LHC*b* Data Management on the Computing Grid

Andrew Cameron Smith

Thesis submitted for degree of Doctor of Philosophy

The University of Edinburgh

Abstract

The LHC*b* detector is one of the four experiments being built to harness the proton-proton collisions provided by the Large Hadron Collider (LHC) at the European Organisation for Nuclear Research (CERN). The data rate expected, when the LHC experiments are fully operational, eclipses that of any previous scientific experiments and has motivated the adoption of a grid computing paradigm to store and process the data. Managing PetaBytes of data in a distributed environment provides a rich set of challenges related to scalability, reliability and performance. This thesis will present the data management requirements for executing the workload of the LHC*b* collaboration. We present the systems designed that support all aspects of the grid data management for LHC*b*, from data transfer, to data integrity, and efficient data access. The distributed computing environment is inherently unstable and much focus has been made on providing systems that are robust and resilient to observed failures.

To Mishka.

Declaration

This work represents the effort of many members of the LHCb collaboration at CERN. I have been a integral part of a small team of people working on the DIRAC project to support all aspects of the LHCb Computing Model. The design and implementation of the DIRAC Data Management System presented in Chapter 4 was entirely my work with the exception of some components discussed in Section 4.1, which existed prior to my contribution. The Replica Manager, Storage Element and protocol plug-in components were re-implemented to expose reviewed interfaces. The work in Chapters 5 and 6 and the writing of this thesis is entirely my own work.

Introduction to LHCb

The Large Hadron Collider is a proton-proton collider that will operate at a centre of mass energy of 14 TeV. The LHC has been built in the 27km tunnel, under Switzerland and France, originally created for LEP, see Figure 1. The LHC will support four major experiments: ALICE, ATLAS, CMS and LHCb. The aim of these four experiments is to explore the current understanding of the Standard Model and probe what might lie beyond it.

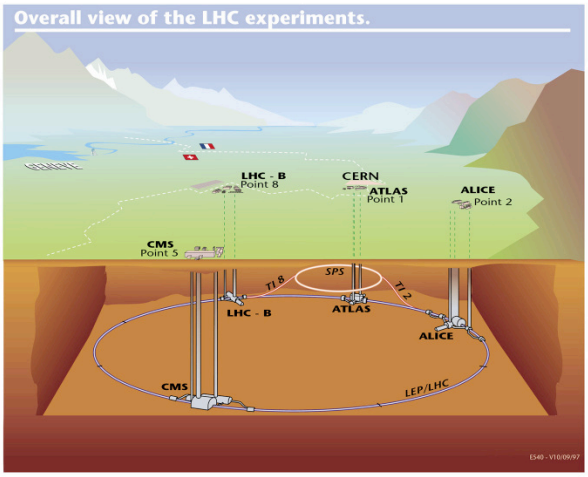


Figure 1: The LHC.

LHCb is a specialised B physics experiment that hopes to take advantage of the large quantity of b hadrons produced at the LHC and will concentrate on the precise measurement of CP violation parameters and rare decay processes. The statistically large sample of B particles will allow LHCb to investigate decay modes with low branching ratios, that were not possible with the previous generation of B-factories.

The LHCb detector [1], shown in Figure 2, is a single arm spectrometer. It consists of a dipole magnet, two Ring Imaging CHerenkov (RICH) detectors, a Vertex Locator (VELO), an electromagnetic and hadronic calorimeter, five muon chambers and four tracking stations. The role of each of these is outlined below.

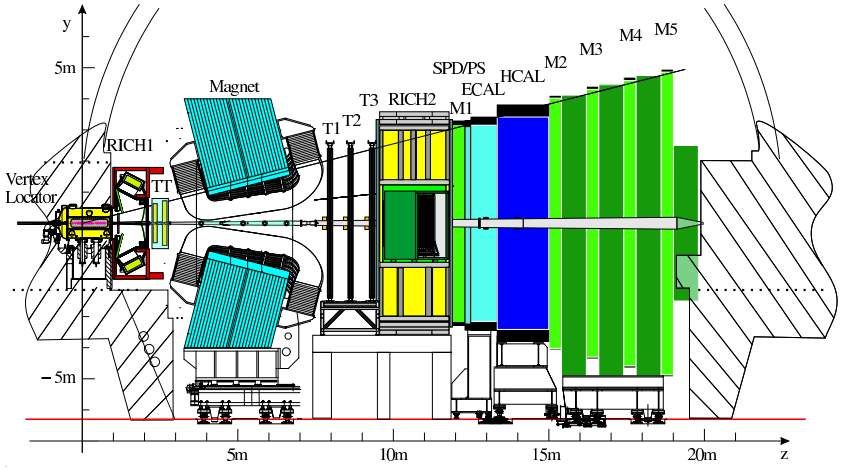


Figure 2: The LHCb detector.

Track parameters are measured using the tracking stations [4, 5]. Using the track curvature (induced by the magnetic field [2]) charged particle's momenta can be determined. The VELO [3] is located next to the beam pipe, close to the proton-proton interaction point. It measures precisely the passage of charged particles, allowing reconstruction of primary and secondary vertices. The decay vertices (secondary vertices) of long lived particles, such as B mesons, allow LHCb to study time dependent properties of the decays.

The particle identification is performed using information from three sub-detectors. The two RICH detectors [6], use the Cherenkov effect to determine the velocity of the particles. The first RICH detector is located in front of the magnet and will be used to measure low momentum particles, while the second RICH is located after the magnet for higher momentum particles. The Electronic CALOrimeter (ECAL) measures the energy of photons and electron while the Hadronic CALOrimeter (HCAL) measures the energy of pions, kaons and protons [7]. The Muon system [8] consists

of five stations to identify the particles which penetrate fully through the calorimeters.

A key element of LHCb is the trigger system [9]. The trigger contains two levels: the Level0 (L0) hardware trigger and the High Level Trigger in software (HLT). The L0 primarily uses information from the calorimeter and muon systems to select particles with high transverse energy and momentum, reducing the 40MHz LHC crossing rate to 1MHz. The software trigger further reduces this rate to 2kHz. This RAW event data is written to the Online system [10] and is the primary commodity of LHCb. This data is the starting point for everything that will be discussed in this thesis.

Contents

1 Grid Computing; History, Standards and Middleware 1
- 1.1 Introduction . . . 1
- 1.2 From IMP to I-WAY . . . 2
- 1.3 Defining the Grid . . . 4
- 1.4 Types of Grid . . . 5
- 1.5 Typical Components of a Data Grid . . . 6
  - 1.5.1 Fabric Layer . . . 6
  - 1.5.2 Requirements and Standards . . . 8
- 1.6 Selected Grid Projects . . . 11
  - 1.6.1 Infrastructure Projects . . . 11
  - 1.6.2 Grid Middleware Projects . . . 12
- 1.7 gLite Middleware . . . 16
  - 1.7.1 Information . . . 16
  - 1.7.2 Security . . . 17
  - 1.7.3 Workload Management . . . 17
  - 1.7.4 Data Management . . . 19
- 1.8 Data Intensive Applications . . . 21
  - 1.8.1 Distributed Aircraft Maintenance Environment . . . 21
  - 1.8.2 Biomedical Informatics Research Network . . . 21
  - 1.8.3 Particle Physics Applications . . . 22
- 1.9 Summary . . . 27

2 LHC*b*: Computing Requirements and Solutions 28
- 2.1 Gaudi: Architecture and Framework . . . 28
  - 2.1.1 Interface Model . . . 29
  - 2.1.2 Data and Algorithms . . . 29
  - 2.1.3 Services . . . 30
- 2.2 Physics Applications . . . 30
  - 2.2.1 Gauss . . . 30
  - 2.2.2 Boole . . . 31
  - 2.2.3 Brunel . . . 31
  - 2.2.4 DaVinci . . . 31
- 2.3 Logical Workflow and Dataflow . . . 31

2.3.1 RAW data . . . 32
2.3.2 Reconstruction . . . 32
2.3.3 Stripping . . . 33
2.3.4 Simulated data . . . 34
2.3.5 Analysis . . . 35
2.4 Computing Model . . . 35
2.4.1 Tier-0 . . . 36
2.4.2 Tier-1 . . . 36
2.4.3 Tier-2 . . . 37
2.5 2009 Resource Requirements . . . 37
2.5.1 Data Taking . . . 38
2.5.2 Re-processing Requirements . . . 39
2.5.3 Re-stripping Requirements . . . 41
2.5.4 Simulation Requirements . . . 41
2.5.5 Aggregate 2009 Resource Requirements . . . 42
2.6 Accumulated Data . . . 43
2.6.1 Tape . . . 43
2.6.2 Disk . . . 44
2.7 Summary . . . 45

3 DIRAC: A Community Grid Solution 46
3.1 Brief History . . . 46
3.2 DIRAC Design Principles . . . 47
3.3 DIRAC Architecture . . . 49
3.4 DIRAC Systems . . . 50
3.4.1 Framework . . . 50
3.4.2 Workload Management . . . 52
3.4.3 Production Management . . . 55
3.4.4 Data Management . . . 57
3.5 The Advantages of a Community Solution . . . 57

4 DIRAC Data Management Design 59
4.1 Core Data Management Components . . . 59
4.1.1 Storage Element . . . 60
4.1.2 File Catalog . . . 62
4.1.3 Replica Manager . . . 65
4.1.4 Request Database . . . 66
4.1.5 Data Management Agents . . . 66
4.2 Bulk Transfer Framework . . . 67
4.2.1 Transfer Database . . . 68
4.2.2 Replication Scheduler . . . 69
4.2.3 FTS Submit / Monitor . . . 70
4.2.4 Request finalisation . . . 71
4.2.5 Request Persistency . . . 71

4.3 Data Driven Replication . . . 72
4.3.1 RAW Data Splitting . . . 74
4.3.2 DST Data Broadcast . . . 74
4.4 Ensuring Data Integrity . . . 74
4.4.1 RAW Data . . . 75
4.4.2 DIRAC Resource Consistency . . . 78
4.5 Datasets . . . 80
4.5.1 Dataset Use Cases . . . 81
4.5.2 Dataset architecture and implementation . . . 82
4.6 Summary . . . 84

5 DIRAC Resources Performance 86
5.1 Replica Catalog Performance . . . 86
5.1.1 Initial File Registration . . . 88
5.1.2 Registering Replicas . . . 89
5.1.3 Retrieving Replica Information . . . 90
5.1.4 Removing Replica Information . . . 93
5.1.5 Removing Files . . . 94
5.1.6 Distributed Read-Only Mirror Performance . . . 95
5.1.7 Data Set Dereferencing . . . 96
5.1.8 Summary . . . 97
5.2 Storage Resource Manager Performance . . . 98
5.2.1 Retrieving File Metadata . . . 99
5.2.2 Issuing Prestage Requests . . . 101
5.2.3 Retrieving Transport URLs . . . 102
5.2.4 Removing Files . . . 105
5.2.5 Uploading Files . . . 107
5.2.6 Summary . . . 108

6 DIRAC Bulk Transfer Framework Performance 110
6.1 Peak Network Requirements . . . 110
6.2 Commissioning Exercise . . . 111
6.2.1 Transfer Throughput Scalability . . . 112
6.2.2 Channel Performance . . . 116
6.2.3 Discussion . . . 120
6.3 Replication Strategies . . . 120
6.3.1 Introduction to Graph Theory . . . 120
6.3.2 Spanning Tree Generation . . . 122
6.3.3 Edge Weight Generation . . . 125
6.3.4 Comparing Strategy Performance . . . 127
6.3.5 Selected Strategy . . . 131
6.4 Summary . . . 132

7 Conclusions 134

A SRM Evolution **137**
A.1 Key Concepts . . . 137
A.1.1 LFNs, PFNs, Source URLs and Transfer URLs . . . 139
A.2 Protocol Development . . . 140
A.2.1 SRM Version 1.x . . . 140
A.2.2 SRM Version 2.x . . . 141
A.2.3 Additions in SRM Version 2.2 . . . 143
A.3 SRM in LCG . . . 143
A.4 Summary . . . 144

B SRM 2.x Commands **145**
B.1 Data Transfer Functions . . . 145
B.2 Space Management Functions . . . 146
B.3 Permission Functions . . . 147
B.4 Directory Functions . . . 148
B.5 Status Functions . . . 148

C RAW Upload Performance **149**

D SRM Performance Results **153**
D.1 Retrieving File Metadata . . . 153
D.2 Issuing Prestage Requests . . . 153
D.3 Retrieving Transport URLs . . . 153
D.4 Removing Files . . . 153

List of Figures

1 The LHC ii
2 The LHC*b* detector iii

1.1 The first 4 nodes on ARPANet 3

2.1 LHC*b* Computing Model Schematic 36
2.2 Network rate into the Tier-1 sites for RAW data 39
2.3 2009 Tier-1 Data Produced 40
2.4 Tier-0 and Tier-1 Tape Resources 2009 - 2011 44
2.5 Tier-0 and Tier-1 Disk Resources 2009 - 2011 44

3.1 Interaction Of DIRAC Services, Agents and Resources 49
3.2 DISET Layered Architecture 51
3.3 DIRAC Workload Management System Architecture 53
3.4 DIRAC Production Management System Architecture 56

4.1 Storage Element Class Diagram 60
4.2 File Catalog Class Diagram 63
4.3 Architecture for Ensuring Availability of Replica Information 65
4.4 Replica Manager Class Diagram 66
4.5 Distributed Request DB Service Schematic 67
4.6 The DIRAC bulk transfer framework components 68
4.7 Components to perform data driven replication 73
4.8 Ensuring RAW data integrity schematic 76
4.9 RAW Integrity Activity Diagram 77
4.10 Schematic representation of the data integrity suite 79
4.11 Dataset Architecture Schematic 83

5.1 The relationship between LFN, file GUIDs, PFNs and Storage Element names within the LFC 87
5.2 Time to register a file with a single replica in the LFC within a single transaction 89
5.3 Insert rate of replicas in the LFC for a varying number of replicas within an authenticated session 90

5.4 Retrieval rate of replica information from the LFC for a varying number of files . . . 91
5.5 Retrieval rate of replica information from the LFC for a varying number of files using bulk method . . . 92
5.6 Removal rate of replicas from the LFC for a varying number of replicas within an authenticated session . . . 93
5.7 Removal rate of files from the LFC for a varying number of files . . . 94
5.8 Retrieval rate of replica information from the read-only LFC instances for a varying number of files using bulk method . . . 96
5.9 Time to obtain replicas for entire directory from the LFC for a varying number of files in the directory . . . 97
5.10 Retrieval time for file metadata from site SRMs for a varying number of files using GFAL . . . 100
5.11 Time to issue prestage request to the site SRMs for a varying number of files using GFAL . . . 102
5.12 Retrieval time for transfer URLs from site SRMs for a varying number of files using GFAL . . . 104
5.13 Time to remove 1 byte files from site SRMs for a varying number of files using GFAL . . . 105
5.14 Removal rate of data from site SRMs for a varying number of 2 GB files using GFAL . . . 106
5.15 Histogram of time taken to upload a 1 Byte file to site SRM from local worker node . . . 109

6.1 Cumulative transferred data during commissioning exercise grouped by destination site . . . 112
6.2 Transfer throughput during commissioning exercise grouped by destination site . . . 113
6.3 Mean number of times FTS requests were monitored by the FTS monitor agent grouped by day . . . 114
6.4 Transfer throughput during commissioning exercise grouped by source site . . . 116
6.5 Channel throughput and success rate by source and destination site . . . 119
6.6 Introduction to graph theory . . . 121
6.7 Complete weighted graph example . . . 124
6.8 Solutions to weighted graph example . . . 125
6.9 Mean time to transfer files within each request for the strategies evaluated . . . 128
6.10 Average request spanning tree properties . . . 129
6.11 Aggregated assigned files per edge . . . 130
6.12 Edge files scheduled vs. performance by source . . . 132

C.1 Read rate from online storage with varying number of executing agents 150
C.2 Lemon network monitoring of Castor pools accessed by the DIRAC agents deployed at the gateway 150
C.3 Lemon network monitoring of Castor pools accessed by the DIRAC agents 151
C.4 Histogram of the time for files to be migrated to tape after being uploaded to Castor disk pools 152

List of Tables

2.1 Events information per processing step 34
2.2 Resource consumption for each stage of LHC*b* activity 35
2.3 2009 Pledged Computing and Tape Resources by Tier-1 . . . 38
2.4 Network rates associated to the production of DSTs during data taking (MB/s). 40
2.5 2009 re-processing data rates for transferring rDST and DST data 41
2.6 2009 re-stripping data rates for transferring DST data 42
2.7 2009 Simulation network requirements 42
2.8 Total required network rates during re-stripping exercise for each site 43

5.1 CPU and RAM configuration of LFC read only instances at CERN and Tier-1 sites 88
5.2 The time taken for updates to the read-write master LFC to reach the read-only mirrors 95
5.3 Success rates for file metadata retrieval from site SRMs for varying number of files using GFAL 100
5.4 Success rates for tURL retrieval from site SRMs for varying number of files using GFAL 103

6.1 Aggregated throughput achieved during final phase of commissioning exercise 117
6.2 Channel success rate during final phase of commissioning exercise 117
6.3 Mean file transfer time of 2GB file during final phase of commissioning exercise 118
6.4 Estimated channel throughput during final phase of commissioning exercise 119

A.1 Files Types Allowed in SRM Space Types 139
A.2 SRM Space Type Properties 139
A.3 SRMv1.1 Transfer Functions and Their Operation 141
A.4 SRMv1.1 Status Functions and Their Operation 141

D.1 Mean, standard deviation and median of the retrieval time and the success rate for file metadata queries from site SRMs for a varying number of files using GFAL 154
D.2 Mean, standard deviation and median of the operation time and the success rate for pre-stage request to site SRMs for a varying number of files using GFAL 155
D.3 Mean, standard deviation and median of the retrieval time and the success rate for transport URL queries from site SRMs for a varying number of files using GFAL 156
D.4 Mean, standard deviation and median of the removal time and the success rate for 1 byte file removal requests from site SRMs for a varying number of files using GFAL 157
D.5 Mean, standard deviation and median of the removal rate and the success rate for 2GB file removal requests from site SRMs for a varying number of files using GFAL 158

Chapter 1

Grid Computing; History, Standards and Middleware

The thesis will concentrate on the LHC*b* experiment, their computing requirements, implemented solutions and experience gained. An introduction to and history of Grid computing is presented in Chapter 1 along with an overview of past and present Grid projects. The LHC*b* Computing Model will be discussed in Chapter 2. The DIRAC system, which supports all facets of the Computing Model, will be covered in Chapter 3. The core of this thesis is contained in Chapter 4, which outlines the DIRAC Data Management System, Chapter 5 that presents the performance of the grid resources (with the full results presented in Appendix D) and Chapter 6 which reports the performance of the DIRAC bulk transfer system. An overall summary and conclusions will be given in Chapter 7. Appendix A discusses the evolution of the SRM standard and Appendix B gives a summary of the functionality provided by version 2 of the protocol. Appendix C gives the performance of handling RAW data by the DIRAC Data Management system.

1.1 Introduction

> If such a network as I envisage nebulously could be brought into operation, we could have at least four large computers, perhaps six or eight small computers and a great assortment of disc files and magnetic tape units - not to mention remote consoles and teletype systems - all churning away. ∼ J.C.R. Licklider (1960)

Since the earliest days of computing, dominated by monolithic beasts built to perform a single repetitive computational tasks, the aggregation of computational resources has been dreamed of by computer engineers. Communication protocols [11] existed alongside the early computational machines to allow humans to interact at distance, but it was not till much later that inter-machine communication was attempted. In 1965, Lawrence Roberts at the MIT Lincoln Labs performed the first inter-computer communication [12] allowing machines to work together for the first time. Although the scale of Licklider's vision may appear quaint in the current computing environment the underlying desire still remains the same; to connect computing and storage resources and enable them to work together. This chapter will review the technology, standards and projects that have paved the way from connected machines to inter-connected Grids.

1.2 From IMP to I-WAY

After performing the first inter-computer communication Roberts moved to ARPA (Advanced Research Projects Agency) and began work on ARPANet [13, 14]. ARPANet was mandated to be a packet (then called message) switched network using Interface Message Processors (IMP) [15] to act as the digital interface between hosts and the underlying (analogue telephone) network. By 1969 the first messages had been sent on this packet switched network (shown in Figure 1.1), opening the door for many subsequent networking protocols and the services built on top of them.

The Network Working Group (NWG), formed just before ARPANet went live, was responsible for considering the protocols to be used on the network. One of the first to be proposed was a tentative protocol for transferring files across networks [17] but it was not until the first inter-networking protocol, Transmission Control Protocol (TCP) [18], was proposed that standards started to emerge. After several years of discussion and updates the original TCP specification was split into IP [19], to perform the network routing, and TCP [20] to ensure the reliable end-to-end delivery of packets. These two protocols form the basis of most networking applications on the modern internet and after many intermediate proposals the File Transfer Protocol (FTP) [21] was defined to sit on top of TCP/IP. Another important step was the proposal of the Domain Name System (DNS) [22, 23] and subsequent

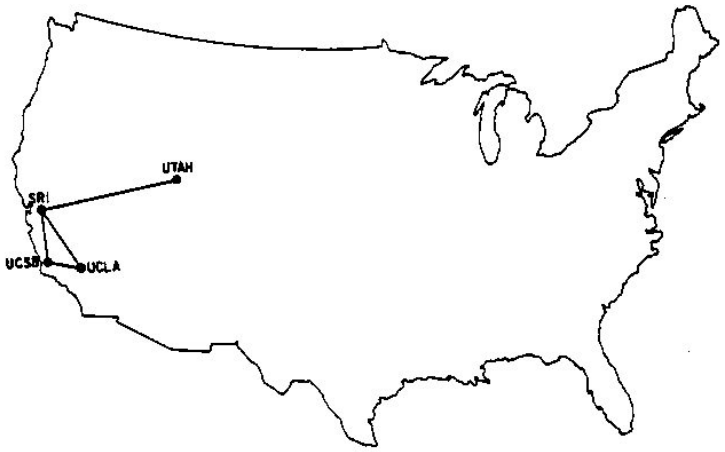


Figure 1.1: The first 4 nodes on ARPANet: UCLA, SRI, UCSB, UTAH.[16]

modifications [24, 25] which provisioned for a scalable way of resolving machine names to their numerical network address.

The early 1990s internet, as pioneered by ARPA and the NWG, provided the standards and protocols for building and using Wide Area Networks (WANs). Another major step was the advent of the World Wide Web (WWW), first proposed as an information management system [26] and led to the development of the HyperText Markup Language (HTML) [27], Uniform Resource Locators (URLs) [28], the HyperText Transfer Protocol (HTTP) [29, 30] and the formation of the WWW Consortium (W3C) [31] to develop standards. One of the important roles of the W3C was the fostering of Web Services (WS) to 'support interoperable machine-to-machine interaction over a network' [32]. This broad description is often taken to mean the use of client-server communication based on XML [33] messages following the SOAP [34] standard. In this architecture an application WS exposes a description of the functionality offered by the service in Web Service Description Language (WSDL) [35] to allow prospective clients to discover services relevant to their needs.

In 1995 the Factoring via Network-Enabled Recursion (FAFNER) [36] project used parallel factoring algorithms to compute RSA130 solutions for public key encryption making use of web servers to distribute tasks and re-

ceive results from clients over HTTP. Clients could easily contribute their hardware to the project via the HTTP interface and this paved the way for many future projects applying an idle cycle utilisation paradigm. The first mainstream project of this type was Entropia [37] followed by the massively successful SETI@home project [38] based on the Berkeley Open Infrastructure for Network Computing (BOINC) [39] platform.

A contemporary to FAFNER was the Information Wide Area Year (I-WAY) project [40] which aimed to virtualise resources across 17 participating sites using varying network resources and protocols. Each of the participating sites installed gateway machines running the I-WAY software environment (I-Soft) to overcome resource heterogeneity. Client computational tasks were submitted to a central scheduler which maintained communication with local schedulers installed as part of I-Soft. This architecture allowed heterogeneous tasks to be performed on heterogeneous resources and the experience gained led to the proposal and development of grid toolkits.

1.3 Defining the Grid

Experience from I-WAY highlighted the need to define a common set of components and corresponding interfaces to achieve a scalable system capable of managing heterogeneous resources. The first attempt to define the requirements of a Grid [41] came in parallel with their first implementation in the Globus Toolkit (GT) [42] by the Globus Alliance [43]. This first attempt was rather nebulous and a more concrete description, including a crystalisation of the nomenclature to describe the Grid and a proposed layered architecture [44], accompanied the release of GT2. In parallel, attempts to define standards for the grid were conducted by the Global Grid Forum (GGF) [45], which approached this task with several research and working groups, and later the Enterprise Grid Alliance (EGA) [46], founded by a host of commercial software and hardware vendors.[1] In 2006, GGF and EGA merged to form the Open Grid Forum (OGF) [47] with the aim of 'leading the global standardisation effort for grid computing' mirroring the role of the W3C for WWW in the Grid world.

[1] Founding members were: SUN Microsystems, EMC, Fujitsu Siemens Computers, HP, NEC, Network Appliance, and Oracle.

Although several groups attempted to enumerate a Grid in their own particular way [41, 44, 48, 49, 50, 51] the standards based approach brought almost everyone into agreement that the adoption of the Open Grid Services Architecture (OGSA) defined a Grid [52] (with a few detractors [53]). OGSA aims to provide virtualisation of heterogeneous resources by adopting a Service Oriented Architecture (SOA) based on the web service technologies of WSDL and SOAP. Initially, Open Grid Services Infrastructure (OGSI) [54] was used as the underlying implementation of OGSA to extend existing web service technology to support stateful Grid services, but this was later replaced by the use of Web Services Resource Framework (WSRF) [55, 56].

Independent of the architecture and technologies employed to realise a Grid the expectation of what you get ultimately defines it. One such definition is offered in [57].

> A Grid is a hardware and software infrastructure that provides dependable, consistent, and pervasive access to resources to enable sharing of computational resources, utility computing, autonomic computing, collaboration among virtual organisations, and distributed data processing, among others.

1.4 Types of Grid

Attempts to define Grids usually prove to be vague and nebulous, as is shown above, because the specifics of what Grids are depends significantly on the application and practitioners. In this sense, the Grid means many things to many people and provides a wide umbrella to support significant academic and enterprise interest in the subject. Within the definition of Grids several major types of Grids emerge:

- Computational Grids provide an architecture for transparent access to distributed heterogeneous CPU resources. Two examples of such Grids are TeraGrid and DEISA (discussed further in Section 1.6.1) which provide distributed access to US and European super-computing facilities respectively.

- Storage Grids provide standardised access to reliable distributed storage space. The Storage Resource Broker (SRB) [58], from the San Diego Super-computing Center, provides a logical namespace to manage data resident across distributed administrative domains. Another type of large scale storage Grid facility is Amazon Simple Storage Service [59] which offers a web service interface to an abstract storage facility.

- Networking Grids provide access to networking resources across disparate infrastructures. 'LambdaGrids' [60] use optical links carrying different wavelengths to provide network resources with an explicit quality of service which may be scheduled by user applications.

- Data Grids provide the infrastructure for users to perform the distributed processing of large volumes of data across geographically separated sites.

Although these types are different they are not mutually exclusive. A Data Grid requires access to computational, storage and network resources which reinforces the importance of the adoption of SOA. The focus of this thesis is Data Management within Data Grids. To elucidate Data Grids a summary of the typical components is given in the Section 1.5.

1.5 Typical Components of a Data Grid

The raison d'être of a Data Grid is the processing and storage of large volumes of data. The core of a Data Grid, often referred to as the *fabric* layer, are the CPUs on which the *processing* is performed, the servers which store the data and the network components that connect them. Fabric level components in a typical Data Grid are given in Section 1.5.1.

1.5.1 Fabric Layer

The fabric of a data grid is more than a collection of raw resources, but also includes the first levels of abstraction; this ranges from the operating systems, application libraries, and common networking protocols used by the CPUs, to the batch systems which aggregate and manage the resources

and schedule the tasks running on them.

Batch Systems

There are many more batch systems in existence than can be enumerated here, but the most common is the Portable Batch System [61], which shares its ancestor with the Tera-scale Open-source Resource and QUEue manager (TORQUE) [62], that provides resource management and a simple scheduler. Both may be combined with the Maui/MOAB scheduler to provide user and group based job priorities and scheduling algorithms. The Condor High-Throughput Computing System [63] is a resource manager that can aggregate dedicated clusters and idle desktop machines into a virtual cluster. The common commercial products available are LSF [64] from Platform which is a commercial integrated resource manager and scheduler, and Sun Grid Engine (SGE) [65] which provides an open source resource manager and scheduler that can be used to share resources between distributed SGE systems.

Storage Systems

In an analogous way to batch systems, storage systems provide a layer between the physical storage media and the client. Familiar file systems, as used by individual PCs, manage the underlying data storage device (most commonly a spinning disk, although flash based devices are now common) to map files and directories to the underlying *blocks* and maintain metadata regarding the contents. As the requirements of storage systems increase, mechanisms for accessing the file system contents across several nodes must be adopted. There are two main approaches to this problem: shared disk file systems and distributed file systems.

Shared disk systems (or more commonly Storage Area Networks (SAN)) provide uniform access from all servers to all the data blocks in the SAN. The file information in this system may be maintained in a centralised server or distributed across the servers comprising the SAN. SANs often adopt high-bandwidth networking technology, such as Fibre Channel [66] or In-

finiBand [67], for performance and lock managers to allow concurrent read and write operations to blocks while avoiding data corruption.

In contrast to SANs, distributed (or network) file systems allow file access using the wide area network where files may be distributed across different remote network nodes. Each of the network nodes may address given parts of the file system and therefore, to provide transparent access to the client, should provide automatic file replication across nodes, with the added advantage of increasing fault tolerance. This data replication allows increased performance by striping the data from multiple servers to the client. The most widely used network file system is NFS which implements the NFS protocol [68, 69, 70]. The newest version of NFS provide a concurrent file read/write as well as the possibility for data access parallelism using Parallel NFS (pNFS). Andrew's File System (AFS) [71] approached scalability by downloading files to the client cache from the server when accessed allowing multiple updates without server interaction. General Parallel File System (GPFS) [72], is a high performance system from IBM, originating from the HPC environment which uses block parallelism to increase read and write rates from multiple disks.

The fabric layer resources mentioned above are in no way exhaustive, but the variety of available products emphasises the need for a standards based approach to allow interoperability.

1.5.2 Requirements and Standards

The provision of a Grid relies on the availability of underlying resources and a software infrastructure, referred to as *middleware*, to allow utilisation. This section will give a high level overview of the middleware requirements for a Data Grid, along with the relevant standards, while the specifics of selected middleware projects will be given in Section 1.6.2.

Information Service

The knowledge of resources is essential to any Grid system, making information providers important components. The ability to discover resources available for a particular task requires that resource capabilities are pub-

lished and are retrievable by clients. In addition to capabilities, resources must also publish contact information such that they can be invoked. Finally, it must be possible to discover the dynamic state of a resource to determine the optimal resource at any time. A single standard schema for representing this information is required for transparent access to disparate information providers. The Grid Laboratory Uniform Environment (GLUE) Working Group [73] at OGF has produced an abstract information model to describe attributes pertaining to computing and storage resources in the GLUE schema [74]. The schema is independent of implementation and may be used for interoperability between Grids.

Data Management

The variety of storage systems available in the fabric layer implies the need for a standard protocol to manage these resources. The Grid Storage Management Working Group [75] at OGF produced the Storage Resource Manager (SRM) [76] interface to provide a uniform middleware layer between clients and the underlying resources. Early versions of the protocol dealt with simple file management operations and later versions included the ability for clients to dynamically reserve and manage space (the evolution is discussed in Appendix A). SRM provides a control protocol for managing space and files, but does not define a transfer protocol. The mechanism for protocol negotiation in the SRM specifications allows the use of any transfer protocol that the client and the server mutually support.

The necessity to transfer large volumes of data is inherent in a Data Grid and the definition of a transfer protocol suitable for this was proposed by the GridFTP Working Group [77]. The GridFTP protocol [78, 79] uses Grid Security Infrastructure (GSI) authentication on both control and data channels for secure data transfer. In addition, the protocol builds on the FTP specification by allowing multiple data channels (striped transfers) for increased throughput and network utilisation. In addition to large data transfers over the WAN a Data Grid must also provision the ability to read data from files during processing activities. The OGSA ByteIO Working Group [80] defined the ByteIO protocol [81] to enable POSIX-like access to data files to allow clients for transparent access on disparate storage systems.

