time to DIRAC. Without pacing the submission of the jobs to DIRAC, the distribution of start times is dominated by the time taken to submit all the jobs through the LCG Resource Broker. This amounted to approximately 5 seconds per job between submissions.

Secondly, to ensure similar conditions for each experiment, multiple users submit jobs in turn, with each user submitting with a different mode as

described above. With analysis jobs being chaotic in nature, testing on the real system was carried out with each user submitting jobs using the same algorithm and the same number of datasets. This ensures a fair comparison between the different modes of submission. With each user submiting jobs in turn, any temporary problems on the Grid affect each user in the same way.

For testing the WMS, it was decided to create a workload that places the highest load on the system. Since DIRAC has been proven to cope with long Production jobs, a study of the various DIRAC modes of submission was performed using short analysis jobs. This serves to test the other extreme whilst also placing a higher load on the WMS.

5.4.5 Results and Performance

The results presented here based on a data sample of ten distinct experiments of three users submitting one hundred jobs for each mode, with three thousand jobs submitted in total.

Figure 5.5 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 effectively 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. With each job in Resubmission mode requiring to be scheduled through the LCG RB, it is clear why delays occur. The RB must perform complex calculations in order to schedule jobs to sites in the *PUSH* approach. However, not all jobs in the Filling and Multi-Threaded modes must go through this procedure.

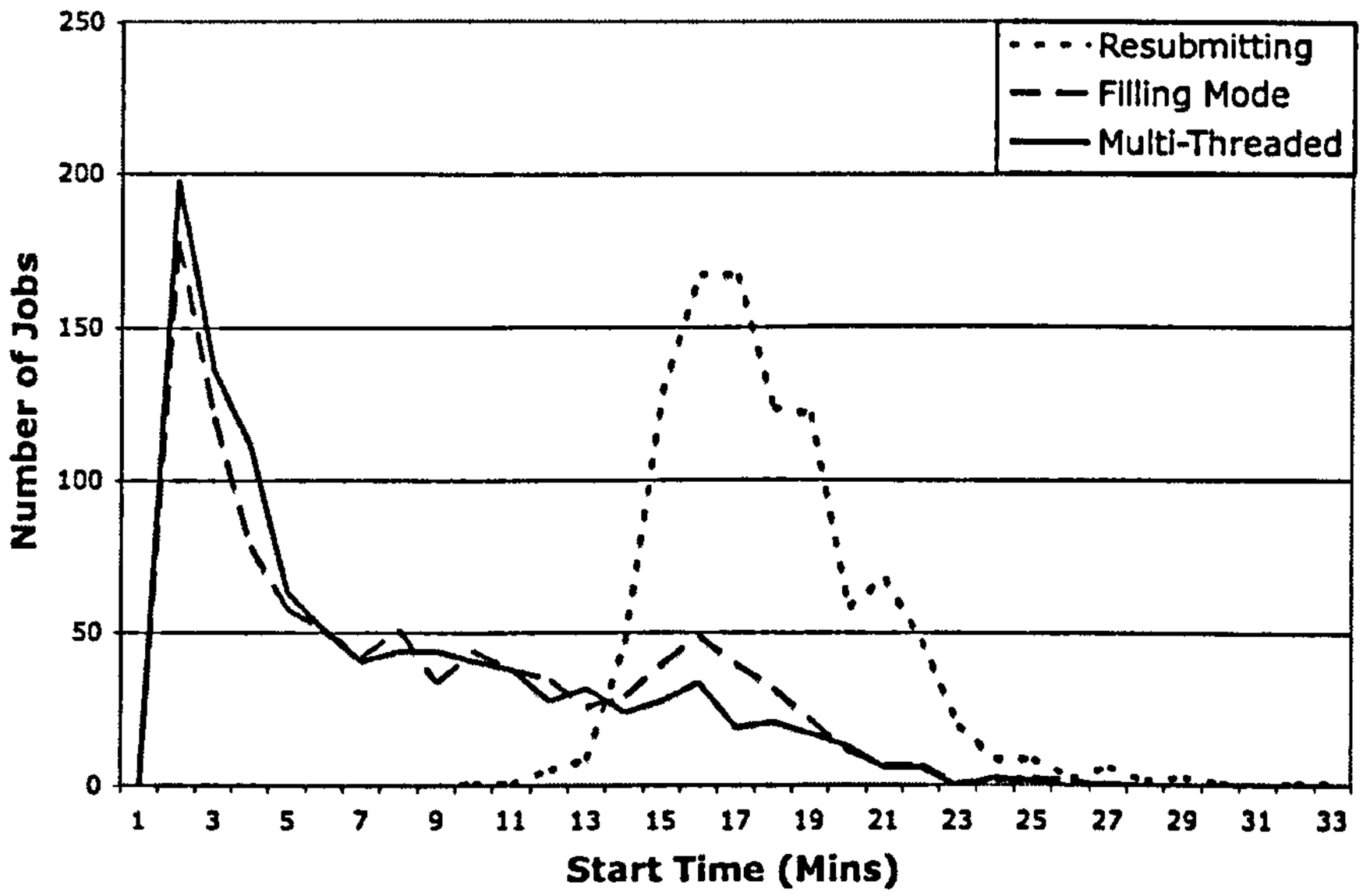

Figure 5.5: *Start times by submission mode for a total of 3000 jobs submitted to DIRAC by 30 users.*

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 in the steady mode of operation. It is important to note that all three thousand jobs completed successfully in this test, so the real goal is now to minimise the start times.

Figure 5.6 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 reproducibility of the results.

These results show that even when LCG is performing well, there is a significant improvement with the DIRAC optimisations. Furthermore, Table 5.1 shows that fewer Pilot Agents need to be sent for the Filling and Multi-Threaded modes compared to Resubmission mode and so the load on LCG can be reduced.

Comparing the number of Pilot Agents sent versus the number of jobs

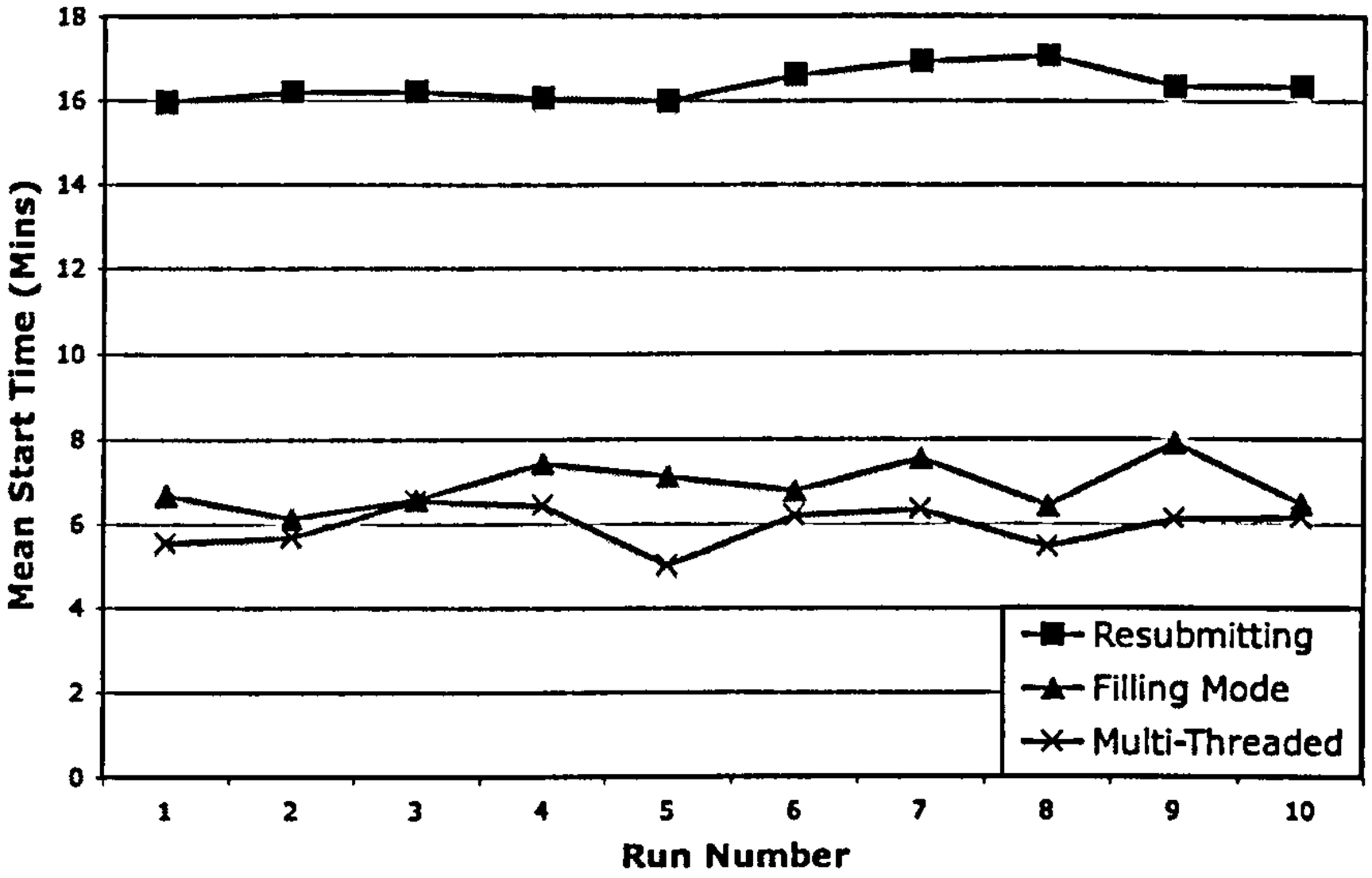

Figure 5.6: *Mean start times for 10 experiments submitting a total of 3000 jobs to DIRAC from 30 users.*

executed, Table 5.1 shows a reduction by a factor three for the Filling and over a factor of four for the Multi-Threaded mode in these experiments. The reduction factors depend on the amount of the available resources and on the Job characteristics.

The standard method to cope with high priority tasks on the Grid is to create 'short' queues, in a similar fashion to a normal batch system. This means that Grid resources are being allocated to serve jobs with a small processing time, *e.g.* a few hours. As a result, these resources are often idle, waiting for short, high priority tasks to arrive. The advantage of the Filling and Multi-Threaded modes of submission described above, is that no short queues are required in order to serve the high priority tasks. The optimisation is performed through maximising the usage of the resources, once they are obtained.

The experiments described here were performed using thirty distinct

Mode of Submission	Number of Agents Submitted
Resubmission	1000
Filling	299
Multi-threaded Filling	238

Table 5.1: *Number of Pilot Agents sent for each mode of submision for the experiments in Figure 5.6, involving a total of 30 users.*

users. Optimising 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 5.5 and 5.6 reflect the optimisation on a one hundred job basis. We can conclude that optimisation at this scale is effective but not as powerful as optimisation at the level of the VO could be.

Figure 5.7 shows the potential benefit of optimising the workload at the level of the VO instead of the level of each user. In this experiment two

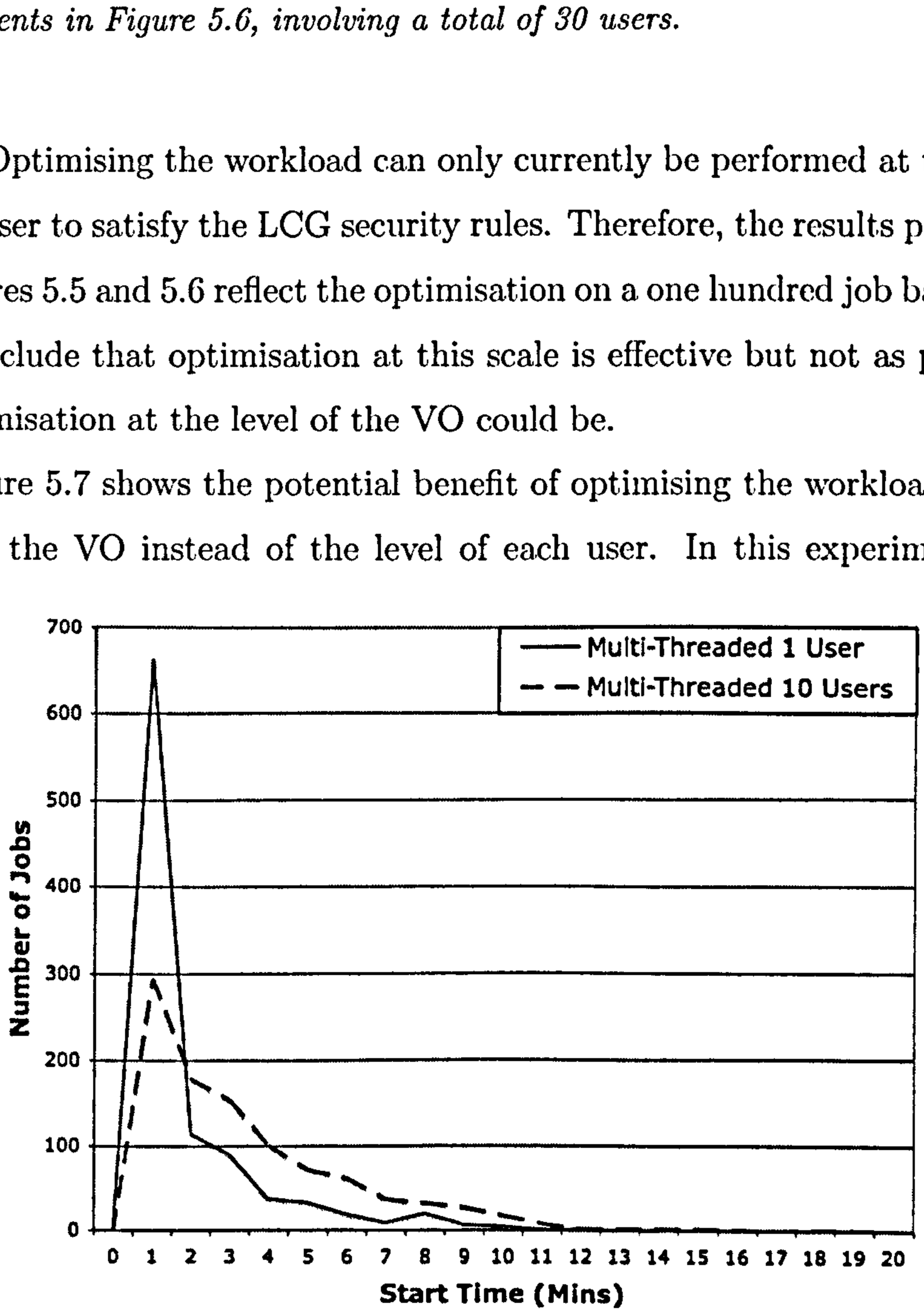

Figure 5.7: *Effect on the start time of jobs of optimising the workload on the level of the VO, versus multiple users.*

thousand jobs were submitted in Multi-Threaded mode. Half of the jobs were from a single user (equivalent to optimisation at the level of the VO) and the remainder were from ten distinct users. A clear improvement in efficiency is observed in the first case.

The results in Figure 5.7 were produced using the same conditions as those in Figure 5.5 and reflect the benefit of optimising the workload at the level of the VO. Since Pilot Agents submitted for the single user (equivalent to optimisation at the VO level) are able to pick up all of the jobs from that user, it is more likely that jobs are waiting in the system when requests are made. For the ten distinct users, each job was required to wait for the specific Pilot Agents submitted with the correct credentials to start. The possibility of sending generic LHCb Agents on behalf of the VO and using DIRAC to choose the priority of tasks in the central queue will be discussed in Section 5.4.6. In these tests, the system was performing at 100% efficiency. Since the testing occurred over a period of one day, factors affecting the provision of an analysis service over a longer period of time are less prevalent. In the next chapter, results from actual user jobs over a period of months will be presented.

5.4.6 Pre-emption and Future Optimisations

With the Filling and Multi-Threaded Filling modes described above, jobs being submitted to DIRAC have a chance of being picked up immediately, without the submission of any Agents to LCG. This is due to Agents submitted for previous jobs from the same user which can make requests to the WMS for other suitable jobs from the same user.

The term 'pre-emption' is used where the application of policy, *e.g.* high versus low priority jobs, results in DIRAC ensuring an optimised start time.

This was not directly explored due to the LCG security restrictions. However, results can be inferred from those presented in Section 5.4.5. Looking at Figure 5.5 and Figure 5.7, it is evident that in a steady regime, the start times can be minimised.

As shown in the last section, the effect of the number of users on the optimisations is significant. If it was possible to utilise all Agents of the VO, for the members of the VO, including those for the production, stripping and reprocessing activities, the start time could become negligible for high priority tasks. For example, LHCb plans to run the production over extended periods of time with thousands of production jobs starting and finishing daily. By first checking if a higher priority job was waiting, before running a production job, this would allow user distributed analysis tasks to run in advance. Likewise at the end of a production job, assuming there is sufficient time and resource left, an analysis job could potentially be executed afterwards. The Multi-Threaded mode also makes it possible to run jobs in parallel. In the future, this could allow low priority running jobs to be suspended, in favour of running a higher priority job. The lower priority job could then resume execution. From the LCG benchmark result in Figure 5.5 the mean start time is over 15 minutes. This delay may not be as significant for a 24 hour production job but for a 1 hour high priority analysis job this constitutes an overhead of 25%. An improvement of 10 minutes has been gained via optimisation at the 100 job level for each user. Hence, performing this type of optimisation at the VO level could lead to a mean saving of over 15 minutes per job.

This activity is pending, however, until it is possible to send generic LHCb VO Pilot Agents to the Grid. One future possibility for doing this securely on LCG is via *glExec*. This is a component of the gLite Middleware [67],

that will provide the functionality to switch user identities on the Grid WN.

If generic LHCb VO Pilot Agents become possible in the future, the application of policy and priorities becomes simple to apply in the DIRAC WMS due to the central *Task Queue*. There is already a lot of experience with this for standard batch systems such as LSF [171] and Maui [172]. As mentioned above, this solution would eliminate the need for dedicated short queues and is not possible with the standard LCG tools.

5.5 Comparison of Strategies with Previous Simulation

In a recent simulation study [118], the decentralised DIRAC approach was compared to a centralised scheduling system. The model adopted in [118] is slightly different to the real system above, although the results are consistent.

In this model, the centralised scheduling approach works by checking all resource availability upon the arrival of a task and scheduling the job to the least loaded resource. The DIRAC approach modelled here involves two main cases of deployment. The first case is where Agents are deployed to site clusters and make job requests, submitting jobs to the local scheduler of that site when successful. The second case is where the Agents running on the site query the matchmaker service and submit a Pilot Agent to the cluster, wrapped in a simple task, which checks the local environment and requests jobs from the WMS. The difference between the simulation and the real system described above is that the Pilot Agents are submitted through the standard Grid scheduling mechanism from the WMS.

In the ideal case, where the update period of the global information system tends to zero, the simulated study showed an improvement for centralised

versus decentralised scheduling [118]. In Figure 5.5, the results from the real system show that the ideal case is not the everyday experience and therefore the situation exists where the benefits of decentralised scheduling become significant.

From the simulation [118], a 50% improvement was observed with Filling mode for the average job start times when compared to the centralised scheduling approach. Comparing this to the results in Figure 5.6, the improvement observed with the Filling mode for the real system is 58%, consistent with the simulated results. The difference can be attributed to how the Pilot Agents are being deployed and the configuration of the real Filling Mode on the Grid.

One of the issues raised in [118] was that a large number of Agents in the simulation terminate without picking up any jobs in the Filling mode, which places an unnecessary load on the system. However, looking at the total number of submitted Agents in Table 5.1, it seems the opposite is true in the real system. The difference is due to how the jobs were submitted and the variations in Agent deployment described above. With the deployment of Pilot Agents on demand from the DIRAC WMS, the Filling and Multi-Threaded modes actually reduce the load on the Grid.

5.6 DIRAC, Condor and Condor-G

The Condor [72,173] project shares many of the principles on which DIRAC is based. In fact, DIRAC uses Condor ClassAds [75] for the purposes of matchmaking as described in Section 4.2. The aim of this section is to give an overview of the Condor project, with special emphasis on the similarities and differences to the DIRAC approach.

While DIRAC has been designed to accommodate multiple Grids and multiple VOs, the scope of the project currently only includes LCG and LHCb. DIRAC focusses on providing services to a particular community (VO) which overlays the infrastructure of the Grid, whereas Condor has a broader scope which encompasses providing the Grid infrastructure [174].

In a similar way to DIRAC, Condor places no dependence on the assumption that particular resources will work. This accommodates transient, unforeseen failures such as network outages or site misconfigurations. In fact, Condor has been designed with special emphasis on providing reliability through responsible behaviour [175]. Condor realises the centralised scheduling paradigm, where resources advertise their descriptions to a matchmaker service through ClassAds [75]. A machine known as the 'Central Manager' is dedicated to job scheduling and periodically receives resource advertisements and updates of status. The matchmaker service in Condor creates task and resource pairs in order to determine where the job will run. This informs the client and resource of a match and then the client proceeds to claim the resource. At this point, the request can be authorised or rejected, in case the matchmaking process was performed on outdated information. Therefore, Condor machinery is used to overlay a more centralised scheduling system on resources than DIRAC, which adopts the decentralised *PULL* approach.

The Condor project today [176] has several main elements. Condor and the Condor-G agent for the Grid, are described in Sections 5.6.1 and 5.6.2, respectively. Gliding-In, which shares similarities to the DIRAC Pilot Agent approach, will be discussed in Section 5.6.3.

5.6.1 Condor

The Condor high-throughput computing system provides many of the elements which are also common to DIRAC such as:

- Job management mechanisms;
- Scheduling policies;
- Mechanisms for the prioritisation of jobs; and
- Resource monitoring and management [174].

The main contexts of use for Condor are in the so-called 'high-throughput' and 'opportunistic' regimes. Similarly to DIRAC, Condor aims to optimise the use of available resources and provide reliable access to these resources over prolonged periods of time. In this high-throughput context, it is essential that failures are dealt with effectively to minimise the effect on the whole system. Opportunistic computing involves the utilisation of resources without requiring total availability.

Condor offers some advanced features for job checkpointing [176], which increases fault tolerance and also serves to keep a record of progress made. Condor also allows the possibility to migrate a job from one machine to another based on the recorded checkpoints. These are useful features which provide valuable redundancy during the processing of jobs. By comparison, DIRAC does not support these features and relies on the rescheduling of jobs in case of failures during execution.

Another area which suits the architecture of both DIRAC and Condor is the utilisation of CPU through 'cycle-stealing'. Condor is more developed than DIRAC for this activity, and can be configured to run jobs on desktop workstations when the keyboard and CPU are idle [174, 177]. However, the

Condor approach does require machinery to be in place at all times to detect the available resources and is therefore more relevant for individual organisations or institutions. For example, outside of allocated computing slots on Grid WNs, it would not be possible to have machinery in place specifically for one VO on these resources.

The Condor high-throughput computing system can be compared to the DIRAC Site Agent approach. Site Agents are typically used outside of the Grid and are more suited towards individuals, organisations and institutions for use on PCs and site clusters. As mentioned above, Condor offers more advanced features than DIRAC, but requires heavier machinery to be in place to achieve this. Moreover, while DIRAC overlays a decentralised system, with Agents making requests for jobs through the *PULL* approach, Condor overlays a more centralised architecture which is less scalable [118]. One drawback of the Condor system, as mentioned in [175], is that there must be a reliable network connection between submission and execution sites for the entire lifetime of a job. The job is not lost completely if it is broken, although a significant amount of work must be repeated. In fact, Condor-G was developed to deal with issues such as temporary network disconnections and will be described in the next section. As discussed in Section 4.4, DIRAC does not suffer from this feature and Agents only require outbound connectivity. In DIRAC, the monitoring of 'heartbeats' sent to the WMS indicates that jobs are still running, and it is configurable how long to wait before marking a job as stalled and taking further action.

5.6.2 Condor-G

The Condor-G [178] system was built through collaboration between Condor and Globus [48], with the result being a Grid-enabled agent for accessing re-

mote batch systems. As stated in [175], 'resilience introduces complexity' and Condor-G can cope with temporary network disconnections at the expense of additional machinery being in place. Several components of Globus were introduced to Condor-G in order to manage large numbers of jobs in a fault tolerant way. These include: Grid Security Infrastructure (GSI); Grid Resource Allocation Manager (GRAM) and Global Access to Secondary Storage (GASS).

The Condor-G agent has a user interface that allows the Grid to be considered as a local resource. In a similar way to the DIRAC API, described in the next chapter, the Condor-G API permits users to submit, cancel and monitor jobs as well as obtain detailed logs in case of failures [178]. In contrast to the DIRAC approach, Condor-G deploys a running process on the local machine to submit and manage jobs. DIRAC has no explicit dependence on the Globus components integrated into Condor-G. All that is needed is a valid proxy registered in the LHCb VO. In fact, once a DIRAC job has been submitted to the WMS, there are no processes running on the local machine by design.

With the DIRAC approach, the WMS takes care of all job requirements as discussed in Section 5.3. The Condor-G agent aims to provide similar functionality via the processes running on the local machine, such as: staging I/O files or executables; and also monitoring and recovery from failures.

The Condor-G agent provides seamless access to many types of batch system and is actually used by the LCG Resource Broker as a job submission service [176]. For the purpose of providing a VO with services, DIRAC offers a more lightweight approach that is less intrusive for users.

Condor and Condor-G have distinct advantages and disadvantages as discussed in [175]. For example, Condor allows advanced environments for

checkpointing, resource description and discovery, whereas Condor-G aggregates remote resources that do not have to be a Condor pool. The next section describes how a combination of Condor and Condor-G can be used in a complementary fashion.

5.6.3 Gliding-In

Gliding-in involves building a traditional Condor pool on top of a Condor-G system [175]. The Condor software is packaged into a 'glide-in job' and passed to Condor-G, which uses GRAM to submit the jobs. The glide-in jobs can also be a portable shell script [178], which retrieves Condor executables from a central repository. In this sense, 'glide-in jobs' perform the same function as DIRAC Pilot Agents.

In the glide-in approach, the user estimates approximately how many machines they wish to use [175] and then submit a number of glide-in jobs. In DIRAC, the WMS handles the deployment of Pilot Agents automatically through the *Agent Director* and *Agent Monitor* services. The number of Pilot Agents submitted is determined on demand, depending on how many waiting jobs are in the central *Task Queue* and the mode of submission. Both approaches ultimately serve the same purpose by creating a personal pool of resources that can execute the user jobs. The size of the pool is determined by the number of Pilot Agents or glide-in jobs which successfully start on the remote system. In both cases, the jobs terminate gracefully if no work is available.

While the glide-in approach has many similarities to the DIRAC Pilot Agent approach, heavier machinery is involved by establishing a Condor pool on remote resources and more intrusive processes are left running by Condor-G on the submission system. Furthermore, the establishment of a Condor

pool is overlaying a centralised approach to scheduling in order to aggregate resources. The DIRAC *PULL* approach simplifies scheduling through the use of Agents on resources. DIRAC, although lacking some advanced features such as job checkpointing, provides the necessary functionality for the LHCb VO by design. Since Pilot Agents are submitted on demand from the WMS, a steady pool of resources can be maintained and managed for LHCb automatically.

5.7 Implementation of DIRAC Paradigms in Other Experiments

Some of the trends for distributed data analysis systems were discussed in Section 3.4.4. In this section, two systems which exhibit similar functionality to DIRAC will be explored. The first of these is the Collider Detector at Fermilab (CDF) [179] production and analysis framework, GlideCAF, which uses the Condor Glide-In approach described in Section 5.6.3. The second is the ATLAS Panda system which also aims to facilitate production and distributed user analysis on the Grid and has been strongly influenced by DIRAC.

5.7.1 GlideCAF

GlideCAF is based on the CDF Central Analysis Farm (CAF) [180], that provides a CDF specific submission infrastructure on top of dedicated Condor pools outside of the Grid domain. Since CAF was based on Condor the decision was taken to use the glide-in mechanism on the Grid. The resulting system, GlideCAF [181, 182], made some additions to the standard glide-in

approach. One such extension was a 'glide-in factory', in order to create a virtual private Condor pool in a similar way to the DIRAC *Agent Director* service. Glide-ins are submitted on demand as new jobs arrive in the system, analogous to the way in which Pilot Agents are submitted by the *Agent Director*.

An interesting limitation with the GlideCAF system, stemming from the underlying Condor daemons, is that bi-directional network traffic was required with the remote sites. As a result, GlideCAF needs to be installed on every Grid site of interest in order to access site resources [181]. Since the DIRAC Agents only require outbound connectivity, this is a limitation that DIRAC does not suffer from. It is also interesting to note that GlideCAF uses a single CDF service proxy for all the glide-in jobs on the Grid. This means that each Grid site has no way to trace the actual users and is not in compliance, for example, with the LCG security rules. Before the extensions for distributed data analysis, this was the way in which DIRAC operated. In this regime only Condor, for GlideCAF, or in the case of DIRAC, the WMS, has complete knowledge of all users. GlideCAF also expects to make use of *glExec*, as discussed in Section 5.4.6, lending extra weight to the argument for generic VO agents in order to minimise the start times of high priority tasks.

5.7.2 Panda

As described in Section 3.4.4, the recent ATLAS Panda [123, 183] system has been strongly influenced by DIRAC. Panda has also been developed in Python, utilises the *PULL* scheduling approach (including the use of Pilot Agents) and has a very similar, service-oriented architecure to DIRAC. Panda also started as a Production system and has been extended for distributed

data analysis tasks. There are further similarities with DIRAC, such as the use of a MySQL database to store all job related information. A comparison of some of the DIRAC and Panda components is given in Table 5.2.

DIRAC WMS Component	Panda Equivalent
Task Queue	Task Buffer
Matcher	Broker / Job Dispatcher
Agent Director	Job Scheduler
Job Database	PandaDB

Table 5.2: *Comparison of the DIRAC WMS components and the ATLAS Panda system equivalents.*

Similarly to DIRAC, Panda also has data management components. In DIRAC, much of the data management infrastructure is part of the system although in the case of Panda there is an interaction with the Don Quijote Data Manager [184]. Both DIRAC and Panda also have a job monitoring infrastructure. In a similar way to GlideCAF, the Panda system currently runs using a single proxy [183].

More recently there has also been interest from the gLite [67] project in DIRAC workload management approaches, such as the Pilot Agent paradigm. In the future, it is possible that this will become the default mode of submission to the Grid.

5.8 Summary

The DIRAC infrastructure of Jobs, Steps and Modules was introduced in Section 5.2. This allows the creation of any necessary workflow for LHCb through different topologies of DIRAC Steps, which form a DAG, described in Section 5.2.1. The workflow of DIRAC jobs from submission to comple-

tion was examined in Section 5.3 with reference to the WMS components described in the last chapter. The methods used to ensure reliable access to input data were summarised in Section 5.3.1, with redundancy in place at each stage in case of failures.

In Section 5.4, DIRAC workload management optimisations were described, including Resubmission, Filling and Multi-Threaded modes. It is important to note that these optimisations are not possible with the standard LCG tools.

Extending the DIRAC production system to cope with distributed data analysis tasks has been demonstrated to be effective. From the results in Section 5.4.5, it is also evident that a significant improvement on the job start times could be obtained via optimisation of the workload at the level of the VO, rather than the individual user. This could lead to an average improvement of 15 minutes per job for the start time when compared to the LCG benchmark result in Figure 5.5. Optimisations on the VO level could become possible in the future through the use of the *glExec* component of the gLite Midddleware.

The results obtained for the workload optimisations were compared to a recent simulation of DIRAC in Section 5.5, which highlighted consistent results when compared to the centralised scheduling approach.

In Section 5.6, the DIRAC system was compared to the Condor and Condor-G systems, as well as the glide-in approach. The use of DIRAC paradigms in other experiments was described in Section 5.7 with a comparison to the recent ATLAS Panda and CDF GlideCAF systems.

In the next chapter the DIRAC Analysis Service will be discussed, using data from real user jobs. It was decided that the Analysis system would run using the Filling mode based on the results in Section 5.4.5. The implemen-

tation and maintenance of the DIRAC Analysis Service will be described, as well as possible future developments and a description of the experience gained from operating the system.

Chapter 6

DIRAC Analysis Service

DIRAC, the LHCb Workload and Data Management system for Monte Carlo simulation, data processing and distributed user analysis, was described in Chapter 4. In particular, the extensions necessary for DIRAC to accommodate the distributed user analysis activity were highlighted. The results of testing possible workload management optimisations were presented in Chapter 5. In this chapter, the DIRAC analysis service for LHCb will be discussed, starting with the current and future deployment strategies, in Section 6.2. Users interact with WMS services via the DIRAC API, this is discussed in Section 6.3. This section also describes some of the policy decisions that were made for LHCb Grid users, *e.g.* limits on sandbox sizes.

The system performance on LCG is explored in Section 6.4, and data from the real user jobs is used to explain analysis usage in Section 6.5. The experience gained from having real users will be covered in Section 6.6 and the strategies for maintaining the system will be detailed in Section 6.7. Finally, the future directions of the DIRAC system will be summarised in Section 6.8.

6.1 Introduction

The infrastructure of DIRAC and possible workload optimisation strategies were described in Chapters 4 and 5 respectively. This chapter will focus on how the system is currently utilised. As a production system, DIRAC operated with a single instance of the WMS services on one machine. The decision was made to create a separate instance of the WMS for the distributed data analysis tasks, which will be discussed in the next section.

In the study presented in the previous chapter a successful job completion efficiency of 100% was obtained. These tests were performed over a period of hours and therefore cannot be representative of the daily users experience of the system. The results of real users and the performance of the DIRAC system over an extended period of time will be presented in subsequent sections. This is intended to highlight the effect of problems that do not present themselves regularly but can still have a significant effect on the overall system performance.

The data sample used is from the experience of real users submitting jobs over a period of several months before and during the Data Challenge 2006 (DC06) activity. Since the LHCb production and analysis workflows are distinct, the user analysis jobs are in direct competition with the production jobs for the same resources.

6.2 Implementation of DIRAC Service

The current implementation of DIRAC involves two separate instances of the WMS, one for production activities and the other dedicated to distributed data analysis tasks. A third DIRAC instance exists for testing purposes and is principally used by developers of the system. With separate WMS

instances, it is important to note that production and distributed analysis tasks are in direct competition for the LHCb allocation of LCG resources. In the future, it may be necessary to have multiple machines for one instance of DIRAC. As mentioned in Section 4.3.2, the service oriented architecture of DIRAC allows services to be migrated to other machines if there are high loads. However, a single server has been sufficient thus far.

The introduction of generic LHCb Pilot Agents would make a strong case for implementing a single instance of DIRAC, since the production and analysis workloads could start to be optimised in a mutually beneficial way. As discussed in Section 5.4.6, this could lead to situations such as: running distributed data analysis tasks before and after production jobs at suitable sites; or halting a running production job in one thread and running a higher priority task in another. The latter would depend on a secure messaging system as described in Section 4.5.4, and the ability to switch the user identity on the Grid WN. Another possible scenario would be to keep the WMS instances separate and use generic LHCb VO Pilot Agents to first poll the DIRAC Analysis system for user tasks, before requesting a job from the Production system. While LCG security issues must be resolved to facilitate these optimisations, the potential gains for LHCb are considerable.

Currently, workload optimisations can only be made on the level of individual users to comply with LCG security rules. Therefore, the DIRAC Analysis System operates in the Filling Mode, introduced in Section 5.4.2.

The next section will describe the DIRAC API by which users interact with the WMS. This can be used in the same way for all DIRAC WMS instances, with a configuration option determining which one is used. For CERN LXPLUS users, a shared installation of the DIRAC client is available with an associated '`DIRACEnv`' command in order to set up the environment

correctly. This automatically points to the instance of the WMS that is dedicated to user tasks.

6.3 DIRAC API

The DIRAC API consolidates new and existing functionalities in DIRAC, providing users with a transparent way to submit jobs to the Grid. The DIRAC API is principally a scripting language but may also be used from the Python prompt. It allows users to securely submit, monitor, retrieve and delete Jobs. Input data is specified by LFN and the full complexity of the treatment of data on the Grid, as outlined in Section 1.6.1, is masked from the user. Exploiting the DIRAC Job, Step and Module topology, described in Section 5.2, the DIRAC API allows users to construct complicated workflows (DAGs) to perform, for example, private production tasks on a small scale.

In order to run user tasks on the Grid, several key elements must be considered. Firstly, any input data requirements must be satisfied, which will be explained in Section 6.3.1. Secondly, files needed for application 'steering', such as options files or DLLs, must be delivered to the computing resource to successfully execute the task. This is achieved through a sandbox mechanism described in Section 6.3.2. Lastly, the results of running a job, *e.g.* output files, must be stored in the file catalogue (LFC) or transported back to the user in an efficient manner, this will be explored in Section 6.3.3.

6.3.1 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 that can access this data, as illustrated in Figure 6.1. The DIRAC API automatically appends LFNs to

the user options file used to steer the application, before execution on the WN.

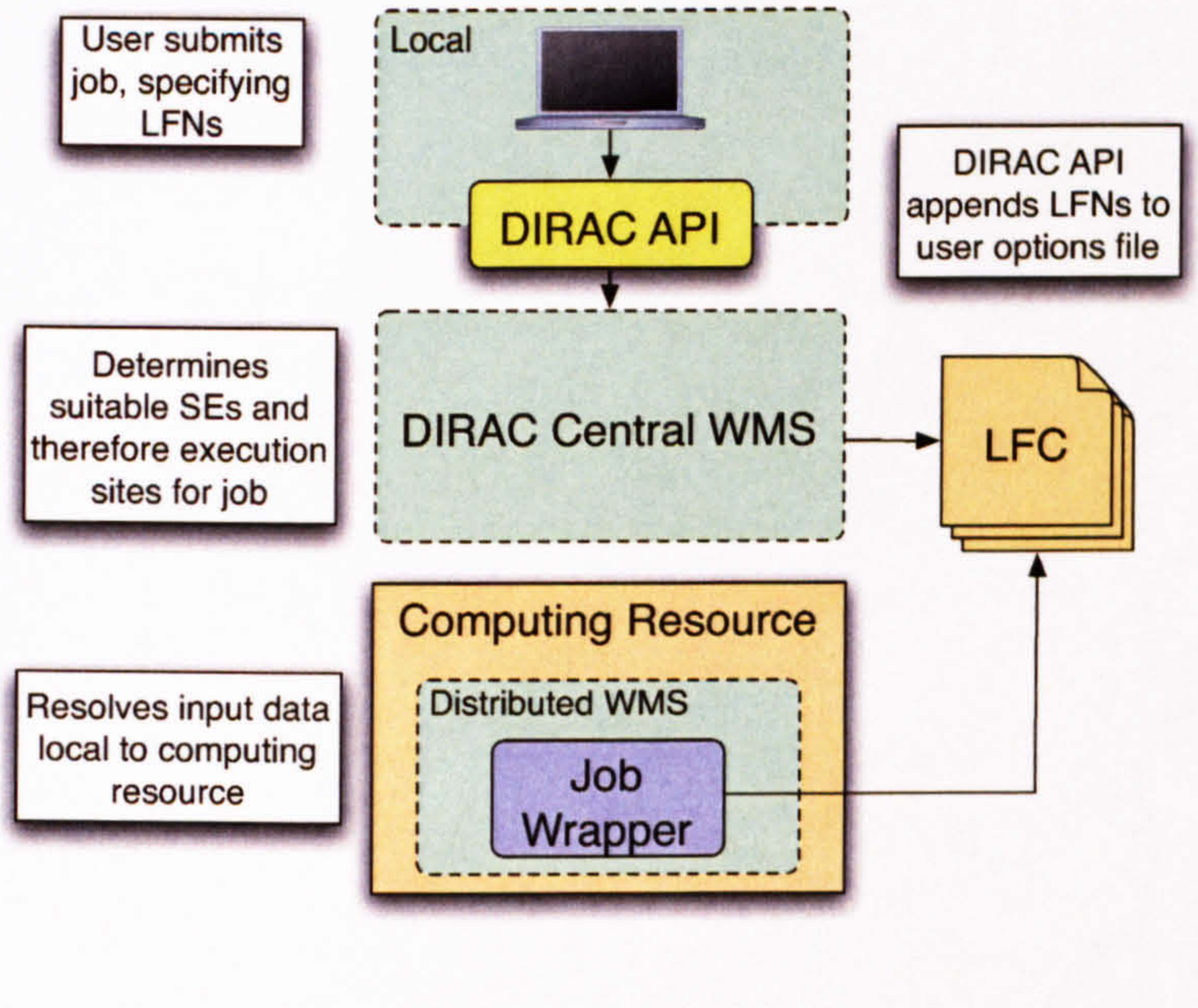

Figure 6.1: *Overview of treatment of input data by Logical File Name in DIRAC. The full complexity of providing access to data in a Grid environment is masked from the user.*

Once the job is on the WN, there is still some work to be done by the DIRAC *Job Wrapper* as mentioned in Section 5.3.1. The mechanism in Figure 6.1 is completely transparent from the user perspective. 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 that have access to the data and also resolves the LFNs local to the site that executes the job.