A different approach to data management on the Grid is being pursued by the Grid File System Working Group [82]. The Grid File System (GFS) architecture, as proposed in [83], wishes to create a federated logical resource namespace of all data in distributed file systems. A GFS system would store resource endpoint, protocol and authentication information to map GFS file entries to the underlying resource. These entries would then be used to deliver the file to a client when attempting to access a file. In addition to distributed file systems GFS also may contact transfer protocol servers for files via HTTP, FTP or GridFTP or negotiate with SRMs. When used in conjunction with SRMs the GFS becomes a centralised replica catalog, for which there is no proposed standard interface.

Job Management

The approach of deploying a standardised interface that abstracts the underlying resource from the client is also adopted in job management. The Basic Execution Services Working Group [84] developed a specification [85] which allows the creation, monitoring and control of execution tasks on a computing resource. The specification provisions the abstraction layer allowing transparent submission of tasks to different batch implementations. Tasks submitted to a Basic Execution Service are described in Job Submission Description Language (JSDL) [86].

Security

Ensuring that resources are secure and used appropriately is fundamental to every grid component. The de-facto standard authentication mechanism in grid computing is the use of public key infrastructure (PKI) [87]. Globally trusted Certification Authorities (CA) issue grid users with X.509 certificates composed of public key and a password protected private key. The public and private key pair can then be used when authenticating with remote services. They can also be used to generate limited lifetime *proxy* certificates which allow users a *single sign-on* to avoid repeatedly typing passwords. Mechanisms to ensure that clients are authorised to perform actions must also be used based on client affiliation and role.

1.6 Selected Grid Projects

The first round of Grid projects was funded by national grid initiatives which sought to foster development of middleware and the procurement of hardware resources. An early example was the UK e-Science [88] program which provided funds for each of the national Research Councils to demonstrate use-cases across a broad range of research with major drivers coming from the fields of physics, astronomy and bio-informatics. Similar initiatives were undertaken in Italy (the Grid.it project brought together six multi-disciplinary scientific institutes to provide a middleware platform and applications for the participating institutes) and Germany (funding the UNiform Interface to COmputing REsources (UNICORE) [89] project). A selection of infrastructure projects are discussed in Section 1.6.1 and middleware projects Section 1.6.2.

1.6.1 Infrastructure Projects

HPC

Grid infrastructure projects were initiated to integrate resources at leading super-computing centres. In the United States, the TeraGrid project [90] links 9 'Resource Providers' via a dedicated optical network and allocates computing resources to U.S. based academics through a multidisciplinary peer-review process. The TeraGrid infrastructure is integrated using the 'Coordinated TeraGrid Software and Services' middleware based on the Globus Toolkit. The Distributed European Infrastructure for Super-computing Applications (DEISA) [91], the European analogue to TeraGrid, links 11 national super-computing centres via 10Gb network and allows users to use Globus or UNICORE middleware for submission of computational tasks. The China National Grid (CNGrid) [92] set out to create a combined HPC and Grid environment linking two super-computing centres with the computing resources of a further 6 universities using Globus. Eventually, CNGrid produced proprietary middleware [93] based on OGSA. Recently, similar projects have been adopted in Canada (Compute Canada [94]) and India (Garuda [95]).

Commodity

Several projects are building Grid infrastructures from commodity clusters and storage. The two major projects in this genre are Open Science Grid (OSG) [96] in the United States and Enabling Grids for E-sciencE (EGEE) [97] led by CERN.

The OSG is a consortium of universities and national laboratories in the United States that evolved as a continuation of the Grid3 project, a collaborative initiative between the Grid Physics Network (GriPhyN) [98], Particle Physics Data Grid (PPDG) [99] and the international Virtual Data Grid Laboratory (iVDGL). The Virtual Data Toolkit (VDT) [100], produced by the Grid3 collaborators, forms the basis of the OSG "Software Stack" and includes Globus, Condor-G [101] and VOMs [102] and is installed at over 60 sites.

The EGEE project provides the middleware and the underlying infrastructure to support the European multidisciplinary research community. The EGEE resources are contributed by National Grid Initiatives (NGI) across Europe, North Africa, Asia and Latin America to over 300 sites providing 80,000 CPU cores. Two major contributors are the UK e-Science program who, through GridPP [103], provides resources and support across 20 UK institutions and the Italian Grid.it project which established over 30 sites within INFN-GRID [104]. There are a plethora smaller contributors to EGEE who provide additional resources.

1.6.2 Grid Middleware Projects

Globus Toolkit

The Globus Toolkit was an early implementation of the Grid infrastructure attempting to provide a bag of services on which Grids could be built. The success of the Globus Toolkit can be ascribed to the proximity of the Globus Alliance to the first attempts to define standards. The widely used GT2 was replaced by a Web services based OGSI implementation which was largely ignored due to issues of scalability and reliability and was in turn replaced by a WSRF implementation in GT4[105]. Several GT services are widely used by other projects:

- Grid Security Infrastructure (GSI) [106] based on X.509 public key infrastructure (PKI) for authentication.
- MyProxyServer [107] for credential delegation and management.
- Grid Index Information Service (GIIS) [108] for publishing and discovering resource status.
- Globus Resource Allocation Manager (GRAM) [109] for managing job execution, monitoring and output.
- Grid File Transfer Protocol (GridFTP) for high throughput data transfers.

The most important of these components with respect to this thesis is GridFTP which has become the primary mechanism for file transfer on the Grid. The name GridFTP refers to both the standard discussed in Section 1.5.2 and the Globus implementation [110].

UNICORE

The UNICORE [89] project was initiated to give German scientists access to distributed HPC resources. The middleware was built on proprietary technologies in parallel to the emerging standards. As it became widely used in scientific and commercial settings across Europe, the European Union provided funding through the UniGrids project [111] to migrate it towards a web service architecture which was compliant with OGF standards. The UNICORE middleware is based on a three tier model: client (for user creation of workflows/tasks), gateway (for authentication and authorisation of user requests) and service container that runs next to the Grid resource. The service container, containing Unicore Atomic Services (UAS), offers a framework for hosting WSRF compliant Grid services and is therefore extensible to any new developments.

NAREGI

The Japanese National Research Grid Initiative (NaReGI) [112] began in 2003 to develop Grid middleware to be used for e-Science. The success of the UNICORE project, and the complete solution it offered, provided the basis for the first middleware deployed on the NaReGI testbed. As the

OGSA standard was defined the NaReGI middleware evolved to implement WSRF architecture and to emphasis interoperability with other emerging middlewares. The major addition made to the original UNICORE architecture is the Super Scheduler (SS) which aggregates globally available resource information and performs match making based on the job requirements. The SS itself contains four services which perform the following tasks:

- Job Management Service (JM) accepts client jobs.
- Candidate Set Generator (CSG) determines all resources required by the job.
- Execution Planning Service (EPS) schedules the job based on the compute resources returned by CSG and their state.
- Reservation Service (RS) performs resource reservation and job submission for the list of sites generated by the EPS.

Data management in NaReGI is based on Gfarm [113] which implements a Grid File System [83] architecture. This approach is based on a set of file server nodes, mapping the network topology, which trigger dynamic file replication in response to high activity on a single node [114]. To facilitate interoperability with other production Grids, GridFTP access to data is also provisioned.

ARC

The Advanced Resource Connector (ARC) [115] project began in 2001 with the aim to provide a Grid solution for the Nordic countries. At the time of inception none of the existing middleware solutions met ARC's requirements so a new project was started based on (the then standard) GT2. The ARC system is designed to have no single point of failure and no centralised management where brokering, like UNICORE, is performed in the client layer. Clients query the top level information services (Index Services) to obtain URLs for the resource level Local Information Services (LIS) which are then queried in turn to obtain possible resource candidates and their status. Once a resource is selected the task is forwarded to the Computing Service (CS) corresponding to that resource via GridFTP. The Grid Manager (GM), the main component of the CS, interprets the job description

and retrieves the input files and job executable in a directory dedicated to that task. The GM communicates the executable to the Local Resource Management System (LRMS), installed on the batch system worker nodes, and is responsible for uploading output data and cleaning the task directory.

The GM encapsulates data management operations required for fetching input data and upload of output data using the DataMove component. This library has a plug-able architecture for supporting different protocols and supports third party transfers. Similarly the GM can automatically register output data to the Data Indexing Service (DIS) with plug-able backend support. ARC also provides a Smart Storage Element (SSE), integrated with the DIS, which has both a WS and SRM interface.

CROWN

The China R&D Environment Over Wide-area Network (CROWN) project [116] adopts an approach similar to UNICORE and ARC, placing middleware services next to the compute resource and performing brokering in the client. The Node Server (NS), hosting a WSRF compliant service container, provides the services to execute and monitor tasks and retrieve output from underlying computing resource. Each of these nodes belongs to a domain which has an associated Resource Locating and Description Service (RLDS), to publish and retrieve system state information, which are organised into a hierarchical tree structure. The novel contribution of CROWN is a logical overlay network above the set of RLDS services. In this system resources of common properties form groups which are known to their respective RLDS improving the resource location efficiency and scalability of the global information service.

gLite

The history of gLite starts with the European DataGrid (EDG) project which sought to provide a distributed computing system for three data intensive applications: high energy physics, biological/medical image processing and Earth observation science. Early versions of EDG middleware were heavily based on GT2 but later enhanced to support the data-centric appli-

cation use cases. By the end of the EDG project several production quality middleware components were taken forward into the LHC Computing Grid (LCG) project. In addition to the DataGrid components LCG blended its own developments with the Virtual Data Toolkit (VDT) (produced by PPDG, GriPhyN and iVDGL) into the supported middleware. The second version of the LCG software (LCG-2) later became the basis for the gLite middleware supported by the Enabling Grids for E-sciencE (EGEE) project [97]. Current gLite middleware components of importance to this thesis will be discussed in Section 1.7.

Next Generation Projects

The adoption of open standards and a SOA allows interoperability between components developed by different middleware projects. In addition to the development projects discussed above, a new type of middleware project has emerged with the focus on consolidating existing software, one such example is Open Middleware Infrastructure Institute (OMII) [117]. The aim of OMII is to evaluate Grid services from existing infrastructures and provide impartial evaluation. Another interesting project, NextGRID [118], had the ambitious aim to define an architecture for future commercially viable enterprise Grids in which "Grid-based applications execute on inter-enterprise, heterogeneous Grid infrastructures which encompass at run time, the different business models used by different stakeholders. This implies a cost-effective and universally applicable technology supporting diverse and sustainable business models." [119].

1.7 gLite Middleware

The gLite middleware is based on a SOA architecture providing services in four main groups: Information, Security, Workload Management and Data Management.

1.7.1 Information

The foundation of the gLite information system is the resource information publishers which generate status information compliant with the GLUE schema. This information is published to a Globus GIIS present at each

site which is subsequently published, using LDAP technology, to a Berkeley Database Information Index (BDII) [120] which may then be queried by information consumers. The BDII is based on the GIIS design, but uses two LDAP databases to allow update and query operations to be performed simultaneously providing increased stability. The gLite implementation of the BDII splits static and dynamic information to reduce database load and further increase stability.

1.7.2 Security

The security mechanism in gLite is based on X.509 certificates and GSI for authentication. The authorisation mechanism uses the Virtual Organisation Management Service (VOMS) [102] to maintain and verify user groups and roles attributes. MyProxy is used as a credential store for long life proxy certificates.

In general, the security containers are embedded within gLite services and do not invoke external web services. On contacting a gLite service, with a proxy containing VOMS added extensions, the authentication is performed within the security container. Once accepted authorisation is performed, to ensure the client has permission to access the resource. This authorisation step can involve the use of a Local Centre Authorisation Service (LCAS) which performs a primary check of the users' status at the site. The Local Credential Mapping Service (LCMAPS) then maps the grid credential, based on the proxy certificate Distinguished Name (DN) and VOMS attributes, to a credential understood by the local resource. Once successful the requested service functionality is invoked. Some services, such as FTS described in Section 1.7.4, may then use delegated client credentials for interaction with other services.

1.7.3 Workload Management

The two key components in the gLite Workload Management system are the gLite Workload Management System (WMS) which provides advanced resource brokering and the Computing Resource Execution And Management (CREAM) computing element currently being adopted. Clients of the gLite WMS describe their computing tasks using the gLite Job Description

Language (JDL), based on the ClassAd language [121], which contains job attributes (such as input sandbox, executable etc.), data attributes (input files, storage elements etc.) and job requirements (required operating system, CPU time etc.).

gLite WMS

The gLite WMS [122] provides the job management service for matching user tasks with available resources, submitting the tasks, monitoring their state and where possible managing retries. The user job requirements, which may be represented in gLite JDL, or JSDL, are used to determine candidate resources and are assigned to 'task queues' mirroring these candidates. The state of the underlying resources is cached in the WMS server (the Information Supermarket) and periodically updated synchronously and asynchronously from the gLite information system. The scheduling algorithms for matching the jobs to resources can be performed in one of two ways. The first, *eager* scheduling, matches the best resource for the job based on the requirements and the current information in the supermarket. To do this the Job Submission Service forwards jobs to the highest ranked resource using Condor-G [101]. The update period of the contents of the information supermarket renders absolute scheduling decisions liable to computing resource *flooding* and to combat this a *fuzzy rank* mechanism is adopted to stochastically select from the best available resources. Alternatively, a *lazy* scheduling approach is supported which waits for resources to become available and matches the most appropriate job from the task queues. The gLite WMS also provides advanced features such as bulk submission, real-time job interaction, output peeking and proxy renewal.

CREAM CE

Like any computing element CREAM [123] provides an abstract layer on top of a computing resource, extending the native functionality to include Grid security and increased reliability. It supports typical job management capabilities of submission, cancellation and the ability to pause and resume jobs (where supported by the underlying batch system). The management of tasks can use a legacy WS or BES interface with support for gLite JDL and JSDL for job description. Job state information is available through

the CE Monitoring (CEMON) service which also provides an asynchronous call-back mechanism to clients. The Journal Manager component maintains all requested actions for a local job and executes them on the computing resource using the BLAH interface to translate to the computing resource client commands. CREAM is designed to be a stand alone component but is integrated with gLite WMS using the Interface to CREAM Environment (ICE) component within the WMS. ICE translates the job requests in the WMS to CREAM methods, subscribes to the CEMON call-back to receive state update information and performs resubmission in the event of transient failures.

1.7.4 Data Management

The gLite supplied data management components allow the storage, cataloguing, transfer and access of files on the Grid. The four components that provide these services are the LCG File Catalog (LFC), the gLite Disk Pool Manager (DPM), the Grid File Access Library (GFAL) and the gLite File Transfer Service (FTS).

LCG File Catalog

The LCG File Catalog (LFC) [124] is a lightweight and scalable file metadata and replica catalog. It exposes a hierarchical, logical namespace and maps constituent files to Globally Unique IDentifiers (GUIDs). Each logical file entry may have associated metadata (size, permissions/ownership, status, checksum, access and modification time) and replica information (host, pfn, number of accesses, status). It offers secure authentication and VOMS based fine grained authorisation with ACL support at the file level. To reduce server load it implements authenticated sessions to reduce repeated handshaking and bulk methods to reduce round trips. Both MySQL and ORACLE backend databases are supported with the additional opportunity to use ORACLE Streaming Technology to replicate LFC contents to remote servers [125].

Disk Pool Manager

The Disk Pool Manager (DPM) [126] is a lightweight disk storage management system. The DPM architecture consists of a *headnode* containing the

DPM Name Server (DPNS), the DPM Server which manages request queuing and space management and the SRM interface servers. The nameserver ensures that authentication and authorisation policies are respected while physical access is obtained through GridFTP and RFIO daemons running on the disk servers. At small sites all the DPM services can be installed on a single machine while in larger instances all services (including the database server) can be distributed across different machines.

Grid File Access Library

The Grid File Access Library (GFAL) and the LCG Utils (*lcg_utils*) [127] are the data management clients supplied in gLite. The GFAL library gives a POSIX interface to local and remote data allowing users to open/read/write/close files. As SRM became accepted as the standard interface to Grid storage elements, GFAL was extended to include SRM client capabilities. The lcg_utils are a higher level client that builds on the GFAL capabilities to allow file replication with SRM based storages. The lcg_utils also provide optional coupling with the LFC such that end users may use logical file references to remove and register physical files.

File Transfer Service

The gLite File Transfer Service (FTS) [128] provides reliable point-to-point file transfers. The FTS architecture is based around a concept of *channels* that group files into task queues with common sources and destinations. The channels are generally between a source-destination pair (although catch all channels are supported) and have their own transfer parameter configuration. This configuration allows control of the number of concurrent transfers, the number of GridFTP streams, TCP buffer size and VO shares. Channel configuration allows site managers to control the load generated against their SRM and GridFTP servers and to meet SLA agreements regarding VO throughput. Clients interact with the FTS web service to submit source-destination URL pairs. These file pairs are assigned to channels and third party GridFTP transfers executed by transfer agents. To ensure authentication and authorisation are respected the FTS interrogates the SRMs and executes the transfers using the client's delegated VOMS credentials.

1.8 Data Intensive Applications

As the understanding of the potential offered by Grid computing has spread the adoption of Grid architecture as a practical tool for academic and industrial applications has increased. The earliest adopters of Grid technologies were in physical and medical sciences where the large volumes of scientific data produced by experiments required storage and analysis. The catalyst for the expansion of Grid usage was provided by national grid initiatives (such as the UK e-Science programme) which provided substantial outreach and industrial crossover funding. In the following section a number of data intensive applications are discussed: an industrial application, a biomedical project and finally an overview of the applications being developed for particle physics.

1.8.1 Distributed Aircraft Maintenance Environment

The Distributed Aircraft Maintenance Environment (DAME) [129], brought together university researchers and commercial collaborators to produce a distributed decision support system for aircraft maintenance. The system performs real-time distributed analysis of in-flight aero-engine monitoring data. This is combined with fault diagnosis and prognostic software to provide a 'predictive maintenance system' to reduce aircraft turn around time and increase safety. The aero-engine monitoring data, up to 1GB per engine for transatlantic flights, produces petabytes of data per year. The OGSA compliant (GT3 based Grid) analyses engine signal data in-flight and allows faults to be discovered and repairs scheduled before the arrival of the flight at the destination. Anomalous engine signal data for engines that later developed problems are retained to ensure future prognosis are more accurate.

1.8.2 Biomedical Informatics Research Network

The Biomedical Informatics Research Network (BIRN) [130] is a collaboration of biomedical researchers with the aim of maximising the research potential of neuro-imaging data produced by various scanning technologies. The data produced varies in type, format and size and can produce over 0.5TB per day. The volume of data and the computing resources required to process this data led to the adoption of Grid technologies. In addition, the

Grid paradigm was suited to the distributed nature of collaborators, their medical imaging machinery and the computing resources available to them. BIRN uses components of Globus Toolkit for security, information systems, resource management and data transfer and the Storage Resource Broker for data storage. The key problem solved by BIRN application software is the 'information mediation' required for researchers conducting studies to discover relevant existing imaging data. A 'virtual database' federates different underlying databases to provide a single query point for researchers that translates their queries into a domain relevant selection for the underlying database.

1.8.3 Particle Physics Applications

The latest generation of particle physics experiments being built at CERN (ALICE, ATLAS, CMS, LHC*b*) have computing and storage requirements an order of magnitude higher than previous particle physics experiments. During the first full year of data taking, it is expected that 140 million SPECint2000 (SI2k) years of processing power[2] and 10PB of storage will be required. The combined scale of the experiment requirements was a factor in the decision to use a Grid computing paradigm. The Models Of Networked Analysis at Regional Centres for LHC Experiments (MONARC) project proposed the use of a hierarchical model [131] that could share the processing and storage responsibility across member countries and institutes:

- The central site in the model is the Tier-0 from which the experiment data originates.
- Tier-1 sites are 'Regional Computing Centres' and typically have a national scope.
- The Tier-2 sites are institutional or university computing centres which associate to a Tier-1.
- Individual computing clusters that belong to physics groups are referred to as a Tier-3.
- The lowest level in the model, the Tier-4, are individual machines which are typically laptops or desktops of group members.

[2]The kilo SPECint2000 will be used for the remainder of this thesis and is denominated as kSI2k. The kSI2k equates to the power of a Pentium Xeon 2.8GHz.

When the model was created the assumption was that the amount of computing, storage and manpower, as well as stability, would decrease from the Tier-0 down to the Tier-4s.

Each experiment has a 'Computing Model' defining the anticipated data-flows and the estimated computing and storage requirements required at each Tier [132, 133, 134, 135]. The experiments use the gLite middleware, but due to historical instability, have build their own frameworks to support their data-flow. The LHC*b* experiment framework (DIRAC), which will be discussed in Chapter 3, provides a combined workload and data management system supporting all aspects of the data-flow in a single project based on a common framework. The ALICE experiment also adopts a single framework approach with the ALIEN project [136], which began just after the publication of the MONARC report.

The ATLAS and CMS experiments, which together demand over 75% of the computational and storage requirements, have adopted specialised systems for different aspects of their computing models. For example, ATLAS initially split the responsibility of their Grid computing projects into four areas; Tier-0 processing, Distributed Production, Distributed Analysis and Distributed Data Management. In addition, within the Distributed Production and Distributed Analysis areas, competing projects were developed for use in different geopolitical regions. The fragmentation of projects was resolved with the adoption of a common Distributed Production system for all ATLAS communities, the Production and Distributed Analysis (PanDA) [137] project based on DIRAC, to allow more effective resource management. The PanDA project is also likely to assume the responsibility for Distributed Analysis to allow the entire ATLAS distributed computing workload to be managed by a single system (the advantages of which will be discussed along with the DIRAC architecture in the Chapter 3). The Distributed Data Management project in ATLAS, responsible for bookkeeping file based metadata and data replication, has a similar scope to PhEDEx project from CMS. These two projects are the most relevant to this thesis and will be discussed in more detail in the following sections.

ATLAS Distributed Data Management

The ATLAS Distributed Data Management (DDM) system [138] has the primary responsibility of bookkeeping ATLAS files and controlling the movement of production data according to the Computing Model, as well as the secondary responsibility of managing local SE access from running jobs. The requirement for data movement in the first full year of data taking is a sustained rate of 600MB/s from CERN to the Tier-1 sites. This requirement increases to 1.1GB/s in the third full year of data taking. In addition, the system must control Tier-1 to Tier-1 data movement of 200MB/s ranging up to 700MB/s in the third year of data taking [132]. The bookkeeping of file metadata is based on heavily on the concept of *datasets*. Within the context of DDM a dataset is defined in the following way:

> Whenever new data files are retrieved....it is necessary to group them together. This grouping of files serves as a type of "native" metadata.....that we define as a "data set"... [139]

With this approach, every file created becomes a member of a 'primary' dataset, grouped to represent some 'common characteristics' of all constituent files. A file may belong only to a single 'primary' dataset and therefore, newly created datasets must contain completely new files or existing datasets. Datasets created from existing datasets are referred to as 'derived' datasets.

To manage this dataset architecture, DDM employs 6 catalogues, each designed to determine specific properties of datasets or their contents.

1. Repository catalog - simple dataset metadata mapping to the versions of the datasets present.
2. Selection catalog - specific metadata for users to locate datasets of interest.
3. Content catalog - constituents of each dataset and version.
4. Location catalog - site location of datasets.
5. Local catalog - location of dataset constituents at a local site.

6. Subscription catalog - dataset subscriptions to trigger data replication.

The ATLAS dataflow is based completely on the concept of subscription, i.e. sites 'subscribe' to datasets which in turn triggers replication of that dataset to a site. The dataset, and all constituent files, become visible in the 'Location' at the site once all files are present, enforcing physical properties of datasets. At each site the physical location of the dataset constituents is mapped by the 'Local' catalog containing only the replica information (i.e. PFN) for constituents at that local site. To gain access to replicas of a given dataset the client interrogates the 'Repository' to ensure the data set exists followed by the 'Location' catalog followed by the 'Local' catalog.

To manage bulk replication DDM builds on top of gLite FTS and the dataset catalogues and deploys a series of agents each handling a small part of the replication chain. Each agent in the chain refers to a single database table which is populated by the preceding agent. A Subscription Resolving Agent finds pending subscription requests. The Replica Resolver determines possible source replicas and chooses based on the number of recent accesses, producing source and destination SURL pairs. The Partitioner determines the state of the FTS channel and partitions the number of files to be submitted, which are in turn submitted by the Submitter. The Pending Handler checks the status of FTS jobs which are treated by the Verifier once complete. If the transfer of a file fails it is populated back to the Replica Resolver to be retried.

CMS Physics Experiment Data Export

The Physics Experiment Data Export (PhEDEx) [140] project, the CMS equivalent of the DDM, handles data placement and transfer for CMS. The data export rates, as required from the CMS Computing TDR [133], from Tier-0 to Tier-1 centres is 625MB/s and a massive 1GB/s between Tier-1s. CMS tackled the data transfer problem early and PhEDEx has developed into mature software. The CMS bookkeeping system uses *datasets* as an abstraction to group *event collections* that would naturally be analysed together and is the smallest unit visible to end users reducing micromanagement of data. The dataset architecture uses three catalogues:

1. Dataset Bookkeeping System - contains metadata about existing data.

2. Data Location Service - contains information on where datasets exist.

3. Local file catalog - contains the location of dataset constituents at a local site.

The Dataset Bookkeeping System (DBS) relates 'datasets' to their constituent files. This catalog is analogous to the combined functionality of the Dataset repository, Dataset selection and Dataset content catalogues used by DDM. The Data Location Service (DLS) maps *file blocks* (groups of files rather than the constituents of a file) to the sites at which they are present but, does not contain any information about the physical location of files (PFNs). It can be considered like the Dataset location catalog of the Atlas DDM. Finally, the Local File catalog maps the constituents of the 'file block' to the physical location on the storage element. Within the CMS system this has been reduced to a 'trivial file catalog', using convention, to map CMS LFNs to PFNs by prepending the SE access information.

The PhEDEx system design uses a network overlay system to describe the topology of their storage resources and their interconnections, independently of the underlying network fabric. Although this approach pre-dated the development of gLite FTS both adopt the same overlay principle which resulted in the FTS being adopted by PhEDEx as the underlying transfer mechanism. With the overlay approach PhEDEx is able to determine, for each node in their network, a minimum spanning tree, or routing table, to all other nodes. These minimum spanning trees are effectively static, with the number of participating sites, but are updated periodically to reflect problematic network links. The PhEDEx architecture is based on a central transfer management *blackboard* where requested tasks are located and distributed agents request pending work, execute tasks and update status. At each site, or node in the overlay network, a number of services and agents are required. The main agents are the Allocator, that assigns files to their destinations based on the transfer requests, and the File Router which determines the closest replica based on the routing tables. The execution of the transfers is performed by 'pulling' the data to the target. The Download agent at the destination requests the Export agent at the source to prepare the source file, to ensure it is available on disk and perform basic integrity checks. Once available the Download agent executes the transfer.

1.9 Summary

This chapter has given a historical account of the Grid computing paradigm from the earliest networking and distributed computing systems, given in Section 1.2, to contemporary data intensive applications in Section 1.8. The question of defining the Grid was discussed in Section 1.3 along with a review of the standards bodies and the architectures they have proposed. It was argued that the expectation of a Grid is what defines it and different types of Grids in use were discussed in Section 1.4. It was noted that Data Grids are the most relevant to this thesis and the requirements and associated emerging standards to build such a Grid was given in Section 1.5. Section 1.6 gave a summary of projects aiming to build Grid infrastructures and those developing the middleware to use them. Special attention was paid to the gLite middleware in Section 1.7 which being used by the four LHC experiments at CERN and many small collaborations.

Chapter 2

LHC*b*: Computing Requirements and Solutions

This chapter details the computing requirements of the LHC*b* experiment. LHC*b*'s computing activity is composed of a number of logical steps, each with specific computing requirements and running distinct software applications. Together these steps represent the chain of LHC*b*'s *workflow* and the data processed and produced at each step creates the logical *dataflow*.

The resources available to LHC*b* to perform its computing operations are located at geographically distributed institutes. The Computing Model associates each step in the workflow with a resource profile in accordance with the tiered computing architecture. The Model also states the physical dataflow to ensure the resilience and availability of the data.

This chapter presents the logical workflow along with the software applications used at each step. The Computing Model dataflow will also be described including the resources required.

2.1 Gaudi: Architecture and Framework

The LHC*b* software strategy follows a stringent architecture approach to ensure changing requirements can be adopted over the lifetime of the experiment. The architecture aims to define a common set of services and components with clearly defined interfaces as well as consistent data and

algorithm handling mechanisms to be used by all event processing applications. Gaudi [141] implements this architecture, fully respecting the design principles, as the framework for LHC*b*'s software applications.

2.1.1 Interface Model

The Gaudi architecture includes a well defined interface model which allows components to be effectively de-coupled from each other. This approach allows transparent inclusion of new software or technologies as long as they offer the same interface. In addition, well defined interfaces for all components allows run-time dynamic library loading to be used in the software applications (both for testing application development code and running individual physicist's private code).

2.1.2 Data and Algorithms

An important aspect of the Gaudi architecture is the de-coupling of objects describing data and the methods for manipulating the data. This approach runs contrary to common object oriented programming paradigms. Within the domain of LHC*b* physics analysis the data structures remain relatively constant while the algorithms to process these may evolve rapidly. Decoupling these allows rapid development, testing and deployment of new or updated algorithms.

The *DataObjects* are containers for physical quantities which may be either persistent or transient. The persistent data objects may be stored using a variety of underlying technologies which are shielded from the consumer of the data using *transient data stores*. Three transient data stores are defined in Gaudi, based on varying access patterns and lifetime:

- Event - single event data with lifetime corresponding to the time to process
- Detector - detector behaviour valid for the time to process a series of events
- Histogram - statistical data produced during event processing

The methods for manipulating data objects are contained in *AlgorithmObjects* which evolve more rapidly than the data objects. All algorithms

offer a standard interface they can be instantiated and executed based on the requirements of the application and are configured using 'job options'. During execution the algorithms get access to the data to be processed through the relevant transient data store.

2.1.3 Services

The service components offer the functionality which is standard and common for all applications developed in the framework. These services are instantiated by Gaudi when the application is executed and used as required. For example the *Histogram Service* gives access to the Histogram data store from any algorithm or application.

2.2 Physics Applications

The Gaudi framework allows physics applications specific to particular steps in the logical workflow to execute in a standard environment. The LHC*b* applications (Gauss, Boole, Brunel and DaVinci) are based on the Gaudi framework. They consume and/or produce data (all of which is conformant to the LHCb event model). These applications are outlined below.

2.2.1 Gauss

The first step in the simulation of physics data is performed by Gauss [142] to study the performance and the behaviour of the detector in response to proton-proton collision events. Within the Gauss application there are two distinct phases: generator and detector response. The generator phase is split into two further phases: event generation of proton-proton collisions using Pythia[143] and B particle decays using EvtGen[144]. The detector response phase uses Geant4[145] to simulate the detector response to particles produced by the generator phase.

For use within LHC*b* the response of Pythia at low energies was tuned and EvtGen, originally developed by the BaBar collaboration[146], was extended to include other B particle decays. To use Geant4 in the response phase the LHC*b* detector geometry is converted to the Geant4 format and the output later reformatted to the LHC*b* Event Model format.

2.2.2 Boole

The Boole application [147] simulates the detector behaviour using *hits* generated by the second phase of the Gauss application. In particular Boole simulates the performance and digitisation of the read out electronics and the L0 trigger hardware. In addition Boole can introduce electronics *spillover* from the preceding and following beam crossings randomly. The output produced by Boole is in the same format as real data coming from the detector after the L0 trigger.

2.2.3 Brunel

The Brunel application [147] reconstructs the digitised output from the Boole application into tracks, clusters and performs particle identification. The Brunel application contains a series of independent processing phases: Reconstruction, Relations and Monitoring. This approach allows the simulation specific processing in the Relations phase, where clusters are associated to MC particles, to be skipped when processing real data. The Monitoring phase, which allows sub-detector specific reconstruction performance to be histogramed, may be executed selectively. This design allows a consistent set of algorithms and conditions data to be used on both real and simulated data in the reconstruction phase and provides the tools to investigate the response of the different sub-detector systems.

2.2.4 DaVinci

The DaVinci application [148] provides the physics analysis framework within LHC*b*. The application supports the selection of events based on supplied criteria, either defined through job options or using supplied user algorithms. DaVinci is configurable to produce different forms of output: an output file containing event data selected to be used for later processing or Analysis Object Data (or Ntuples) files containing physics objects for later processing.

2.3 Logical Workflow and Dataflow

This section outlines the logical dataflow and workflow model for all stages in the processing of real and simulated events as required by LHC*b*. The

nomenclature used for each step of the logical workflow and the data produced at each step is introduced.