The mechanism described in Section 5.3.1 is a temporary solution. In the future, an SRM-aware version of Gaudi will be available. This will allow the specification of LFNs or SURLs in the application options file, that will be resolved to TURLs at the execution site through the use of GFAL (Grid File

Access Library). From the perspective of the user, this will be a transparent change and the DIRAC machinery will not require significant modification to support the new mechanism.

6.3.2 Input and Output Sandbox Handling

A typical user job will have small (*i.e.* less than 10MB) input files in order to steer the application as discussed in Section 2.2.2. These normally include an application options file and any DLLs containing compiled source code. In order to successfully run the job, these input sandbox files must be transported to the Grid WN or computing resource. Figure 6.2 highlights the DIRAC sandbox mechanism. When a job is submitted to the DIRAC WMS, input and output files are specified via the DIRAC API. The *Input Sandbox* service uploads the specified files from the user / local area and these are stored in the DIRAC MySQL database until requested by an Agent. Once an Agent has successfully requested a job from the WMS, the *Input Sandbox* service transfers it to the computer resource (WN) so that job execution can commence. All data transfers in this mechanism are performed using the XML-RPC Protocol.

After a job is finished, the specified output files are transferred by the *Output Sandbox* service to the WMS, and again stored in the DIRAC MySQL database. Job progress may be tracked in two ways: firstly, through the DIRAC API; and secondly, through the DIRAC Monitoring web pages [168]. Once a job has completed, the *Sandbox* service is again invoked to return the output to the local area of the user.

The *Sandbox* services could move to Grid storage in the future. However, there are some concerns about this such as:

- This will be slower;

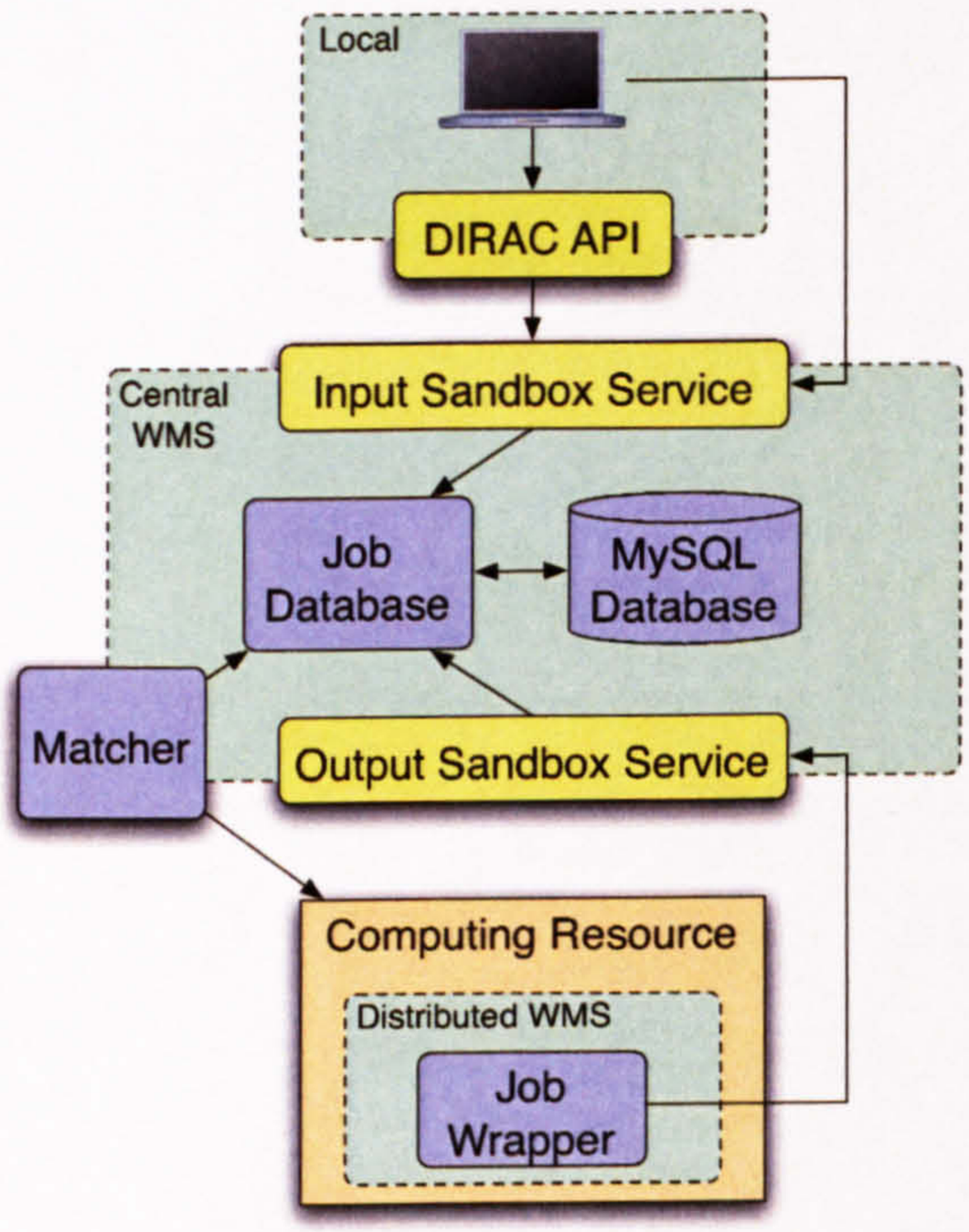

Figure 6.2: *Input / Output Sandbox handling as part of the job submission procedure in DIRAC. All data transfers in this mechanism are performed using the XML-RPC Protocol.*

- Extra dependency on LFC; and
- Increased client-side interaction with Grid SEs.

This will be implemented and tested, with the final decision being made based on performance. Another advantage of the current mechanism is the avoidance of flooding Grid SEs with small files. Since Grid SEs are often MSSs, they generally do not cope well with storing and providing frequent access to many small files. The current mechanism can also support large sandboxes as discussed in the next section.

Treatment of Large Sandboxes

The above mechanism is utilised for input and output sandboxes of less than 10MB. A protection mechanism is in place for input sandboxes over 10MB and it is possible to specify the input sandbox as an LFN via the DIRAC API. This allows users to upload an input sandbox to Grid storage, and the DIRAC *Job Wrapper* will automatically resolve and download this LFN before executing the application.

When a user specifies output files for a job their size may not be known in advance. To ensure no data is lost, whilst also protecting the DIRAC MySQL database against overloading, users can specify output data files for permanent Grid storage which will be described in the next section. In the case of a specified output sandbox file exceeding 10MB, the file is instead transferred and registered in Grid storage through the output data mechanism and notification is sent to the user through an additional file in the output sandbox.

6.3.3 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 6.3 shows the treatment of specified output data files by DIRAC.

The same process is used for output sandbox files that exceed 10MB, with the decision being made by the DIRAC *Job Wrapper* on the computing resource. If the specified files are found after the job has finished, they are automatically transferred to Grid storage and registered in the LFC. If failures occur, for whatever reason, the missing files are reported to the *Job*

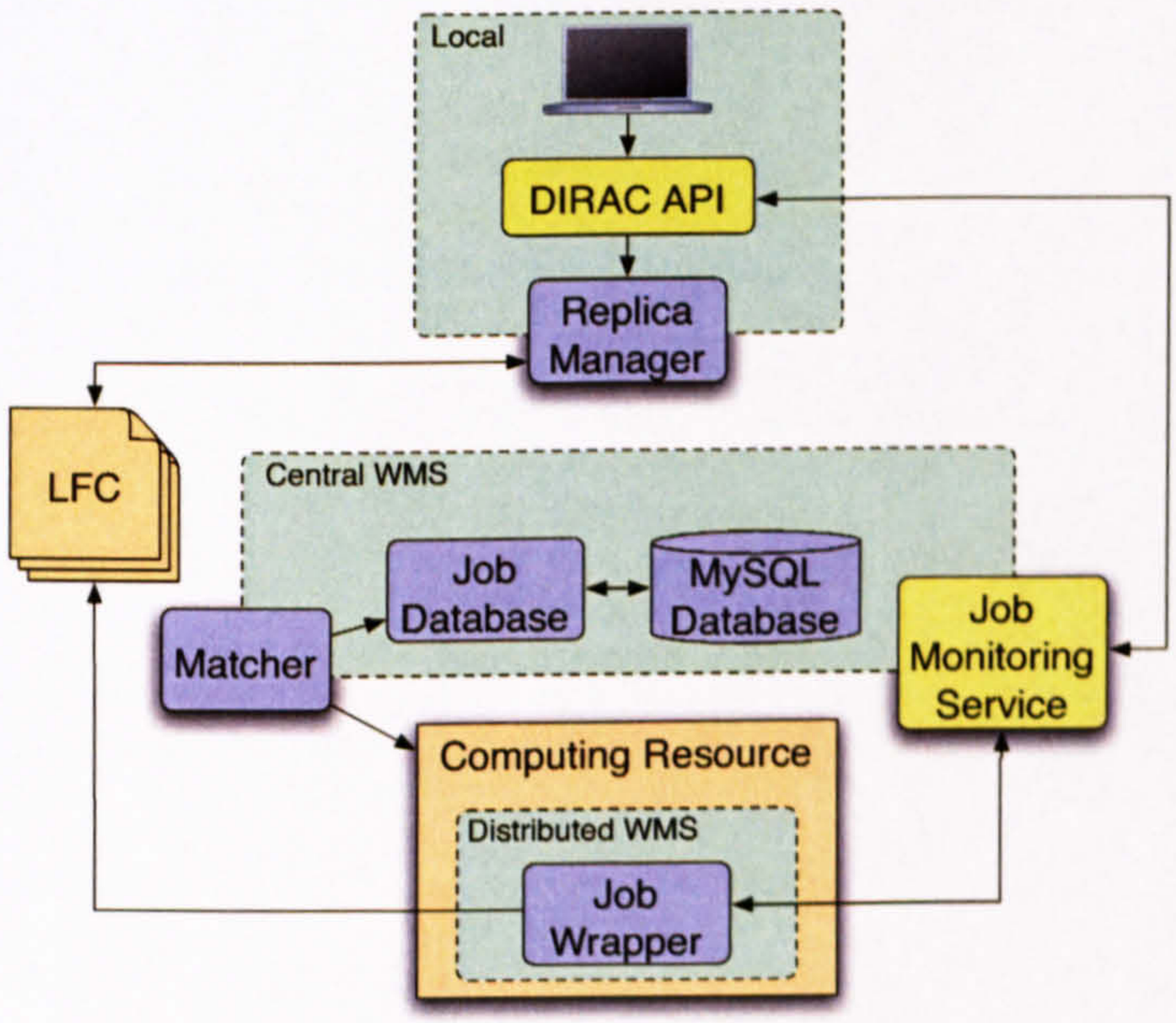

Figure 6.3: *Treatment of output data files in DIRAC. Large output sandbox files are also passed through this mechanism.*

Monitoring Service and the job will fail with a meaningful error message. For each output data file successfully stored in the LFC a file with a meaningful name is sent via the output sandbox, which contains pertinent information such as: size of file; LFN; and the Grid SE on which it is stored. This allows users to keep track of their Grid files and can be used to contain instructions for retrieval of the files if desired.

The temporary solution in place to cope with user output data files is a Grid SE directory, with group read/write access. In the future, this should become secure, with definite ownership of files at the user level. The convention used for registering the files is as follows:

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

This allows the user to submit a job to the WMS with output data from a previous job, specified as input data for the next. For example, private user productions could produce DST files suitable for later analysis jobs. This convention is mandatory and automatically prepended to all LFNs by the *Job Wrapper*. In fact, the convention may be overridden but this is strongly discouraged and could become restricted to a smaller group of trusted users, specified through VOMS.

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. In this way, users need never know the PFNs (SURLs) of data files, and the complexity of handling data in a Grid environment is therefore masked from the user.

6.3.4 Interface to LCG

The DIRAC API is entirely written in Python and there is no explicit dependence on a LCG User Interface (LCG UI) if a valid proxy is already present. However, to generate a Grid proxy, the commands '`grid-proxy-init`' or '`voms-proxy-init`' must still be used. In the future, it may be possible for DIRAC to generate proxies on behalf of LHCb users for use via DISET, although this is not currently in place.

Complications can arise with the DIRAC Client when users attempt to access output data files in Grid storage, since this involves access to the LFC. As discussed in Section 4.6.2, a Python interface to the LFC is shipped with the LCG middleware and this is used to implement the Replica Manager in DIRAC. In order to access the LFC without the presence of a LCG UI, a

distribution of LCG tools, including the LFC Python interface, is maintained as part of the DIRAC distribution. This also contains components such as GridFTP that are commonly used to manipulate files in a Grid environment. With a DIRAC Client installed on a correctly configured LCG UI, Grid-based operations such as accessing files stored in the LFC are immediately possible. However, to facilitate the use of DIRAC without the LCG UI environment, a few extra steps are required during the installation of DIRAC to configure the necessary LCG utilities.

Another important issue affecting LCG usage for both centrally maintained (*e.g.* on LXPLUS at CERN) and private DIRAC installations is CRLs. On an LCG UI, CRLs are automatically kept up to date, although for private DIRAC installations these must be periodically updated by hand. This will be discussed further in Section 6.7, with other aspects of maintenance.

6.3.5 Generic Gaudi Application Job

The DIRAC Job, Step and Module topology, described in Section 5.2, has been 'tailored' for user jobs with the equivalence demonstrated in Figure 6.4. In this way a simple DIRAC API script transparently creates one module to install application software, and another to execute the application, with the underlying complexity hidden from the user. This infrastructure also naturally accommodates redundancy, since failures during software installation can be reported before the application starts to execute. If failures cannot be recovered, the job can be automatically rescheduled.

The first few lines of the DIRAC API script import the relevant module and create instances of the `Dirac()` and `Job()` classes. To specify user input files, the default root location is taken as the directory in which the script

```
from DIRAC.Client.Dirac import *
dirac = Dirac()
job = Job()

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

job.setInputSandbox(['<Input File>'])

job.setInputData(['<LFN1>','<LFN2>'])

job.setOutputSandbox(['<Output File>'])

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

jobid = dirac.submit(job,verbose=1)
print "Job ID = ",jobid
```

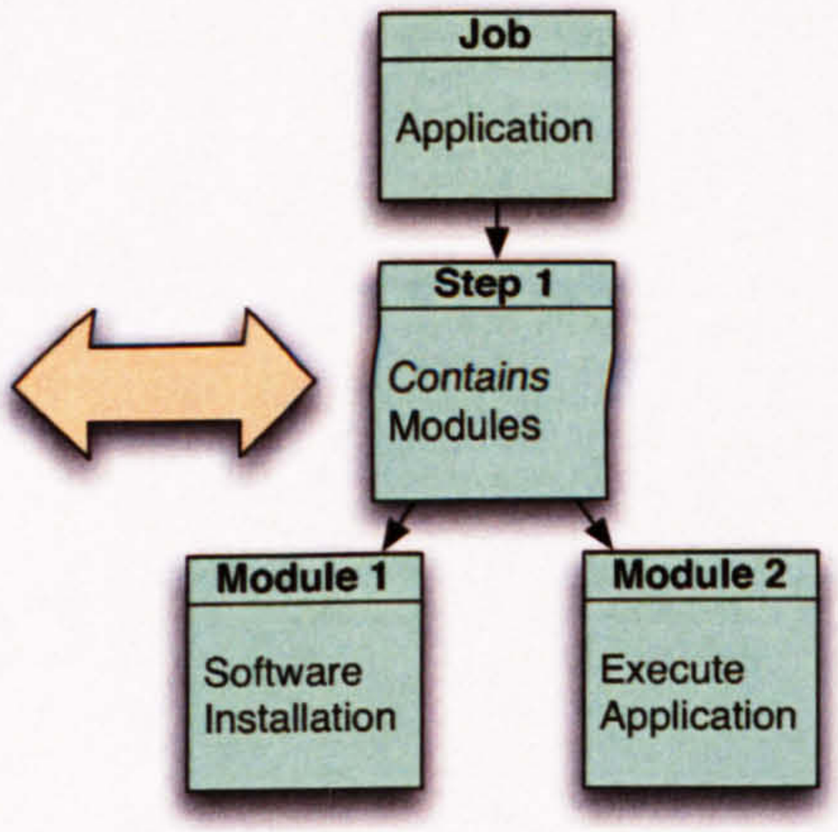

Figure 6.4: *A generic DIRAC API script for LHCb Gaudi-based applications shown with the equivalent Job, Step and Module structure.*

is executed. An application log file is automatically created by DIRAC and can be returned in the output sandbox. Appendix B shows the DIRAC API script required to create a more complicated workflow such as that shown in Figure 5.2.

6.3.6 Interface to GANGA

While it may be exploited directly by users, the DIRAC API also serves as the interface for the Ganga (Gaudi / Athena and Grid Alliance) [185, 186] Grid front-end to perform distributed user analysis for LHCb. Athena is the ATLAS software framework based on Gaudi. This common framework between the two experiments allows for cooperation in the configuration and management of tasks and, as such, Ganga is a joint project between ATLAS and LHCb.

Ganga provides a seamless way to submit jobs to several 'backends', these include: LSF; PBS; LCG; gLite; Condor and DIRAC. However, for LHCb

Grid jobs, the default mode of submission is via DIRAC and Ganga makes use of the DIRAC API to configure, submit and monitor jobs. The Ganga client also offers a GUI, that will provide a seamless way for users to query the LHCb Bookkeeping Database for LFNs, as well as client-side splitting of jobs into smaller tasks. The functionality to support more complex workflows, such as in Figure 5.2, is currently not available via Ganga. However, this is anticipated in the future.

With Ganga submitting Grid jobs via the DIRAC API to the WMS, LHCb has a seamless system that allows users to transparently submit jobs to batch systems, such as LSF, and the Grid. The DIRAC job status machine, highlighted in Appendix C, is very refined in order to aid in the debugging of Grid jobs and improve redundancy. Ganga provides a simplified view of this for the user, in order to mask the underlying complexity. Following the paradigms of the Grid outlined in Chapter 1, users should not be concerned with the finer details of what is going on behind the scenes, the priority is to ensure jobs are successfully run.