2.3.1 RAW data

The High Level Trigger (HLT) receives data at 1MHz and applies increasingly selective algorithms to reduce the output stream to 2kHz. Each event contained in the stream is expected to be 25kB corresponding to an output data rate of 50MB/s. The output stream consists of 4 categories of data; Di-Muon, D* sample, b-inclusive and b-exclusive which are all transferred to Mass Storage (MSS) at CERN in quasi-real-time. The b-exclusive sample, produced at 200Hz, is fully reconstructed at the online farm and the resultant rDST (discussed below) and the ancestor RAW, known as the *hotstream* is transferred to MSS. The provision of resources for LHCb computing is based on an effective running period of 1×10^7 seconds per year. Considering a HLT rate of 2kHz we obtain 2×10^{10} events, which at 25kB per event gives 500TB of RAW data per year.

The RAW data and the hotstream that are transferred to MSS are read from the Online Storage System[149] which provides a temporary buffer for all data output by the HLT. The rate at which data is transferred from the Online Storage to MSS is:

$$\begin{aligned} datarate &= RAW + hotstream \\ &= (2kHz * 25kB) + 2 * (200Hz * 25kB) \\ &= 60MB/s \end{aligned}$$

2.3.2 Reconstruction

Once the RAW data has been created, either real or simulated, it is then 'reconstructed', using the Brunel application, to provide physical qualities and particle identification. The reconstructed events, expected to be 25kB, are written to output files called reduced Data Summary Tapes (rDST). The reconstruction is performed in pseudo real-time when the RAW data is produced. This step may be repeated during reprocessing to include improved reconstruction algorithms, calibration and alignment information.

The Brunel reconstruction CPU requirement per event is 2.4kSI2k.seconds[135]. To reconstruct the entire RAW event sample from the annual running period requires 1.5MSI2k.years and produces a further 500TB of data. As mentioned above this will be done in quasi-real time during the data taking period. In addition, it is expected to completely re-reconstruct the entire sample (at least) once per data taking period, requiring another 1.5MSI2k.years. This will take place out-with normal data taking during which the Online Event Filter (OEF) will be available for 2 months. This provides a power of 5.4MSI2k, corresponding to 0.9MSI2k.years, and 60% of the required resources for re-reconstruction with the remainder being provided at LHCb distributed computing centres.

2.3.3 Stripping

The events stored in the rDST produced at the reconstruction phase are subjected to pre-selection criteria proposed by physics working groups called *stripping*. The events passing this pre-selection, using DaVinci, are fully re-reconstructed, using Brunel, and have their associated RAW data added back in to provide all available event information for analysis. It is therefore required that the stripping activity has access to rDST data files and their ancestor RAW files. The output from the stripping activity is known as a full DST. In addition to the DST output an Event Tag Collection (ETC) is created containing event summary characteristics for a quick event reference. The stripping activity will be performed in pseudo real-time as the reconstructed data becomes available and will also be repeated for any re-processed rDST data. In addition, the stripping may be performed up to an additional two times per year with improved or additional preselection algorithms, calibrations and cuts.

The pre-selection step of the stripping process, performed by DaVinci, is computationally light requiring 0.2kSI2k.seconds[135] per event. For the entire rDST event sample, the selection requires 0.13MSI2k.years. The pre-selection reduction factor varies per event stream, as shown in Table 2.1.

In total $2.09\text{x}10^9$ events are output. Using the CPU requirement per event given in Section 2.3.2, the reconstruction step of the stripping requires

	Reconstruction		Stripping		
Event Stream	Events	Size (kB)	Reduction	Events	Size (kB)
b-inclusive	$9x10^9$	25	100	$9x10^7$	100
b-exclusive	$2x10^9$	25	10	$2x10^8$	100
Dimuon	$6x10^9$	25	5	$1.2x10^9$	50
D*	$3x10^9$	25	5	$6x10^8$	50

Table 2.1: Events information per processing step. The output of the reconstruction produces 25kB events for all Event Streams. In the stripping phase the reduction factor and the size of the resultant stripped events differs between streams.

a total of 0.16MSI2k.years of CPU. Therefore stripping of the entire rDST event sample requires a total of 0.29MSI2k.years. The event size of the stripped events varies with the sample (see Table 2.1), but in total each stripping pass creates 119TB of output data.

2.3.4 Simulated data

The general performance of the detector will be studied using the large number of dimuon and D* samples collected during data taking (see Table 2.1). Therefore the simulation strategy is to concentrate on particular channel decay modes of interest to physics groups. To insure the Monte Carlo statistical error does not dominate the total error a large sample of signal ($2x10^9$) and b-inclusive ($2x10^9$) events are to be produced.

Simulated data is produced from a detailed Monte Carlo model of the LHCb detector incorporating the best understanding of the response. This simulated data is produced using chained Gauss and Boole applications to produce data that is identical in format to the real RAW data from the HLT, but also includes additional simulated hit and and truth information. These events are then reconstructed with Brunel. The Gauss step dominates the CPU requirement with 50kSI2k.seconds per event while Boole consumes 1kSI2k.seconds with the two in total requiring 6.5MSI2k.years to produce all signal and b-inclusive events. During the simulation of the trigger a 10% reduction factor is included in the design which reduces the number of events to be reconstructed to be $4x10^8$, but requires an additional CPU of 30.4kSI2k.years.

The reconstructed MC event size is expected to be 400kB with an additional 1kB for tag information. In total the simulation program creates 160TB of output data and requires 6.5MSI2k.years.

2.3.5 Analysis

Physics analysis uses the DSTs produced by the stripping phase. This is performed using the DaVinci package and produces personal DSTs or Ntuples, used for fitting and to obtain physics results. An estimation of the number of active physicists along with their possible workload gives an estimated annual CPU usage of 0.78MSI2k.years producing 200TB of output data.

A summary of the computing and storage requirements for each stage of the LHCb's computing activity, over the course of a full data taking period, is given in Table 2.2.

	CPU (MSI2k.years)	Storage (TB)	Events
RAW	-	500	$2\text{x}10^{10}$
rDST	1.5	500	$2\text{x}10^{10}$
DST(ETC)	0.3	119 (11.9)	$2.09\text{x}10^{9}$
mcDST(ETC)	6.5	160 (0.4)	$4\text{x}10^{8}$
User	0.8	200	-
Total	9.0	1491.3	$42.5\text{x}10^{9}$

Table 2.2: Resource consumption for each stage of LHCb activity. The CPU column gives the requirement to produce the data at a given step. The Storage column gives the storage requirement for the data and the Events column gives the total number of events at each step.

2.4 Computing Model

The LHCb Computing Model [150] was defined as a result of an iterative process to maximise the use of resources available to the collaboration. LHCb initially requested resources based on the integrated computational and storage requirements, outlined above. The LHCb member states pledged resources to LCG to be located at national Tier-1 and regional Tier-2 centres.

The pledged resources across all Tier-1 centres matched the requirements for data processing and analysis. As a result, the role of each Tier of the Computing Model was determined (see Figure 2.1).

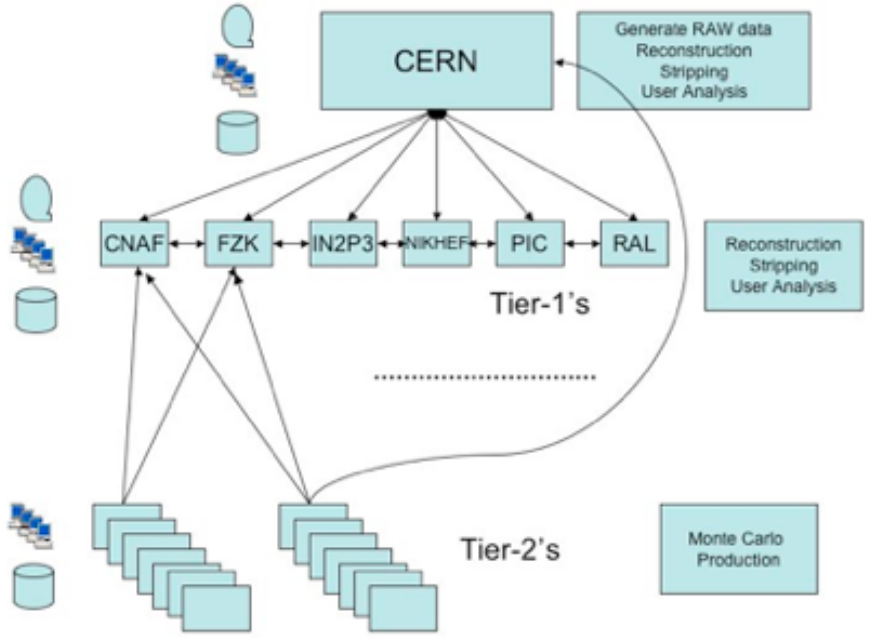


Figure 2.1: LHC*b* Computing Model Schematic.

2.4.1 Tier-0

CERN takes the role of a Tier-0 center and data warehouse. As the RAW data is produced at the LHC*b* detector it is uploaded to the CERN MSS and archived. The Tier-0 maintains the master copies of all RAW data and archive copies of all DSTs produced.

2.4.2 Tier-1

In all other respects CERN is considered a Tier-1. The external Tier-1s are: CNAF (Italy), FZK (Germany), IN2P3 (France), NIKHEF (Netherlands), PIC (Spain) and RAL (United Kingdom). To ensure the persistency of each RAW file it is replicated in quasi real-time to the MSS of one of these external centres. This creates a distributed replica of the entire RAW *dataset*. This data is reconstructed, according to pledged computing resources, where the resulting rDST is archived at the Tier-1 where it was

produced and subsequently stripped. The output of stripping step is a DST file which is maintained on disk and tape at the local MSS. To ensure high data availability for analysis, DST files are replicated to all Tier-1 centres where they are maintained on disk. The user analysis activity, making use of the DST files, is performed at all Tier-1s.

2.4.3 Tier-2

The current computational requirements of the reconstruction, stripping and analysis activities match that of the Tier-1 pledged resources. Therefore the simulation activity required is performed at the Tier-2 centres. The data produced at these sites is uploaded to the associated Tier-1 (based primarily on national affiliation), CERN and two other Tier-1 centres.

2.5 2009 Resource Requirements

The total resources to perform the computing activities of LHC*b* have been given in the previous section. The LHC is expected to provide 14TeV proton-proton collisions for the first time in 2009. The resources pledged at each site for LHC*b* activity during 2009 (retrieved from [151]) provide the basis for the calculation of the site specific requirements of LHC*b*'s activity.

The 2009 resources provided for LHC*b* at the Tier-1 sites is shown in Table 2.3. The total computing resource available is 6.1MSI2k, the aggregated tape and disk storage resources (not including the 0.5PB required to store the entire RAW dataset stored at CERN) is 5.0PB and 3.7PB respectively. The ratio of CPU, disk and tape is approximately equal across external Tier-1 sites with tape pledges exceeding disk pledges by a factor 1.1 and CPU to disk ratio (kSI2k/TB) of approximately 1.8. These ratios are constant across the external Tier-1s as the Computing Model links the ability to process data with the ability to store the output. As the Tier-0 center CERN has a tape copy of all RAW files and a tape and disk copy of all Monte Carlo DSTs which increases the ratio of required storage to CPU.

In the following sections the network rates required while performing the real-time data taking, re-processing, re-stripping and Monte Carlo generation will be presented. These rates are derived from the pledged resources at each site and the overall requirements for the LHC*b* computing activity.

	CPU(kSI2k)	%pledged	%-CERN	Disk(TB)	Tape(TB)
CERN	1056	17.3	-	991	1770
CNAF	583	9.5	11.5	253	286
GRIDKA	812	13.3	16.1	460	512
IN2P3	1342	22.0	26.6	745	829
NIKHEF	1277	20.9	25.3	666	986
PIC	307	5.02	6.1	170	189
RAL	733	12.0	14.5	415	462
Total	6110	-	-	3700	5034

Table 2.3: 2009 Pledged Computing and Tape Resources by Tier-1.

2.5.1 Data Taking

Tier-0 resources

During data taking the 500TB of RAW data produced is archived on tape at the Tier-0. Network resources of 60MB/s are required into MSS from the LHC*b* DAQ and 50MB/s required to export the RAW dataset to the external Tier-1 sites.

Tier-1 resources

The Computing model requires that a distributed copy of the RAW data is present at the external Tier-1s. The share of the data that each site receives is based on the percentage of CPU pledged (ignoring the CERN contribution). This share is given in Table 2.3. The network rates required to transfer this data, which is proportional to the pledged CPU, is shown in Figure 2.2.

The pledge of CPU across all Tier-1s (including CERN) is used to determine what percentage of the RAW data is reconstructed at each site (given in Table 2.3). After the reconstruction step has completed, the stripping step is performed, producing DSTs.[1] The share of the data produced at each site, this time including the CERN contribution, is shown in Figure 2.3.

The DST produced by the stripping phase is uploaded to MSS of the Tier-1 where it was produced and remains available on disk. The rate at

[1]During data taking the quasi real-time reconstruction and stripping activities consume 1.5MSI2k.years and 0.29MSI2k.years respectively.

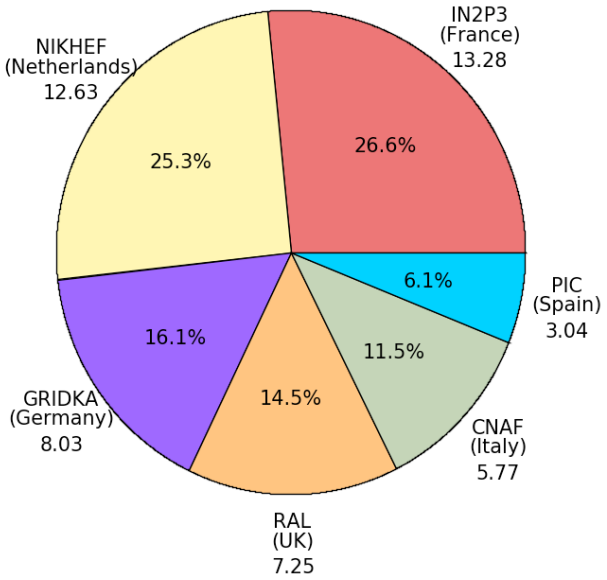


Figure 2.2: Network rate into the Tier-1 sites for RAW data (MB/s).

which DSTs are produced at each Tier-1 is given in Table 2.4. Each DST produced must be replicated to all other Tier-1s resulting in an outbound and inbound network traffic for each site. The rate at which the DSTs are produced derives from the pledged CPU at the site.

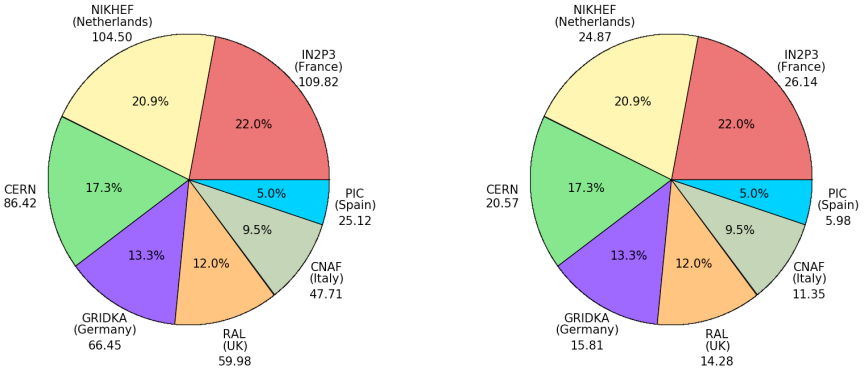
2.5.2 Re-processing Requirements

Re-processing of the entire RAW data set is performed once a year, during the LHC maintenance period. During this exercise the RAW data is re-reconstructed and stripping performed on the resulting rDSTs. This produces 500TB of rDST, 119TB of DST and 20TB of TAG.

Tier-0 resources

During the LHC maintenance the LHC*b* Online Event Filter (OEF) is available and provides 5.4MSI2k for the 2 month period[2]. The RAW data files

[2]Assumed to be 5.18x10^{6}s



(a) rDST (b) DST

Figure 2.3: 2009 Tier-1 Data Produced (TB).

	Produced	Outbound	Inbound
CERN	2.1	12.3	9.8
CNAF	1.1	6.8	10.8
GRIDKA	1.6	9.5	10.3
IN2P3	2.6	15.7	9.3
NIKHEF	2.5	14.9	9.4
PIC	0.6	3.6	11.3
RAL	1.4	8.6	10.5

Table 2.4: Network rates associated to the production of DSTs during data taking (MB/s).

to be analysed at the OEF (42% share of the 500TB) must be transferred from the Tier-0 MSS to the online storage. This requires a sustained rate of 40.5MB/s throughout the period of the exercise. The 210TB of rDST and 58.4TB of DST/TAG produced at the OEF must be transferred back to CERN MSS. This requires a sustained rate of 50.1MB/s during the re-processing period.

Tier-1 resources

The remaining 7.4MSI2k required for the re-processing is obtained from the Tier-1s. The pledge of CPU share determines how much is analysed at each site. A portion (82.7%) of rDSTs produced at the OEF must be moved

from CERN to the Tier-1 centres. This requires an additional 33.5MB/s out of CERN to the Tier-1s. All of the DSTs produced must be replicated to all Tier-1 centres, adding significant network requirements out of CERN to support the transfer of DSTs produced at the OEF.

The breakdown of rates at which rDSTs and DSTs are transferred is given in Table 2.5.

	rDST(out)	rDST(in)	DST(write)	DST(out)	DST(in)
Online	40.5	-	9.6	-	-
CERN	33.5	40.5	1.7	67.8	11.7
CNAF	-	3.9	1.5	9.1	21.4
GRIDKA	-	5.4	1.4	8.8	21.5
IN2P3	-	8.9	2.2	13.3	20.7
NIKHEF	-	8.5	4.7	28.0	18.3
PIC	-	2.0	0.8	4.6	22.2
RAL	-	4.9	1.1	6.4	21.9

Table 2.5: 2009 re-processing data rates for transferring rDST and DST data (MB/s).

2.5.3 Re-stripping Requirements

Two periods of re-stripping are performed during the data taking period and run concurrently with real-time data taking. Each of these strippings is expected to be carried out over a month[3] and produce 119TB of DST and 20TB of TAG. The short period of this exercise implies significant network resources to be available between the Tier-1s, shown in Table 2.6.

2.5.4 Simulation Requirements

The Monte Carlo production takes place throughout the entire year[4] and produces 160TB of DSTs. These DSTs are produced at Tier-2 sites and uploaded to the associated Tier-1. To create a distributed copy each DST is also uploaded to archive storage at Tier-0 and on disk at another two Tier-1s. This implies a modest site transfer rate, given in Table 2.7.

[3] Assumed to be 2.59×10^6s

[4] Assumed to be 31.5×10^6

	DST(write)	DST(out)	DST(in)
CERN	7.9	47.7	38.0
CNAF	4.4	26.3	41.6
GRIDKA	6.1	36.6	39.8
IN2P3	10.1	60.6	35.9
NIKHEF	9.6	57.6	36.3
PIC	2.3	13.9	43.6
RAL	5.5	33.1	40.4

Table 2.6: 2009 re-stripping data rates for transferring DST data (MB/s).

	In from Tier-2	Out to Tier-1	In from Tier-1
CERN	0.88	2.63	4.20
CNAF	048	1.45	1.21
GRIDKA	0.68	2.03	1.02
IN2P3	1.12	3.35	0.58
NIKHEF	1.06	3.18	0.63
PIC	0.26	0.77	1.44
RAL	0.61	1.83	1.08

Table 2.7: 2009 Simulation network requirements (MB/s).

2.5.5 Aggregate 2009 Resource Requirements

The first full year of data taking beginning in 2009 may yield 500TB of RAW data and 1000TB of rDST from two periods of reconstruction. Over 4 periods of stripping 476TB of DSTs will be produced with 80TB of associated TAG. Over the entire year the simulation program will create 160TB of DSTs. Peak network rates will be seen during re-stripping periods where the network rates accumulate the following activities:

- RAW data transfer from CERN to external Tier-1s
- Distribution of DSTs from quasi-real time processing to all Tier-1s
- Distribution of the DSTs from re-stripping to all Tier-1s
- Distribution of the Monte Carlo DSTs to three Tier-1s

A summary of the peak network rates during the re-stripping period is given in Table 2.8. During this time the network resources out of CERN

will be greater than 110MB/s. In addition, significant network rates out of IN2P3 and NIKHEF are required. The required rate of data import is approximately constant across the sites and is around 60MB/s.

	In from Tier-[0-1]	Out to Tier-1
CERN	52.9	112.6
CNAF	59.8	34.6
GRIDKA	59.9	48.1
IN2P3	60.1	79.6
NIKHEF	60.1	75.7
PIC	59.7	18.2
RAL	59.9	43.5
Total	412.3	412.3

Table 2.8: Total required network rates during re-stripping exercise for each site (MB/s). The aggregate total network rate is 412.3MB/s in and out.

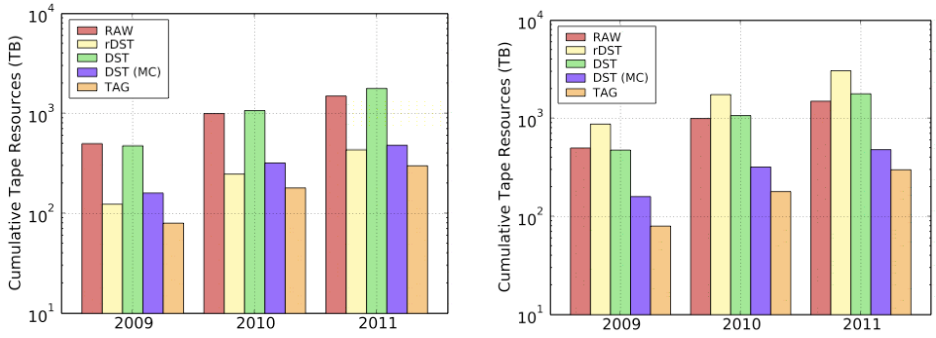
2.6 Accumulated Data

The previous section outlined the resources required in 2009 while performing LHC*b*'s computing activities. This section will provide a summary of the cumulative storage resources required from 2009 until 2011.

2.6.1 Tape

The tape resources required at Tier-0 and the external Tier-1s is given in Figure 2.4.

As the Tier-0 center, CERN maintains a tape copy of all data generated. This is cumulative over subsequent years. The activities performed in 2010 are the same as those performed in 2009 with the additional requirement of reconstructing the 2009 stripped DSTs. This produces an additional 139TB of DST/TAG over the previous year and must be archived at CERN. In addition, a distributed archived copy should be spread across the Tier-1s. The 2011 requirements are the same as those of 2010 with the additional requirement of reconstructing the 2010 stripped DSTs. Each subsequent year the amount of new data produced is the same as the previous year plus 139TB, required to reconstruct the stripped DSTs of the previous year.



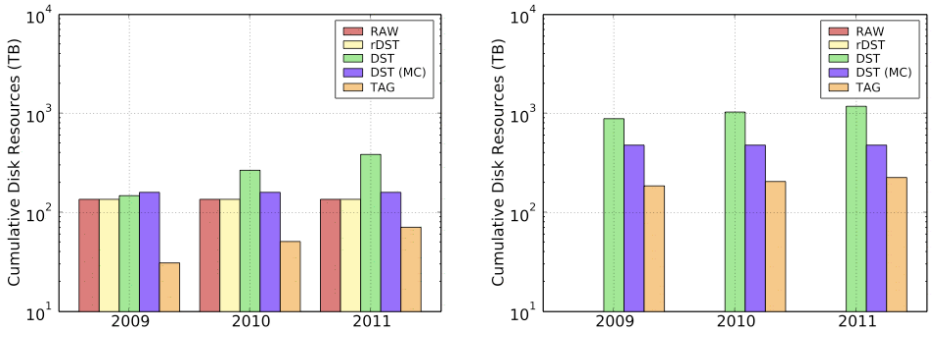
(a) CERN (b) External Tier-1s

Figure 2.4: Tier-0 and Tier-1 Tape Resources 2009 - 2011.

2.6.2 Disk

The requirement for retaining data on disk is not cumulative for all data, see Figure 2.5. The hotstream RAW and rDST, introduced in Section 2.3.1, retained on disk at Tier-0 is replaced as new data becomes available. The required space for this data is 136TB for RAW and rDST. The simulated DSTs, stored on disk at CERN with three distributed copies at Tier-1s, are similarly replaced as new data is generated.

The most recent DSTs from real data, along with the next latest version of the B stream are always kept on disk at all the sites. The most recent DSTs from all previous years data taking are also stored on disk at all sites. The cumulative disk resources are mostly from this requirement.



(a) CERN (b) External Tier-1s

Figure 2.5: Tier-0 and Tier-1 Disk Resources 2009 - 2011.

2.7 Summary

This chapter introduced the computing requirements of LHC*b*. The physics applications used by LHC*b* are based on the Gaudi framework which implements the Gaudi architecture. This allows common services and tools to be shared and new components to be easily adopted. The logical workflow and dataflow to perform LHC*b*'s computing activities was discussed and seen to consist of three major activities: reconstruction, stripping and simulation. The LHC*b* Computing Model assigns specific roles to each of the tiers in the MONARC model, performing all data processing activities at Tier-1 centres and with Tier-2 centres the mainstay of simulation. The resources pledged to LHC*b* for 2009 were presented and from these the networking rates expected during each of the computing activities was derived. Finally the storage requirement for the first three years of data showed cumulative disk and tape resources of several petabytes.

Chapter 3

DIRAC: A Community Grid Solution

The Distributed Infrastructure with Remote Agent Control (DIRAC) is a grid system that supports all aspects of the LHC*b* Computing Model. This chapter will present a brief history of DIRAC from a Monte Carlo generation system to its most recent incarnation as a complete grid solution. The design principles and architecture adopted by DIRAC will be discussed in Sections 3.2 and 3.3. The DIRAC Workload, Data and Production Management systems, that provide the functionality to support the LHC*b* Computing Model, and build upon a common DIRAC framework will be presented in Section 3.4. Finally, the advantages of a using a single system to support the computing activities of a single community are given in Section 3.5.

3.1 Brief History

The DIRAC project began in 2002 with the aim of performing Monte Carlo production for LHC*b* physics community with distributed daemons pulling configuration parameters and workload descriptions from a central database [152], much like the FAFNER project. The success of this exercise provided the motivation for a new generic grid system interoperable with the emerging OGSA standards. As the EDG project was ending and the middleware architecture for the gLite project to replace it was also being proposed. The Architecture Road-map for Distributed Analysis Requirement Technical Assessment Group (ARDA-RTAG) produced a report [153], with input from

the four LHC experiments, that proposed a de-coupled and modular architecture leveraging experience gained by the experiments. It was foreseen that DIRAC (in addition to AliEn) would contribute their developments allowing LHC*b* to integrate their systems with gLite. Instead, gLite chose to develop a Service Oriented Architecture, resulting in the experiments providing *overlay* systems to support their computing models.

During 2004 LHC*b* conducted a physics data challenge (DC04) [154] using an updated version of DIRAC [155] from that used during 2003. Two classes of resources were used for this exercise, firstly those provided by the (newly introduced) LCG and those pledged to LHC*b* directly from university groups and managed by DIRAC. This exercise was the first successful demonstration of a heavy workload supported by LCG. In 2005 the DIRAC Review was conducted producing a set of recommendations [156] for the continued development of DIRAC. During 2006, and continuing into 2007, a Data Challenge (DC06) [157] was performed to produce data for the LHC*b* physics book. DC06 exercised many aspects of the Computing Model, making use of significantly expanded resources available in LCG. Monte Carlo production activity was conducted at Tier-2 sites, data transfer from Tier-0 to Tier-1 [158] and reconstruction and stripping of data at the Tier-1s. By the end of DC06 DIRAC [159] has matured to include production management capabilities [160], as well as a workload management system capable of supporting large production activities and user analysis [161, 162] all built on a secure service framework [163].

After most of the DC06 exercise was complete the energy of the DIRAC team was directed towards the implementation of the DIRAC Review recommendations. Instead of incrementally evolving DIRAC, the decision was taken to meet the recommendations building all components freshly upon the updated secure framework. The product of these developments is DIRAC3.

3.2 DIRAC Design Principles

The DIRAC3 system was primarily designed to support all aspects of the LHC*b* Computing Model. One of the core philosophies within the development program was to keep the system generic, adopting a pluggable archi-

tecture to support specific LHC*b* policies. This design principle was taken with the assumption that all future use-cases could not be foreseen and that they should be easily supported, whether from LHC*b*, or any other community. Another core principle was to ensure that DIRAC provides all the services required to perform distributed computing on heterogeneous resources. Therefore, DIRAC can be used as an overlay network on top of existing middleware (for example gLite) or in a standalone environment. The adoption of the pluggable architecture also allows an overlay network to be converted into standalone environment (and vice versa) with no development required.

The programming language used throughout the lifetime of DIRAC is Python. This language was chosen because it is lightweight, portable and freely available for many computing platforms. Similarly, DIRAC itself is lightweight and portable with few external dependencies and can be used with many operating systems and architectures, even Microsoft Windows [164]. The term lightweight is relevant in terms of the code base itself, but also in the context of resource management. To maximise the resources available to LHC*b* the threshold for participation in the computing activities of DIRAC is low. When running within existing grid infrastructures (such as gLite) DIRAC requires no additional services to be installed at a site. In a completely standalone environment a computing resource, such as a university computing cluster, can be integrated by allowing DIRAC to query the underlying batch system. Where remote access to the batch system is unavailable, only a single DIRAC agent needs to be installed with access to the batch system client.

A key lesson learned over the lifetime of DIRAC is that utilising computing and storage resources in a distributed environment is not deterministic. The first DIRAC versions adopted a *pull scheduling* paradigm, to be discussed in more detail in Section 3.4.2. This allowed DIRAC to mitigate instabilities in the underlying computing resources by only attempting to perform tasks in a sane execution environment. This philosophically sceptical approach results in a more stable system at the level of end users who are shielded from the underlying instability. This approach has infused all levels of DIRAC with redundancy. Rather than attempting to predict all possible future sources of failure DIRAC aims to provide the mechanisms to

recover. With this approach DIRAC has evolved into a flexible and resilient system.

3.3 DIRAC Architecture

DIRAC implements a Services Oriented Architecture (SOA) decomposing functionality into 4 broad categories: Resources, Agents, Services and Interfaces.

Resources in the DIRAC context corresponds to the underlying computing and storage available to DIRAC. In a standalone environment this maps to the fabric layer resources. DIRAC supports many common batch systems through a plugin mechanism that translates standard interface to the specific relevant commands. Similarly, in the data domain, with no grid services available, various common transfer protocols are supported. Where no transfer protocols are supported by the underlying resource DIRAC can overlay a mounted file system through a simple Storage Element service. In a grid overlay configuration DIRAC has been instrumented to submit jobs directly to standards-based computing elements and resource brokers.

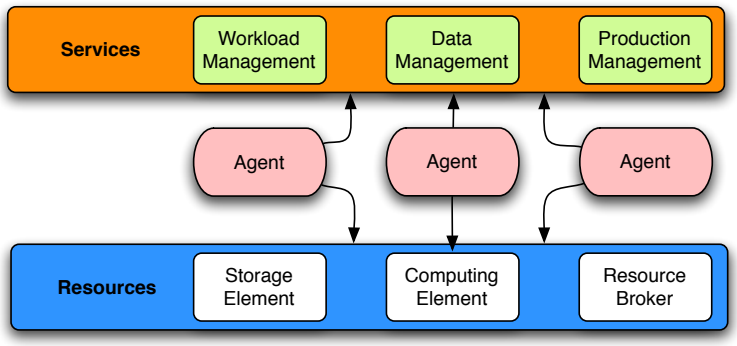


Figure 3.1: Interaction Of DIRAC Services, Agents and Resources.

The Resources described above are passive awaiting invocation. The primary consumer of Resources are DIRAC Agents which are active components that perform a specific function within the system. Agents also interact with DIRAC Services that perform operations on request. The interaction between DIRAC Services, Agents and Resources is shown in Figure

3.1. DIRAC services can be stateless or stateful, with state usually maintained in database backend[1]. While agents may be deployed anywhere in the distributed environment services are centrally managed in a controlled environment. Each service offers a client interface used by agents while performing their activity. These clients are also used by DIRAC Interfaces which give end users access to the system through programmable Python APIs or the Web.