6.4 Performance on LCG

As described in Section 5.4.4, measuring performance on the Grid involves taking many factors into account, including: network outages; power failures; and site configuration problems. The data sample used for this analysis is from real user jobs[1] submitted to the DIRAC Analysis System over a six month period, between February and August 2006. The tests performed in Chapter 5 were obtained in a single day, and whilst the results yielded 100%

[1]This primarily consists of user analysis jobs but also contains a small number of private user production jobs, which account for 3% of the sample.

job efficiency this is not necessarily representative of real user experience. The focus of this section will be to reveal issues that can manifest over an extended period of time.

6.4.1 Job Start Times

A good measure of system performance is the start time of jobs over an extended period. The start time is defined as the time between submission to the DIRAC WMS and the job starting to execute on an LCG WN. The DIRAC Analysis system operates in Filling mode, introduced in Section 5.4.2. Figure 6.5, highlights the job start times for 3000 real user jobs over a six month period. On the whole these results are encouraging, with the majority

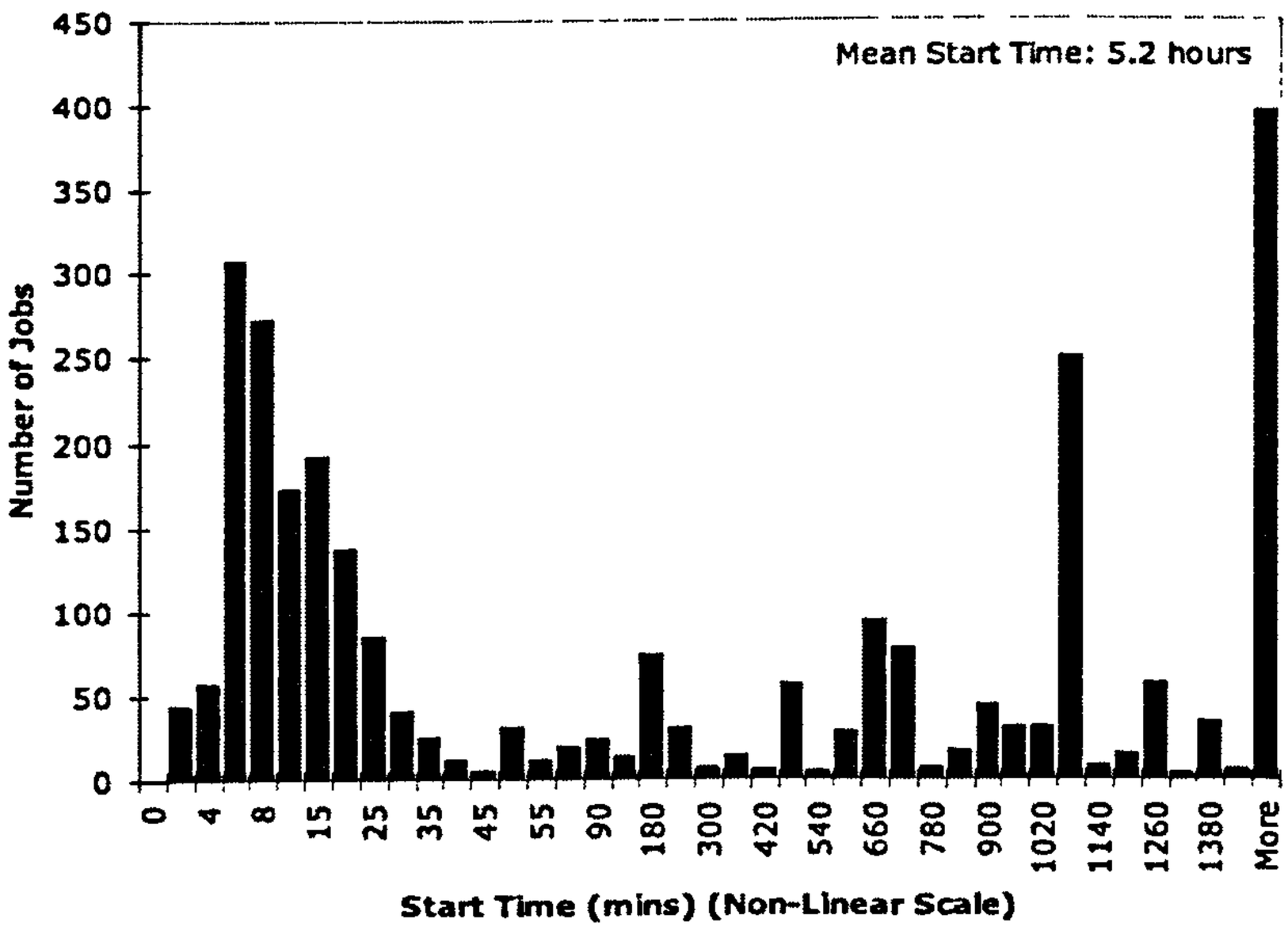

Figure 6.5: *Job start times on the DIRAC Analysis System for a sample of 3000 real user distributed analysis jobs, collected over a six month period. The mean start time, excluding rescheduled jobs over 24 hours, is just over 5 hours.*

of jobs starting in under one hour. However, the secondary peak in Figure

6.5, where jobs have a start time of over 24 hours, requires further comment.

A mechanism is currently in place for services in the WMS to always make use of the longest available Grid proxy. Each time a user performs an operation, such as submitting further jobs to the WMS, the existing proxy is checked. The existing proxy is replaced if it is valid for less time than the newest proxy which is currently available. This means that if a user job fails to start on one day for whatever reason, or the user submits jobs using a proxy of very limited validity, the jobs can enter a waiting 'proxy expired' state. Subsequent operations can therefore recover these jobs through renewing the available proxy. The side effect of this is that job start times can appear to be more than 24 hours. This is also caused by users rescheduling their jobs. Rescheduling means that the job is treated as new except for the job identifier and, significantly, the time of submission.

Large job start times can also be the result of no available WNs on the Grid. The effect of the DC06 activity on system performance will be explored in Section 6.4.5.

6.4.2 Total Job Times

The total job time is defined as the time between submission to the DIRAC WMS and the subsequent final reported job state, and can therefore be dominated by the job start times explored in the last section. Since users decide job parameters, including:

- Number of input datasets per distributed analysis job;
- Number of events for private MC production jobs; and
- Complexity of submitted algorithms.

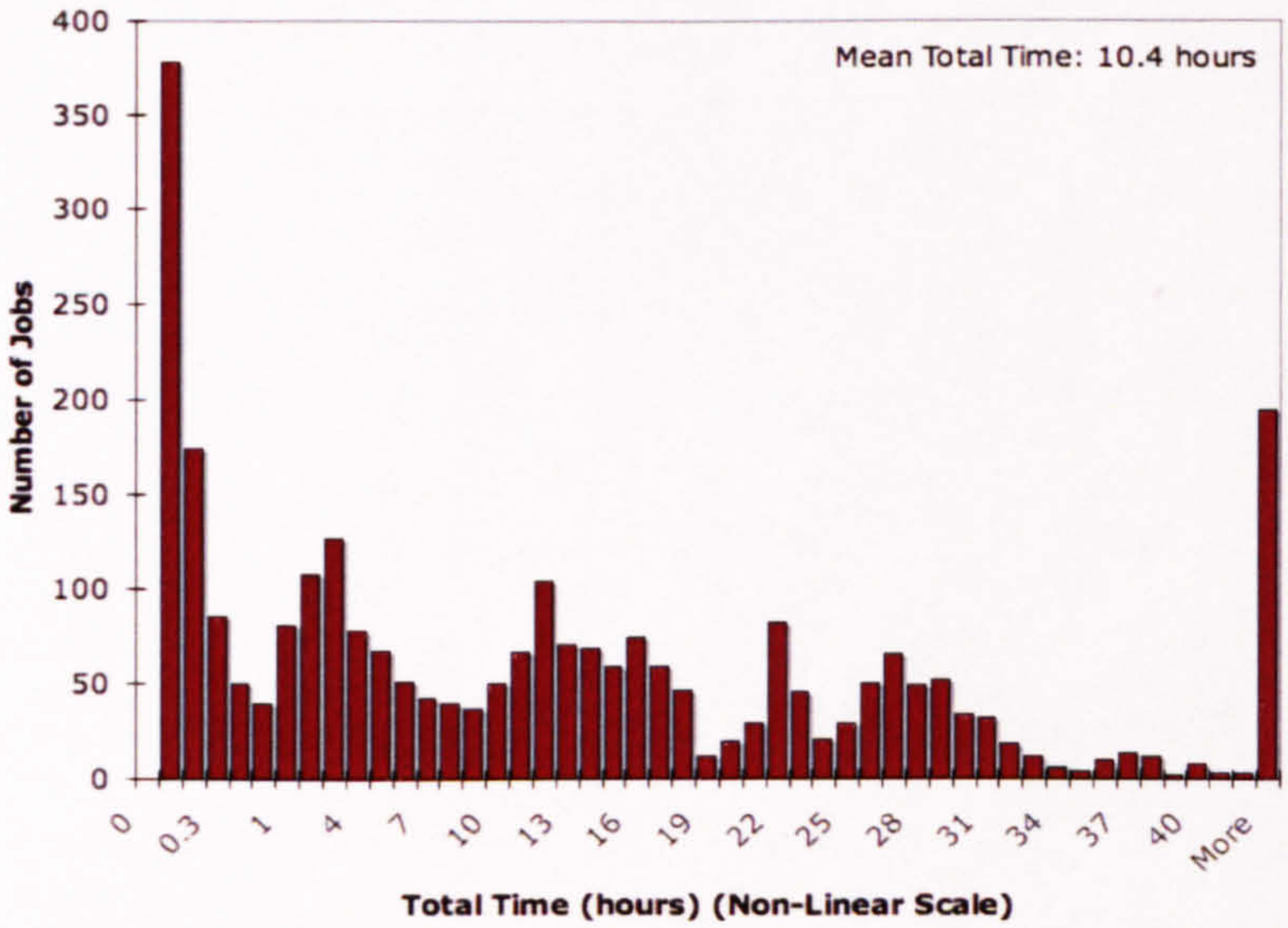

Figure 6.6: *Total job times for 3000 real user distributed analysis jobs collected over a six month period on the DIRAC Analysis System. Users determine the length of jobs via the complexity of algorithms and number of input datasets. The mean total time, excluding rescheduled jobs over 48 hours, is approximately 10 hours.*

The total job times are often chaotic in nature, illustrated in Figure 6.6.

The peak of jobs finishing in under one hour can be attributed to jobs submitted with a small number of datasets as well as jobs that fail shortly after submission. An example of the latter can occur if the specified input data is not available, after consulting the LFC. Such cases will be examined in Section 6.4.4, with a breakdown of the causes for job failures.

In a similar fashion to Figure 6.5, there is a peak of jobs lasting longer than 48 hours, which can be attributed to jobs that have been rescheduled. Other factors to consider include temporary effects such as data access problems, often caused by site misconfigurations, which can delay running jobs.

6.4.3 Matching Times

Matching time is defined as the time between a Pilot Agent requesting a job from the WMS and the job being delivered to the computing resource. LHCb Monte Carlo production jobs have fairly uniform requirements, normally including a particular amount of CPU time, as discussed in Section 3.1.2.

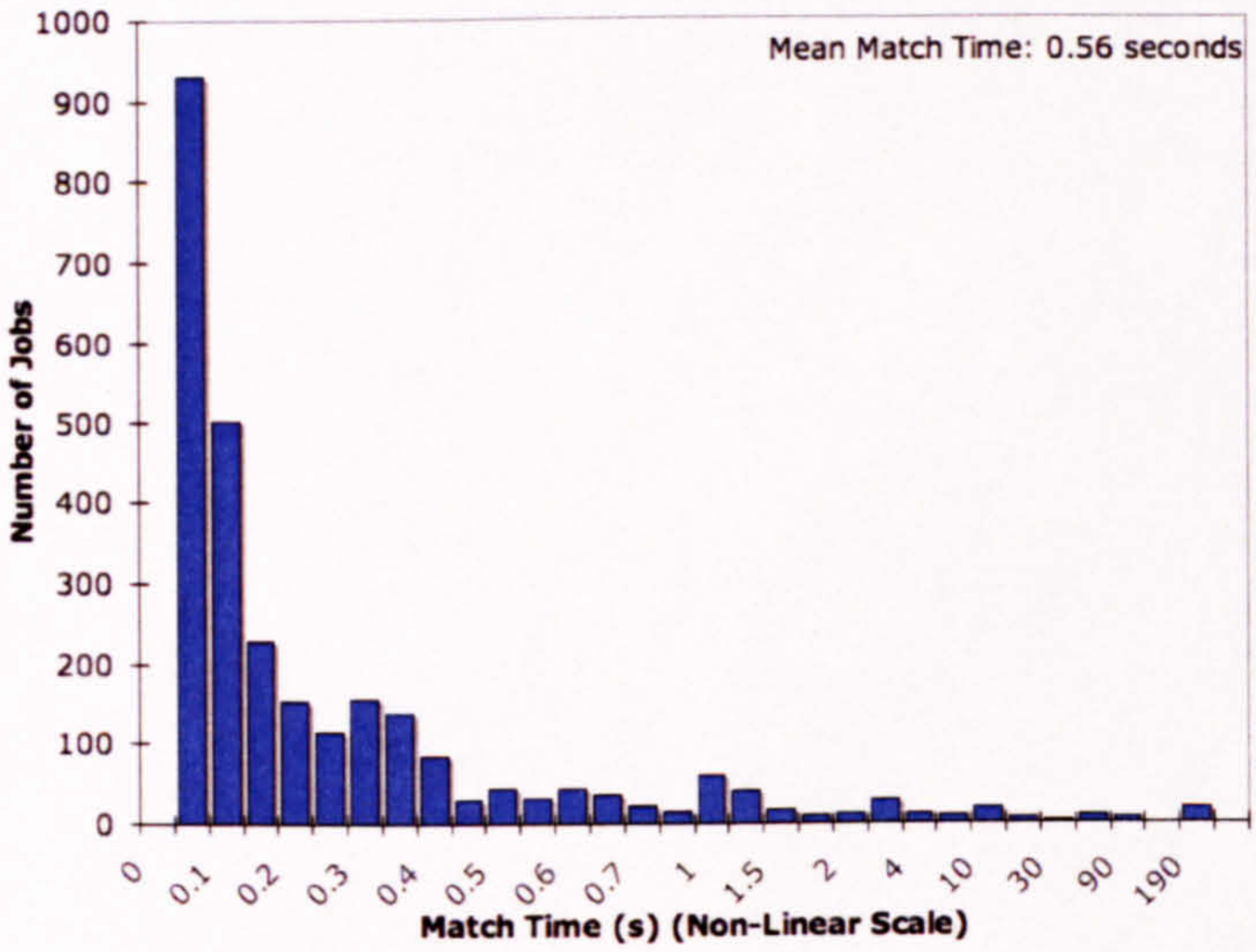

Figure 6.7: *Matching times on the DIRAC Analysis System for 3000 real user distributed analysis jobs collected over a six month period.*

With uniform production jobs, using *Task Queues* to enable job scheduling is natural. However, distributed analysis tasks have chaotic requirements and present a more demanding task for the *Matcher* service. Figure 6.7 shows the matching times for 3000 user distributed analysis jobs. Comparing this plot to the matching times for production tasks in Figure 3.3, reveals no loss in performance with 96% of jobs being matched in under 2 seconds. Moreover, the mean matching times are consistent, with a value of 0.56 seconds for Figure 6.7, compared to 0.42 seconds for Figure 3.3. The double-matching

mechanism where Agents may also impose requirements on the jobs, reduces the load on the *Matcher* service by making requests more specific.

Looking at the plots for start times and total job times in Figures 6.5 and 6.6, it is evident that the matching time is a negligible factor in the overall lifetime of DIRAC jobs. Furthermore, this demonstrates that the *PULL* scheduling paradigm originally employed for LHCb production tasks scales well to the chaotic requirements of distributed data analysis jobs.

6.4.4 Job Completion Efficiency

Whilst individual short-term usage of a system may yield high efficiency, temporary issues can arise over a longer period of time, affecting the success rate of user jobs. Some of the user experience gained will be discussed in Section 6.6. However, in this section the focus will be on how the sample of jobs fared over a six month period. Figure 6.8 illustrates the job completion efficiency breakdown. Each of the cases in Figure 6.8 will be explored below:

- **Successful (68%)** Successful jobs are those that have entered the final state without errors. This includes jobs which from a DIRAC perspective are successful, but may not be a success from the user perspective. For example, if a user does not specify the output data file name correctly, the system will mark a job as failed although the application has executed properly.

- **Input data not available (10%)** Jobs with input data requirements can 'fail' because input data is not available. This means either the specified data is incorrect or there are inconsistencies in the file catalogue. A common mistake is to specify PFNs (SURLs) instead of LFNs, since users have experience with using PFNs on batch systems.

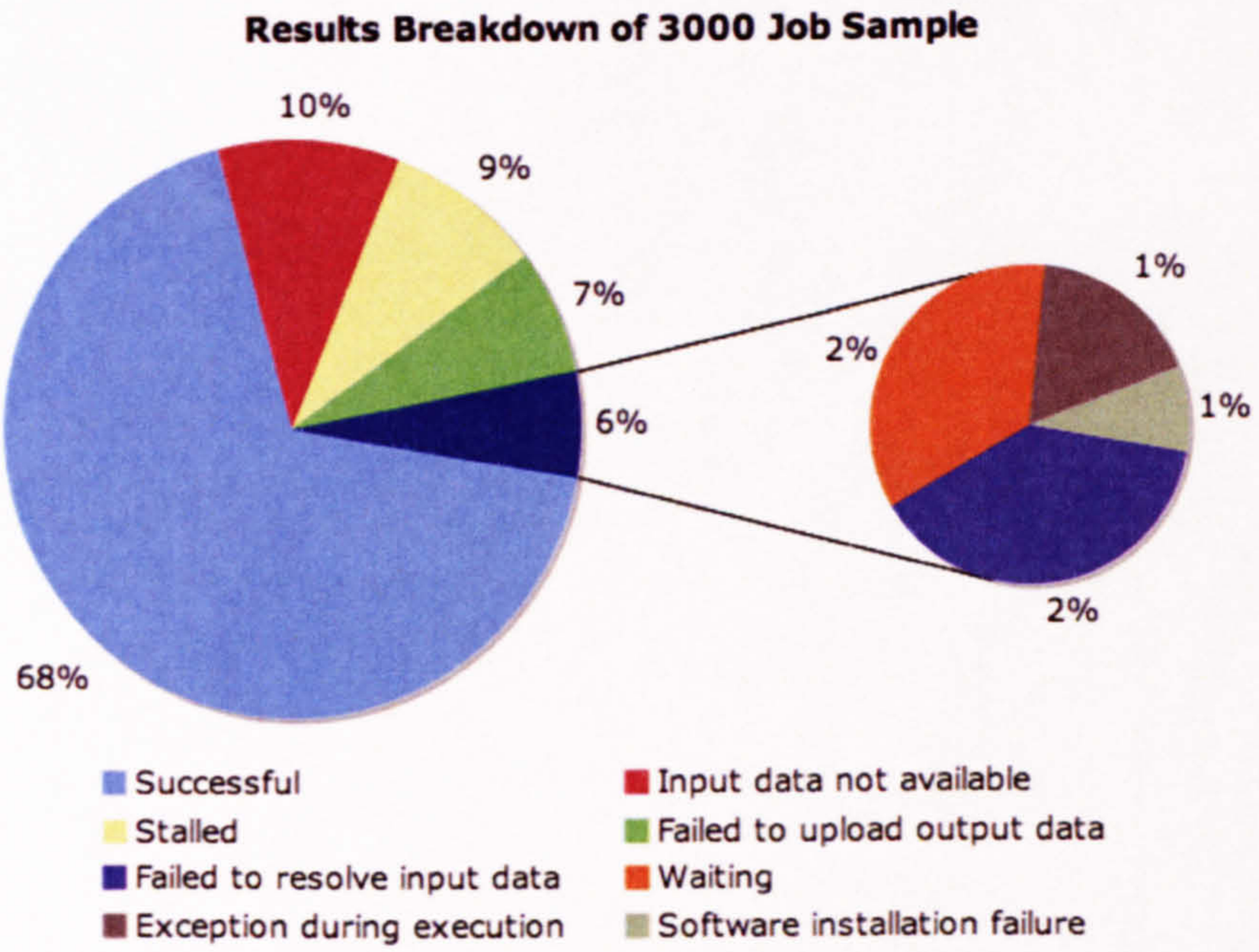

Figure 6.8: *Breakdown of results for 3000 real user distributed analysis jobs collected over a six month period from the DIRAC Analysis System.*

The latter case where inconsistencies exist in the file catalogue is more complicated and dominates the figure of 10% in the sample. Although these entries have subsequently been fixed, this is a time consuming operation, which must often be performed 'by hand'. In fact, these jobs can be considered successful from the perspective of DIRAC, since Pilot Agents are prevented from being sent to LCG unnecessarily. Although the jobs cannot run immediately, they may be rescheduled and run successfully afterwards.