3.4 DIRAC Systems

The architecture of agents interacting with resources and services provides a flexible and extensible approach. Each DIRAC system groups services and agents to provide a set of required functionalities. The important systems are the Workload Management system, the Data Management system and the Production Management system. These systems all build on the DIRAC Framework which provides a common execution environment and a set of tools for all components allowing rapid application level development.

3.4.1 Framework

At the core of the DIRAC framework is the DIRAC SEcure Transport (DISET) [163, 165] layer. DISET provides the secure communication between DIRAC services and clients using OpenSSL to perform authentication and encryption using X509 certificates. The DISET protocol provides Remote Procedure Call (RPC) functionality, embedded into DIRAC services and clients, which supports user and group level authorisation capabilities. DISET also provides file transfer capabilities with bulk support for directory transfer. A schematic of the DISET architecture is given in Figure 3.2.

DISET is the lowest layer in the DIRAC software stack. On top of this, the framework provides a set of utilities commonly required in application programming environments. Every DIRAC component has access to these tools, that may be referenced from any level in the code. Of primary importance is Configuration which must be retrieved by any executing DIRAC

[1]The database backend is usually MySQL with Oracle supported for DIRAC Bookkeeping Service.

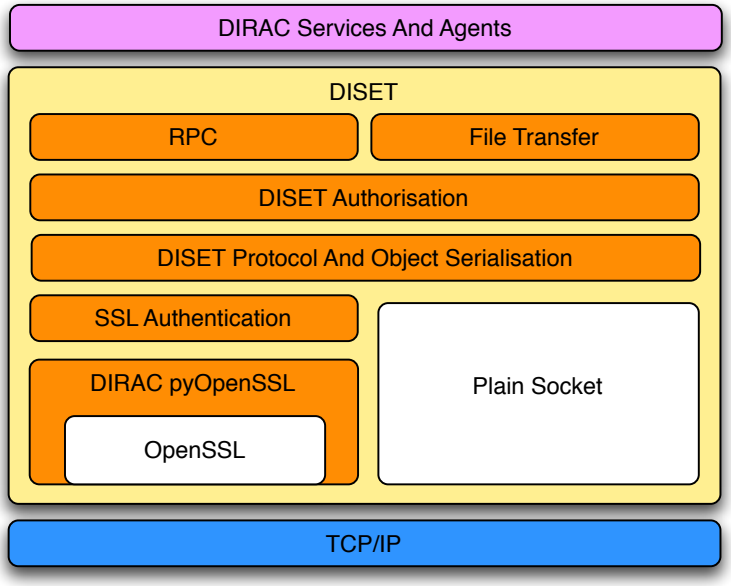


Figure 3.2: DISET Layered Architecture.

component to obtain timely data regarding resources or for service discovery. To ensure the availability and resilience of the configuration data the Configuration Services (CS) are organised in a hierarchical structure with a single master with multiple slaves. The geographically distributed slaves are synchronised automatically with the master CS and provide the necessary redundancy to ensure 100% availability of the configuration data to DIRAC services and agents.

The remaining three utilities are a global Logging service that allows applications to send logging messages of varying status to standard output or to a remote service, the Monitoring service which allows clients to send periodic messages regarding the current state of the component and the Accounting service which allows complex metrics to be formulated and later reviewed. To further aid the development of application services and agents, these tools are built into a set of base classes that provide a fully configurable and secure execution environment [166].

3.4.2 Workload Management

The Workload Management System is the raison d'être for DIRAC. It delivers capabilities to clients to submit and monitor computation jobs and to retrieve their output. To deliver increased stability DIRAC employs a series of mechanisms to shield users from the transient failures on the grid. To facilitate increased stability the job in the DIRAC context is separated from the task executed on the underlying resource. This allows multiple tasks to be executed on behalf of a single DIRAC job, thus diminishing the role of transient errors.

The first mechanism to provide increased stability is that of shallow rescheduling, whereby jobs which failed with transient errors are automatically retried by the system.

The most important mechanism for delivering stability is the adoption of *pull scheduling*. Pull scheduling assumes the execution environment found on distributed computing resources may not be standard or consistent. Only tasks with requirements matching those available on the worker node should be considered for execution. This *late binding* also allows DIRAC to apply scheduling decisions based on priorities with significantly reduced latency since the decision on which job to execute is made when the resource is already available. To achieve this DIRAC uses light agents, or *Pilot Jobs*, which are sent to the underlying computing resource to check the execution environment and then request an eligible job from DIRAC. In the case of the WLCG grid, Pilot Jobs are sent as regular jobs not requiring any special treatment. Typically, pilot agents are executed within a batch system with an allocated CPU and wall clock time limit. Where DIRAC is available to optimise the use of a batch slot, multiple jobs may be run within the time limits thereby increasing their usage efficiency [161, 167].

The Workload Management system is composed of four major services and a series of agents that together facilitate a *pull scheduling* architecture. This is shown in Figure 3.3.

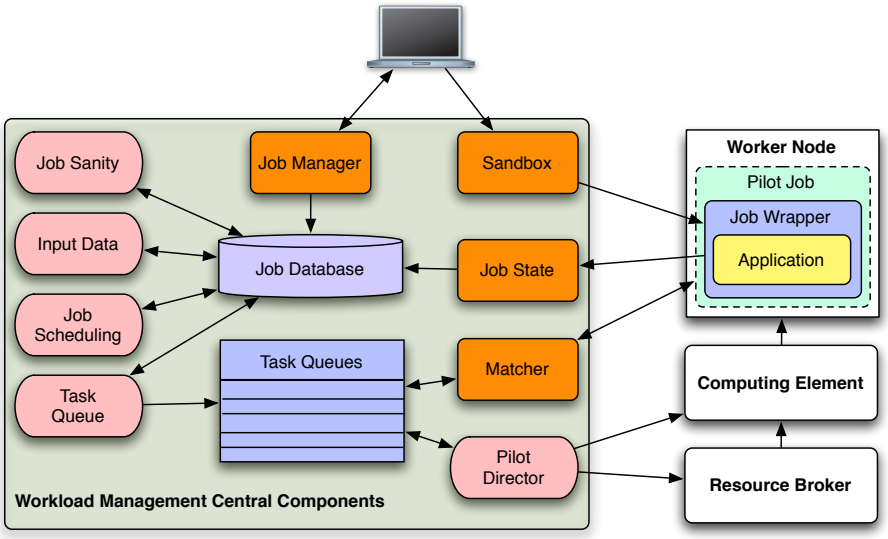


Figure 3.3: DIRAC Workload Management System Architecture.

Services

The Job Manager service is the entry point for clients and allows the submission of jobs to DIRAC. The DIRAC API provides a client side wrapper to allow users to formulate their tasks in a programatic way. These are translated into Job Description Language (JDL) format and are provided in a RPC connection to the Job Manager. The Job Manager is responsible for creating a new entry in the database backend describing completely the requirements of the job and the identity and group of the job owner. The DIRAC API also allows users to specify *input sandbox* files to be used during job execution which are uploaded to the Sandbox service[2].

A newly entered job is treated by a series of agents, to be discussed in Section 3.4.2, and is placed in a task queue grouping jobs with the same requirements. This job is then eligible to be matched to Pilot Jobs reporting back to the Matcher service. The Matcher translates the resource capabilities reported by the Pilot Job into a list of eligible task queues and selects a task queue and job based on the priorities assigned to each. The job JDL is then returned to the pilot agent which then executes the payload described.

[2]The Sandbox service is deployed as two distinct services to deal with input and output sandboxes

During execution the job periodically reports back *heartbeat* and progress information to the Job State Update service to ensure that the job is executing as planned. Once the task has completed, the Pilot Job may check the time remaining in the batch slot and attempt to match a new job with the updated CPU limit requirement.

Agents

As mentioned above DIRAC logically splits a DIRAC job from the Pilot Job that may eventually execute it. In this respect two broad categories of agents exist in the Workload Management system: those involved with job scheduling and those involved with Pilot Job management. The agents on the left of Figure 3.3 manage job scheduling and those on the right manage Pilot Jobs.

Once a job is received by the Job Manager it must be treated by a series of agents before being assigned to a task queue. Each agent performs a specific and de-coupled action in the scheduling process of each job. The requirements of each job determine which of these agents needs to be applied before making the final scheduling decision. The Job Path agent determines which subsequent steps are to be performed, assigning each job a *path* through the agents, to ensure all relevant information is available for making the scheduling decision. This design allows new steps to be added and policies to be modified without significant development. The most common examples of these agents are given here.

The Job Sanity agent performs a check of all of the provided requirements to ensure that the job can execute successfully. This ranges from ensuring that any input sandbox files are correctly uploaded and available, checking the input data is correctly specified or that any specified output data does not already exist. This step removes any jobs that a priori would fail from being attempted. The Input Data agent is invoked when input data is supplied in the job description and provides a key example of the advantages of a flexible scheduling process. This agent retrieves the replica information for the supplied files and persists this information in the database. The Job Scheduling agent collects all the information gathered during the previous steps and selects the sites at which the job is eligible to be executed. Fi-

nally, once the eligible sites have been assigned the job is assigned to the task queues supporting all of the job requirements.

The management of Pilot Jobs is performed by two agents: the Pilot Director and the Pilot Monitor. The Director evaluates the task queues, their requirements the number of jobs and their priorities to determine the number of Pilot Jobs to be submitted. These Pilot Jobs can be submitted to a grid resource broker or directly to a computing element and are given requirements matching those of the task queue. This ensures that only resources capable to execute the workload are reserved. The Pilot Monitor checks the status of the submitted Pilot Jobs to ensure that too many or too few Pilots are not submitted for the workload present in the task queues. Failed Pilot Jobs are replaced by newly submitted ones recovering the grid job inefficiencies.

3.4.3 Production Management

The Workload Management system provides users the ability to run their jobs with an increased level of stability and performance. In addition to the ability for running individual jobs the LHC*b* Computing Model prescribes the need for large, centrally managed production to generate Monte Carlo data and to process RAW physics data produced at the LHC*b* detector. The Production Management system is built on top of the Workload Management system creating and submitting jobs to fulfil production requests.

The Production Manager server offers a service interface to allow the definition of production requests to be stored in the database backend. Each production is defined by a *Workflow* that describes:

- The applications to be run and their configuration
- The data to be produced and the destination location
- Any input data requirements

When a Data Processing type production is defined any input data specified is retrieved, from the LHC*b* Bookkeeping, and the input files are placed in a dedicated table in the database. As new files are produced they are

registered in the Production Manager service and assigned to productions with matching input data requirements.

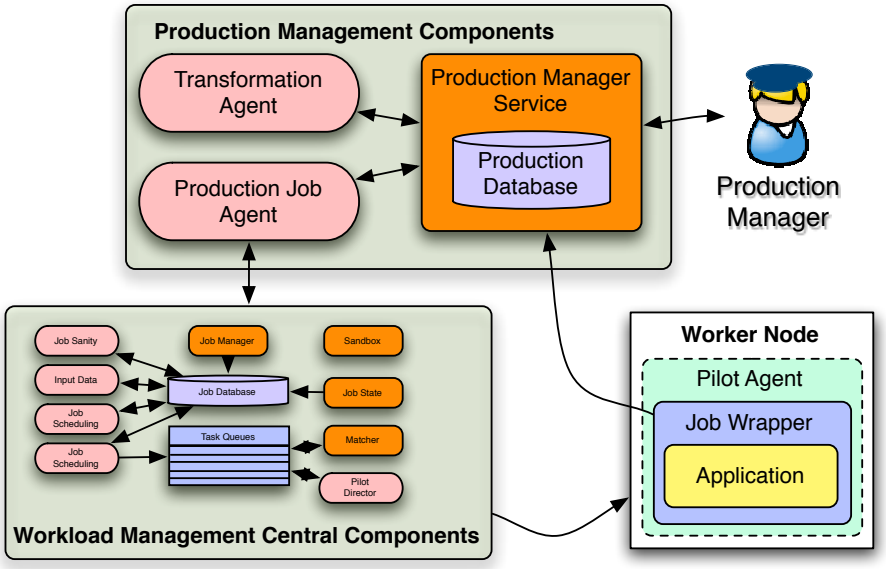


Figure 3.4: DIRAC Production Management System Architecture.

The Transformation Agent periodically polls for active productions from the Production Manager. For Monte Carlo productions the agent creates jobs to meet the number specified in the production definition, corresponding to the number of events requested. For Processing productions the agent groups candidate input files according to their location and creates jobs to process a given number of input files, as defined in the production. These jobs are created and stored in the database backend such that they can be submitted to the Workload Management system asynchronously by the Production Job agent. The architecture of the Production Management system is shown in Figure 3.4.

The Production Management system supports the Computing Model in a data driven and coordinated way. The real-time reconstruction and stripping of RAW physics data can be performed as RAW data becomes available and is registered in the Production Manager. The existence of the RAW files will produce a reconstruction job, which in turn will create

a rDST file, which will trigger the creation of a stripping job. Therefore, two production definitions are required to manage the real-time processing of LHC*b* data. As the jobs generated from these productions complete the input data is marked as *processed* in the Production Manager, while failed jobs may mark their input data such that they may be used to create a fresh job.

3.4.4 Data Management

The Data Management system is responsible for file transfer and cataloguing of all of LHC*b* data according to the LHC*b* Computing Model. The responsibilities of the data management system start with the files written at the Online system that must be reliably uploaded to mass storage at CERN. It ensures that the data flows described by the computing model, between the LHC*b* computing centres, are performed. Finally, it manages the data produced using DIRAC by all production and user activities. The DIRAC Data Management system will be discussed fully in Chapter 4.

3.5 The Advantages of a Community Solution

DIRAC has attempted to provide all the software necessary for a community to perform distributed computing. The systems that constitute DIRAC are logically separated and can be used independently of each other. Together they provide a complete grid solution. Grid computing projects undertaken within the particle physics community, like those discussed in Section 1.8.3, have concentrated on supporting specific activities within experiment computing models. The experiment computing models require end-to-end data processing and data transfer. The specialised systems, that support specific parts of these activities, must be coupled with their counterparts to achieve this. Building on a single common framework, that provides a secure application environment, allows rapid development of the functional code and provides implicit communication between the systems.

A combined solution also allows a community to manage their computing resources in a consistent way. For example, the DIRAC workload management system allows the submission of production and user type jobs which have different resource usage patterns. In addition, the production and user

activities have varying priorities per production and per user (perhaps down to the level of individual jobs). A workload management system that supports all community activities allows priorities and fair-share to be easily applied, managed and updated as use-cases evolve. This places the management of the available resources in the hands of the community. Community policy may also be applied to transfer activities where data required by urgent productions can be replicated across available storage elements, with priority, to increase data availability and processing throughput.

Chapter 4

DIRAC Data Management Design

The previous chapter discussed the DIRAC architecture, the design principles and the components that make up the Workload and Production Management systems. This chapter will present the DIRAC Data Management System and the design choices made.

The components that form the core of the data management system will be described in Section 4.1. These are the basic tools on which the remainder of the functionality is built. The bulk transfer framework, which supports all data replication activity within LHC*b* will be described in Section 4.2 and the mechanism for supporting the coordinated dataflow outlined by the Computing Model is described in Section 4.3. Ensuring the integrity of LHC*b*'s data is vital and the tools designed to maintain the consistency of the DIRAC resources is given in Section 4.4. Finally, the architecture of the system for managing logical *datasets* within LHC*b* is presented in Section 4.5.

4.1 Core Data Management Components

The DIRAC architecture consists of Resources, Services, Agents and Interfaces. The core Resources used by the DIRAC Data Management system are grid storage elements (SE) and file catalogues. To support the use of these resources a Storage Element and a File Catalog abstraction layer have

been employed. These functionalities are combined by the Replica Manager to perform simple file management on the grid. The Storage Element, File Catalog and the Replica Manager will be discussed in the following sections. When file management operations fail, because of transient resource level errors, a mechanism for persisting and retying the operations asynchronously is presented.

4.1.1 Storage Element

The DIRAC Storage Element is an abstraction layer used to mask underlying storage technologies and protocols. Primarily, it is used to perform the upload, download and replication of files and directories to/from grid SEs. In addition, it offers the ability to obtain file and directory metadata and to perform file manipulation (pre-stage, removal etc.)

To support a variety of standard and proprietary protocols, the Storage Element adopts a plugin architecture, whereby each protocol supported by DIRAC has an associated plugin, implementing a standard interface. These may be used interchangeably within the Storage Element. This design allows new protocols to be adopted without significant development. A Unified Modelling Language (UML) class diagram of the Storage Element is shown in Figure 4.1.

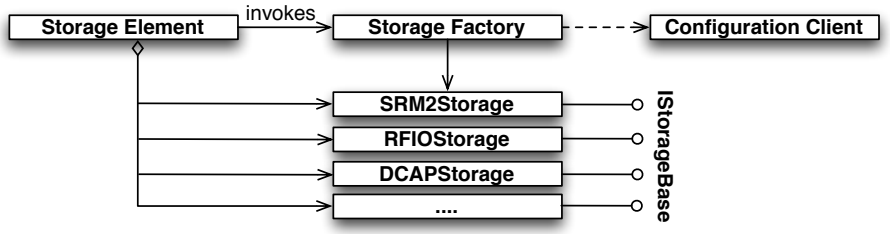


Figure 4.1: Storage Element Class Diagram.

When creating a DIRAC Storage Element a *name* must be supplied which references an entry contained in the DIRAC Configuration Service (CS). This name is provided to the Storage Factory to create storage plugin objects (e.g. SRM2Storage), based on the connection details found in the configuration. These plugin objects are then returned to and persisted by the Storage Element. When operations are performed, each of the storage

plugins is attempted in turn until the operation is successful. The order of preference in which they are executed is based on the configuration, i.e. 'AccessProtocol.1' is attempted before 'AccessProtocol.2'. This adds to the resilience of the Storage Element.

An example configuration, describing the SE at CERN for storing LHC*b*'s RAW data, is given below. For this SE there are two protocols supported, SRM2 and RFIO.

```
CERN-RAW
{
  BackendType = Castor2
  AccessProtocol.1
  {
    Access = remote
    ProtocolName = SRM2
    Protocol = srm
    Host = srm-lhcb.cern.ch
    Port = 8443
    WSUrl = /srm/managerv2?SFN=
    Path = /castor/cern.ch/grid
    SpaceToken = LHCb_RAW
  }
  AccessProtocol.2
  {
    Access = local
    ProtocolName = RFIO
    Protocol = castor
    Host = castorlhcb
    Port = 9002
    Path = /castor/cern.ch/grid
    SpaceToken = lhcbraw
  }
}
```

There are two mandatory elements of the protocol configuration for Storage Factory to instantiate plugin objects. The 'ProtocolName' element is used to locate the plugin to be instantiated. The 'Access' element allows Storage Element to know whether a protocol must must be used locally or whether it may also be used remotely. This distinction is required as SEs may block certain protocol use from off-site for security reasons. In the

example above, SRM2 is a 'remote' protocol while RFIO is 'local'. When performing an operation, the Storage Element object first determines the location in which it has been invoked and its relation to the underlying grid SE, based on the DIRAC configuration, to evaluate which protocols are suitable for use.

The remaining elements in each protocol configuration are the connection elements. The number of elements required may vary between plugins as they are used to construct the URLs understood by the protocol libraries e.g. the SRM2 plugin requires 6 connection elements to construct the URL while the RFIO plugin requires only 5 (the File plugin, not shown above, requires only 2).

During the WLCG workshop in Mumbai 2006 [168], the LHC experiments agreed that the interface to grid SEs used in LCG should be SRM (the evolution of which is discussed in Appendix A). The SRM protocol is a control protocol with the aim of providing transparent access to disparate SE implementations. For this reason, SRM2 is the primary storage plugin used by the Storage Element. The SRM2 plugin uses the GFAL Python binding to contact SRM services and *lcgutils* to perform file transfer. The performance of the LHC*b* SRMs using the GFAL client is presented in Section 5.2.

4.1.2 File Catalog

The DIRAC File Catalog, like the Storage Element, is an abstraction layer, used to maintain a single point of access to different replica and metadata catalogues. It adopts a plugin architecture, similar to that of Storage Element, to allow the support of new catalogues as requirements dictate. The philosophy of File Catalog differs from Storage Element in that each operation should be carried out on all available catalog plugins to maintain their mutual consistency. The plugin architecture relies on each plugin realising a File Catalog interface. A UML class diagram of File Catalog is shown in Figure 4.2.

When a File Catalog object is created it searches the DIRAC configuration to obtain the current list of known catalogues. An example con-

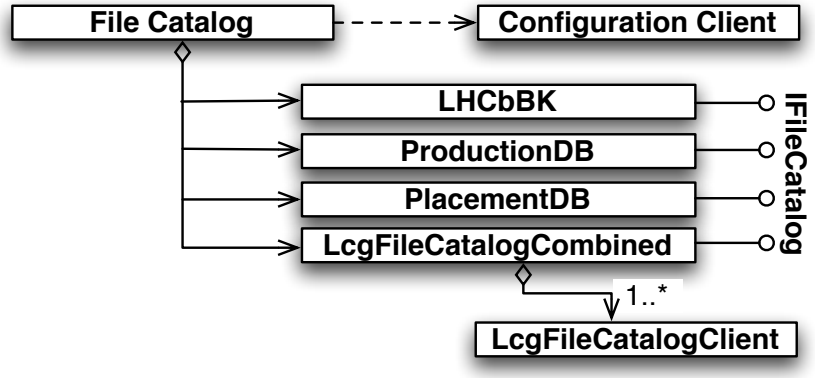


Figure 4.2: File Catalog Class Diagram.

figuration slice, containing two catalogues, 'LcgFileCatalogCombined' and 'ProductionDB', is shown below.

```
FileCatalogs
{
  LcgFileCatalogCombined
  {
    Status = Active
    AccessType = Read-Write
    Master = True
    LcgGfalInfosys = lcg-bdii.cern.ch:2170
    MasterHost = lfc-lhcb.cern.ch
    ReadOnlyHosts = lfc-lhcb-ro.cern.ch
  }
  ProductionDB
  {
    Status = InActive
    AccessType = Write
    Master = False
  }
}
```

The catalog name is used to locate the catalog plugin to be instantiated. For each catalog there are two mandatory and one optional configuration elements. The mandatory elements are:

- Status - whether or not a catalogue is 'Active'
- AccessType - whether a catalogue is 'Read', 'Write' or 'Read-Write'

The 'Master' element is optional and allows the File Catalog to determine which catalog to consider as primary. Within LHC*b* the LFC contents are assumed to be primary and correct. Therefore, if registration in the LFC does not succeed no other registration operations should be performed.

The File Catalog separates the read and write catalogues based on the 'AccessType' configuration element. File Catalog acts as a simple dispatcher, directing queries to 'Read' catalogues and registration/removals to 'Write' catalogues. This separation is performed as read and write operations have a fundamental difference: 'Write' operations should be performed on all catalogues to ensure mutual consistency while 'Read' operations should only be directed to catalogues for which the operation is sane.

LCG File Catalog

The LCG File Catalog (LFC), described in Section 1.7.4, is the primary catalog of LHC*b*. The LFC is used as a replica catalog to store the DIRAC SE names and PFNs for each replica associated to registered LFNs[1]. In addition to replica information, the LFC is used to store file metadata (size, GUID, checksum, status) and replica metadata (status). The most consumed LFC information is the replica information, used for job scheduling, data upload and replication. As described in Section 1.8.3, other grid Data Management systems have chosen an architecture with distributed 'site-specific' catalogues due to the volume of data being managed. In comparison, DIRAC manages a smaller data volume and has a single, centralised catalogue containing all managed files. This approach has shown to be scalable to the current level of O(10M) replicas and has several advantages. A single central catalogue simplifies the operations required to obtain replica information and reduces the number of components required to be operational. The centralised architecture employed by DIRAC does have a single, major drawback: single point of failure.

To ensure maximum availability of replica information a distributed LFC service is employed, shown in Figure 4.3. To provide redundancy in the availability of replica information DIRAC uses distributed read-only catalog

[1]A full explanation of the mappings between LFNs, GUIDs, PFNs and DIRAC SE names is given in Section 5.1.

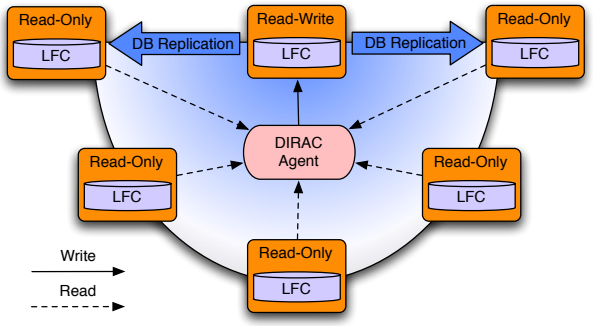


Figure 4.3: Architecture for Ensuring Availability of Replica Information.

mirrors at LHC*b*'s Tier-1s, populated from the central instance using Oracle Streaming Technology [125]. Producers of replica information must contact the read/write master instance while queries can use any of the read-only mirror catalogues, with the additional benefit of reducing the load on the master. In the event the master instance is unreachable, registration *requests* are persisted using the mechanism presented in Section 4.1.4. These requests are retried asynchronously until successful and provide additional redundancy for the master instance.

The performance of the LFC for registration, querying and removing file and replica metadata is fundamental to the scalability of the DIRAC Data Management system. A study of the capability to perform all of these operations is presented in Section 5.1.

4.1.3 Replica Manager

The Replica Manager (RM) combines the functionalities of the Storage Element and File Catalog to provide a coherent interface for performing file based data management and accounting. This ensures that the consistency of the File Catalog and Storage Element is maintained. A UML class diagram of the RM is shown in Figure 4.4.

The Replica Manager interface allows clients to upload files to grid SEs and register their contents, download data specified as an LFN from a grid SE with best replica selection, replicate and register LFNs to grid SEs. It also allows the removal of files and replicas from grid SEs and catalogues. The success or failure of each attempted operation is returned to the client

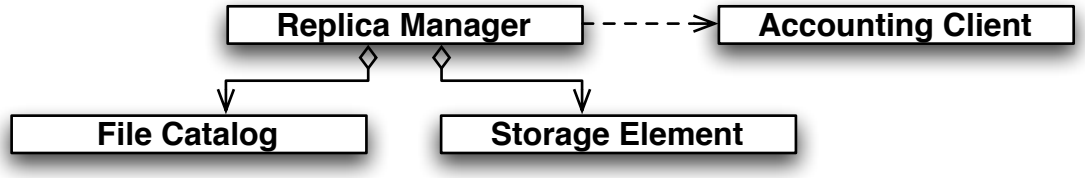


Figure 4.4: Replica Manager Class Diagram.

along with any timing information available. This timing information is used to create and send accounting reports for operations attempted.

The Replica Manager and the underlying Storage Element and File Catalog use redundancy wherever possible to complete the requested operation. To provide additional resilience to transient failures any failed Replica Manager operation may be persisted in a Request Database, discussed in Section 4.1.4 and retried asynchronously by specialised Data Management agents, described in Section 4.1.5.

4.1.4 Request Database

Each data management operation that may be attempted in the Replica Manager interface can be persisted as a *request* in the event of failure. This request is a generic XML [33] representation of the information required to execute the operation at a later time. Once created the request must be *set* to a Request Database service which maintains the XML contents either in a MySQL or file based database backend. A distributed 'failover' mechanism (shown in Figure 4.5), using Request DB services deployed on each of LHC*b*'s Tier-1 VO boxes, ensures the persistency of requests.

In addition to data management operations the XML schema may represent any client-server operation possible in DIRAC. This mechanism is used across DIRAC to ensure failed server updates can be retried asynchronously.

4.1.5 Data Management Agents

The requests persisted within the Request Databases are retrieved and executed by a series of agents, each capable of handling requests of a particular type. When creating a request the 'type' parameter must be provided. This field corresponds to the agent that should execute the request e.g. *transfer*

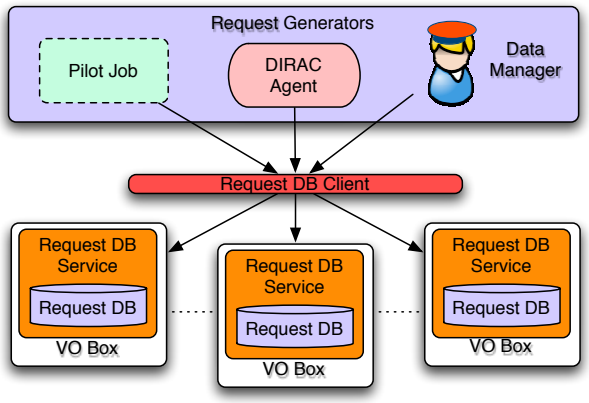


Figure 4.5: Distributed Request DB Service Schematic.

requests are retrieved by the Transfer Agent, *removal* requests by the Removal Agent and *register* requests are retrieved by the Registration Agent. An 'operation' field must also be supplied which informs the agents what functionality is being requested, which in the Data Management context maps to the Replica Manager interface.

The decision to have distinct agents for each type of request was taken to fully exploit the bulk functionalities available at the Resource level (these will be presented in Chapter 5). Individual agents with a specific scope can group requests for similar operations making use of bulk interactions with the Resources wherever possible. The Registration Agent uses the bulk functionalities provided in the LFC, while the Removal Agent uses both the LFC *and* SRM bulk functionalities. The network requirements to perform data transfer derived from the Computing Model necessitates the use of a bulk transfer mechanism. The bulk transfer framework will be discussed in the next section.

4.2 Bulk Transfer Framework

The bulk transfer framework architecture has been defined by an iterative process. The first bulk transfer mechanism employed, managed centrally coordinated transfer activity. It obtained transfer requests and submitted a job to the gLite File Transfer Service (FTS), discussed in Section 1.7.4, for all the files contained in the request. The system provided a simple wrapper

to submit and monitor FTS jobs.

Transfer requests generated by jobs were collected in the distributed Request Databases present on LHC*b VO boxes* at the Tier-1s, where they were executed by Transfer Agents. This system provided the advantages of a distributed 'failover' mechanism with the disadvantage that there was no global knowledge of the behaviour of the other agents. The load placed on the SEs with this system proved unpredictable and resulted in degraded global performance.

The eventual solution combined the two mechanisms previously employed into a centralised repository with distributed failover. The components comprising the bulk transfer framework will be discussed in this section and their interactions are represented schematically in Figure 4.6.

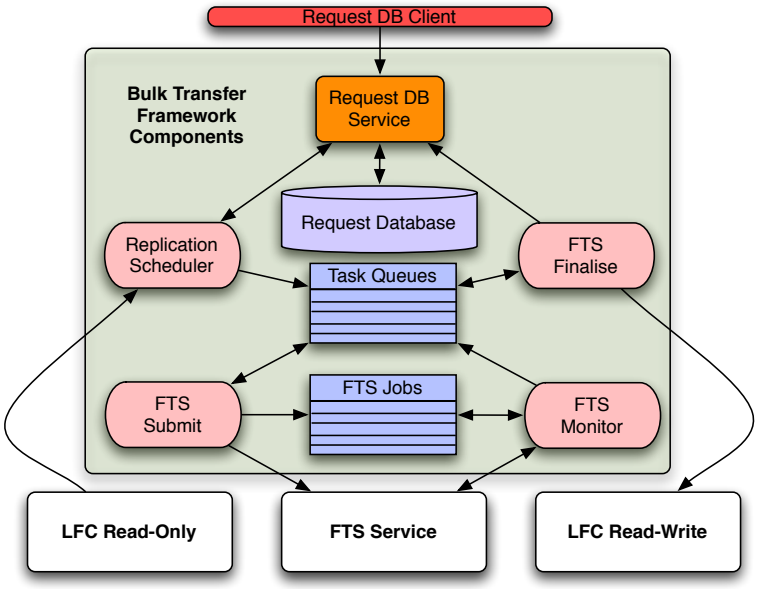


Figure 4.6: The DIRAC bulk transfer framework components.

4.2.1 Transfer Database

A centralised repository allows an aggregation of similar transfer requests into bulk operations and provides an overall picture of global activity and performance. This aggregation occurs at the central Transfer Database

(Transfer DB) which offers the same interface as the Request DB. The Transfer DB inherits the MySQL backend of the Request DB, extending it to include *task queues*. The task queues contain the files waiting to be transferred on specific point-to-point transfer routes between grid SEs.

The default bulk transfer mechanism uses gLite FTS. In this mode of operation the task queues reflect the available underlying FTS channels. The Transfer DB schema contains tables to store details of the FTS jobs submitted, their status and the files they contain.

4.2.2 Replication Scheduler

The Replication Scheduler interacts with the Request DB interface to retrieve transfer requests waiting to be executed (using the same mechanism as the Data Management agents mentioned in Section 4.1.5). The destination SE (and where available the source SE) is retrieved from the request along with the list of LFNs or *datasets* (see Section 4.5). The replica information for data to be scheduled is retrieved from the File Catalog and the task queues are populated with the source and destination SURLs for each file.