- **Stalled (9%)** A job is marked as 'stalled' if the DIRAC *Job Monitoring Service* stops receiving 'heartbeats', which are regular notifications of the job being in an acceptable state. The main cause of this is due to user proxy expiration on the WN. Proxy expiration is a major issue

that requires further thought and will be discussed in Section 6.8. If a user submits jobs with a 'short' (*i.e.* default 12hr) proxy submitted Pilot Agents may wait in a site batch queue for a significant portion of this time before starting to execute. If the proxy expires while the application is being executed, the job will stall. A simple solution for users is to submit jobs with a proxy that lasts several days, however, it is potentially dangerous from a security point of view for this to become standard practice. Other causes for stalled jobs include: power cuts; network outages; and also site misconfigurations. The latter are generally more subtle problems, sometimes only affecting a small number of jobs at any one time. For example, major power cuts on the site level are obvious to spot but more minor interruptions, affecting individual WNs, become more difficult to identify.

- **Failed to upload output data (7%)** The failure to upload specified output data is caused by the transfer and register operation to the LFC failing. This can happen due to network outages, power cuts, site misconfigurations, and also LFC availability. In fact, of the 7% of jobs in the sample which failed to upload output data 94% occurred during one day, when the LFC was unavailable. Therefore, 7% is not representative of the high level of service over the six month period.

- **Waiting (2%)** Jobs which are in the 'waiting' state have not failed, they simply have not begun to execute. Waiting jobs are either: submitting Pilot Agents; waiting for Pilot Agents to respond; or have an expired proxy. On further examination of the 2%, all had an expired proxy and this can be attributed to the ongoing DC06 activity. The effect of DC06 on system performance will be described in Section 6.4.5.

- **Failed to resolve input data on the WN (2%)** Jobs arriving at the WN first install any required application software then resolve input data as described in 5.3.1. The vast majority of the 2% of jobs failing to resolve input data are due to transient site configuration problems. For example, if available site protocols are not correctly set up, this results in not being able to construct TURLs for the software application to access input datasets. To recover from this type of failure, the usual course of action is to ban those sites until issues are resolved and reschedule affected jobs.

- **Exception during execution (1%)** This is perhaps the most difficult error to debug, since affected jobs report no problems until failure. Further examination of the 1% of jobs in this sample revealed that the most likely cause of these failures is application failure. However, this could also have been caused by power cuts.

- **Software installation failure (1%)** The software distribution mechanism, introduced in Section 2.5.4, has proven to be successful for both production and distributed analysis tasks. However, this is not immune to network outages, which prevent the transfer of binary distributions to the WNs. All of the 1% of jobs failing in this manner appear to be caused by network problems.

Most of the causes of failure examined above are consistent with the experience of running jobs on the Grid, however, the inconsistencies in the LFC accounting for 10% of the sample requires further comment. LFC inconsistencies can be attributed to the fact that the system is relatively new, although does suggest that an extra mechanism to check catalogue entries would be advantageous. Without this, it is unlikely that problems affecting

particular SURLs at individual sites can be discovered until users start to run over all available datasets. The resolution of this problem should also be automated to some degree in the future to enable swift recovery and allow affected jobs to run as soon as possible.

Overall, the breakdown of causes for job failures is fairly encouraging. Taking factors with temporary causes into account such as: input data not available; failure to upload output data; waiting jobs; and the failure to resolve input data on the WN, the efficiency becomes much higher. Also, further examination of the stalled jobs in the sample showed that around 4% of the 7% were due to job submission with a short-term proxy. Therefore, an estimate of the efficency of the system, having omitting the temporary problems from the sample, becomes 91% with the remaining 9% caused by intermittent power cuts and network outages, outside of the control of DIRAC, over the six month period.

6.4.5 Effect of DC06 Activity on Performance

In order to establish the effect of DC06 on performance of the system, some factors in the sample require consideration. For the job start times shown in Figure 6.5, all jobs in the sample starting in over 24 hours were ignored. Similarly for the total job times, shown in Figure 6.6, jobs in the sample finishing in over 48 hours were omitted. While it is possible that these jobs did have delays of over 24 and 48 hours respectively, in fact, the vast majority have been rescheduled. The percentage of these ignored 'rescheduled' jobs is taken into account in Table 6.1, as another measure of system performance.

Figure 6.9 displays the number of jobs running on the DIRAC Production system during 2005-2006. The sharp increase in usage during May 2006, corresponds to the start of DC06. Therefore mid-May was used in order

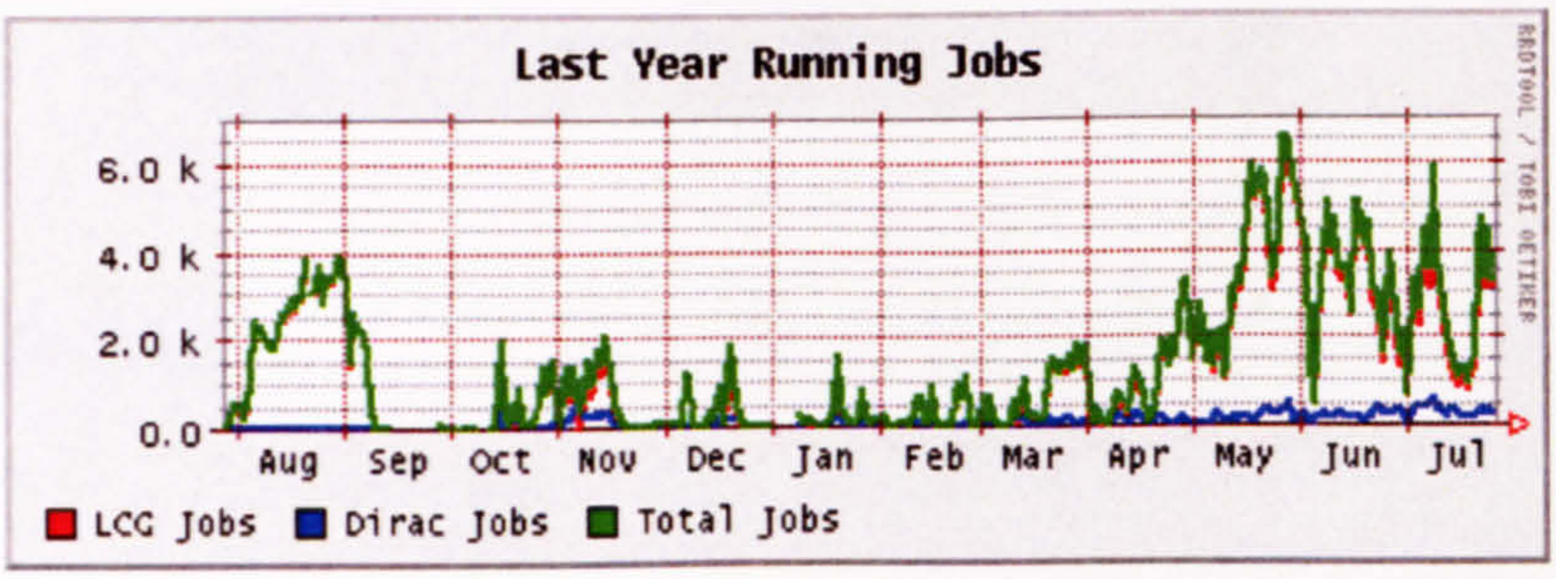

Figure 6.9: *Number of jobs running on the DIRAC Production system during 2005-2006. A sharp increase in usage is observed in May 2006, corresponding to the start of the LHCb DC06 activity. The small number of 'DIRAC jobs', corresponding to jobs using DIRAC, but outside of the Grid, is also shown.*

to split the sample of user distributed analysis jobs to determine how the DC06 activity affects the performance of the system. The production and analysis tasks are in direct competition with each other, since the ramp-up of production tasks can saturate all CPUs available for LHCb on LCG. Table 6.1 highlights the changes in mean start time and total time for jobs before and after the DC06 activity.

Parameter	**Before DC06**	**After DC06**
Percentage of Jobs in the Sample	29%	71%
Mean Start Time (hours)	0.5	7.0
Mean Total Time (hours)	2.6	12.8
Percentage of Rescheduled Jobs	<0.5%	12%

Table 6.1: *Effect of the DC06 activity on the mean start time and total time of jobs. The sample was split using mid-May as the starting point of DC06, from Figure 6.9, and also takes into account the overall percentage of rescheduled jobs in the sample.*

It is clear from Table 6.1 that a drop in system performance is observed corresponding to the DC06 activity. In this regime, Pilot Agents submitted

for distributed analysis tasks frequently end up in long site batch queues before starting to run, regardless of how many are sent per job. As will be explored in Section 6.5.1, the sample of distributed analysis jobs is dominated by a peak corresponding to the large number of jobs in May from the production system in Figure 6.9. To further analyse this effect, Figure 6.10 shows the mean start times versus the number of submitted jobs, per day.

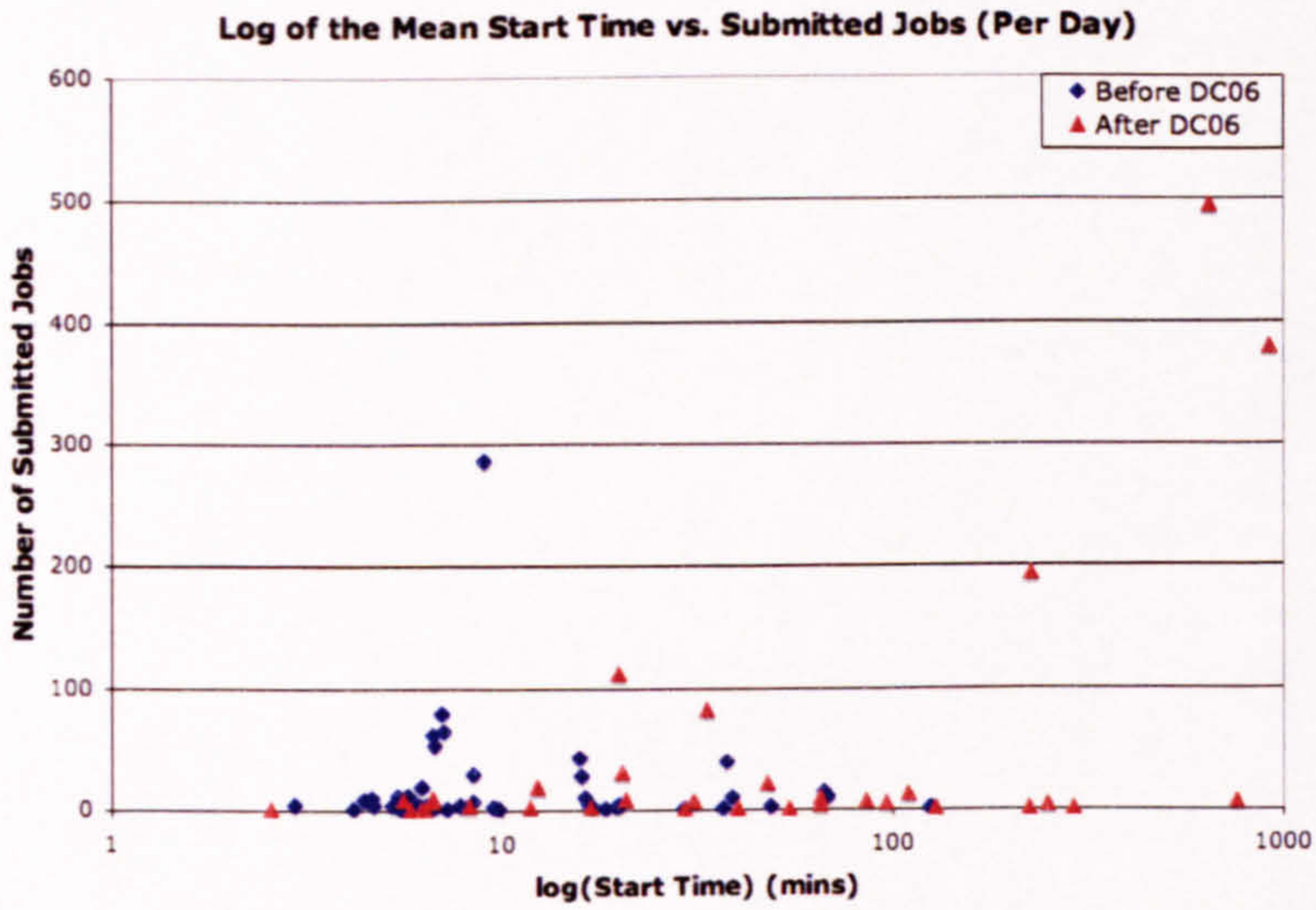

Figure 6.10: *Plot of mean start times versus the number of submitted jobs submitted to the DIRAC Analysis System on a daily basis, during a six month period. Rescheduled jobs with start times over 24 hours were omitted from sample.*

While the majority of the days considered in Figure 6.10 show a reasonable average start time, the outlying points in the sample are dominated by those which correlate with the start of the DC06 activity. A possible solution to this problem was mentioned in the last chapter and involves the use of generic LHCb VO Pilot Agents. If it were possible for all Pilot Agents sent by members of the LHCb community to potentially run jobs from other members of the VO, it would facilitate the execution of higher priority tasks.

Furthermore, the production activity (shown in Figure 6.9) shows regular use of LCG resources during DC06. Therefore, the steady number of Pilot Agents sent for the production activity could be utilised by the higher priority distributed analysis jobs, offering a potentially negligible start time, as described in Section 5.4.6.

6.5 DIRAC Analysis Usage

This section will examine the user patterns of data analysis using the sample of distributed analysis jobs submitted to the DIRAC Analysis System. Section 6.5.1 will examine the frequency of submission of jobs in the sample and detemine the effect, if any, of number of users on job start time. In Section 6.5.2, the number of datasets submitted per job in the sample will be investigated.

6.5.1 Frequency of Submission

Figure 6.11 shows the number of jobs submitted to the DIRAC Analysis System over the six month period. The highest peak occurs at the start of the DC06 activity, during May 2006. Although the statistics are lower than the numbers used for testing the optimisation strategies in Chapter 5, the chaotic nature of real usage of the system is apparent.

To determine if a correlation exists between the number of users and start times of jobs, Figure 6.12 shows the number of unique users submitting jobs to the DIRAC Analysis System, averaged over two weeks for the six month sample. This is plotted against the mean job start times averaged over the same period.

While the statistics are relatively low, Figure 6.12 demonstrates that

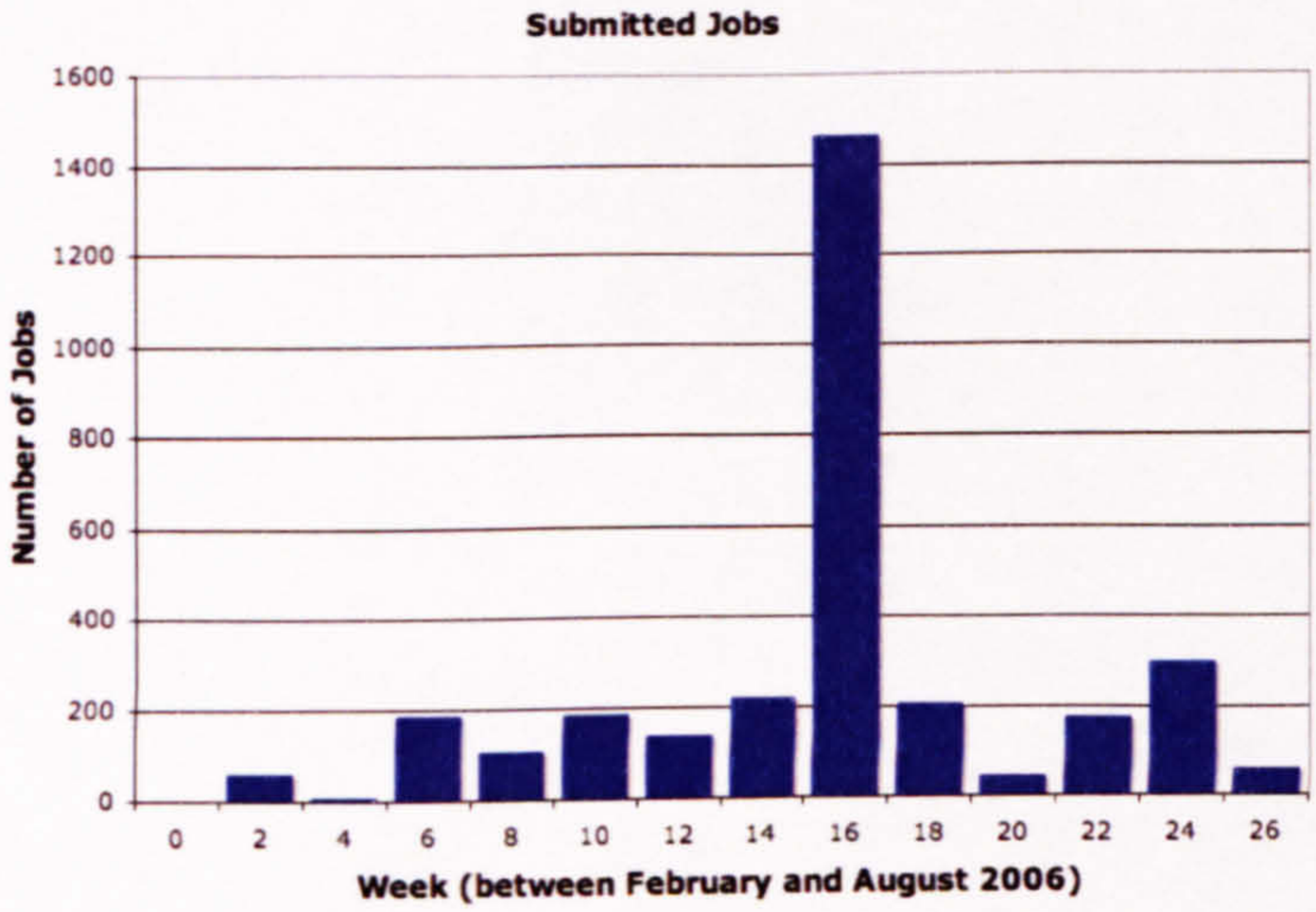

Figure 6.11: *Number of jobs submitted to the DIRAC Analysis System every two weeks between February and August 2006.*

the number of users is independent of the start time (excluding the outlier discussed further below). On further inspection of the sample, variations in the start time can be attributed to the DC06 activity. Each outlying point *e.g.* where the number of users equals 4, 8, 9, 10 and 11, occurs after the start of DC06. The furthest outlying point in Figure 6.12 coincides with the largest submission of jobs in Figure 6.11 and is the most strongly affected by the apparent lack of available Grid resources.