The Replication Scheduler uses a plugin architecture to apply a variety of replication strategies. These strategies perform a scheduling decision to determine the channel(s) on which to transfer a file. The strategy used can be defined as a request parameter otherwise the general agent configuration is used. The strategies are invoked for each individual file with the replica information and a list of target SEs supplied. The strategies can apply any algorithm, but must return a standard structure containing the source-destination pair of each point-to-point transfer and the dependencies of the transfers, in the case of multi-hop transfers. The approach allows new replication strategies to be implemented and tested simply. In addition, the individual treatment of files allows higher granularity scheduling than would be possible if scheduling was performed on the level of requests.

There are two broad types of strategies which have been implemented: static and dynamic.

Static Strategies

Static strategies create a replication tree based on information contained only within the plugin itself. An example is the *simple* strategy that creates a transfer from the source SE to each of the supplied destinations.

Dynamic Strategies

Dynamic strategies use input information that changes with time. The input information available is the observed bandwidth of the active channels, the number of files in the task queues and their total size. The premise of the dynamic strategies is that optimal use of the available network resources can be made using intelligent scheduling based on the current state of the task queues and recently observed throughput.

For example, the channel defined from A to B may have a high *time to start* (many files in the task queue and/or low observed throughput) while the task queues linking A to C and C to B may be more performant. If a file is to be replicated from site A to sites B and C and the combined *time to start* of the A-C-B route is lower than that of the direct A-B route, it is more efficient to schedule the replication first from A to C, then from C to B.

A number of dynamic strategies exist which handle the input information differently. Each dynamic algorithm may evaluate the time to start based on the number of files in the task queues, their size or both. Also, the time over which the recent throughput is determined can be varied for a shorter or longer term view. Finally, with multi-hop transfers there is a *delta* time between the completion of a hop and the start of the next hop which may be evaluated in the time to start calculation. Variation of this value changes the branching ratio and depth of the resultant replication trees.

An investigation of the performance of a number of strategies will be presented in Section 6.3.

4.2.3 FTS Submit / Monitor

Files are retrieved from the task queues by the FTS Submit agent which controls the submission of transfer jobs to the FTS. To ensure that edge effects

do not reduce the achieved transfer throughput the agent maintains multiple (configured to be 2) FTS jobs on each channel concurrently. The agent retrieves files from a task queue based on a hierarchy of importance; task queues with fewer submitted jobs are preferred and subsequently the task queue with more files. Once the FTS jobs are submitted, the agent populates the Transfer DB with the details of that job and the files contained within it.

To monitor the status of the bulk transfer jobs the FTS Monitor agent retrieves details of the FTS jobs from the Transfer DB. To ensure these are monitored fairly the least recently monitored active job is always taken. The agent first contacts the relevant FTS server to obtain a summary of the job (job status and a count of the file statuses). If the job is in a final status the full job output is retrieved and the status of individual files is updated in the Transfer DB. An accounting message is sent containing the data volume transferred since the job submission time.

During the prototyping stage the submission and monitoring were performed sequentially by the same agent. This approach incurred a lag between submission attempts and was abandoned in order to execute the available work as quickly as possible.

4.2.4 Request finalisation

The transfer requests initially sent to the Request DB are broken down by the Replication Scheduler to allow their aggregation in the task queues. After the replication of the files has been accomplished the initial requests must be updated. In addition, the operation of the original requests is checked and where appropriate the file is registered in the File Catalog, or in the case of a move operation, the source file is removed.

4.2.5 Request Persistency

The disadvantage of a central scheduling architecture is that it acts as a single point of failure. The persistency of the transfer requests is ensured using the distributed Request DB service, described in Section 4.1.4. These requests are forwarded to the central Request DB by specialised agents present on the VO Box.

If the central service remains unavailable for long periods Transfer Agents located on the VO Boxes can be activated to process the requests. This mode of operation is heavily throttled to avoid overloading SEs while still allowing some activity to be executed.

4.3 Data Driven Replication

In the previous section the bulk transfer framework was outlined. This provides a service to ensure data replication is attempted until success is attained. The bulk transfer framework knows nothing about the LHC*b* Computing Model and therefore must be supplied with requests to transfer data. DIRAC supports the LHC*b* Computing Model dataflow using a *data driven* approach (in contrast to the *subscription* based approach used by ATLAS, described in Section 1.8.3). The approach is data driven because replication operations to be performed are triggered by the availability of the data. The mechanism for performing data driven replication is outlined in this section and a schematic of the components mentioned given in Figure 4.7.

A specialised catalog called the *Placement Database* (Placement DB) offers the File Catalog interface and is populated transparently by File Catalog. Files registered here are candidates for data driven *transformations*. The Placement DB tables contain the information required to execute the transformations. The most important of these elements are:

- The *file mask* which is used to determine which files are eligible
- The *plugin* with which to process the eligible files

As files are registered in the Placement DB the *file mask* element of each of the active transformations is evaluated against the LFN of the file. Files matching are 'added' to the transformation. The *plugin* element is used to locate the module to be instantiated when the transformation is processed. Each plugin may require different metadata fields to operate and these are stored in an additional table.

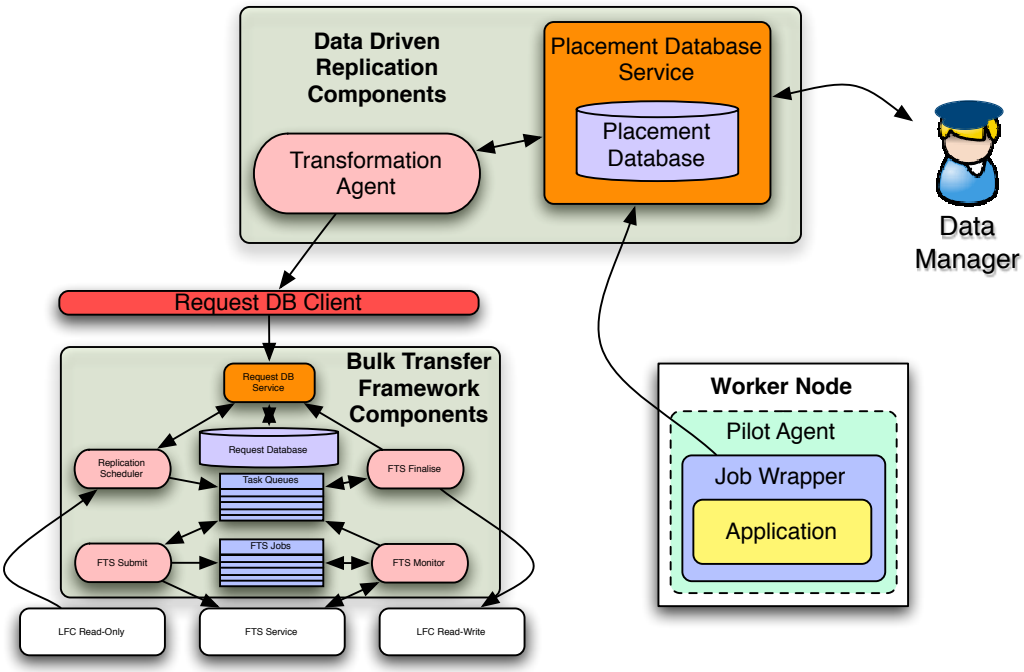


Figure 4.7: Components to perform data driven replication.

The Transformation Agent retrieves the active transformations from the Placement DB. For each active transformation the desired plugin is instantiated with any required metadata. The unused files for the transformation are retrieved from the Placement DB and passed to the plugin module. A standard structure is returned by the plugin representing the operations to be performed. This structure is used to create transfer requests which are submitted to the Request DB service for execution.

Within the Computing Model, data driven replication has two main uses cases:

- transfer of the RAW data from CERN to the Tier-1s to create a distributed copy.
- transfer of DSTs (both from real and simulated data) to multiple Tier-1s.

These will be discussed in the following sections.

4.3.1 RAW Data Splitting

The LHC*b* experiment dataflow requires the RAW data must have a distributed replica at the Tier-1s. The quantity of replica data at each site is based on pledged computing resource. To support this use case the transformation file mask selects only RAW files. The destination Tier-1 sites, and their share of the data, is defined using additional metadata in the transformation. A specific *load balance* plugin is used which splits the input files into chunks according to the defined Tier-1 share. As will be seen in Section 4.4.1 only verified RAW data is registered in the File Catalog. Therefore, as files are verified safe, they are eligible for export to the Tier-1 sites.

4.3.2 DST Data Broadcast

The stripping phase of the LHC*b* dataflow at the Tier-1 centres produces DST files. DST files produced from real physics data are replicated to all Tier-1 centres to allow load balanced distributed user analysis. To support this a *broadcast* plugin is used with all destination SEs defined as transformation parameters. The file mask for the transformation matches LFNs of real DSTs which are registered when the output is uploaded at the end of a stripping job.

The DSTs produced from simulated physics data is present on selected Tier-1 centres only. This can vary per production. For each of the productions (or set of productions) a transformation is defined that reuses the *broadcast* plugin with a reduced set of destination SEs.

As can be seen from the RAW and DST use-cases described, the plugin architecture is extensible and allows new operations to be defined and executed with minimal development.

4.4 Ensuring Data Integrity

The previous sections discussed the bulk transfer framework and the mechanism for performing data driven replication. These activities aim to provide greater access to data and to ensure resilience from data loss or corruption. This section will discuss the mechanisms for ensuring the integrity of data

and recovering from any loss or corruption. The first part of this section will present the components that ensure the integrity of the RAW physics data. The second part will discuss a general framework for ensuring the consistency of the Resources used by the DIRAC Data Management system.

4.4.1 RAW Data

As the Tier-0, CERN performs the role of RAW data warehouse and maintains the master copy of every RAW file. The integrity of each file in the offline computing environment must be ensured. A file erroneously migrated to tape may not be recovered once the disk copy is garbage collected. Therefore the defining principle for ensuring the integrity of the RAW data can be stated as:

> All of LHC*b*'s RAW data must verified on tape before any intermediate copies can be removed.

The High Level Trigger (HLT) writes 2GB RAW physics data files to the Online Storage System [149] approximately every 30 seconds. For each file an entry is made in the Online Run Database (RunDB), which manages the file state machine within the online system [169]. The Data Mover queries the Run DB for files awaiting transfer to offline MSS at CERN[2] and initiates the RAW data replication. This is done by setting a request, containing all the information required, to put the file on the Storage Element (file location, destination storage element name) and register in the metadata catalogues (LFN, GUID, file size, *adler32* checksum[3]), to the DIRAC RequestDB (see Figure 4.8).

Once this request has been set a series of DIRAC components manage the file upload and the verification process. A UML activity diagram of the the steps involved is given in Figure 4.9, with the actors described below.

[2]This is managed by CASTOR2. [170]

[3]The adler32 checksum, proposed by Mark Adler, is composed of two sums accumulated for each byte of the required data. The first (s1) is the sum of all the bytes in the data and the second sum is the sum of all the s1 values. This algorithm is faster than comparative cyclic redundancy checks, for example CRC-32 but is open to intentional forgery.

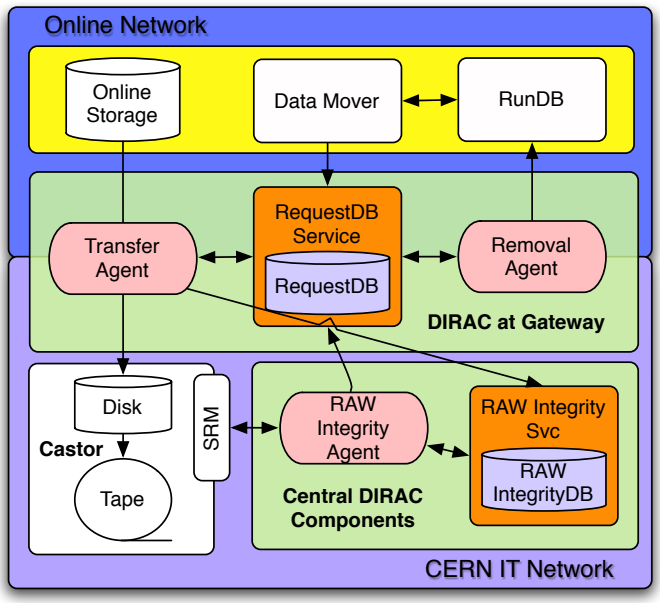


Figure 4.8: Ensuring RAW data integrity schematic.

Transfer Agent

The Transfer Agent is deployed on a gateway machine to have access to the Online Storage and the private Gb/s network connected to CERN IT fabric. The agent is configured to know of a single Storage Element, ('CERN-RAW'), with a single protocol, CASTOR's proprietary RFIO, which ensures the transfer is performed to dedicated disk pools. The configuration also contains a single entry for the File Catalog, 'RAWIntegrityDB', so files awaiting verification do not pollute production catalogues. The agent retrieves requests from a local Request DB and performs the p̂ut to the CERN-RAW Storage Element and on completion registers the file in the RAW Integrity Database (RAW Integrity DB).

The RAW Integrity DB maintains a list of files awaiting migration to tape. It contains all the metadata supplied in the initial request to perform a registration in all catalogues once a file is verified. The RAW Integrity agent retrieves the list of active files from RAW Integrity DB and, using the Storage Element, obtains the metadata from the Castor SRM interface. When migrating a file to tape CASTOR calculates the *adler32* checksum. This is compared with the checksum calculated by the RunDB when the

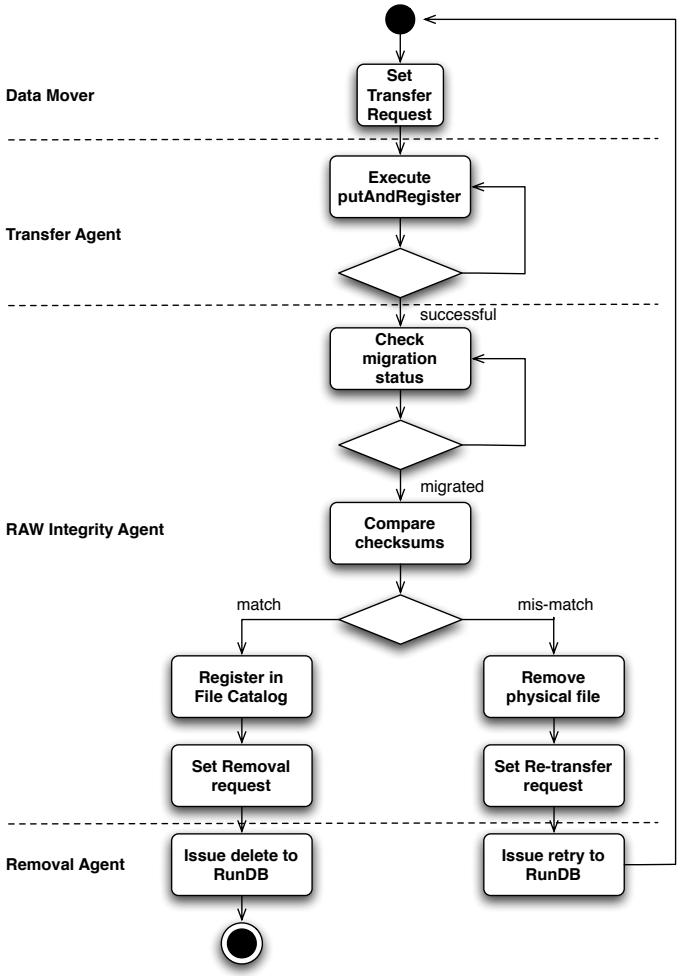


Figure 4.9: RAW Integrity Activity Diagram.

files were initially written. If the two checksums match the integrity of the file in CASTOR, both on tape and disk, can be assumed. In this case a removal request is created and set to the gateway machine Request DB. Then the file is fully registered in all of the LHC*b* catalogues. In the event of a checksum mismatch the physical file is removed from CASTOR and a *re-transfer* request is set to the gateway Request DB.

Removal Agent

The final component in the chain retrieves and executes the removal requests placed by the RAW Integrity Agent. The agent is configured to know only

of the online storage protocol, a simple wrapper around plain RPC service, exposed by the RunDB. This wrapper offers two service methods which either instruct the RunDB that a file may be considered for removal or to request a re-transfer.

During the migration and verification process files may be used for calibration and detector performance studies and their use is registered with the RunDB to ensure their availability at the Online system. When 'removing' data the Removal Agent merely advises the RunDB that the file may be removed. The physical removal is performed asynchronously when space is required on the Online storage.

Commissioning

This system for uploading the RAW data from the online storage and ensuring it's integrity was tested during a computing preparedness (the Common Computing Readiness Challenge) exercise to mimic the data being produced from the detector. This exercise showed The results are given in Appendix C.

4.4.2 DIRAC Resource Consistency

The mutual consistency of Resources managed by DIRAC is vital in the provision of reliable data management. The architecture of the File Catalog ensures the mutual consistency of each of the underlying catalogues. The distributed Request DB service and related agents ensure than any failed operation is persisted as a request and retried until success. In addition, individual file catalog and storage element plugins contain rollback mechanisms to clean up the resource in the event of failure.

To ensure the mutual consistency of these resources is maintained over time a suite of agents is deployed to verify the contents of the three main Data Management Resources: the grid storage elements (SE), the LFC catalogue and LHC*b*'s Bookkeeping and provenance database. This suite of agents report any inconsistencies found to an Integrity Database (Integrity DB). Files or replicas entered in the Integrity DB are also marked as *problematic* in all of DIRAC's catalogues. In the case of the LFC the replicas become invisible to replica queries. This ensures that while the integrity of

an entry is being investigated no further attempts to access it can be made. The suite of agents and service and their interaction with the Resources is shown in Figure 4.10.

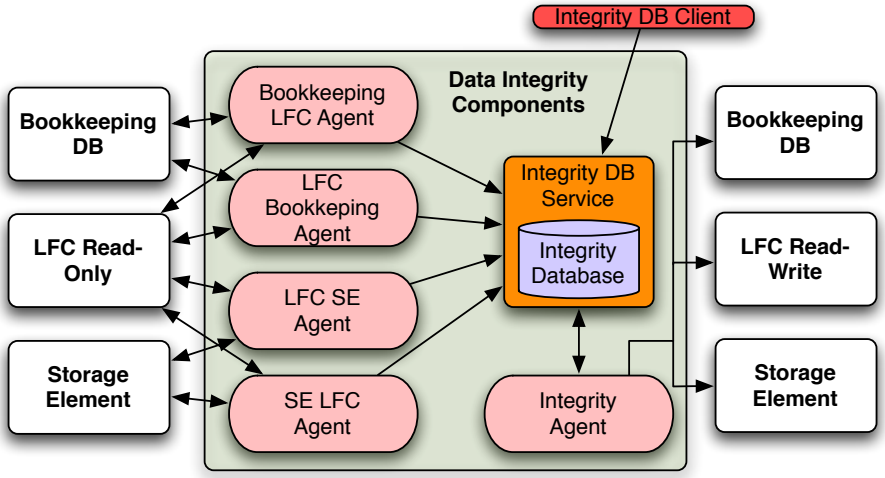


Figure 4.10: Schematic representation of the data integrity suite.

The motivation to maintain the mutual consistency of the Bookkeeping, LFC and Storage Elements is given below.

Bookkeeping and LFC

LHC*b*'s Bookkeeping DB contains *provenance* information regarding all LHC*b*'s files and jobs. The Bookkeeping provides an interface for physicists to query for files with particular properties of interest. Files that do not physically exist are flagged in the database and are invisible to searches. To ensure physicists are given only files that exist, the consistency of the Bookkeeping must be maintained with the LFC. An agent is deployed to verify the existence in the LFC for the data visible in the Bookkeeping. Similarly, data registered in the LFC and not visible in the Bookkeeping can never be found by a physics search, which requires an agent to use the LFC namespace and check whether the files found exist in the Bookkeeping.

LFC and storage elements

The LFC is used to obtain the storage element and PFN for replicas associated to a file. If the replica information is incorrect attempts to access files that do not physically exist may occur. The PFNs registered in the LFC are checked on the storage elements, to ensure they exist, and the *adler32* checksum returned by the storage may be verified.

When managing PetaBytes of data, using the storage resources efficiently is paramount. The LHC*b* disk requirements, presented in Section 2.6, assume a disk utilisation efficiency of 100%. Orphan physical files on storage resources, without registered replicas in the LFC, can never be seen and are a source of waste. To combat this, the contents of the SE namespace are obtained and the physical files present checked against the LFC contents.

Resolving Data Integrity Problems

The series of agents above report any inconsistencies found to a central repository, the Integrity DB, with a note of the observed prognosis. In addition to these agents, which try to pre-emptively discover problems, the Integrity DB can be populated by any DIRAC component when a problem is found with a file or replica. In this way every Workload Management and Data Management agent as well as every job is an integrity check.

To resolve the problem files found in the Integrity DB a further agent is responsible for determining the core reason for the inconsistency. This agent attempts to resolve the fundamental inconsistency by performing (re)replication, (re)registration, physical removal or catalogue removal of files. If the problem is successfully resolved the status of the file or replica in the LFC and other DIRAC catalogues is updated. Some pathologies may not be resolved automatically and remain in the DB for manual intervention.

4.5 Datasets

The final section of this chapter will look at a mechanism for clients to simplify the management of large volumes of data by using *datasets*.

The concept of *datasets*, where a single tag may reference large numbers of files grouped by their inherent properties, provides several advantages for end users:

- Managing large numbers of files is reduced to managing a smaller number of datasets.
- Dataset handling is less error prone.
- Dataset sharing between collaborators is easier.

This section will give the use-cases for using datasets in LHC*b*, the design considerations and the eventual implemented architecture.

4.5.1 Dataset Use Cases

The LHC*b* Computing Model and the analysis activities of the LHC*b* physics community provide a number of use-cases for datasets.

Physics analysis

The primary motivation for defining and supporting datasets is to reduce the physicist's exposure to large lists of files. The analysis activities routinely performed by physicists are envisaged to take two forms:

- Algorithm tuning; repeated analysis of a subset of events for relevant decay channels to determine the effect of modifications.
- Full analysis; run over all available events for the relevant channels for increased statistical precision.

The full analysis activity provides a use-case for accessing large numbers of files and analysing them as an atomic unit. To support the algorithm tuning use-case it must be possible to select a subset of files with a required level of repeatability.

Reprocessing activities

During periods of re-processing, to take place between data taking periods, large numbers of files with similar properties (i.e. RAW/rDST data produced during the latest data taking period) are reprocessed.

Managing the 'hot stream'

The LHC*b* computing model makes provision for a sample of RAW and rDST data to be available on disk storage to allow data monitoring, data quality checking and tuning of application algorithms. For this case a set of files would maintained on disk for low latency access.

4.5.2 Dataset architecture and implementation

Within LHC*b* a dataset is defined with the following criteria:

1. A dataset is a logical grouping of files defined by a specific set of common properties.
2. Dataset constituents must physically exist and be accessible.
3. Datasets can be either 'open' where the constituents may be added or removed or 'closed' where data may only be removed (subject to 2).

The logical grouping of files with similar properties is a common approach to aggregate data to simplify management and processing. The second criteria implies that constituents can be distributed across many locations, but maintain their logical grouping. This approach was taken because logical groupings of files may exist with a distributed copy across many sites, i.e. all RAW data has a distributed copy at the Tier-1 centres. No a priori physical property (i.e. location) of a dataset is implied by being a member of a dataset.

Defining datasets as being either 'open' or 'closed' allows a simple dataset evolution architecture that does not require a complex system of dataset versioning. A dataset is 'open' while it is initially being populated and during this period no repeatability, as perceived from a physics analysis, can be guaranteed. A dataset should be 'closed' once the common conditions defining a dataset end, e.g. run period ends or a stripping production ends. Before being 'closed' an integrity check of the entire dataset content is performed (see Section 4.4.2).

Design considerations

The following points were considered when defining the dataset architecture.

- Determining the contents of datasets (dereferencing) should be 100% available.
- Creating datasets is important, but less critical than dereferencing.
- Dataset metadata should be searchable.

For datasets to support the use-cases defined in Section 4.5.1 the availability of the dataset architecture must be ∼100% and not susceptible to a single point of failure. To reduce the number of privately created datasets, a mechanism for end users to query the dataset architecture for existing datasets must be available (at lower quality of service).

Dataset architecture

A schematic representation of the dataset architecture is given in Figure 4.11.

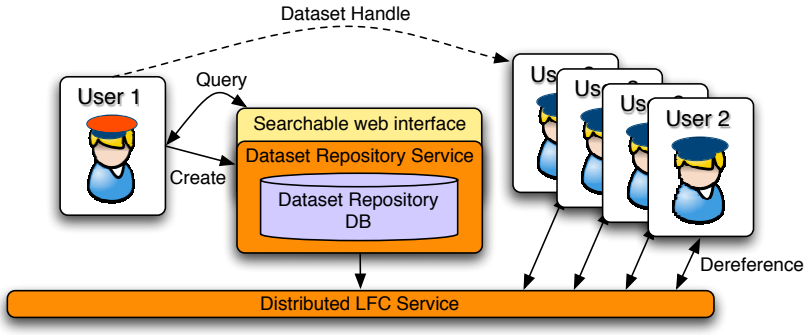


Figure 4.11: Dataset Architecture Schematic.

It was decided that the dataset architecture would use the LFC service to store the datasets. The primary reason for this was the overall resilience of the distributed LFC service, which provides 100% read access, across seven read-only instances. In addition, the file metadata (replica information) stored in the LFC matches the information required by the dataset consumers with the consequence that resolving datasets requires a single service call. Each dataset is defined in a dedicated directory within the LFC containing soft links to production LFNs, or other dataset directories. A short dataset 'handle' may also be used to point to the LFC dataset directory, i.e. the 'handle'

DC06/StrippedDSTs

would be located at

/lhcb/dataset/DC06/StrippedDSTs

in the LFC namespace. The performance of dataset dereferencing was investigated and will be given in Section 5.1.7.

An additional service offers a browsable interface to the dataset metadata for the less frequent operation of users searching for available datasets. It is expected that dataset searches will be performed much less frequently than dereferencing so the criticality of this service is lower. Similarly, the creation of datasets is expected to be a low frequency operation and can be managed by the Dataset repository service. If a dataset is not present the user may create a private dataset by providing the LFNs they wish it to contain, and any metadata tags for other users to search for. Once the service creates the links in the LFC, and publishes the correct metadata, a dataset 'handle' is returned to the requesting user.

One of the major advantages of defining datasets is to perform reference counting of the most popular data. Assuming that the number of datasets is very much smaller than the number of files, tracking the number of times a dataset is accessed gives a scalable and accurate picture of most frequently used data. At the point that a dataset is dereferenced a message is sent to the dataset repository service to increment the dataset reference count. This does not affect the availability of the dataset dereferencing system, since the failure to contact the dataset repository is not considered fatal. It is wrapped as a request and set in the RequestDB service for asynchronous execution. Knowledge of the most popular datasets with time allows more efficient resource planning and consumption.

4.6 Summary

This chapter presented the DIRAC Data Management system and the components required to support LHC*b*'s computing activities.

The core components of Storage Element and File Catalog provide an abstraction of the underlying resources and when combined can be used to perform file management on the grid using the Replica Manager. To ensure that operations are not effected by transient failures, operations can be stored as requests and executed asynchronously. The bulk transfer framework provides redundant file replication making intelligent use of available network resources. The support of the Computing Model dataflows is provided using a data driven approach that triggers replication as data becomes available. The solutions of ensuring data integrity was described for RAW physics data and a general suite for maintaining the consistency of the DIRAC resources was given. The dataset architecture was presented which allows users to reduce their exposure to unmanageable list of data files and provides the data management system with the knowledge of which data is the most popular within the collaboration.

The following Chapters will present the performance studies relating to the components described in this Chapter.

Chapter 5

DIRAC Resources Performance

The DIRAC Data Management System manages the physical files and file based metadata for all LHC*b* physics data. The LCG File Catalog (LFC), discussed in Section 1.7.4, is the replica catalog of choice within LHC*b*. The performance of the LFC for meeting the use-cases of LHC*b* is presented in Section 5.1.

The LHC*b* data, from RAW to DSTs are stored on grid Storage Elements offering the Storage Resource Manager (SRM) interface. The performance of the SRM interface at the LHC*b* Tier-1 sites is given in Section 5.2.

5.1 Replica Catalog Performance

The LCG File Catalog (LFC) stores trivial file metadata and replica information for all files that physically exist and that are under management of the DIRAC Data Management System. The primary role of the replica catalogue is to map a Logical File Name (LFN), and its Global Unique IDentifier (GUID), to its replicas on grid Storage Elements. A file entry may have any number of replicas. Each replica registered has two important associated fields: the Storage Element name and the Physical File Name (PFN), or Storage URL (SURL). The Storage Element name is used by the Workload and Data Management systems to make scheduling decisions based on the availability of data. The PFN information is used to gain access to the file,

either within the actual job or data transfer task. The relationship between the LFN, GUID, PFN and SE name is shown in Figure 5.1.

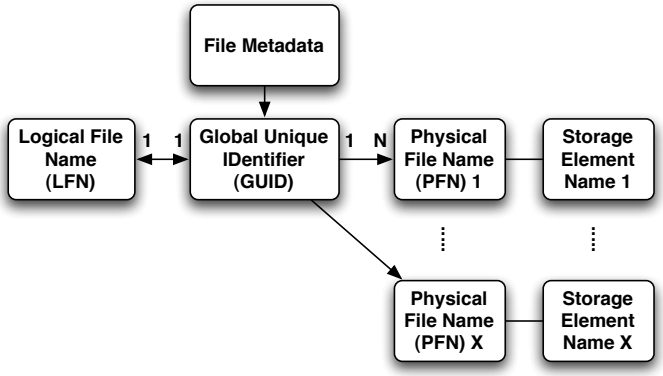


Figure 5.1: The relationship between LFN, file GUIDs, PFNs and Storage Element names within the LFC. Each LFN is associated with a single GUID. The GUID has associated file metadata and replica information. Each replica entry has a PFN and Storage Element name.

The ability to register file and replica information in a timely manner is fundamental to all operations performed by DIRAC. The speed at which this information can be retrieved places a limit on how quickly scheduling decisions can be made. In addition, as data ages, or new (and better) processings exist, the relevance of the older data decreases. The need to remove replicas and eventually files is therefore required. The performance of all these operations will be discussed in the following sections. All results discussed in this section use a Python binding to the LFC client.

All tests were performed from a single host with Intel Xeon 3.0GHz quad-core with 8GB of RAM. The master LFC instance used for write operation tests (Sections 5.1.1, 5.1.2, 5.1.4 and 5.1.5) was a DNS load-balanced service with two Intel Xeon 3.0GHz dual-core hosts with 4GB of RAM. The read only instances used for query tests (Sections 5.1.3, 5.1.6 and 5.1.7) are given in Table 5.1. All LFC instances were used an Oracle RAC as the database backend.

	CPU	RAM
CERN	Intel Xeon 3.0GHz dual-core	4GB
CNAF	Intel Xeon 2.4GHz dual-core	4GB
GRIDKA	Intel Xeon 2.33GHz dual-core	5GB
IN2P3	IBM System x3550 2xdual-core	2GB
NL-T1	Dell 1850	4GB
PIC	AMD Opteron 2218 2.6MHz dual-core	8GB
RAL	Intel Xeon 2.8GHz dual-core	4GB

Table 5.1: CPU and RAM configuration of LFC read only instances at CERN and Tier-1 sites.

5.1.1 Initial File Registration

The combined LHC*b* computing activities, over the course of a year, will produce a total of 25M files which must be registered in the LFC. This implies a file registration rate of 0.8Hz sustained over that period.

Registration of a file in the LFC is a multi-step process. The LFC uses a GUID (of the UUID format proposed in [171]) as the primary file identifier to which LFNs are associated. Once the GUID is created, file metadata, such as the file size and checksum information, is added. The final step is the registration of the first replica. To avoid corrupted entries, due to transient errors, all the above steps are performed within a single transaction. The registration of 180k files with single replicas was performed and the time for the complete registration was recorded.

A histogram of registration time is shown in Figure 5.2 and shows the registration time is well defined with good precision. A tail in the distribution can be seen extending out to 1.4 seconds. The most likely cause for the tail's existence is the variable load on the LFC service, although further investigation would be required to confirm this. The mean registration time of 0.85s implies a registration rate of 1.17Hz which is 30% above the required rate.

No errors were observed during the registration of the files which is testament to the stability and robustness of the LFC service.