6.5.2 Size of User Jobs

One of the largest determining factors in the length of user jobs is the number of specified input datasets. Figure 6.13 shows the number of input datasets for each job in the sample, over the six month period. Jobs submitted with no input datasets relate to private user production jobs. These make use of the DIRAC Job, Step and Module architecture, described in Section 5.3.1, to construct the workflows.

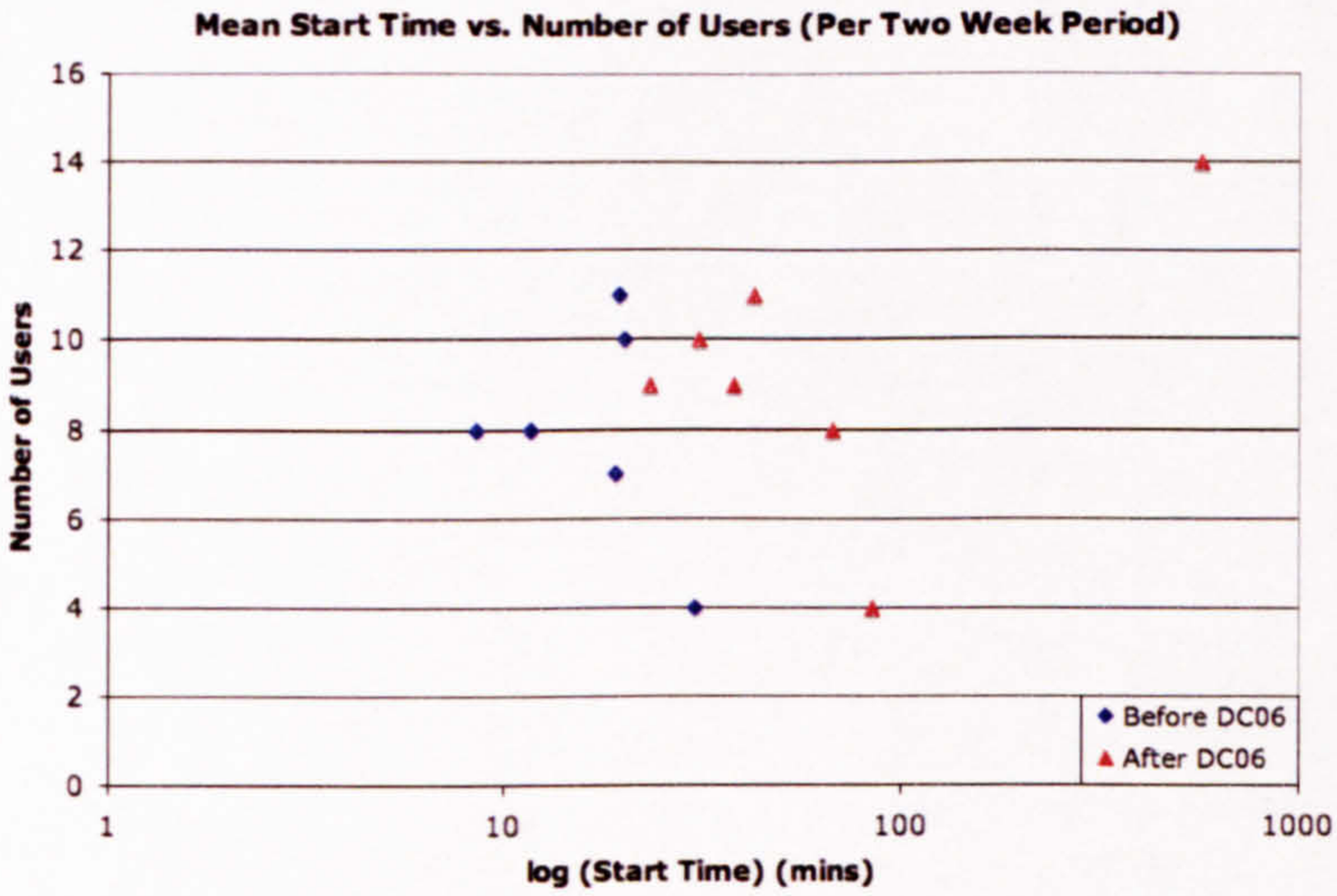

Figure 6.12: *Mean start time versus number of users submitting jobs to the DIRAC Analysis System every two weeks between February and August 2006. Over the period there was a total of 44 distinct users.*

While Figure 6.13 comprises of jobs submitted by over 40 users, it is remarkably ordered with a peak at 20 input datasets. With each LHCb dataset containing approximately 500 events, this suggests running over 10,000 events. As mentioned in Section 3.2, the LHCb Computing Model [100] predicts that approximately 140 physicists will submit 2 jobs per week which will process $\sim 10^6$ events per job, increasing to $\sim 10^7$ events for larger samples. This corresponds to jobs submitted with between 2000 and 20,000 LHCb input datasets. It is envisaged that these jobs can be split into smaller 'chunks' in order to be run in parallel which, from Figure 6.13, appears to be how users are proceeding.

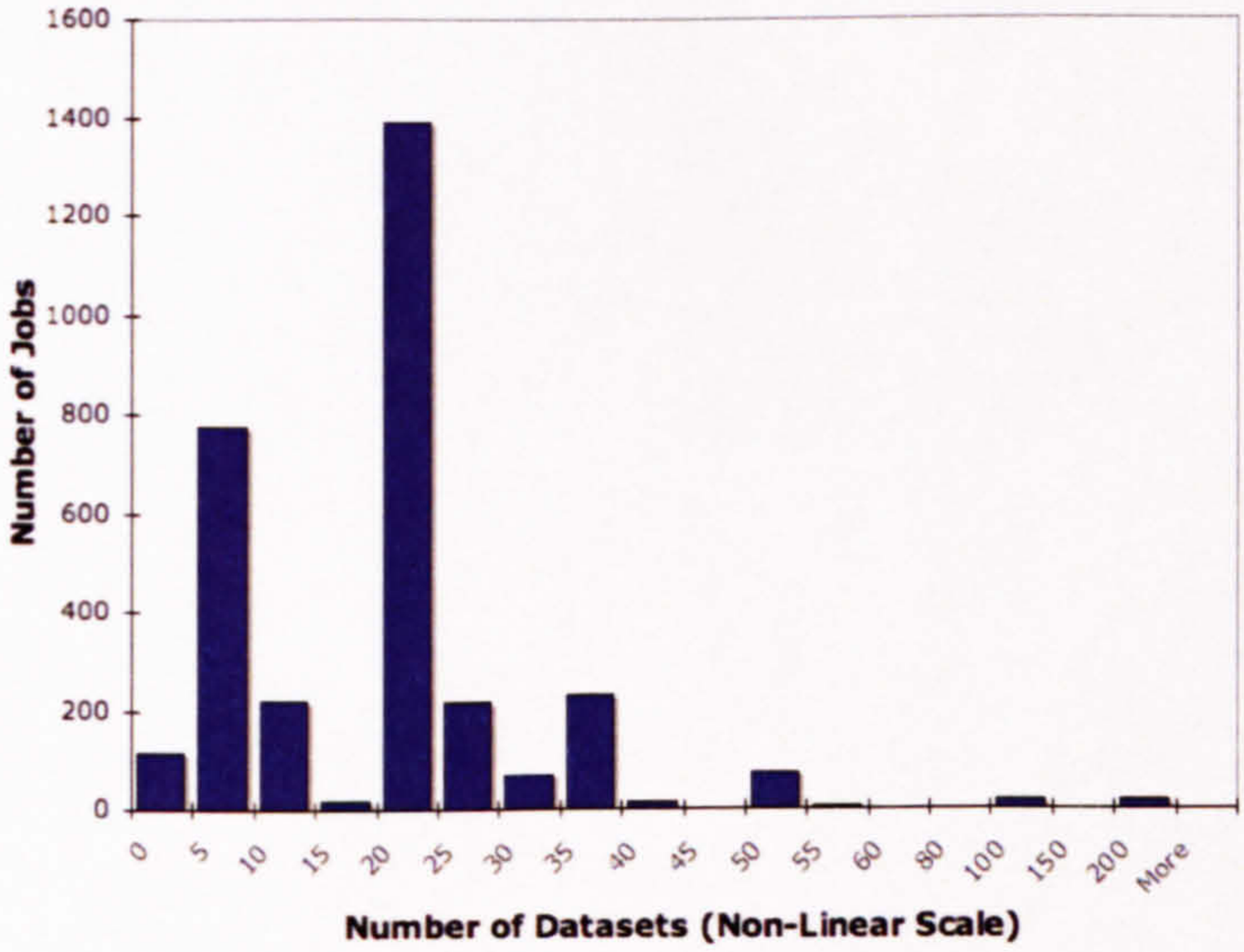

Figure 6.13: *Number of datasets submitted per job for over 3000 real user distributed analysis jobs collected over a six month period on the DIRAC Analysis System.*

6.6 User Experience

Outwith the sample considered in Section 6.4, users have gained experience of the system through individual use. Some of their results will be explored in this section. Due to the extended time period considered in Section 6.4.4, and the coincidence of analysis with production jobs during DC06, the performance is of lower quality than that found from the tests in Chapter 5.

Figure 6.14 shows the results of tests[2] [187] performed before the start of DC06, with Ganga submitting to the WMS via the DIRAC API. This analysis comprised of 500 jobs running over a total of 5000 input datasets, which corresponds to 5 million events.

As Figure 6.14 shows, 90% of results were back in under 3 hours. After

[2]Testing performed by Dr. Ulrik Egede, Imperial College, London.

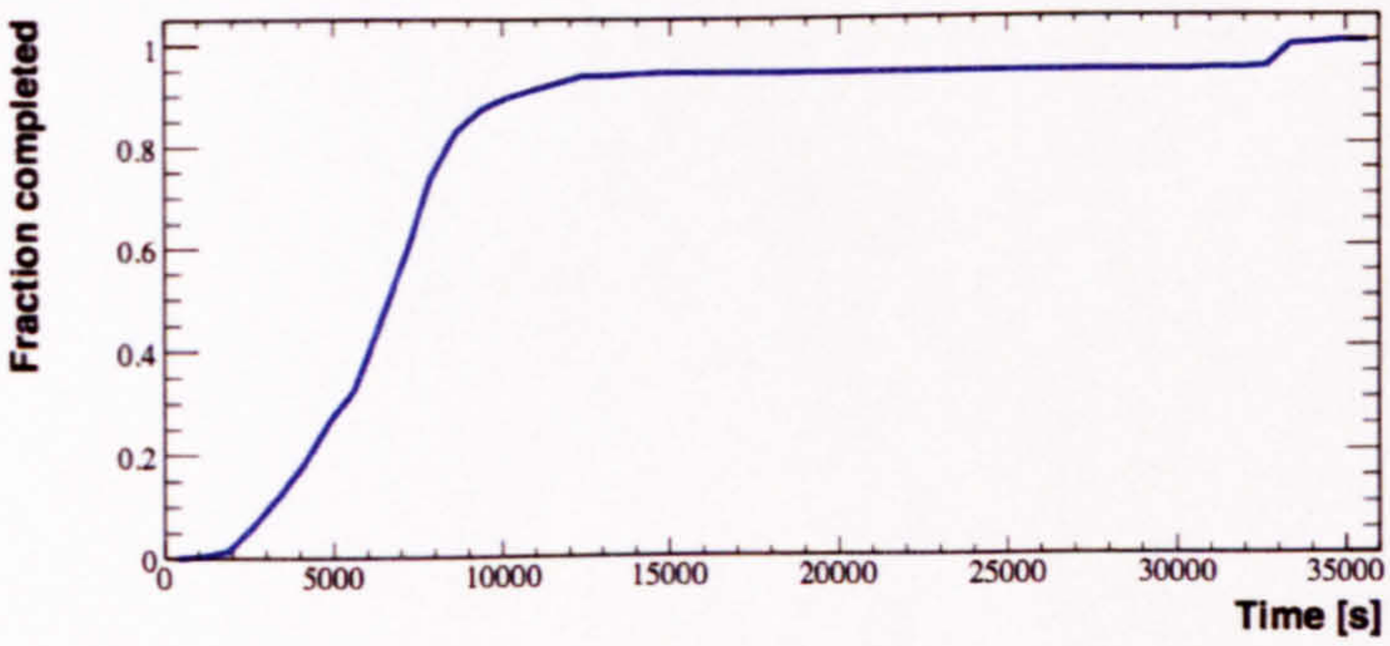

Figure 6.14: *Results of 500 jobs running over a total of 5000 datasets submitted to the WMS via the DIRAC API using Ganga. This analysis ran over a total of 5 million events, from [187].*

four hours this rose to 95% with the last 5% completing after 10 hours. The delay of the last 5% of jobs was caused by temporary file access problems at one of the LHCb Tier-1 sites. The efficiency of the sample, shown in Figure 6.15, is defined to encompass the submission, running, and retrieval of output, through the DIRAC API, all working as expected.

The job completion efficiency, shown in Figure 6.15, was measured at 95% with the remaining 5% due to inconsistencies in the LFC (similar to the observation in Section 6.4.4). For the last 5%, the jobs failed meaningfully before any Pilot Agents were submitted to the Grid. From the perspective of DIRAC, this test can be seen as 100% efficient since after the LFC inconsistencies are resolved, the remaining jobs are free to execute.

Another study was carried out by five University of Cambridge summer students [188], using Ganga to submit jobs via the DIRAC API to the WMS. With no prior knowledge of Grid computing, the summer students successfully utilised the Ganga-DIRAC system to submit jobs to LCG in a transparent way. Over a period of three months, 75 million events were processed with around 5,500 jobs submitted to the system. This yielded similar

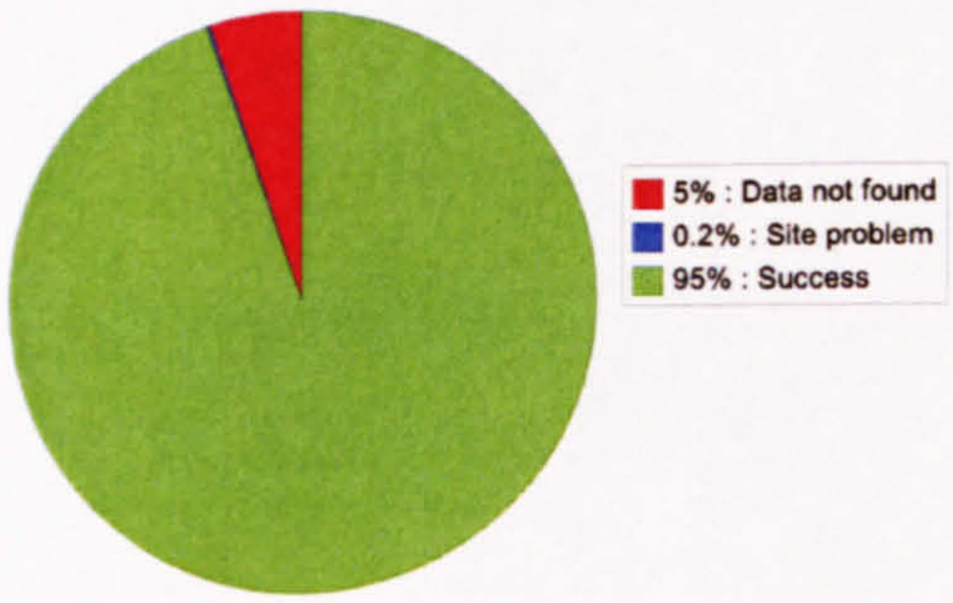

Figure 6.15: *Efficiency of 500 jobs running over a total of 5000 datasets submitted to the WMS via the DIRAC API using Ganga. This analysis processed a total of 5 million events, from [187].*

results with a final job success rate of 95% after taking into account errors such as LFC inconsistencies.

6.7 Maintenance of Service

This section will present an overview of the issues relating to the maintenance of the DIRAC analysis service. As mentioned in Chapter 4, services in the DIRAC WMS are managed via the *runit* tools, which ensure a high degree of reliability. Occasionally however, power cuts can affect the hosting machine to such an extent that *runit* cannot recover normal service completely. Currently, the DIRAC WMS will remain in this state until maintenance is performed 'by hand'. One option to consider is WMS 'mirroring' where a backup instance of key central services could come into play only when the primary instance is unavailable. A backup instance of the WMS has not been required thus far, so development on this is pending until deemed necessary.

As mentioned in Section 6.3.4, CRLs must be maintained in order to preserve access to LCG resources. The current mechanism means that all

DIRAC installations must have access to the latest CRL in order to perform Grid-based operations. For DIRAC public installations, *e.g.* on LXPLUS, there is a centrally updated CRL. However this becomes more cumbersome for individual users and in both cases can result in obscure failures when an out of date version is being used.

The issue of proxy expiration was touched upon in Section 6.4.4. A solution to renew short-term proxies has been attempted through MyProxy Server [49]. However this mechanism places an extra burden on users in order to get started on the Grid. The temporary solution to 'pipe' longer term proxies with jobs has been sufficient up to this point for distributed analysis jobs to solve this problem. However, a more formal mechanism should be established in the future. One option would be for the DIRAC *Job Wrapper* to monitor the remaining time on the proxy of a running job. A request to the WMS for an extended proxy could then be made using only the existing proxy as a credential. This could be fully automated and would shield users from the problem. However, this may duplicate the functionality of MyProxy Server and could result in fresh security concerns. If negligible start times could be guaranteed for all jobs, perhaps proxy expiration would not be as pressing an issue.

6.7.1 User Training

User training is one of the most important aspects of maintaining a service for a community. An LHCb software training course was held at the University of Cambridge in February 2006 [115]. The participants included LHCb collaboration members and the five summer students whose results were described in Section 6.6. This was essential to encourage users to use the Grid whilst also building confidence in the system. The format of the

training course involved an equal balance of lectures and practical activities. This resulted in users with no prior Grid knowledge successfully running jobs on LCG. Encouraging feedback was also gathered from users about the Ganga-DIRAC system.

6.8 Outlook

The project to develop the DIRAC Monte Carlo Production system for users to perform distributed analysis on the LHC Computing Grid has been very successful, and a working system has been released and is under use. However, there is still much work to be done and some possible future developments are described in this section.

As the number of users increases, so too does the number of use cases which must be accommodated by the system. One such example is Event Tag Collections (ETCs) that require reliable yet sparse access to many more datasets than standard analysis jobs. Access could be governed by a POOL XML slice, similar to that currently used to resolve input data, as described in Section 5.3.1, although extra mechanisms should be in place to ensure a high degree of success through redundancy, and a means should be introduced to handle ETCs during job submission.

Another emerging issue from the experience of users in a Grid environment is the desire to run application software outwith that provided as standard by the VO. Although the current mechanism for software distribution scales well for the Gaudi-based applications of LHCb, it may be necessary in the future to support additional software. One example would be supporting stand-alone ROOT. The LHCb software depends on a particular version of ROOT. However, there is no current support for running ROOT on its

own. A centrally maintained Pacman cache could be one solution to support additional software in the future.

As the start of the LHC approaches, the number of users, but also the frequency of usage, is set to increase significantly. In the saturated regime, where production and analysis jobs are competing for available Grid resources, it becomes paramount to establish effective priority mechanisms. In parallel with the increase in users will be an increase in the number of submitted jobs. Therefore, it is also essential to have a management system for user storage quotas on the Grid. For LHCb, policies for both these cases can naturally be applied inside the DIRAC WMS.