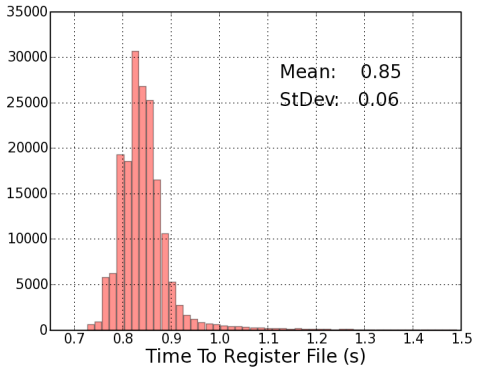


Figure 5.2: Time to register a file with a single replica in the LFC within a single transaction.

5.1.2 Registering Replicas

The LFC uses the file GUID as the primary file identifier. The registration of replicas requires the GUID to associate the replica to a file. DIRAC uses LFNs as the file identifiers and therefore the file GUID must first be retrieved before replica registration can be performed. This places the requirement for an additional client-server interaction.

To avoid multiple client-server authentications[1] the LFC provides the ability to create an authenticated *session*. These sessions allow to perform a single client-server authentication which is then reused for all subsequent operations until the session is closed.

The Computing Model defines that the RAW data should have a distributed replica across the external Tier-1 sites. The DST and TAG files produced from these RAW files should be present at all 7 Tier-1s while the DSTs produced by the simulation activities are to be replicated at 3 Tier-1s. Each additional replica of a file must be registered in the LFC requiring, over the course of a year, a total of 4.4M replicas to be registered. Over the period of a year this equates to a registration rate of 0.14Hz.

To test the performance of the replica registration within an authenti-

[1]The client server authentication requires mutual verification of the client and server identity. This involves CPU intensive operations at the SSL layer and adds significantly to the load on the LFC service.

cated session a series of replica registrations was performed, each attempting to register groups of replicas, increasing from 1 to 5,000, within a session. A new session was created for each group of files and destroyed after the group had completed and the time to register the replicas was recorded. This series of registrations was repeated 200 times giving a total of ∼177k replica registrations performed during the test. The mean rate at which replicas were registered, with the RMS of this value represented as error bars, is plotted against the number of replicas registered within a session in Figure 5.3. This plotting style will be adopted for the remainder of this Chapter.

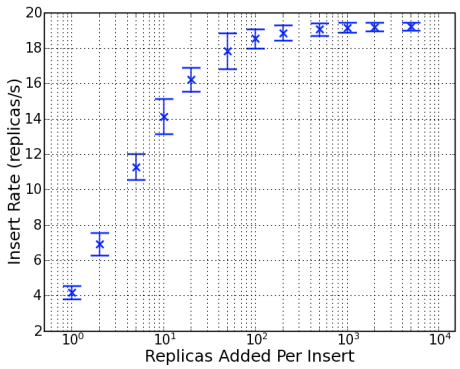


Figure 5.3: Insert rate of replicas in the LFC for a varying number of replicas within an authenticated session.

The rate at which replicas are registered increases with the number of replicas registered per session until reaching a stable maximum of around 19Hz. The time to perform the authentication handshake is included in the recorded registration time and contributes a significant portion of the time for smaller groups of registrations. As the number of registrations increases the relative cost of the handshaking decreases until the database insertion rate limit is reached. The lowest insertion rate of 4Hz, observed for a single replica registration within a session, is an order of magnitude above the rate required by the Computing Model.

5.1.3 Retrieving Replica Information

The core consumers of file replica information are the agents performing job and transfer scheduling. Retrieval of replica information should not limit

the ability to perform these activities. The Computing Model activities (excluding user activity) require the job and transfer scheduling agents query for 17.3M file replicas which represents a sustained rate of 0.54Hz.

Two methods for retrieving replica information were compared. The first uses an authenticated session and polls for replica information for each file serially. The second is a bulk method, taking a list of input files and involves a single client-server round trip. The files used in the test each had seven replicas and the time taken to retrieve replica information for bunches of files, from 1 to 10k, was recorded. This series was repeated 200 times (resulting in ∼377k file replica queries for each method evaluated). The rate at which replicas were retrieved plotted against the number of files in the group is shown in Figure 5.4.

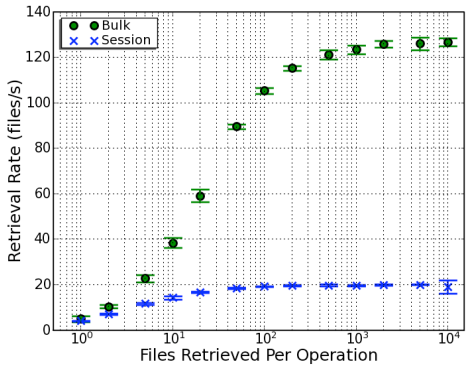


Figure 5.4: Retrieval rate of replica information from the LFC for a varying number of files. Two methods were used: bulk query method and query within an authenticated session.

The rate at which replica information is retrieved increases with the number of files for both methods used. The lowest retrieval rate of 5Hz, observed for a single file within a session, is an order of magnitude greater than the requirement. The session approach shows a plateau of ∼20Hz. The bulk method shows a plateau at over 120Hz providing a factor of 6 performance increase over the session approach. This increased performance can be attributed to two factors.

- The session method requires a client-server round trip for each file. Short round trip times can become significant when accumulated over

all files. The bulk method requires only a single round trip.

- The bulk method performs a single query to the database backend. This allows the database backend to retrieve many entries at once.

Section 2.4 outlined the necessity to store stripped DSTs at all seven of LHC*b*'s Tier-1 sites (including CERN). It is therefore expected that a file may have up to seven replicas registered in the LFC. The bulk method, found to be most performant in the previous test, was used to determine the retrieval rate for files with varying numbers of replicas. Distinct sets of files were registered with varying numbers of replicas, varying from 1 to 7. For each set of files the performance profile of replica retrieval was determined for varying groups of files in each query. For each set this was repeated 200 times. The performance profile for each set of files is given in Figure 5.5.

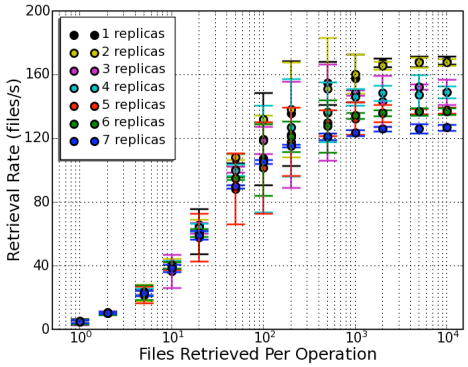


Figure 5.5: Retrieval rate of replica information from the LFC for a varying number of files using bulk method. Seven sets of files were used with the number of replicas varying one to seven.

The first feature of Figure 5.5 is that it verifies the ∼120Hz performance plateau of the bulk method for seven replicas, presented in Figure 5.4. The retrieval rate for the different file sets only begins to differentiate with more than 50 files in a single query. At groups over 10^3 files the different sets show variable plateau ranging from ∼120Hz (for files with 7 replicas) up to ∼170Hz (for files with 1 and 2 replicas). From a file centric view the retrieval rate decreases by one third as the number of replicas for each file increases (∼170Hz to ∼120Hz). From a replica centric view this is a 5 fold increase for each retrieval, corresponding to an absolute rate change from

$\sim$170Hz to $\sim$840Hz (7$\times$120).

5.1.4 Removing Replica Information

The computing model foresees periodic reprocessing of data with the newest version of the reconstruction software. Data produced by earlier processing activities must be removed from selected storage elements to recover space as their relevance decreases. Once removed from the storage elements the associated file replica information must be removed from the LFC such that no further attempts to access the physical file are made. In addition, small files produced by the stripping are to be merged into larger files with the removal of the input files once the merger is complete. These activities imply the removal of 15M replicas over the course of a year giving a requirement 0.5Hz.

The LFC maintains file replica information in a separate database table, which has the PFN as a primary key. The removal of a replica can be performed by providing only the PFN and does not require a resolution of the file GUID (as is the case for replica registration). This removal can be performed within an authenticated session.

As before, the number of replicas removed in a single group was varied, from 1 to 5000, and the time to remove all replicas in the group recorded. This was repeated 200 times and the removal rate profile shown in Figure 5.6.

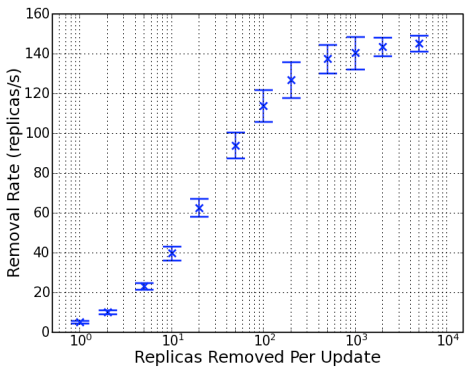


Figure 5.6: Removal rate of replicas from the LFC for a varying number of replicas within an authenticated session.

The removal rate reaches a plateau at ∼140Hz for groups of files greater than 10^3. This rate is over 2 orders of magnitude higher than required. This removal rate within a session is significantly higher than the insert rate or the replica retrieval rate for the following reasons:

- The removal of a primary key is optimised at the database level.
- The registration of replica information requires an additional database query to obtain the file GUID.
- The retrieval of replica information requires a join of two database tables (the files table containing the LFN and the replicas table).

5.1.5 Removing Files

The removal of all replicas associated to a file will leave the file metadata registered in the namespace. The 15M small files produced by the stripping must be completely removed after the merging activity is complete giving a requirement of 0.5Hz sustained over the year.

The metadata removal performance was analysed using two methods: the authenticated session and bulk method taking a list of LFNs. Using both approaches the number of metadata entries removed was varied. The results are shown in Figure 5.7.

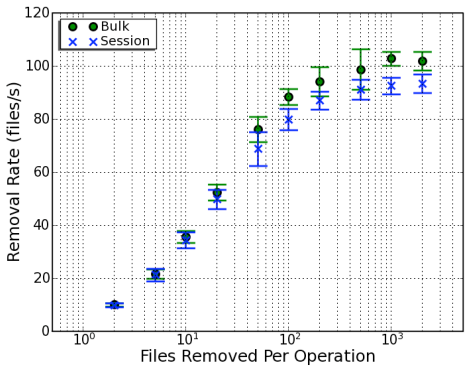


Figure 5.7: Removal rate of files from the LFC for a varying number of files. Two methods were used: bulk removal and removal within an authenticated session.

The bulk method shows better rate of removal than the session with a 10% performance improvement observed for large numbers of files. Both methods give performance more than 2 orders of magnitude over the requirement. The removal rate for both methods is lower than the replica removal rate, as seen in Figure 5.6. This is because a database level constraint ensures that metadata information may not be removed unless all replicas have first been removed. This constraint requires checks to be performed at the database level resulting in decreased performance.

5.1.6 Distributed Read-Only Mirror Performance

The use of a distributed LFC service with a single read-write master and multiple read-only mirrors was discussed in Section 4.1.2. The distributed mirrors are updated using Oracle Streaming technology as changes are made in the master. The consistency of the distributed mirrors lags that of the master server as the updates are streamed. If this lag period becomes too high then the information served by the mirrors might be inconsistent.

To determine the lag period a series of 4000 file inserts and file removals was performed on the master. An authenticated session was created to each of the mirror instances and replica information was retrieved in a loop, with a small wait time O(0.1s), until the modifications became visible. The six mirrors used for this test were present all Tier-1 sites (other than GRIDKA), and the observed times are shown in Table 5.2.

	Time For Insert To Propagate	Time For Removal To Propagate
CERN	00.30± 0.12	00.32 ± 0.26
CNAF	19.99 ± 7.00	22.12 ± 8.39
IN2P3	22.09 ± 8.70	22.02 ± 8.41
NL-T1	22.65 ± 8.20	23.16 ± 8.44
PIC	21.98 ± 8.49	22.48 ± 8.45
RAL	22.27 ± 8.40	22.04 ± 8.38

Table 5.2: The time taken for updates to the read-write master LFC to reach the read-only mirrors (s).

The time measured was similar across all mirrors for both insert and removal operations, with the exception of CERN. At CERN, the mirror shares

the database backend with the master instance and therefore the 'replication' is observed to be instantaneous.

For the replica retrieval tests, performed in Section 5.1.3, the CERN mirror instance was used. The performance of all of the distributed mirrors must be satisfactory. Using the bulk method for a varying number of files, the performance for replica retrieval was evaluated at each available mirror. The results are shown in Figure 5.8.

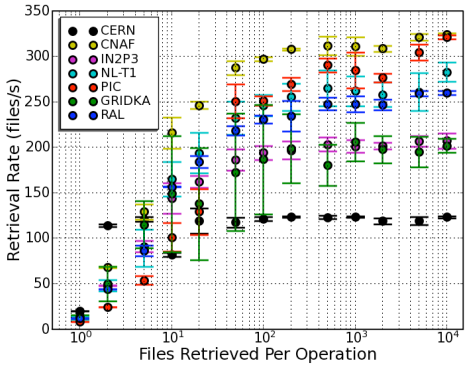


Figure 5.8: Retrieval rate of replica information from the read-only LFC instances for a varying number of files using bulk method.

The profile for each mirror instance can be seen to be the similar with a rapid increase of performance with the number of files until reaching stability for groups of files over 10^3. The CERN mirror shows the lowest performance of all tested due to sharing the database backend with the master instance. The other six mirrors showed a maximum retrieval rate between 200 and 300Hz, which varied with the memory available to the database on the machine supporting the service.[2]

5.1.7 Data Set Dereferencing

As discussed in Section 4.5.2 the dataset architecture supported by DIRAC maps a dataset handle to a directory in the LFC. The final study performed was to evaluate the performance of a specific LFC binding method that allows the retrieval of file and replica metadata information for all files in

[2]The IN2P3 mirror, one of the least performant, is hosted by a 4 core IBM x3550 with 2GB of RAM. The PIC mirror, one of the most performant, is hosted by dual core AMD 2218 machine with 8GB of RAM.

a directory. The performance of retrieving replicas for directories containing varying number of files was evaluated for the CERN and the CNAF mirrors. These mirrors were chosen as they showed the lowest and highest retrieval rate, respectively, in Figure 5.8. The results of this test are shown in Figure 5.9.

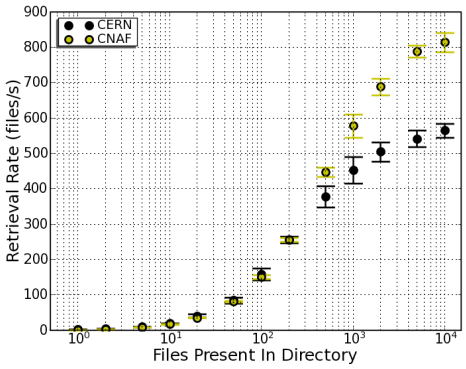


Figure 5.9: Time to obtain replicas for entire directory from the LFC for a varying number of files in the directory.

The maximum observed retrieval rate with this approach ranges from 550Hz to 800Hz for large directories. In some cases this gives a quadrupling of performance over the bulk method by file (as seen in Figure 5.8). The bulk method by file allows any files to be used in an ad-hoc way, but the directory method is advantageous when all the files are contained in a single directory. Grouping files into a dataset within a single directory allows the performance to be maximised. This underpins the architectural design of the Dataset system, presented in Section 4.5.2.

5.1.8 Summary

The performance of the LFC was evaluated for all LHC*b* use cases and the required rates surpassed for all. The requirement for file registration was exceeded by 30% while the other use-cases were exceeded by more than 2 orders of magnitude.

Where bulk methods are available they always out perform session based approaches. Several use-cases have no associated bulk methods, namely the registration of replicas and the removal of replicas. The performance of

replica registration also suffers because the client side retrieval of GUIDs associated to LFNs is required before registering a replica. To address this, requests for additional functionality have been made to the LFC developers.

It has been seen in all tests that the rate at which operations can be performed increases when acting on large groups of files. This was observed for session based, bulk and directory based queries. The data management agents, discussed in Section 4.1.5, have been designed to allow the aggregation of requests such that wherever possible large groups of files are acted upon. This design gives an increased performance to the data management system and reduces the load on the LFC master instance.

5.2 Storage Resource Manager Performance

The Storage Resource Manager (SRM) protocol is the accepted standard interface to Grid storage for the LHC experiments. The SRM protocol allows a diverse range of space and file management capabilities (see Appendix A) which are supported to varying degrees by the deployed implementations in the EGEE Grid infrastructure. Although SRM was designed to provide a standard interface, the storage elements which implemented the SRM interface interpreted the standard in different ways, due to ambiguities in the specification.

The GFAL library, discussed in Section 1.7.4, was extended to provide SRM client capabilities to the LHC experiments. GFAL offers a subset of the SRM specification, providing core file and directory management capabilities: path metadata, obtaining URLs for transport, issuing prestage requests and removing files. These functionalities allow DIRAC to manage data on grid storage elements. The performance of the GFAL library and the underlying SRM implementation is central to the design of data management system components. The four functionalities were evaluated against the SRM services at each of LHC*b*'s Tier-1 sites. Across the seven sites three flavours of SRM are deployed:

- Castor SRM - at CERN and RAL
- dCache SRM - at GRIDKA, IN2P3, NIKHEF and PIC

- StoRM - at CNAF

The response time for each of the SRM services was measured for a varying number of files, each hour, over a period of 20 days. This approach was chosen to provide a large statistical basis for the measurements and to determine any time-variability of services' performance. The time taken for each operation was recorded and the size of the file grouping plotted against the mean time taken. The errors bars presented are the standard deviation of the time distribution. All of the results discussed in this section are contained in full in Appendix D.

5.2.1 Retrieving File Metadata

The retrieval of file metadata in GFAL is a wrapper around the srmLs method and is the most solicited of the available functionalities. The information returned is similar to a POS-IX like *ls* providing file size and permissions data in addition to the locality of the file on disk and/or tape. The file metadata information is used within the Data Management agents such as the RAW Integrity Agent (to determine when a file is migrated and to obtain the checksum), the Stager Agent (to determine when a file is available on disk cache) and the Data Integrity agents (to verify the existence of files and cross check their size and checksum information). The functionality is also used when performing file uploads to ensure:

- The target file does not already exist before executing the transfer.
- The upload was successful on its completion.

The time to obtain metadata was measured for a varying number of files. This was done each hour, over a period of 20 days and the average response calculated. This involved ∼3k SRM interactions and the metadata of ∼90k files being retrieved at each site. The *mean* response time for each site is shown in Figure 5.10, and the observed success rates are given in Table 5.3.

The two Castor services, present at CERN and RAL, show the best performance. They have well defined response times below 10 seconds. At both sites a decreased success rate was observed for group of files of 50 or above. For files groups of 100 CERN displays a success rate of almost zero

Files	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	1.00	0.99	1.00	0.97	1.00	0.98	1.00
2	1.00	0.99	1.00	0.98	1.00	0.98	1.00
5	1.00	0.99	1.00	0.98	1.00	0.98	1.00
10	0.99	0.99	1.00	0.99	1.00	0.98	1.00
20	1.00	0.98	1.00	0.99	1.00	0.98	1.00
50	0.46	0.98	1.00	0.96	0.94	0.96	0.95
100	0.01	0.99	0.88	1.00	0.67	0.87	0.52

Table 5.3: Success rates for file metadata retrieval from site SRMs for varying number of files using GFAL.

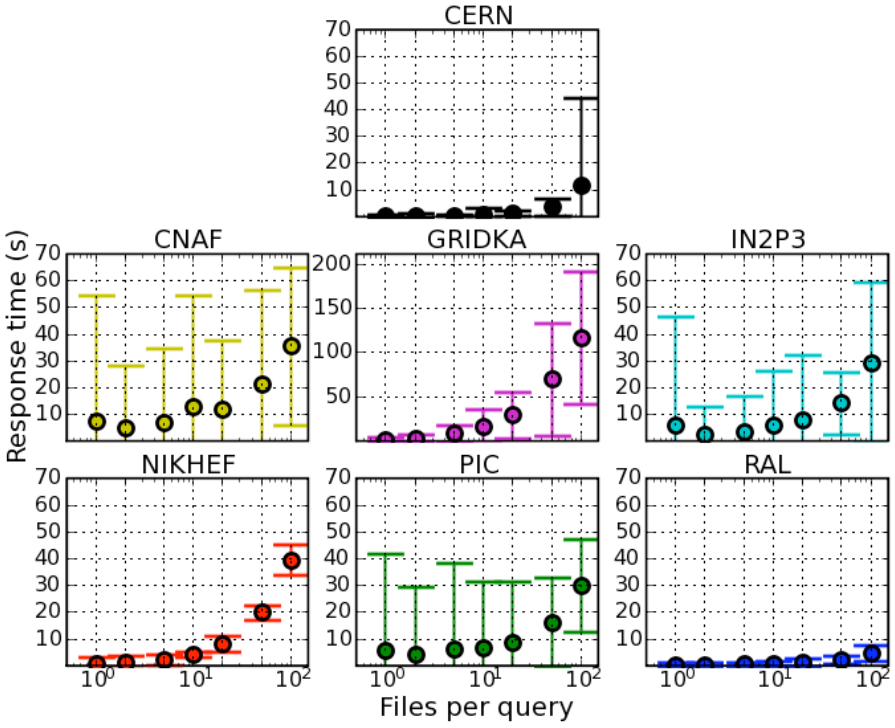


Figure 5.10: Retrieval time for file metadata from site SRMs for a varying number of files using GFAL. Note the different scale for GRIDKA.

which explains the significant response time variation. The dCache services, present at GRIDKA, IN2P3, NIKHEF and PIC display a similar profile. The response time is approximately flat for file groups ranging from 1 to 20, but increases rapidly above this. This effect is particularly noticeable at

NIKHEF and GRIDKA, and is accompanied by a decrease in success rates. For both Castor and dCache file groups over 20 result in an increased failure rate. To avoid this a default file grouping of 20 files has adopted within the DIRAC Storage Element plugin.

5.2.2 Issuing Prestage Requests

The DIRAC Stager System centrally manages staging requests for all DIRAC activities. The stager performs two distinct actions: the submission of prestage requests and the monitoring of file status to discover when files become available. When performing large pre-staging operations, as is envisaged for re-processing activity, the goal of the Stager is to allow the storage element tape system to optimise tape recalls by providing large numbers of prestage requests. During the re-stripping exercise the entire sample of 500k RAW and rDST data files must be recalled from tape over the period of one month. This gives a requirement of issuing a prestage request for a single file every 5 seconds. The performance of issuing prestage requests for varying numbers of files is given in Figure 5.11.

The time to issue prestage requests is significantly lower than retrieving file metadata with most implementations returning positive responses within 10s for all file grouping sizes at all sites. For groups of 100 files this gives a rate of 10Hz which is two orders or magnitude in excess of requirements. The Castor services at CERN and RAL show the same profile with an increased response time with larger groups of files. Operations containing more than 50 files exhibited lower success rates, contributing to the increased variance. The dCache services all show a flat profile as the group size increases. The IN2P3 service shows better response times than all other services, closely followed by PIC.

An interesting feature of the dCache sites is the variation in response time between sites, that previously showed similar responses to metadata queries. The dCache service does not provide tape management, but *attaches* an external tape management system through a plug-in mechanism. The difference in performance of GRIDKA and NIKHEF compared to the other dCache services reflects the communication between dCache and the

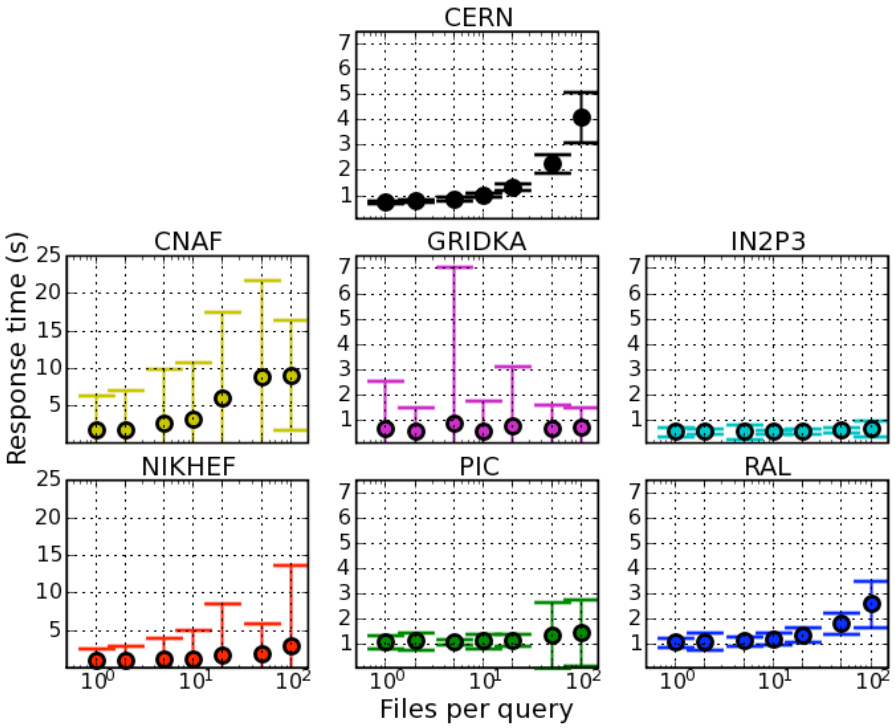


Figure 5.11: Time to issue prestage request to the site SRMs for a varying number of files using GFAL. Note the different scale for CNAF and NIKHEF.

underlying tape system manager[3].

5.2.3 Retrieving Transport URLs

The most important functionality provided by a storage element is allowing access to data under management. The SRM interface provides a mechanism for negotiating the transport protocol used and provides a transport URL (tURL, see Appendix A) for the agreed protocol. The SRM server accepts an ordered list of protocols from the client and performs a logical AND with the protocols it supports. The SRM then performs a brokering with the storage element backend to find the best disk server to serve the file from. This information is then used to create the tURL which is returned

[3]NIKHEF and GRIDKA both use the TSM tape manager from Tivoli while IN2P3 uses HPSS from IBM and PIC uses Enstore developed by Fermi National Laboratory

to the client.

The LHC*b* Computing Model dataflow (excluding user activity) gives the yearly requirement for getting access to files through the SRM. Obtaining transport URLs is required when replicating data and when reading data from within a job. These two activities combined require a total of 18M files to be accessed over a year period. By far the largest contribution to this number is the access required to the small files produced by the stripping activities when performing the merging and therefore the requirement is split between the Tier-1s in proportion with the pledged CPU. Taken over a year long period this rate becomes 0.56Hz.

The success rate for for each of the Tier-1 SRMs is given in Table 5.4 and the retrieval time is shown in Figure 5.12.

Files	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	0.98	0.84	0.97	0.98	0.82	0.96	0.99
2	0.98	0.86	0.96	0.97	0.75	0.96	0.98
5	0.98	0.89	0.95	0.97	0.66	0.93	0.98
10	0.98	0.87	0.96	0.97	0.65	0.88	1.00
20	0.98	0.88	0.93	0.96	0.66	0.88	0.98
50	0.00	0.89	0.87	0.97	0.64	0.83	0.69
100	0.98	0.90	0.93	0.97	0.67	0.85	0.99

Table 5.4: Success rates for tURL retrieval from site SRMs for varying number of files using GFAL.

Comparing Tables 5.3 and 5.4 the success rate of obtaining tURLs is lower at all sites. At CERN and RAL, which both use Castor SRM, an interesting *race condition* occurs when retrieving 50 files resulting in a significant drop in success rate. The StoRM service at CNAF shows a constant failure rate of 10-15%. Significant error rate of over 30% is observed at NIKHEF for groups of files greater than 2.

Despite the high failure rate the StoRM service returns tURLs within 20 seconds. The Castor services show a flat response profile, particularly at RAL, but demonstrate occasional out-lying events that result in a large measured deviation. The dCache services demonstrate poor response time

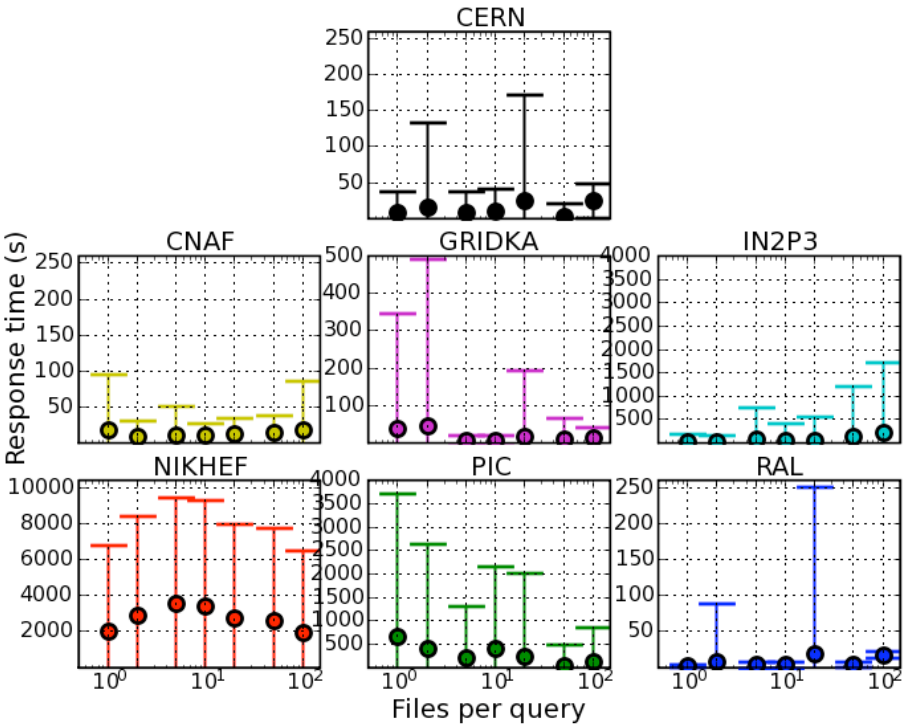


Figure 5.12: Retrieval time for transfer URLs from site SRMs for a varying number of files using GFAL. Note the different scales.

with this operation, i.e. GRIDKA, IN2P3, NIKHEF and PIC. The architecture of dCache is such that to obtain access to a file through the SRM five internal service look-ups are required. The complexity of this operation and the dependency on multiple components results in a significantly increased response time. This is effect was most marked at NIKHEF where response times of over 14k seconds were observed for a single file.

The highest retrieval rates were observed for large file groupings for all sites. The best performing sites are CERN, CNAF and RAL each displaying rates around 5Hz. Three dCache sites GRIDKA (2Hz), IN2P3 (0.5Hz) and PIC (0.5Hz) also exceed the total required aggregate rate. The pledged share of resources at NIKHEF, given in Table 2.3, is 21% requiring a retrieval rate of 0.12Hz. At NIKHEF the peak rate retrieval rate observed is 0.05Hz which is twice less than required.

5.2.4 Removing Files

As discussed in Section 5.1.5 15M files must be removed each year as a result of the merging activity giving an aggregated removal rate across all Tier-1s of 0.5Hz. To determine the rate at which files could be removed from the SRM two tests were performed. The first test, to remove single byte files, was to determine the rate at which files could be removed from the namespace with minimal physical bytes. This gives a measure of the SRM overhead for removing files. The second test, with 2GB files, was to determine overall rate at which data could be removed through the SRM. The results of the first test are shown in Figure 5.13.

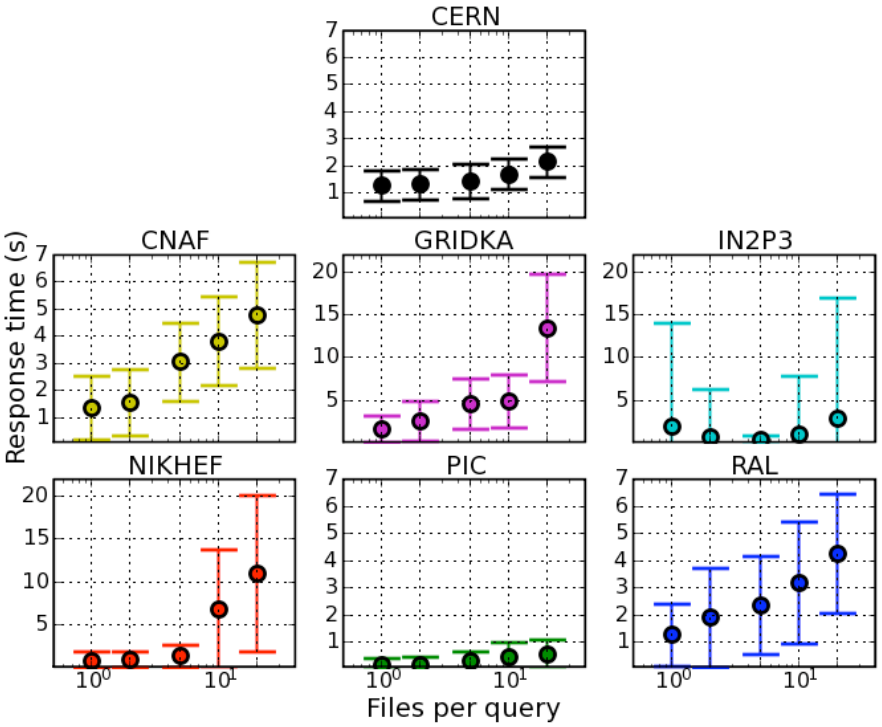


Figure 5.13: Time to remove 1 byte files from site SRMs for a varying number of files using GFAL. Note the different scale for GRIDKA, IN2P3 and NIKHEF.