The arguments for LHCb VO generic Pilot Agents have been made through the exploration of optimisation strategies in Chapter 5, and the effect of the DC06 activity on system performance in Section 6.4.5. If this becomes available in the future, only minor modifications would have to be made to the system in order to reap the potential benefits.

6.9 Summary

In this chapter, the current and future deployment strategies of DIRAC were explored in Section 6.2. This was followed by an introduction to the DIRAC API in Section 6.3 by which users, including the Ganga Grid front-end, interact with the system to perform distributed data analysis for LHCb. The system performance results from a six month period were presented in Section 6.4, highlighting factors which become significant over long periods of time such as power failures and network outages. This also showed that the DC06 activity had an adverse effect on system performance with production and analysis jobs competing for the same resources.

The patterns of user analysis were discussed in Section 6.5, and some real user experience mentioned in Section 6.6. Issues with the maintenance of the system were mentioned in Section 6.7 and future developments were discussed in Section 6.8. The DIRAC infrastructure for supporting distributed analysis activitites in LHCb is in place. Real users are starting to utilise and, more importantly, benefit from the system.

Chapter 7

Conclusions

High energy physicists are driving much of the development of the computing Grid. When the detectors of the LHC experiments begin taking data it will become essential to have reliable and secure access to this data. Furthermore, this data will undergo further re-processing as the reconstruction and analysis requirements of the experiment evolve. Distributed data analysis immediately becomes a high priority task and is essential to the success of the LHC experiments.

Since no single institute has the compute resources to handle the unprecedented amount of data on the Petabyte scale from the LHC, resources must be pooled to form the Grid. Grid technology aims to provide seamless access to computing power and storage capacity across the world. It is the task of Grid middleware to present a uniform view of heterogeneous compute systems to the experiments software as well as transparent access to available resources. Issues such as where jobs run, or on which storage elements data resides, should be masked from users. The start time of user jobs should also be optimised to ensure results are returned to users as soon as possible.

Perhaps the first item to consider when running software applications

on the Grid is how to efficiently distribute the software. The mechanism introduced to DIRAC and described in this thesis, has proven to be very efficient for LHCb software applications and presented a more flexible option than the alternative Pacman approach also considered here. It remains to be seen whether another mechanism is necessary for supporting software outwith the main LHCb data-processing applications.

The EGEE gLite framework prototype was used to perform the first real physics analysis. Although the prototype was in its infancy during this testing, performing user analysis for LHCb was possible. However, this required additional effort from the user when compared to standard batch systems. At the present time, LCG middleware is the basis of all Grid production and analysis for the LHC experiments.

The development of DIRAC for distributed analysis in LHCb has provided a stable and efficient framework for researching and exploiting the possibilities of the Grid for data analysis. The paradigms for distributed analysis such as *PULL* scheduling and the Pilot Agent approach, realised by DIRAC, have proven to be highly successful. For example, this has allowed the principles of workload management to be applied not only at the time of user job submission to the Grid but also to optimise the use of computing resources once jobs have been acquired. The investigation of workload management strategies showed that it is possible to achieve a negligible start time for higher priority distributed data analysis jobs on LCG. In the saturated regime, where no resources are available, this may depend on the possibility of sending generic LHCb VO Pilot Agents or switching the identity of jobs on the WN in the future.

Results from real users show that system performance can be affected when in direct competition with the ongoing DC06 production activities.

It was found that the efficiency of jobs over an extended period of time was 91%, although higher efficiencies have been observed in shorter time frames by individual users. The DIRAC system is now the default mode of submission to the Grid for all LHCb user jobs, and usage is set to increase in the near future. The real test will begin when the first data from the detector becomes available and it is important that users build confidence in the system beforehand. In order to achieve this user training sessions have been held and future sessions are planned.

The DIRAC system requires further development in order to cope with use cases such as Event Tag Collections. The system must also be capable of applying a priority mechanism to ensure fair sharing of LHCb Grid resources. A management system for user storage quotas on the Grid must also be introduced in order to cope with user output data. While future improvements will undoubtedly occur, real LHCb users are already starting to benefit from the use of DIRAC to perform their distributed data analysis tasks on LCG. Furthermore, the extension of DIRAC to accommodate the distributed data analysis tasks, has begun to unlock the potential of the Grid for LHCb users.

Appendix A

Glossary

This appendix contains an alphabetical list of all acronyms used throughout the thesis, their descriptions and the page number on which they were first used. Since the use of acronyms is prevalent in the field of Grid computing, this is designed to improve readability.

ACL	Access Control Lists, 35
ALICE	A Large Ion Collider Experiment, 37
AliEn	ALICE Environment, 87
API	Application Programming Interface, 31
ARC	Advance Resource Connector, 25
ARDA	A Realisation of Distributed Analysis, 88
ATLAS	A Toroidal LHC ApparatuS, 37
BDII	Berkeley Database Information Index, 30
BOINC	Berkeley Open Infrastructure for Network Computing, 11
BOSS	Batch Object Submission System, 86

CA Certification Authority, 18

CAF CDF Central Analysis Farm, 166

CDF Collider Detector at Fermilab, 166

CE Computing Element, 28

CERN European Organisation for Nuclear Research, 37

CKM Cabibbo Kobayashi Maskawa, 40

ClassAds Classified Advertisements, 110

CMS Compact Muon Solenoid, 37

ConDB Conditions Database, 48

CPU Central Processing Unit, 11

CRL Certificate Revocation List, 27

CS Configuration Service, 116

DAG Directed Acyclic Graph, 139

DC04 Data Challenge 2004, 56

DC06 Data Challenge 2006, 108

DCAP Data Link Switching Client Access Protocol, 145

DIAL Distributed Interactive Analysis of Large datasets, 85

DIRAC Distributed Infrastructure with Remote Agent Control, 56

DISET DIRAC Secure Transport, 110

DLI Data Location Interface, 35

DLLs Dynamically Linked Libraries, 50

DN Distinguished Name, 27

DNS Domain Name System, 10

DRS Data Replication Service, 20

DST Data Summary Tape, 49

ECAL Electromagnetic CALorimeter, 43

EGA Enterprise Grid Alliance, 16

EGEE Enabling Grids for E-sciencE, 9

ETC Event Tag Collection, 53

FIFO First In First Out, 126

FTS File Transfer Service, 120

Ganga Gaudi / Athena and Grid Alliance, 182

GASS Global Access to Secondary Storage, 164

GFAL Grid File Access Library, 35

GGF Global Grid Forum, 16

GIIS Grid Index Information Service, 19

GLUE Grid Laboratory Uniform Environment, 29

GRAM Grid Resource Allocation Manager, 20

GridFTP Grid File Transfer Protocol, 20

GriPhyN Grid Physics Network, 24

GRIS Grid Resource Information Server, 30

GSI Grid Security Infrastructure, 18

GT Globus Toolkit, 17

GUI Graphical User Interface, 85

GUID Globally Unique IDentifier, 34

HCAL Hadron CALorimeter, 43

HEP High Energy Physics, 1

HLT High Level Trigger, 44

HTML Hyper-Text Markup Language, 10

HTTP Hyper-Text Transfer Protocol, 59

I/O Input/Output, 151

iVDGL International Virtual Data Grid Laboratory, 24

IVOA International Virtual Observatory Alliance, 22

JDL Job Description Language, 31

JobDB Job Database, 123

L0 Level 0, 44

LAN Local Area Network, 10

LCG LHC Computing Grid, 2

LCG IS LCG Information System, 29

LCG UI LCG User Interface, 180

LCG WMS LCG Workload Management System, 31

LDAP Lightweight Directory Access Protocol, 29

LFC LCG File Catalogue, 33

LFN Logical File Name, 33

LHC Large Hadron Collider, 2

LHCb Large Hadron Collider beauty, 36

MC Monte Carlo, 51

MDS Monitoring and Discovery Service, 19

MSS Mass Storage System, 28

OGF Open Grid Forum, 15

OGSA Open Grid Services Architecture, 16

OGSI Open Grid Services Infrastructure, 16

OSG Open Science Grid, 25

P2P Peer-to-peer, 10

Panda Production ANd Distributed Analysis, 86

PC Personal Computer, 6

PDC1 Physics Data Challenge, 107

PFN Physical File Name, 33

PKI Public Key Infrastructure, 18

POOL Pool Of persistent Objects for LHC, 146

POSIX Portable Operating System Interface, 28

PPDG Particle Physics Data Grid, 25

PS Preshower Detector, 43

R-GMA Relational Grid Monitoring Architecture, 30

RB Resource Broker, 31

rDST reduced Data Summary Tape, 49

RFIO Remote File Input/Output, 145

RICH Ring Imaging CHerenkov, 42

RM Replica Manager, 131

RPC Remote Procedure Call, 109

RTTC Real Time Trigger Challenge, 114

SE Storage Element, 28

SETI Search for Extra-Terrestial Intelligence, 11

SM Standard Model, 37

SOA Service Oriented Architecture, 103

SOAP Simple Object Access Protocol, 109

SPD Scintillator Pad Detector, 43

SRM Storage Resource Manager, 28

SSL Secure Socket Layer, 110

SURL Storage URL, 33

TCP/IP Transmission Control Protocol / Internet Protocol, 10

TDS Transient Detector Store, 47

TES Transient Event Store, 47

THS Transient Histogram Store, 47

TT Trigger Tracker, 42

TURL Transport URL, 33

URL Uniform Resource Locator, 10

VDT Virtual Data Toolkit, 24

VELO VErtex LOcator, 42

VO Virtual Organisation, 2

VOMS Virtual Organisation Membership Service, 27

W3C World Wide Web Consortium, 15

WAN Wide Area Network, 10

WMS Workload Management System, 75

WN Worker Node, 31

WSDL Web Services Description Language, 15

WSRF Web Services Resource Framework, 17

WWW World Wide Web, 1

XML eXtensible Markup Language, 15

XML-RPC XML Remote Procedure Call, 109

Appendix B

Complicated Workflows with the DIRAC API

This appendix contains a script used to perform a private user production job using the DIRAC Job, Step and Module infrastructure through the DIRAC API. The following script creates the workflow outlined in Figure 5.2 where <VERSION> should be replaced by appropriate values. The commands within triple quotes are appended to the respective options files before execution of the application. Sample input and output file names are added to illustrate the processing chain.

```
from DIRAC.Client.Dirac import *

dirac = Dirac()
job = Job()
step = Step()

step.setApplication('Gauss','<VERSION>')
step.setInputSandbox(['Gauss.opts'])
step.setOutputSandbox(['Gauss_<VERSION>.log','GaussHistos.root'])
step.setOutputData(['Gauss.sim'])
step.setOption("""
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','<VERSION>')
step2.setInputSandbox(['Boole.opts'])
step2.setOutputSandbox(['Boole_<VERSION>.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','<VERSION>')
step3.setInputSandbox(['Brunel.opts'])
step3.setOutputSandbox(['Brunel_<VERSION>.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)
print "Job ID = ",jobid
```

Appendix C

DIRAC Job State Machine

This appendix illustrates the DIRAC job state machine[1]. The state machine

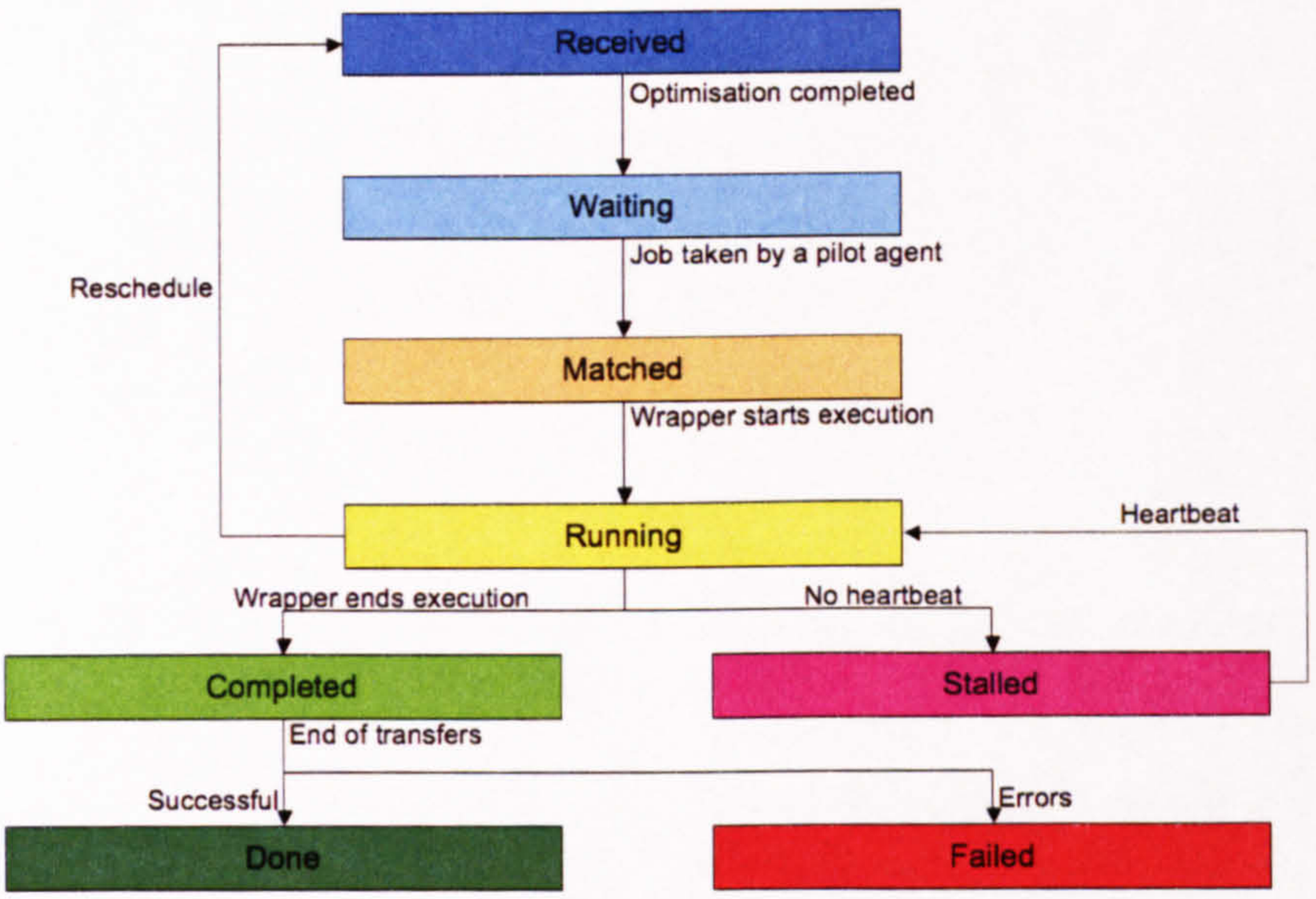

Figure C.1: *Primary job states in the DIRAC status machine where arrows indicate the possible transitions.*

has been designed to be very refined in order to aid in the debugging of Grid

[1]Thanks to Dr. Philippe Charpentier for compiling Figures C.1 and C.2.

jobs. This also serves to improve redundancy since particular causes of failure can be identified and acted upon. DIRAC jobs have both a primary and secondary job state. Figure C.1 outlines the primary job states where 'Received' indicates initial submission to the WMS. The secondary job states are shown in Figure C.2. Rather than explaining each individual state, Figures C.1 and C.2 are included simply to illustrate the many possible outcomes of running Grid jobs, as well as the importance of correctly identifying each case. Secondary job states present a more fine-grained view of how the job

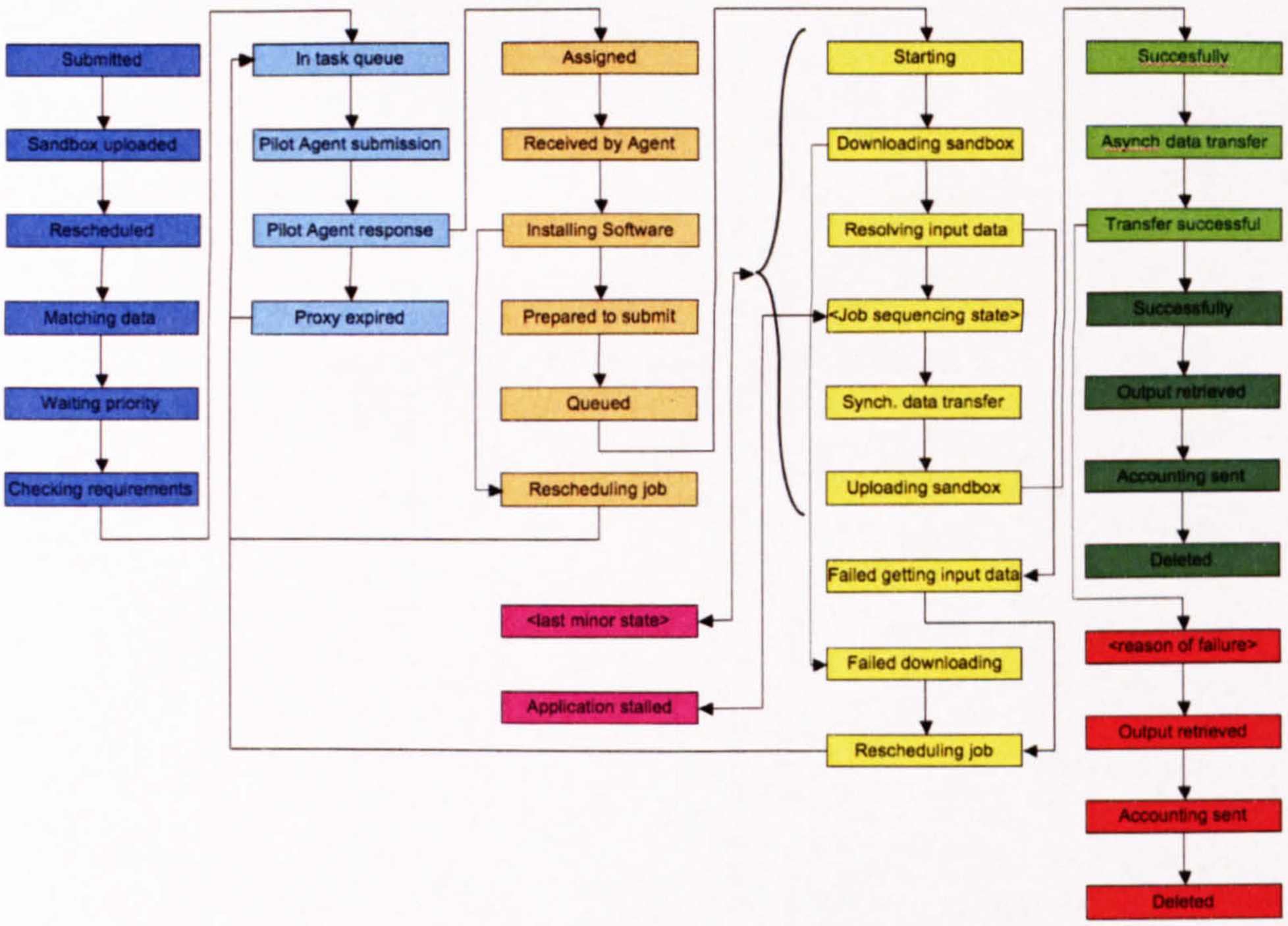

Figure C.2: *Secondary job states in the DIRAC status machine where arrows indicate the possible transitions. The colours used reflect corresponding primary job states in Figure C.1.*

is proceeding. This includes, for example, the activity of the DIRAC *Job Wrapper*. Both primary and secondary states are reported during the life-

time of a job to the *Job Monitoring Service* and this information is also used to construct the LHCb DIRAC Monitoring pages [168].

References

[1] M. Livny. *Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems.* PhD thesis, Weizmann Institute of Science, August 1983.

[2] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1998.

[3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum, 2002.

[4] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[5] The LHC computing grid project - LCG. `http://lcg.web.cern.ch/LCG/`.

[6] I. Foster. What is the Grid? A Three Point Checklist. *GRIDtoday*, 1(6), 2002.

[7] IBM. `http://www.ibm.com`.

[8] IBM Corporation. IBM Solutions Grid for Business Partners. Helping IBM Business Partners to Grid-enable applications for the next phase of e-business on demand, 2002.

[9] Sun Microsystems. http://www.sun.com/service/grid/.

[10] Microsoft. http://www.microsoft.com.

[11] R. Buyya. Grid Computing Info Centre (GRID Infoware). http://www.gridcomputing.com.

[12] Z. Nemeth and V. Sunderam. Characterizing Grids: Attributes, Definitions and Foundations. *Journal of Grid Computing*, 1(1):9-23, 2003.

[13] W.E. Johnston. A Different Perspective on the Question of What is a Grid? *GRIDtoday*, 1(9), 2002.

[14] L. Kleinrock. Quoted in UCLA Press Release. http://www.lk.cs.ucla.edu/LK/Bib/REPORT/press.html, July 1969.

[15] M. Greenberger. The Computers of Tomorrow. *The Atlantic*, May 1964.

[16] M. Chetty and R. Buyya. Weaving Computational Grids: How Analogous Are They with Electrical Grids? *Computing in Science and Engineering*, 4(4):61–71, 2002.

[17] Enabling Grids for E-sciencE (EGEE) project page. http://www.eu-egee.org/.

[18] J. M. McQuillan and D. C. Walden. The ARPANET Design Dccisions. *Computer Networks*, 1(5), August 1977.

[19] V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, 1974.

[20] Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, 19 (5):395–405, July 1976.

[21] P. Mockapetris. Domain Names - Concepts and Facilities. *RFC 1034*, November 1987.

[22] T. Berners-Lee, T. Bray, D. Connolly, P. Cotton, R. Fielding, M. Jeckle, C. Lilley, N. Mendelsohn, D. Orchard, N. Walsh, and S. Williams. Architecture of the World Wide Web, Volume One. *W3C*, (Version 20041215), December 2004.

[23] M. Mutka and M. Livny. The Available Capacity of a Privately Owned Workstation Environment. *Performance Evaluation*, 12,:269–284, 1991.

[24] Entropia. `http://www.entropia.com`.

[25] Entropia. Researchers discover largest multi-million-digit prime using entropia distributed computing grid. Press release, Entropia, Inc., December 2001.

[26] SETI@home. `http://setiathome.berkeley.edu/`.

[27] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[28] BOINC. `http://boinc.berkeley.edu`.

[29] David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, pages 4–10, 2004.

[30] Folding@home. `http://folding.stanford.edu/`.

[31] Compute Against Cancer. `http://computeagainstcancer.org/`.

[32] Fight AIDS@home. `http://fightaidsathome.scripps.edu/`.

[33] Einstein@home. `http://einstein.phys.uwm.edu/`.

[34] LHC@home. `http://athome.web.cern.ch`.

[35] Parabon. `http://www.parabon.com`.

[36] United Devices. `http://www.uniteddevices.com`.

[37] BitTorrent. `http://www.bittorrent.com/`.

[38] Napster. `http://www.napster.com/`.

[39] Gnutella. `http://www.gnutella.com`.

[40] The Free Network Project. `http://freenetproject.org/`.

[41] C. Catlett and L. Smarr. Metacomputing. *Communications of the ACM*, 35:44–52, June 1992.

[42] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11:115–128, 1997.

[43] The World Wide Web Consortium (W3C). `http://www.w3c.org`.

[44] The Open Grid Forum (OGF). `http://www.ogf.org/`.

[45] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm., D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructue (OGSI) Version 1.0. Technical report, Global Grid Forum Drafts, 2003.

[46] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, L. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling Stateful Resources with Web Services. Whitepaper, May 2004.

[47] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution. Whitepaper, March 2004.

[48] The Globus Alliance. `http://www.globus.org`.

[49] James Basney, Marty Humphrey, and Von Welch. The MyProxy Online Credential Repository. In *Software: Practice and Experience*, volume 35(8), 2005.

[50] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol Extension to FTP for the Grid. Global Grid Forum Internet-Draft, March 2001.

[51] The Virtual Laboratory Project. `http://www.gridbus.org/vlab/`.

[52] Rajkumar Buyya. The Virtual Laboratory Project. *IEEE Distributed Systems Online*, 2(5), 2001.

[53] Earth System Grid (ESG). `http://www.earthsystemgrid.org/`.

[54] Grid Enabled Optimisation and Design Search for Engineering (GEODISE). http://www.geodise.org/.

[55] International Virtual Observatory Alliance (IVOA). http://www.ivoa.net/.

[56] AstroGrid. http://www.astrogrid.ac.uk/.

[57] HealthGRID. http://www.healthgrid.org/.

[58] Gridwise Tech. http://gridwisetech.com/.

[59] GridSystems. http://www.gridsystems.com/.

[60] GridPP. http://www.gridpp.ac.uk/.

[61] The GridPP Collaboration. GridPP: development of the UK computing Grid for particle physics. In *J. Phys. G: Nucl. Part. Phys.*, 32 (2006) N1-N20 doi:10.1088/0954-3899/32/1/N01.

[62] GriPhyN - Grid Physics Network. http://www.griphyn.org/.

[63] iVDGL - International Virtual Data Grid Laboratory. http://www.ivdgl.org/.

[64] Particle Physics Data Grid (PPDG). http://www.ppdg.net/.

[65] NorduGrid. http://www.nordugrid.org/.

[66] Open Science Grid. http://www.opensciencegrid.org/.

[67] The gLite project page. http://glite.web.cern.ch/glite/.

[68] The LCG Project. LHC Computing Grid. Technical Design Report CERN-LHCC-2005-024, CERN, June 2005.

[69] Virtual Organization Membership Service (VOMS). `http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html`.

[70] R. Alfieri, Roberto Cecchini, Vincenzo Ciaschini, Luca dell'Agnello, Ákos Frohner, A. Gianoli, Károly Lörentey, and Fabio Spataro. VOMS, an *Authorization* System for Virtual Organizations. In *Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2003, Revised Papers*, pages 33–40, 2003.

[71] Storage Resource Management Working Group. `http://sdm.lbl.gov/srm-wg/`.

[72] The Condor Project. `http://www.cs.wisc.edu/condor/`.

[73] S. Andreozzi. GLUE Schema Implementation for the LDAP Data Model. Technical Report INFN/TC-04/16, INFN, September 2004.

[74] The EGEE Collaboration. EGEE Information and Monitoring Service (R-GMA): System Specification. Technical Report EGEE-JRA1-TEC-490223-R_GMA_SPECIFICATION-v2-r0, CERN, July 2004.

[75] Nicholas Coleman, Rajesh Raman, Miron Livny, and Marvin Solomon. Distributed Policy Management and Comprehension with Classified Advertisements. Technical Report UW-CS-TR-1481, University of Wisconsin - Madison Computer Sciences Department, 2003.

[76] EDG Work Package 1. WP1 - WMS Software Administrator and User Guide. Technical Report DataGrid-01-TEN-0118-1_2, CERN, 2003.

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

[78] ORACLE. http://www.oracle.com.

[79] MySQL. http://www.mysql.com/.

[80] CERN, the European Organisation for Nuclear Research. http://www.cern.ch/.

[81] ALICE - A Large Ion Collider Experiment. http://aliceinfo.cern.ch/.

[82] ATLAS - A Toroidal LHC ApparatuS. http://atlasexperiment.org/.

[83] CMS - Compact Muon Solenoid. http://cmsinfo.cern.ch/.

[84] LHCb - The Large Hadron Collider beauty experiment. http://cern.ch/lhcb/.

[85] S.Amato et al. LHCb technical proposal. CERN-LHCC/98-4, 1998.

[86] R. Antunes Nobrega et al. LHCb Reoptimized Detector Design and Performance TDR. CERN-LHCC/2003-030, 2003.

[87] BaBar experiment. http://www.slac.stanford.edu/BFROOT/.

[88] Belle experiment. http://belle.kek.jp/.

[89] Fermilab Tevatron. http://www-bdnew.fnal.gov/tevatron/.

[90] D0 Collaboration, V. Abazov, et al. Direct Limits on the Bs Oscillation Frequency. In *Phys. Rev. Lett.*, 97, (2006) 021802.

[91] CDF Collaboration, A. Abulencia, et al. Evidence for the exclusive decay Bc+- to J/psi pi+- and measurement of the mass of the Bc meson. In *Phys.Rev.Lett.*, 96 (2006) 082002.

[92] LHCb Collaboration. LHCb Trigger TDR. Technical Report CERN/LHCC 2003-031, CERN, 2003.

[93] LHCb Collaboration. LHCb VELO TDR. Technical Report CERN-LHCC-2001-011, CERN, 2001.

[94] LHCb Collaboration. LHCb Magnet TDR. Technical Report CERN/LHCC 2000-007, CERN, 2000.

[95] LHCb Collaboration. LHCb Outer Tracker TDR. Technical Report CERN-LHCC-2001-024, CERN, 2001.

[96] LHCb Collaboration. LHCb Inner Tracker TDR. Technical Report CERN-LHCC-2002-029, CERN, 2002.

[97] LHCb Collaboration. LHCb RICH TDR. Technical Report CERN-LHCC-2000-037, CERN, 2000.

[98] LHCb Collaboration. LHCb Calorimeter TDR. Technical Report CERN/LHCC 2000-036, CERN, 2000.

[99] LHCb Collaboration. LHCb Muon System TDR. Technical Report CERN-LHCC-2001-010, CERN, 2001.

[100] LHCb Collaboration. LHCb Computing TDR. Technical Report CERN/LHCC 2005-119, CERN, 2005.

[101] LHCb software architecture group. GAUDI LHCb Data Processing Applications Framework. Architecture Design Document LHCb 98-064, CERN, 1998.

[102] R. Brun and F. Rademakers. ROOT - An object oriented analysis framework. *Nuc.Inst.Meth. in Phys. Res A*, 389 (1997) 81.

[103] D. Duelmann et al. The LCG POOL development and production experience. In *IEEE-NSS Rome2004*, October 2004.

[104] M. Cattaneo et al. The new LHCb Event Data Model. LHCb-2001-142, 2001.

[105] Geant 4. `http://geant4.web.cern.ch/geant4/`.

[106] Ian Foster and Carl Kesselman, editors. *The Grid 2 : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 2004.

[107] M. Aderholz et al. Models of *Networked* Analysis at Regional Centres for LHC Experiments (MONARC). Phase 2 Report CERN/LCB 2000-001, CERN, 2000.

[108] J. Closier et al. Results of the LHCb Data Challenge 2004. In *CHEP04*, 2004.

[109] Xen virtual machine monitor. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`.

[110] S. Youssef. Pacman homepage. `http://physics.bu.edu/~youssef/pacman/`.

[111] S. Paterson. Software Distribution for LHCb using Pacman. LHCb Note 2004-066, August 2004.

[112] RPM Package Manager. `http://www.rpm.org/`.

[113] Relink Project Homepage. `http://sourceforge.net/projects/relink/`.

[114] LHCb Installation Procedures. http://lhcb-comp.web.cern.ch/lhcb-comp/Support/html/NEWInstall.htm.

[115] LHCb-UK Software Training. http://www.hep.phy.cam.ac.uk/lhcb/LHCbSoftTraining/2006/.

[116] LHCb Software Distribution Tool. http://lhcbproject.web.cern.ch/lhcbproject/dist/distribution.html.

[117] CMT homepage. http://www.cmtsite.org.

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

[119] I. Stokes-Rees et al. Developing LHCb Grid Software: Experiences and Advances. In *UK e-Science All Hands Meeting 2004*, September 2004.

[120] G. Ganis et al. PROOF - The Parallel ROOT Facility. In *CHEP06*, 2006.

[121] DIAL. http://www.usatlas.bnl.gov/~dladams/dial/design/.

[122] D. Adams, T. Maeno, K. Harrison, G. Rybkine, and D. Liko. DIAL: Distributed Interactive Analysis of Large Datasets. In *CHEP06*, 2006.

[123] D.E. Kaushik et al. Panda: Production and Distributed Analysis System for ATLAS. In *CHEP06*, 2006.

[124] C. Grandi et al. Evolution of BOSS, a tool for job submission and tracking. In *CHEP06*, 2006.

[125] ALICE Collaboration. ALICE Computing TDR. Technical Report CERN-LHCC-2005-018, CERN, 2005.

[126] ALICE Collaboration. ALICE Computing Model. Technical Report CERN-LHCC-2004-038, CERN, 2005.

[127] The ARDA Project. http://lcg.web.cern.ch/LCG/peb/arda/Default.htm.

[128] S. Paterson. Distributed Analysis Using DaVinci in the gLite Framework. LHCb-2005-057, August 2005.

[129] DaVinci - the LHCb analysis program. http://lhcb-comp.web.cern.ch/lhcb-comp/Analysis/default.htm.

[130] EGEE. EGEE Middleware Architecture. EU Deliverable DJRA1.1 EGEE-DRJA1.1-476451-v1.0, 2004.

[131] EGEE. Design of the EGEE Middleware Grid services. EU Deliverable DJRA1.2 EGEE-DRJA1.2-487871-v1.0, 2004.

[132] AliEn homepage. http://alien.cern.ch/.

[133] G. Raven. LHCb technical note. LHCb-2003-118, 2003.

[134] G. Raven. LHCb technical note. LHCb-2003-119, 2003.

[135] The Gaudi Framework. http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/Gaudi/.

[136] LHCb core packages. http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/LHCbSys/default.htm.

[137] Magnetic Field Map. http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/default.htm.

[138] Parameter files. http://lhcb-comp.web.cern.ch/lhcb-comp/.

[139] Xml Detector Description Database. http://lhcb-comp.web.cern.ch/lhcb-comp/Frameworks/DetDesc/default.htm.

[140] gLite Prototype Testbed. http://egee-jra1.web.cern.ch/egee-jra1/Prototype/testbed.htm.

[141] Architectural Roadmap towards Distributed Analysis. Technical Report CERN-LCG-2003-033, CERN, 2003.

[142] J. Andreeva et al. The ARDA Prototypes. In *CHEP04*, 2004.

[143] B. Koblitz et al. First Experience with the EGEE Middleware. In *CHEP04*, 2004.

[144] A.S. Dighe. Proceedings of UK Phenomenology Workshop on Heavy Flavours and CP Violation. In *J. Phys. G: Nucl. Part. Phys. 27*, pages p1341–1344, 2001.

[145] Gaudi/Athena Job Options Editor (JOE). http://ganga.web.cern.ch/ganga/user/v2/JOE/UserManual.html.

[146] gLite Prototype Documentation. http://egee-jra1.web.cern.ch/egee-jra1/Prototype/Documentation/glite_tutorial.html.

[147] P. Saiz. AliEn documentation. http://alien.cern.ch/Alien/main?task=doc\&section=PackMan.

[148] CASTOR project. http://castor.web.cern.ch/castor/.

[149] J. P. Baud, P. Charpentier, J. Closier, R. Graciani, A. Maier, and P. Mato-Vila. DIRAC Review Report. Technical Report LHCb-2006-04 COMP, CERN, 2006.

[150] T. Wenaus et al. Architecture Blueprint RTAG Report. Technical Report CERN-LCGAPP-2002-09, CERN, 2002.

[151] LHCb Bookkeeping Facility. `http://lbnts2.cern.ch/BkkWeb/Bkk/welcome.htm`.

[152] J. Closier, G. Kuznetsov, G. Patrick, and A. Tsaregorodtsev. DIRAC Production Manager Tools. In *CHEP06*, 2006.

[153] A. Tsaregorodtsev et al. DIRAC – Distributed Infrastructure with Remote Agent Control. In *CHEP03*, 2003.

[154] The DataGRID Project. `http://eu-datagrid.web.cern.ch/eu-datagrid`.

[155] XML-RPC Protocol. `http://www.xmlrpc.com/`.

[156] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. Technical report, W3C Note, 2000.

[157] A. Casajus Ramo and R. Graciani Diaz. DIRAC Security Infrastructure. In *CHEP06*, 2006.

[158] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, December 2000.

[159] I. Stokes-Rees, A. Tsaregorodtsev, and V. Garonne. DIRAC Lightweight Information and Monitoring Services Using XML-RPC and Instant Messaging. In *CHEP04*, September 2004.

[160] CERN CVS Repository. `http://www.cern.ch/cvs`.

[161] runit - a UNIX init scheme with service supervision. `http://smarden.org/runit/`.

[162] A. Tsaregorodtsev et al. DIRAC, the LHCb data production and distributed analysis system. In *CHEP06*, 2006.

[163] LHCb VO-box Requirements. `https://uimon.cern.ch/twiki/bin/view/LHCb/LHCbTaskForce#VO_Boxes`.

[164] R. Graciani Diaz and A. Casajus Ramo. Configuration Service Documentation. ECM-UB, June 2005.

[165] V. Garonne, A. Tsaregorodtsev, and I. Stokes-Rees. DIRAC: Workload Management System. In *CHEP04*, 2004.

[166] A. C. Smith and A. Tsaregorodtsev. LHCb Data Replication During SC3. In *CHEP06*, 2006.

[167] A. Tsaregorodtsev. DIRAC Data Management System. LHCb Technical Note, 2005.

[168] LHCb DIRAC Monitoring. `http://lhcb.pic.es/DIRAC/Monitoring`.

[169] dCache project. `http://www.dcache.org/`.

[170] POOL project. `http://lcgapp.cern.ch/project/persist/`.

[171] LSF. `http://www.platform.com`.

[172] Maui Scheduler. `http://www.supercluster.org/maui`.

[173] Condor-G. `http://www.cs.wisc.edu/condor/condorg/`.

[174] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[175] Douglas Thain and Miron Livny. Building Reliable Clients and Servers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[176] Douglas Thain, Todd Tannenbaum, and Miron Livny. *Condor and the Grid*, chapter Grid Computing: Making the Global Infrastructure a Reality, pages 299–335. John Wiley and Sons Inc., December 2002.

[177] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[178] James Frey, Todd Tannenbaum, Miron Livny, Ian T. Foster, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[179] The Collider Detector at Fermilab (CDF). `http://www-cdf.fnal.gov/`.

[180] CDF Central Analysis Farm (CAF). `http://cdfcaf.fnal.gov/`.

[181] S. Belforte, S-C. Hsu, E. Lipeles, M. Norman, F. Wurthwein, D. Lucchesi, S. Sarkar, and I. Sfiligoi. GlideCAF: A Late Binding Approach to the Grid. In *CHEP06*, 2006.

[182] M. Norman, S-C. Hsu, E. Lipeles, M. Neubauer, F. Wurthwein, I. Sfiligoi, and S. Sarkar. OSG-CAF - A single point of submission for CDF to the Open Science Grid. In *CHEP06*, 2006.

[183] Panda project page. `https://uimon.cern.ch/twiki/bin/view/Atlas/PanDA`.

[184] Don Quijote (DQ2) project page. `http://www.triumf.info/hosted/atlas-triumf/index.php/Don_Quijote`.

[185] Gaudi/Athena Grid Alliance. `http://ganga.web.cern.ch/ganga/`.

[186] U. Egede, K. Harrison, D. Liko, A. Maier, J. Moscicki, A. Soroko, and C. Tan. GANGA - a Grid user interface. In *CHEP06*, 2006.

[187] U. Egede, V. Garonne, A. Maier, J. Moscicki, S. Paterson, A. Soroko, and A. Tsaregorodtsev. Experience with distributed analysis in LHCb. In *CHEP06*, 2006.

[188] GridPP News. Cambridge undergraduates run LHCb analysis on the Grid. `http://www.gridpp.ac.uk/news/-1149863627.881156.wlg`, June 2006.