The removal of single byte files is well defined across all services. The peak removal rate varies significantly across the sites: the quickest is PIC

with 20Hz with 20 files and the slowest is GRIDKA with 1.2Hz with 20 files. This slowest rate at a single site is 2.4 times above the requirement for all sites in total. The combined rate possible for all sites is 60Hz which is over 100 times the overall requirement.

The Castor services show a similar profile that increases regularly as the group size increases. PIC is the quickest of all SRMs where 20 files may be removed in 0.6 of a second. For all groups of files IN2P3 shows mean removal times below 5 seconds, but displays significant variance. GRIDKA and NIKHEF show a similar profile with significant increases in response time as the group size increases.

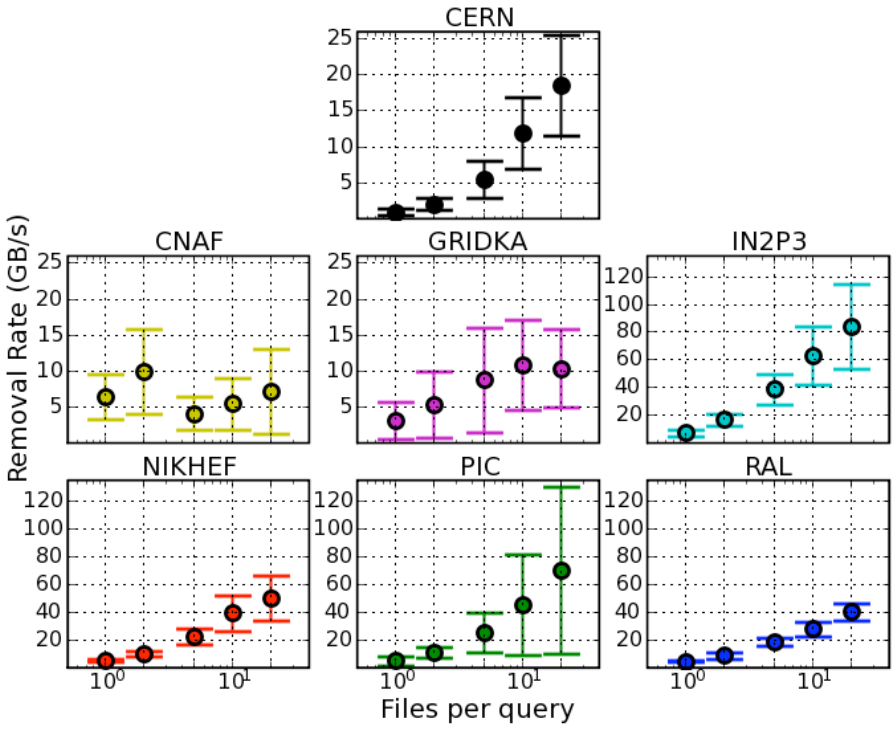


Figure 5.14: Removal rate of data from site SRMs for a varying number of 2 GB files using GFAL. Note the different scale for CERN, CNAF and GRIDKA.

The rate at which data can be removed through the SRM, using 2GB files, is shown in Figure 5.14. At GRIDKA a plateau was observed at

$\sim$10GB/s. At all other sites the removal rate increased with the size of the grouping. Rates up to 80.4GB/s and 70.3GB/s were measured at IN2P3 and PIC respectively. File removal is significantly higher than data can be *written* to the storage element. To highlight this, the removal all of the RAW and rDST data produced during LHC*b* data taking period (1PB) of 1 year would take (assuming a removal rate of 7GB/s) $\sim$39 hours.

5.2.5 Uploading Files

The data produced by a processing job running on a Tier-1 worker node should be uploaded to the SRM at that Tier-1 to ensure the share of pledged storage resources is respected. Uploading files involves two main operations: preparation of the target file and the transfer. The preparation of the target file is done by GFAL issuing asynchronous requests to the SRM[4] to obtain a tURL for writing. Internally, the SRM must decide which disk server to place files on and prepare the tURL to return to the client. To determine the status of the prepare to put request GFAL periodically polls the SRM until the tURL is returned. Once the tURL has been obtained the upload of the file takes place using GridFTP. Within the suite of *lcgutils*, discussed in Section 1.7.4, a method exists that combines the SRM interaction and GridFTP transfer step and is used within DIRAC to perform file uploads. The performance of the upload of single byte files was evaluated at all Tier-1s and is given in Figure 5.15.

The performance of the Tier-1s varied drastically. Both PIC and IN2P3 complete almost all uploads within 10s with CNAF and NIKHEF and GRIDKA completing all within 60s. The CNAF profile, in Figure 5.15(b), shows distinct clustering of results around 5, 15 and 25 seconds which alludes to the periodic polling of the SRM to obtain the destination tURL. The two Castor sites, CERN and RAL, both show much higher upload times than the others with only 80% of CERN uploads and 54% of RAL uploads completed with 60s.

[4]Specifically a srmPrepareToPut operation.

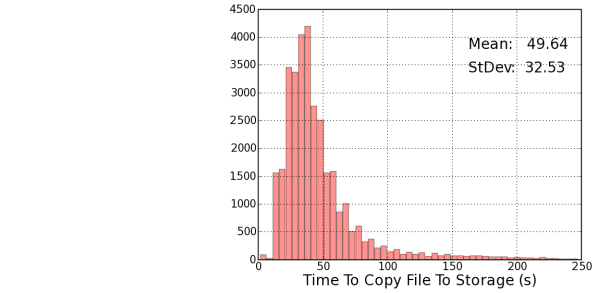
5.2.6 Summary

The performance of the SRM interface to grid storage was evaluated at each of LHC*b*'s Tier-1s. During the tests 3 different SRM implementations (Castor, dCache and StoRM) were tested with four GFAL methods (gfal_ls, gfal_prestage, gfal_turlsfromsurls and gfal_deletesurls). The requirements derived from the Computing Model for the issuing prestage operations was exceeded by 2 orders of magnitude. When removing files from the SRM the lowest observed rate was 2.4 times above the requirement with an aggregate rate 2 orders of magnitude grater that required. It was observed that all of LHC*b*'s RAW and rDST data could be removed in a under two days. The ability to retrieve transport URLs was achieved at all sites other than NIKHEF where the achieved rate is less than half of the requirement.

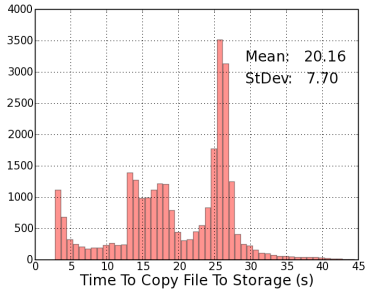
One major concern regarding the performance of the SRM services was discovered. During the transport URL test, discussed in Section 5.2.3, very high response times, performance variability, and failure rate for dCache services was observed. This discovery carries extra weight for LHC*b* as the two largest Tier-1s by pledged resources (NIKHEF and IN2P3) fall into this category. Efforts to diminish the effect of this have been discussed but at the time of writing no clear solutions have been proposed.

Overall, the Castor SRMs were seen to be the most perfomant with significantly less variation for retrieving metadata, submitting pre-stage requests, retrieving tURLs and file removal. For all implementations the number of files in a request affects the operation response time. In addition, larger groups of files are relatively more performant than smaller groups, although groups of 50 or 100 are more prone to error.

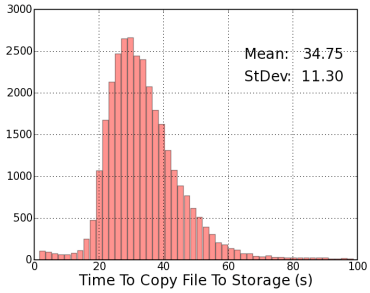
The rate at which pre-stage requests may be submitted compared to the rate at which file metadata can be retrieved had a significant influence on the design of the Stager System. The submission and monitoring of the pre-stage requests was split into two agents. This was done to ensure that retrieving metadata did not hamper the submission of the requests and therefore the ability for the tape system to optimise the recall process.



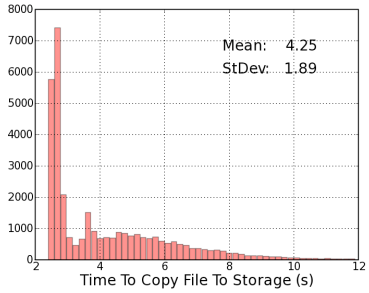
(a) CERN



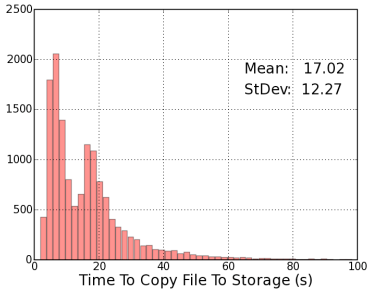
(b) CNAF



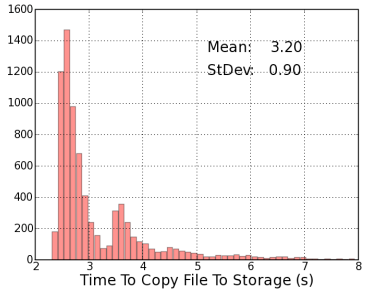
(c) GRIDKA



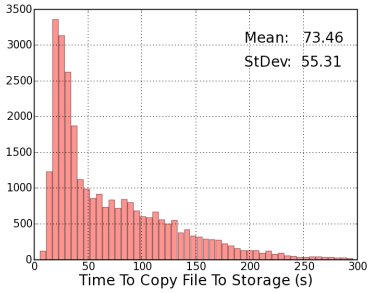
(d) IN2P3



(e) NIKHEF



(f) PIC



(g) RAL

Figure 5.15: Histogram of time taken to upload a 1 Byte file to site SRM from local worker node.

Chapter 6

DIRAC Bulk Transfer Framework Performance

The DIRAC transfer framework, discussed in detail in Section 4.2, is based on the principle of a central scheduler that assigns files to task queues representing source and destination storage element pairs. Transfers are retrieved from the task queues and executed as transfer slots become available. This framework supports the file replication activity of all parts of the LHC*b* dataflow and must be able to meet the network requirements outlined. This chapter presents the performance of the DIRAC transfer framework and discusses the implications for the design choices that were made. Section 6.1, will outline the peak network rates to be supported by the transfer framework. Section 6.2 will give the results of the transfer framework commissioning exercise performed to ensure the scalability of the system. Section 6.3 will discuss the possibility to apply intelligent replication strategies within the transfer framework.

6.1 Peak Network Requirements

The Computing Model, described in Section 2.4, outlines the role of each computing center according to the tiered MONARC model. The distributed processing of data is performed at seven LHC*b* Tier-1 centres and requires sustained data transfer between sites to ensure data persistency and provide increased availability of data to physicists. The peak network requirements occur during the re-stripping period which takes place concurrently with

data taking. During this period, as discussed in Section 2.5, the transfer activities are:

- RAW data transfer from CERN to external Tier-1s
- Distribution of DSTs from quasi-real time processing to all Tier-1s
- Distribution of the DSTs from re-stripping to all Tier-1s
- Distribution of the Monte Carlo DSTs to three Tier-1s

The total required data rate during this period, in and out of each site, are given in Table 2.8. The network rates required into each site are approximately equal at around 60MB/s. The rates out of the sites differ for two reasons. The primary reason is the differing CPU pledges at the sites. A larger CPU pledge results in more DSTs being produced at the site. Since the DSTs must be distributed to multiple final destinations the rates can vary significantly. The CERN export rate is higher than other similarly sized Tier-1s because of the RAW data that is exported at 50MB/s. The values in Table 2.8 are sustained rates over the 1 month period of re-stripping and therefore the transfer framework should demonstrate the capability of peak rates in excess of these.

6.2 Commissioning Exercise

To ensure the transfer framework can support the LHC*b* dataflow a commissioning exercise was performed with two main aims:

- To ensure the transfer framework scales to support PBs of data
- To achieve the required transfer rates into and out of each site

The exercise was set up using 2GB files as seeds at every Tier-1 site to be replicated to each of the other sites. Large files were chosen to mimic the real RAW and DST files which will be transferred during data taking and re-stripping. The target of the exercise was to transfer 1PB. This volume of data was chosen to ensure scalability was addressed and to determine the achievable rates for each site.

To ensure the full transfer framework chain was tested, without interference, the exercise was managed from a stand alone client installation.

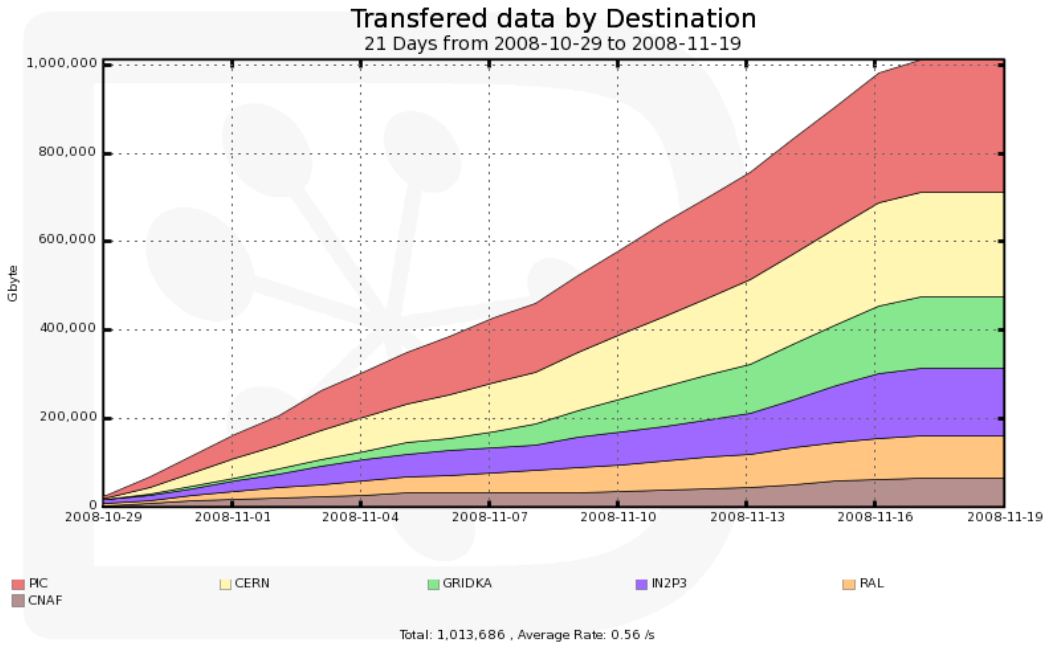


Figure 6.1: Cumulative transferred data during commissioning exercise grouped by destination site.

Requests to transfer the seed files were created by the client and set in the Transfer DB. From here they were assigned to task queues by the Replication Scheduler, the FTS Submit and Monitor agents controlled interaction with FTS and the replica registration was performed on completion. To repeat the cycle the client would periodically check for successfully registered files at the target SEs, remove the physical files, and their catalog entries, then create new requests to retransfer the files. With this approach the system, and each of the underlying source-destination SE pairs, was always loaded.

The exercise completed within 21 days, transferring a total of 1.01PB (see Figure 6.1). This accomplished the first aim of the commissioning exercise; to achieve scalability of the framework to the PB level.

6.2.1 Transfer Throughput Scalability

The metrics used to determine the throughput scalability of the system are:

- The framework should support overall transfer rates 1.5 times the maximum required network rate. This translates to 412×1.5=618MB/s.

- The framework should be able to saturate the available network resources.

The observed throughput over the course of the exercise grouped by the destination site is shown in Figure 6.2.

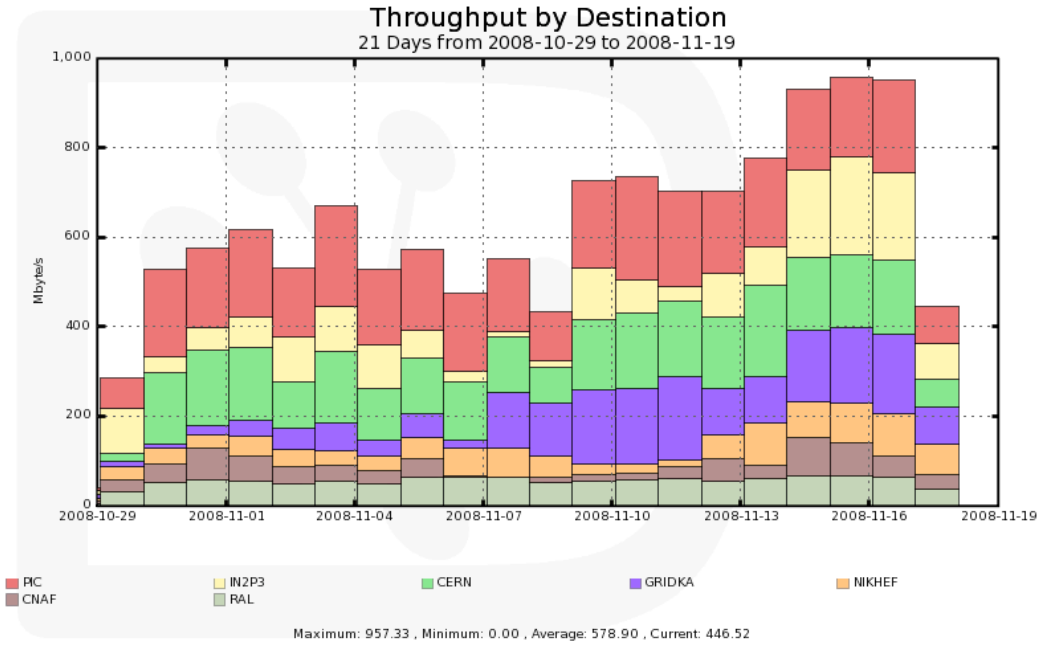


Figure 6.2: Transfer throughput during commissioning exercise grouped by destination site (MB/s).

There are three important periods that can be seen in Figure 6.2. The first period (in the first 11 days) was characterised by stable throughput rates of 500MB/s that decreased to around 400MB/s on the final day. The second phase was characterised by transfer throughput of ∼700MB/s sustained for 5 days. The third phase, during the last three full days of the exercise, displayed sustained daily rates of ∼950MB/s. These three result periods will be discussed in turn.

Phase 1

During the first phase of the exercise the sustained daily rate was in excess of the 412MB/s total peak rate required, with the exception of the first day ramp-up. As the period progressed the daily rate decreased until reaching ∼410MB/s on the final day. The cause of the decreased performance

was investigated and a significant bottleneck was discovered in the central database. The slow database response was limiting the rate at which the agents in the framework could retrieve their tasks. This resulted in a reduction in the rate at which FTS requests were monitored, which in turn limited the rate at which new FTS requests could be submitted.

To increase the performance the database schema was revised, making use of indices for heavily solicited table rows. The effect of the database update was studied by evaluating the number of times each FTS Request was monitored. This is shown in Figure 6.3.

As the first period progressed the mean number of monitors for each request decreases. An associated reduction in the standard deviation is also seen illustrating that the database slowdown affected all FTS requests. The schema change was made on the 9th of November after which the mean number of times each FTS Request was monitored increased. This resulted in a higher turn-around of requests and increased throughput.

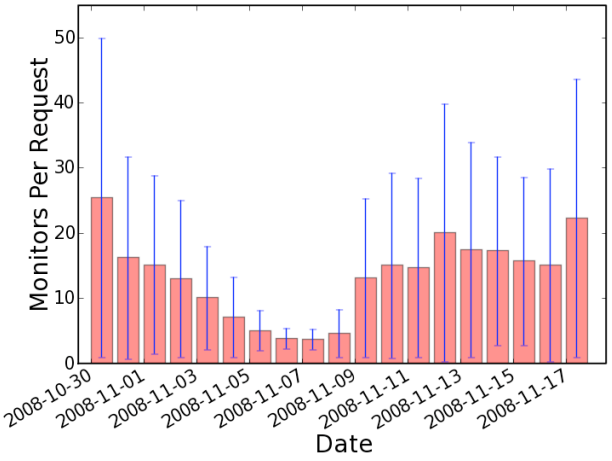


Figure 6.3: Mean number of times FTS requests were monitored by the FTS monitor agent (red) grouped by day. The standard deviation observed during each day is show in blue. The updated database schema was introduced on the 9th of November.

Phase 2

The second period was characterised by daily transfer throughput of ∼700MB/s. Although the throughput was successfully increased it coincided with an increase in the observed error rate. It was discovered that 12.3% of the errors were caused due to the target storage elements being full[1]. On further investigation, the storages exhibiting this symptom were found to all be dCache SRMs. Discussions with the dCache SRM administrators revealed that when files are 'deleted' from dCache the space associated to these files is reclaimed asynchronously. Therefore, even though files had been removed from the namespace dCache thought there was no space available in which to write new files.

The initial test setup was designed to immediately remove files from the target SRM after the replica catalog registration was performed to conserve space. To overcome the problem associated to the asynchronous space recovery the setup was modified to transfer files alternately to three different SRM *spaces*. These different SRM spaces are accounted by dCache independently, effectively allowing access to three times as much physical disk. The modification was made on the 13th, after which, no further errors with this prognosis were observed.

Phase 3

During the final three days of the exercise, ignoring the last day ramp-down, an average daily throughput of ∼950MB/s was achieved. This throughput was limited only by the performance of the site SRMs and underlying GridFTP servers. At this daily average rate the transfer framework was twice the total required rate given in Table 2.8; achieving the first throughput scalability metric given above. A snapshot of the throughput on one of these days is given in Figure 6.4 with peak throughput over 1GB/s, 2.4 times the required sustained rate.

The network rates during the final period of the exercise, in and out of each site, is given in Table 6.1. The network requirements are given in

[1] 600219 transfers were attempted of which 480550 failed. Of those, 59164 failed with the error signature NO_SPACE_LEFT.

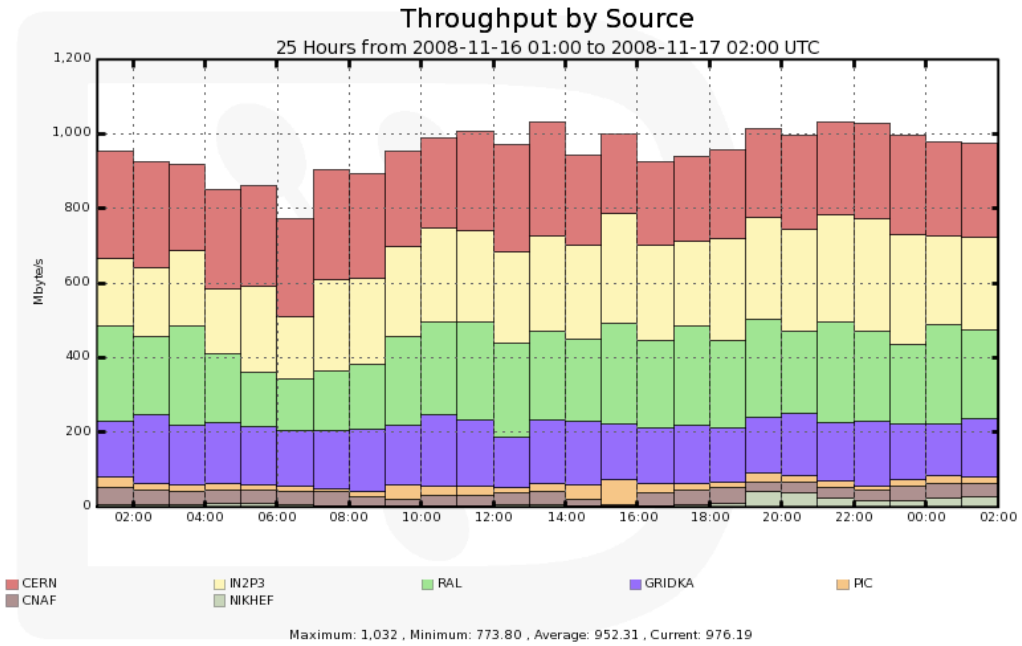


Figure 6.4: Transfer throughput during commissioning exercise grouped by source site (MB/s).

Table 2.8 were met and frequently exceeding requirement by 50%, allowing the system to recover from backlogs effectively. There was one exception: the network rate out of NIKHEF.

As was found in Section 5.2.3, the ability to retrieve tURLs for existing files from NIKHEF is error prone. The mean time to return tURLs at NIKHEF was observed to be over 2000s (for all file group sizes tested). The FTS applies a timeout when retrieving tURLs from the source (and target) SRM which is configured be of the order of 300s. If this timeout is exceeded the transfer is failed by FTS. This severely limits the ability to transfer data out of NIKHEF.

6.2.2 Channel Performance

During the exercise almost all of the raw throughput metrics were accomplished. These metrics were chosen to ensure that the site SRMs could cope with the aggregated required network rates. The performance of the individual FTS channels will be discussed in this section.

	In from Tier-1	Out to Tier-1
CERN	163.1	232.7
CNAF	70.1	66.5
GRIDKA	169.4	155.0
IN2P3	202.9	213.0
NIKHEF	86.6	4.7
PIC	189.8	30.3
RAL	65.0	244.8
Total	946.9	946.9

Table 6.1: Aggregated throughput achieved during final phase of commissioning exercise (MB/s).

The exercise was set up to maintain a sustained equal load on each FTS channel. This ensured that channels were never starved of data. Therefore during the period of the exercise, better performing channels would transfer more data while lesser performing channels would transfer less. To evaluate the performance of each channel several metrics were investigated. The first metric was the success rate of each of the channels observed during the final phase of the exercise, given in Table 6.2.

		Destination						
		CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
Source	CERN	-	0.76	0.96	0.89	0.95	0.97	0.92
	CNAF	0.72	-	0.03	0.44	0.10	0.63	0.14
	GRIDKA	0.78	0.12	-	0.79	0.05	0.45	0.83
	IN2P3	0.87	0.20	0.87	-	0.70	0.80	0.74
	NIKHEF	0.36	0.01	0.03	0.08	-	0.57	0.31
	PIC	0.43	0.04	0.11	0.20	0.31	-	0.47
	RAL	0.85	0.19	0.95	0.96	0.60	0.82	-

Table 6.2: Channel success rate during final phase of commissioning exercise.

It can be seen from the Table that the success rate of 20 of the 42 channels is less than 50%[2]. One of the arguments for using FTS for data transfer is that it allows sites to manage the load on their SRMs and control the network traffic to maintain system stability. To do this sites configure the number of file transfers that are executed concurrently on each channel. A larger number of files allows the possibility to achieve higher transfer

[2]20 (of 42) channels less than 50% success rate, 22 channels less than 60%, 24 channels less than 70%, 29 channels less than 80%, 36 channels less than 90%

throughput with a corresponding increased load on the system.

The second metric measuring the performance of each channel was derived from the individual file transfer times (obtained from FTS logging information). The mean file transfer time (for successful transfers) over the final exercise phase for each channel is given in Table 6.3.

		Destination						
		CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
Source	CERN	-	957.6	211.3	228.0	551.2	159.8	569.1
	CNAF	1550.8	-	1566.7	687.6	1231.3	1276.7	1574.0
	GRIDKA	579.9	1862.1	-	253.8	501.4	603.6	1100.3
	IN2P3	217.5	1848.0	247.4	-	515.5	544.3	769.5
	NIKHEF	1303.8	2673.8	690.4	743.9	-	2247.6	1990.2
	PIC	2002.7	2587.7	455.5	617.6	1661.0	-	2512.0
	RAL	662.0	1472.7	374.0	349.0	882.6	747.5	-

Table 6.3: Mean file transfer time of 2GB file during final phase of commissioning exercise (s).

To obtain a measure of the instantaneous throughput on a channel the mean transfer time was combined with the number of concurrent transfers per channel and the observed success rate. The number of concurrent files transferred on each channel varied by source and destination with most channels configured to transfer between 4 and 15 files[3]. Since the file size was constant (2GB) the formula for obtaining an estimation of throughput is:

$$\text{throughput} = \frac{\text{concurrent files}}{\text{mean transfer time}} \times \text{success rate} \tag{6.1}$$

This estimated throughput for each channel is given in Table 6.4. The estimated throughput and success rate is represented in a scatter plot for each source and destination pair in Figure 6.5. This shows a non-trivial correlation between the estimated throughput and the channel success rate. This suggests that protecting the storage element from high load, increases the overall throughput that can be achieved. The high failure rates seen in Table 6.2 also suggest that the balance between performance and stability

[3]The CERN-Tier1 and Tier1-CERN transfers are managed by the CERN FTS server and are weighted by the relative contributions of the sites to LHCb computing activities. For Tier1-Tier1 activity transfers into CNAF were configured for 10 concurrent files, GRIDKA and RAL for 5, IN2P3 and NIKHEF for 10 and PIC for 15.

		Destination						
		CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
Source	CERN	-	7.92	45.64	109.55	17.26	36.57	32.48
	CNAF	16.75	-	0.16	12.70	1.69	14.74	0.90
	GRIDKA	64.64	0.50	-	62.11	1.85	22.41	7.51
	IN2P3	79.75	0.86	35.08	-	27.03	44.32	9.66
	NIKHEF	6.59	0.02	0.45	2.02	-	7.62	1.58
	PIC	25.96	0.13	2.37	6.49	3.70	-	1.87
	RAL	51.27	1.04	25.31	55.03	13.49	32.90	-

Table 6.4: Estimated channel throughput during final phase of commissioning exercise (MB/s).

has not been achieved yet.

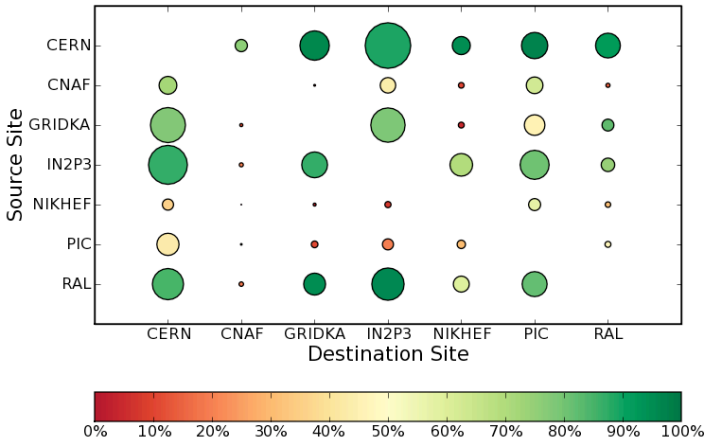


Figure 6.5: Channel throughput and success rate by source and destination site. The normalised network rate is denoted by the area of the dots and the success rate denoted by the colour, ranging from 0% (red) to 100% (green).

The balance between performance and stability is made more complicated by the deployment schema of FTS whereby a channel is managed by the destination site FTS server[4]. In this schema it is possible to directly control the transfer activity into a site, but not the transfer activity out. Therefore, a site may overload another by configuring the channels according to the target site capabilities. This suggests the need for a global

[4]Except channels with CERN as a source which are managed by the CERN FTS.

co-ordination between the sites to obtain realistic channel configuration.

6.2.3 Discussion

In this section the results of the transfer framework commissioning exercise were presented. The bulk transfer framework was designed with a central database to allow replication scheduling policies to be applied to all LHC*b* replication activities. The counter argument to a centralised design is that it creates a possible performance bottleneck that can be avoided with a fully distributed system. It was seen that during 21 days the framework transferred over 1PB of data and supported peak transfer rates of over 1GB/s, approximately 2.4 times the rates required from the LHC*b* computing model. This provided justification for the selected design.

During the commissioning exercise a significant variation in performance of the FTS channels was observed. It was shown that the success rate and observed throughput show a non-trivial correlation suggesting that a coordinated FTS configuration between the sites should be undertaken to protect the storage elements from high load.

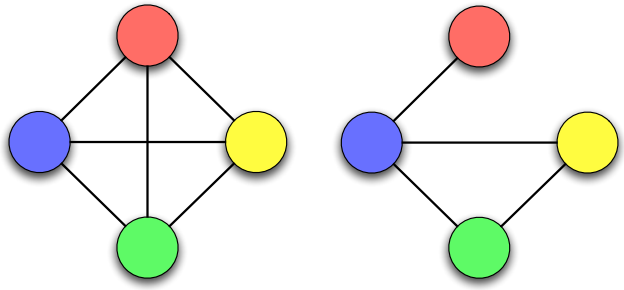
6.3 Replication Strategies

This section will present the mechanism used in the DIRAC bulk transfer framework to efficiently use available network resources. Section 6.3.1 will give a brief introduction to the concepts of graph theory that is used throughout. The subsequent sections present two algorithms employed to execute scheduling decisions, and the generation of the input for the algorithms. In Section 6.3.4 a comparative study of the performance of these algorithms is presented.

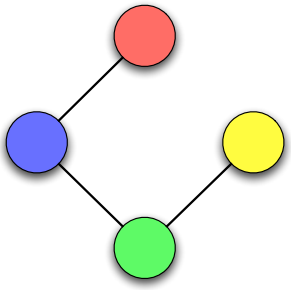
6.3.1 Introduction to Graph Theory

An introduction to the concepts used in the following sections is given in Figure 6.6.

Data produced as a result of the stripping phase of the LHC*b* workflow must be replicated from the generation site to several destinations. In the nomenclature of graph theory the seven LHC*b* Tier-1s are *vertices* in a *com-*



(a) A Complete Graph (b) A Sub-Graph



(c) A Spanning Tree

Figure 6.6: Introduction to graph theory. A complete graph connects all vertices to all other vertices. A sub-graph is a subset of an existing graph. A spanning tree connects all vertices in the graph. An edge connects any two vertices.

plete graph. The FTS channels connecting the Tier-1s are *edges*. When scheduling the replication of a file, a *sub-graph* is created which connects the source SE with all the required destinations.

The edges connecting the vertices have an associated *weight*, representing some metric for transferring data on that channel. To make the most efficient use of the available network the total weight of all spanning tree configurations can be evaluated. Minimising the total weight for connecting all the vertices provides a *minimum spanning tree* and is the most efficient way to transfer the data.

The following section will discuss the algorithms used to generate the spanning trees in the DIRAC bulk transfer framework.

6.3.2 Spanning Tree Generation

The role of the Replication Scheduler is to generate a spanning tree, connecting the primary vertex to all destination vertices. The algorithms used to generate the spanning trees, referred to as strategies, may be used interchangeably and are applied to each file at the point of scheduling. With the small number of available edges[5] the spanning tree generation can be performed deterministically for each file.

The two strategies that will be presented here apply the Prim-Jarnik (P-J) algorithm [174, 175] and a proprietary algorithm named Min-Sigma. The algorithm steps performed for each are given below.

Prim-Jarnik

Starting from the primary vertex the P-J algorithm builds the spanning tree by selecting always the lowest weighted edge to a non-connected vertex. This algorithm will always produce a minimum spanning tree for the LHC*b* setup because all sites are connected to all others. This is represented as follows:

- Input: A complete weighted graph with weights W, vertices V and edges E. The source vertices are V_{sources} and the selected edges are E_{selected}
- Initialise: The source vertices is initialised with only the primary vertex, $V_{\text{sources}} = \{\text{primary vertex}\}$. The selected edges are initialised to be empty, $E_{\text{selected}} = \{\}$
- Repeat the following steps until the source edges contain all the destination vertices, $V_{\text{sources}} = V$:
  - Choose the edge (u, v) from the possible edges E with the lowest edge weight, $MIN(W_{u,v})$ s.t. $u \in V_{\text{sources}}$ and $v \notin V_{\text{sources}}$
  - If there are multiple edges with the same $W_{u,v}$ choose arbitrarily
  - Add the selected destination v to the possible sources V_{sources} and add the edge (u, v) to the selected edges, E_{selected}

[5] The 7 LHC*b* Tier-1s connected to each other create 42 possible edges

- Output: Spanning tree containing the selected sources, V_{sources} and the selected edges, E_{selected}

The P-J algorithm minimises the sum of all consumed edge weights but does not consider the possible parallelism of transfer operations. For example, the time to transfer a file from A to B and C on the A-B and A-C channels is MAX(A-B,A-C) and not SUM(A-B,A-C). If the goal of the scheduling algorithm is to minimise the maximum weight (a proxy measurement for time) to reach all vertices, a different algorithm could be considered. The algorithm designed for this was named *Min-Sigma* (MS).

Min-Sigma

The Min-Sigma algorithm makes a small addition to the P-J algorithm to generate a minimised spanning tree that is not a conventional minimum spanning tree. At each step, the edge with the destination vertex with the lowest sum of all ancestor edges weights is selected. This is written as follows:

- Input: A complete weighted graph with weights W, vertices V and edges E. The source vertices are V_{sources} and the selected edges are E_{selected}
- Initialise: The source vertices is initialised with only the primary vertex, $V_{\text{sources}} = \{\text{primary vertex}\}$. The selected edges are initialised to be empty, $E_{\text{selected}} = \{\}$
- Repeat the following steps until the source edges contain all the destination vertices, $V_{\text{sources}} = V$:
  - Choose the edge (u,v) from the possible edges E with the lowest sum of ancestor weights, $MIN(\sum W_{source,v})$ s.t. $u \in V_{\text{sources}}$ and $v \notin V_{\text{sources}}$
  - If there are multiple edges with the same $\sum W_{source,v}$ choose arbitrarily
  - Add the selected destination v to the possible sources V_{sources} and add the edge (u, v) to the selected edges, E_{selected}

- Output: Spanning tree containing the selected sources, V_{sources} and the selected edges, E_{selected}

The P-J and M-S algorithms both generate replication trees containing edges with ancestors, i.e. *multi-hop* transfers. The cost of additional hops, taken to relate to the turn around time between one transfer completing and the descendent starting, is not included in algorithms outlined above. To model this extra cost an additional parameter, δ, is added to the edge weight if the edge source is not the primary vertex.[6]

The difference between the P-J and M-S algorithms, and the effect of the δ parameter is demonstrated using an example graph, see Figure 6.7. If the Red vertex is taken to be primary, with associated weights on each edge, then the minimum spanning trees generated by the different algorithms are shown in Figure 6.8.

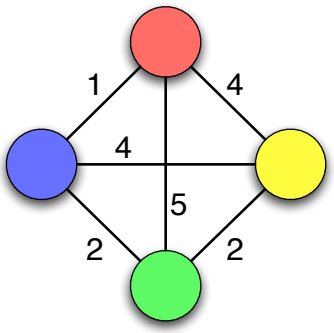


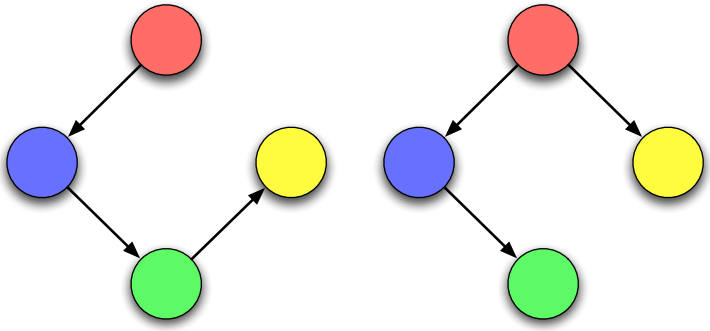
Figure 6.7: Complete weighted graph example. The numbers are the edge weights.

The P-J solution, Figure 6.8(a), can be seen to differ from M-S, Figure 6.8(b), since the M-S algorithm evaluates the Red-Yellow edge as smaller than the sum of the Red-Blue-Green-Yellow edges. The P-J solution would result in a total transfer time (5.2 including δ)[7] that is equal to the time till completion. The M-S solution has a higher total transfer time (7.1 including δ)[8] but, since the Red-Yellow edge will proceed in parallel, the time

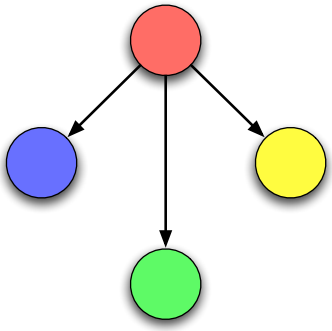
[6] In [176] a similar parameter was shown to regulate the replication tree depth and branching factor. They found increasing δ produced shallower trees with higher branching factors.

[7] 1+(0.1+2)+(0.1+2)=5.2

[8] 1+(0.1+2)+4=7.1



(a) Prim-Jarnik, $\delta = 0.1$ (b) Min-Sigma, $\delta = 0.1$



(c) Min-Sigma, $\delta = \infty$

Figure 6.8: Solutions to weighted graph in Figure 6.7.

till completion is estimated to be lower (the weight of the Red-Yellow edge: 4). The effect of increasing the δ parameter is to reduce the depth of the resulting tree, until at the upper limit, a *multi-unicast* strategy is produced, shown in Figure 6.8(c).

6.3.3 Edge Weight Generation

The edge weights provided as input to the strategy algorithms represent an estimate of the time for a newly assigned file to complete transferring on the edge. This weight is calculated using two sources of information: the observed throughput on the edge and the number of files already assigned to the edge.

Determining Throughput

The throughput on an edge can be calculated over a period of time chosen by the strategy. For a long-term view of the stability of an edge, a long consideration period may be chosen. For more timely information a short period may be chosen.

The throughput is calculated by evaluating the amount of data successfully transferred on the edge in the consideration period, and dividing by the amount of time the data transfer has been attempted for. With this approach, periods where no transfer activity was attempted do not diminish the observed throughput. This also includes failed transfers which reduce the observed throughput.

Assigned Files

Using solely the edge throughput in the scheduling algorithm would result in all files being scheduled on the most performant edges. These edges would appear equally attractive to all files irrespective of the number of files already assigned to the edge. This results in a waste of resources.

Edge Weight Formula

The edge weight is a simple combination of the following two numbers:

$$\begin{aligned} \text{edge weight} &= \frac{\text{volume of waiting data (MB)}}{\text{throughput (MB/s)}}, \\ &\text{or} \\ \text{edge weight} &= \frac{\text{waiting files}}{\text{throughput } (\text{s}^{-1})} \end{aligned}$$

Once a spanning tree is generated for a file the number of files waiting on each of the selected edges is incremented. The updated edge contents are then used in the computation of the spanning tree for the next file to be scheduled.

6.3.4 Comparing Strategy Performance

The algorithms outlined in Section 6.3.2 do not collude to attain optimal global performance, but approximate it by minimising consumed resources for individual files.

Considerations for Comparing Strategies

If the three examples given in Figure 6.8 were to be executed to transfer data in parallel the P-J and M-S ($\delta = 0.1$) algorithms would aid the performance of the M-S ($\delta = \infty$) algorithm. This would happen because the P-J and M-S ($\delta = 0.1$) algorithms do not select the Red-Green and Red-Yellow edges, leaving it free for use by the M-S $\delta = \infty$ algorithm.

To determine the true strategy performance the interference of concurrently executing strategies must be removed. As has been shown, the performance of grid resources are variable. To ensure the status of the resources does not dominate the performance comparison the strategies should be executed with minimal variation in time.

Experimental Setup

To compare the replication strategies a simple experiment was conducted. During this experiment 5k 24kB files were transferred from a single source site (CERN) to four external Tier-1 sites. Requests to transfer files were generated on a client machine and submitted to the bulk transfer framework. The files were split into groups of 100 and a particular strategy specified in the request.

The client submission of requests was done serially, such that a new request was created only when the previous has completed successfully. This was to avoid interference between concurrently executing strategies. Each new request incremented the strategy specified. The total time to complete the requests was recorded.

Three strategies were compared: P-J ($\delta = 0$), M-S ($\delta = 0$) and M-S with ($\delta = \infty$) (which generates simple multi-unicast trees). For the duration of the experiment the throughput consideration period was set to 12 hours

to keep a stable performance estimates between consecutive requests. The results are presented in the following sections.

Mean Time To Transfer

The mean time for a file to reach all of its destinations, per individual request, is given in Figure 6.9.

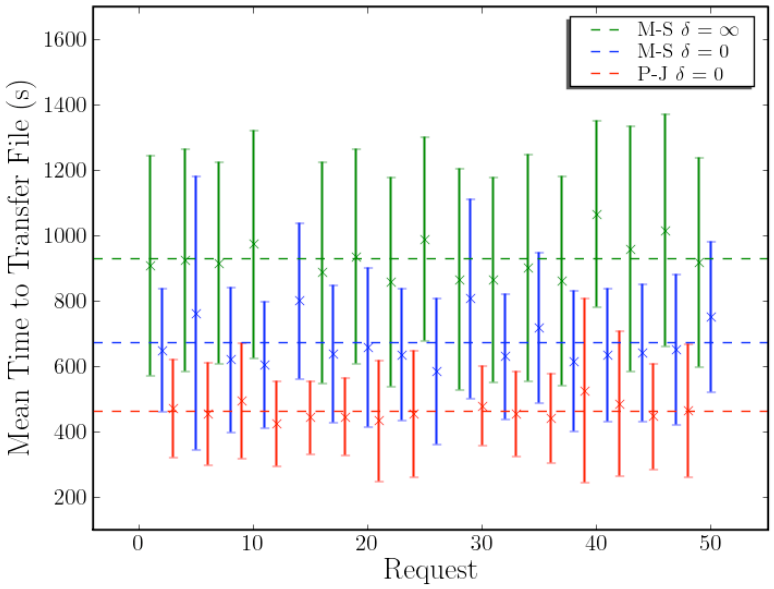


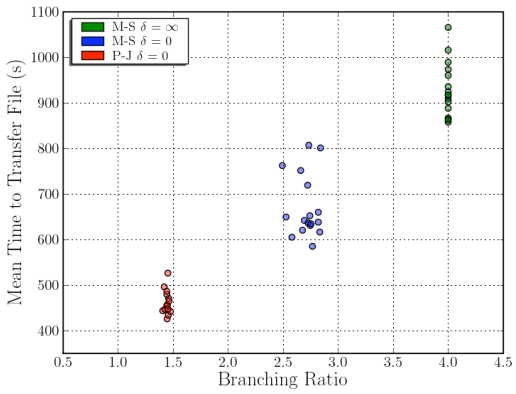
Figure 6.9: Mean time to transfer files within each request for the strategies evaluated. A best fit dotted line was applied to the mean transfer times for the requests of each strategy.

The M-S $\delta = \infty$ strategy consistently under-performs relative to the dynamic strategies, which verifies the premise that intelligent scheduling can be applied.

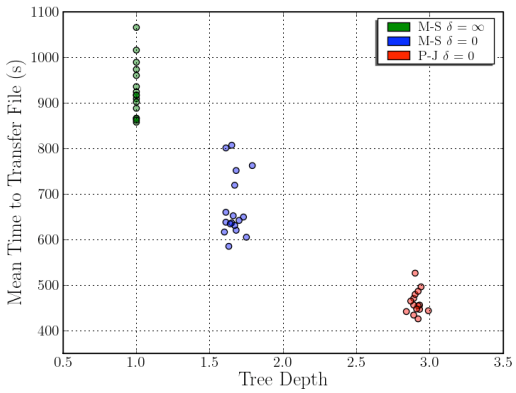
The performance of the M-S $\delta = \infty$ strategy is lower because the time for a file to reach all of its destinations is limited by the least performant of the edges. Of the two dynamic strategies P-J consistently outperforms M-S. To understand the performance difference, the spanning trees generated by each of the strategies were analysed.

Spanning Tree Properties

The spanning trees generated for each file scheduled during the exercise were recorded. The *branching ratio* of the spanning trees, which gives a measure of the number of times a vertex was used as a source, was calculated. The spanning tree *depth*, which gives a measure of how may hops have been performed from the primary vertex to the destinations, was also calculated. The average depth and branching ratio for all the files in a request is plotted against the mean time for the request to complete, Figure 6.10.



(a) Branching Ratio



(b) Tree Depth

Figure 6.10: Average request spanning tree properties.

The average branching factor, shown in Figure 6.10(a), and depth, shown in Figure 6.10(b), show clear correlation with the algorithm that produced the spanning tree. The P-J algorithm produces trees with low branching

ratio and high tree depth while M-S produces trees with high branching ratio and low tree depth (at $\delta = \infty$ limit the trees have a depth of 1 and branching factor equal to the number of destination vertices).

The most performant requests generate spanning trees with low branching ratios with high tree depth.

Edge Utilisation

Analysing spanning tree properties gives a file centric view of the algorithm results, but tells nothing of the overall edge utilisation. The number of files assigned to an edge in each request shows us whether the algorithm makes heavy use of some edges and light use of others. The number of files assigned to each edge for all requests in the exercise by the P-J and M-S algorithms is shown in Figure 6.11.

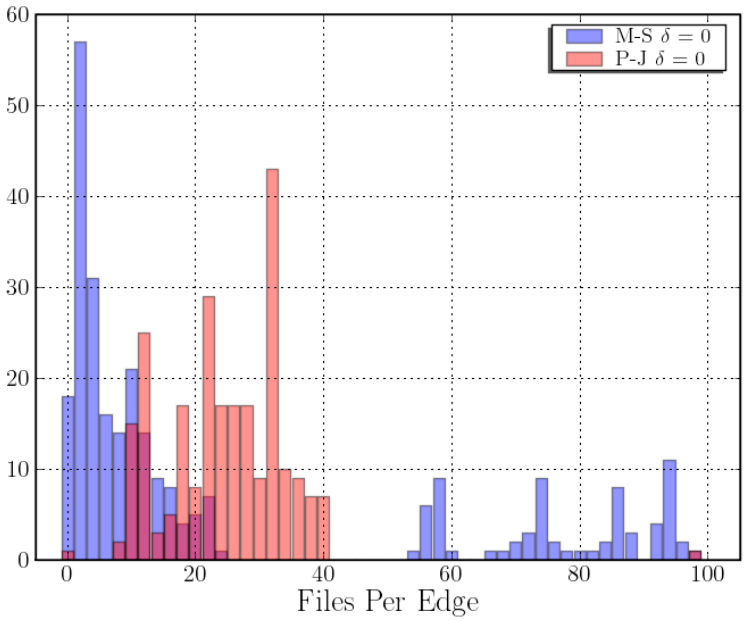


Figure 6.11: Aggregated assigned files per edge. The blue and red sections represent histograms for each algorithm employed with a small overload being observed between 10 and 25.

The M-S algorithm appears to have two groups of edges: heavily used

(more than 50 files per request) and lightly used (less than 20 files per request). The M-S algorithm values parallelism of transfers and the four peaks which can be observed above 50 correspond to the four channels out of CERN. The P-J algorithm displays a different profile with more distributed edge usage never assigning more than 40 files to an edge. The P-J algorithm chooses the cheapest edge making more use of the edges not related to the primary vertex. The result of this is that a file replicated to a destination becomes a new seed to replicate to the other destinations. This reduces the demand on the edges out from the primary vertex.

Edge Performance

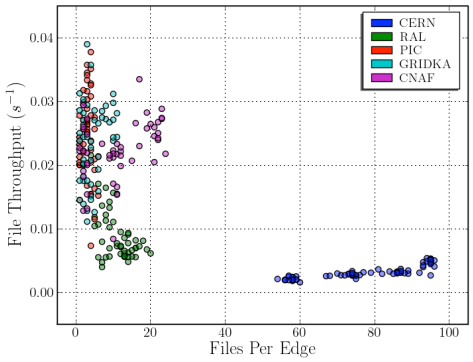
The validity of the scheduling decisions can be evaluated by comparing the number of files assigned to an edge against the performance transferring those files. This is presented in Figure 6.12.

The heavy M-S edges from Figure 6.11 can be seen in Figure 6.12(a) to correspond to the channels out of CERN (bottom right) and the lightly used edges connect the Tier-1 sites (top left). As the primary vertex, the edges out of CERN must be used to gain access other edges. The P-J algorithm makes less use of the edges out of CERN making use of the more performant edges connecting Tier-1s.

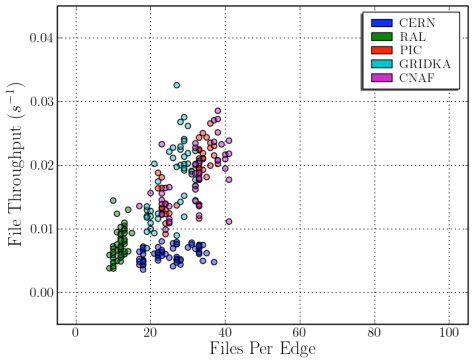
The higher performance of the P-J algorithm can be clearly seen in Figure 6.12(b) where, excluding the required edges out of CERN, the number of files assigned to an edge increases with the performance of the edge, demonstrating the validity of the routing decision.

6.3.5 Selected Strategy

This section has shown that intelligent scheduling is possible using past network performance to minimise the time to get a file to all destinations. Of the two dynamic algorithms that were presented the Prim-Jarnik outperformed the Min-Sigma algorithm. The P-J algorithm creates spanning trees with low branching ratio and high depth making higher use of more performant channels. For this reason this algorithm has been selected to be used by default in the bulk transfer framework.



(a) M-S $\delta = 0$



(b) P-J $\delta = 0$

Figure 6.12: Edge files scheduled vs. performance by source.

6.4 Summary

This chapter presented two facets of the performance of the DIRAC bulk transfer framework. It was seen that the bulk transfer framework scales to support the needs of the LHC*b* Computing Model and demonstrated network rates of over 1GB/s whilst transferring 1PB of data. The required transfer rate metrics, in and out of each site, were met (except data out of NIKHEF) but analysis of the individual channels showed significant variation in performance.

The ability to perform intelligent scheduling to make efficient use of network resources was demonstrated. The Prim-Jarnik algorithm has been adopted as the default strategy to be used. The adopted algorithm uses

edges in proportion to their overall performance and ensures that the variation in the underlying channels does not effect timely distribution of files to all their destinations.

The application of intelligent scheduling has the added benefit of making the required network rates for sites more flexible. It allows to compensate for the transfer metrics not achieved by shifting the export requirement to sites that are more performant.

Chapter 7

Conclusions

The Grid computing paradigm has been presented as a solution to the resource challenge faced by contemporary data-intensive applications. To allow the Grid computing architecture to function a set of standards have emerged that aim to provide seamless access to disparate computing and storage resources. This thesis concentrated on the requirements of the LHC*b* experiment being built at CERN, which will produce 500TB of RAW physics data each year of operation. To analyse this data LHC*b* have adopted a dataflow that splits processing across seven Tier-1s and using a large number of Tier-2 centres to generate simulated data.

The DIRAC project has evolved over that last four years to support all facets of the LHC*b* computing dataflow and workflow. The DIRAC Workload Management system has championed the *pull scheduling* paradigm using Pilot Jobs to mask the instabilities of the underlying Grid middleware and resources. This approach has now been accepted by several of the other LHC experiments as the mechanism for ensuring stability and managing community policy.

The DIRAC Data Management system design has been presented. The core of the system are the components that abstract the underlying resources. The main resources consumed by the data management system are the grid storage elements (offering the SRM interface) and replica catalogue. On top of these components a set of services and agents have been developed to fully support the logical dataflow outlined in the Computing

Model. A key component of this is the bulk transfer system which uses a central scheduler to implement intelligent replication strategies. A system for performing data-driven replication of production files has been developed to automatically distribute data according to LHC*b* policy for different data types. A integrity and accessibility of LHC*b*'s data is ensured by maintaining the mutual consistency of all the catalogues and grid resources used. The mechanisms employed by the Data Management system to ensure resilience to problems with the resources were presented. The distributed failover mechanism has proved so successful that it is now employed by all DIRAC components to relay messages in the event of services being unavailable.

The client interface to the LFC was tested and supports the tested operations far in excess of the needs of LHC*b*. The initial file registration rate required was exceed by 30% and all other tested functionlities exceeded requirements by over 2 orders of magnitude. The SRM interface to the seven LHC*b* Tier-1s was tested showing significant variation in performance across the SRM implementations and for different functionality. The requirement to issue prestage requests and to remove files exceeded requirements by 2 orders of magnitude. The ability to obtain access to data through the SRM was observed to be problematic with NIKHEF achieving only half the required rate and exhibiting a 30% error rate. Significant efforts have been made by the community to increase the stability and scalability of Grid storage systems on the whole, but this still remains a problematic issue.

The bulk file transfer mechanism facilitates the distributed processing of LHC*b* data and has shown to provide both the brute force throughput required and the ability to perform intelligent scheduling, using past network performance, to create replication trees. The transfer framework commissioning exercise demonstrated peak throughput rates of over 1GB/s and sustained rates 2.4 times the requirement from the Computing Model. While the aggregate requirements were met specific metrics of site performance were not met at NIKHEF. The centralised scheduler allows intelligent scheduling to be performed and the Prim-Jarnik algorithm was found to provide more efficient use of network resources and increased performance. It was shown to provide a 50% reduction in total transfer times compared to a simple multi-unicast approach and 33% reduction compared to the propri-

etary Min-Sigma algorithm. The application of intelligent scheduling also allows to redistribute the Computing Model network requirements alleviating the load from problematic sites.

The generation of optimal replication strategies is a fertile ground for further research. The parameter space to investigate using the Prim-Jarnik algorithm is significant, but will primarily concentrate around an investigation of the δ parameter to model the added cost of making multi-hop transfers and the resulting spanning tree performance.

During September 2009 the first beam was observed at the LHC providing LHC*b* with its first data. This data was transferred to MSS at CERN using the mechanisms presented here. Since then, detector commissioning has continued to produce cosmic ray data that is being managed by DIRAC daily. This has provided the first taste of LHC data, and with first collisions approaching, mechanisms to ensure the scalability and use-ability of DIRAC are being extended. The datasets architecture presented here will be made public to users along with a mechanism to manage user grid storage quotas. The coming months will see the LHC return to operation bringing first collisions to the experiments. This will begin the *data deluge* for which all members of the LHC*b* collaboration have been waiting.

Appendix A

SRM Evolution

The standardisation of the interface to grid storage is the key current effort in storage management. The Storage Resource Manager (SRM) interface is the result of this effort. The design considerations first proposed when the SRM specification was created are reviewed here, followed by an overview of the evolution of the specification from the first version to version 2.2 which is being used within LCG.

A.1 Key Concepts

The SRM interface provides uniform access to heterogeneous storage resources on the grid. When the concept of Storage Resource Managers was first discussed the interface was very simple but the imagined functionality that could be performed by the SRM was significant. The fundamental idea behind SRM was the ability for a grid storage system to dynamically manage space. The core concepts in the early discussions were *file pinning* and *space reservation*, *file types* and *space types*.

File pinning is necessary for dynamic management of shared disk caches. It allows clients to inform the SRM of the expected lifetime of a file and how long it should be available. This could be used by the SRM to garbage collect old files as their pins expired. A pin lifetime associated to a file that is only on disk cache implies that this file is temporary. This introduced the need to define file types. Three file types were proposed: *volatile*, *durable* and *permanent*.

Volatile files are temporary and have a pin with an associated lifetime. These files may be removed by the SRM once the pin expires and are effectively managed by the SRM. Permanent files are to be kept forever. Since the SRM can not remove these files they are *owned* by the client. The durable file type is a hybrid of the previous two with an associated lifetime but could not be removed by the SRM after the expiration of the lifetime. The use case for this file type was to allow sharing of temporary resources while protecting important files from automatic deletion.

Space reservation was required to allow a client to dynamically reserve space with different characteristics and lifetimes. To allow the SRM to manage its resources a space type and lifetime, like the pin lifetime, was associated to the reservation. The arguments for defining three file types can equally be applied to spaces resulting in the definition of *volatile*, *durable* and *permanent* spaces. Similar to the arguments for volatile and permanent file types, the need for volatile and permanent space is immediately apparent. Volatile space has a lifetime associated with it. The SRM can reclaim the space and remove the files contained within it once this expires. Permanent is managed by the client and can not be recovered by the SRM. Durable space is client owned temporary space guaranteed by the SRM. Durable space, like durable files, are client managed, but have an associated lifetime. After the expiration of the lifetime the SRM must inform the owner of the space.

When the lifetime of a volatile space expires the SRM can release the space, removing all the files contained in the space. To ensure that all the files in a volatile space can be removed as the space lifetime expires files cannot be pinned for a lifetime longer than the lifetime of the space. When the space lifetime expires the SRM knows the files contained have also expired. With similar logic, durable files may only be placed in durable or permanent space and permanent files can only be stored in permanent space. Volatile files, because of their temporary nature, can be placed in any of the three space types. This is shown in Table A.1.

The ability for clients to reserve space with a given type and (if appropriate lifetime) is core to the principles of SRM. From the client's perspective

File Types	Space Type		
	Volatile	Durable	Permanent
Volatile	Yes	Yes	Yes
Durable	No	Yes	Yes
Permanent	No	No	Yes

Table A.1: Files Types Allowed in SRM Space Types.

this reservation would be guaranteed at the requested size for the lifetime of reservation. This is known as *guaranteed* reservation model. An alternative is a *best-effort* system, where the space is allocated to the client as it is claimed. With this system the initial reservation is advisory which the SRM tries to honour within the reservation time. With a best-effort system the SRM can optimise the use of its resources by allocating space only to those who utilise their reservations. A summary of the properties of different space types can be seen in Table A.2.

Features	Space Type		
	Volatile	Durable	Permanent
Lifetime applies	Yes	Yes	No
Can SRM reclaim?	Yes	No	No
Best-effort	Always	Possible	Possible

Table A.2: SRM Space Type Properties [177].

The motivation for different space and file types was to allow SRM to conduct the management of its resources dynamically. As will be seen in Section A.3 this has not happened in real world applications.

A.1.1 LFNs, PFNs, Source URLs and Transfer URLs

Each unique file on the grid has a globally unique Logical File Name (LFN). An example LFN, the first RAW file written by LHCb is:

/lhcb/data/2008/RAW/LHCb/BEAM/32438/032438_0000081608.raw

Each replica of this file, stored on different storage resources, will have a different Physical File Name (PFN). To allow clients to locate replicas of a file on the grid, a replica catalogue is required to map LFNs to their

corresponding PFNs.

Files on the grid are also represented by a Source URL (SURL). This SURL is made up of the protocol and the PFN. Clients wishing to access a file may use a SURL to begin discussion with the SRM managing the file. Once a client has requested the SRM to pin a file and the protocol negotiation has taken place the SRM returns a transport URL (tURL). This tURL, contains the negotiated transfer protocol followed by the PFN.

A.2 Protocol Development

The Storage Resource Manager Working Group first discussed the functionality and common operations that SRM should support in October 2001. This led to the preparation of the Common Storage Manager Operations [178]. This specification covers basic functionality to allow files to be placed and retrieved from the SRM, to pin and unpin these files, to delete and to archive files. The specification also allowed clients to access request information, file metadata, transfer protocols supported and the estimated time for requests to complete.

This specification document was subsequently reviewed and additional functionality, such as third party SRM copies, termination of part or all of requests and suspension and resumption requests were added. This Joint Functionality Design [179] document detailed the SRMv1.1 specification. The methods outlined in this protocol version are discussed briefly below.

A.2.1 SRM Version 1.x

This version contained two main categories of function: Transfer and Status. These functions are largely superseded by the SRMv2.x protocols which will be discussed in more detail in Section A.2.2 and to avoid redundancy the SRMv1.1 functions are summarised in Table A.3 and A.4 with a brief description of their operation.

Function	Action
Get()	Get file from SRM cache
Put()	Request space and perform put
Pin()	Pin file
Unpin()	Unpin file
MkPermanent()	Make file of 'permanent' type
setFileStatus	Update status by client
AdvisoryDelete()	Advises to delete file (not accessed again)

Table A.3: SRMv1.1 Transfer Functions and Their Operation.

Function	Action
getRequestStatus()	Returns info for files in request
getFileMetaData()	General file info: location and status
getEstGetTime()	Best possible time (remaining) for get request
getEstPutTime()	Best possible time (remaining) for put request
getProtocols()	Get protocols SRM supports

Table A.4: SRMv1.1 Status Functions and Their Operation.

A.2.2 SRM Version 2.x

The second version of the SRM protocol was developed to extend and refine the methods proposed in SRMv1.1. The development of the SRM v2.x protocol began with v2.0[180]. This early version contained the 'Transfer' and 'Status' functions contained in v1.x as well as the addition of four new functions. These functions were added to allow clients to obtain additional details on their requests and increase the allocated lifetime of file pins.

This starting point was greatly expanded with the completion of v2.1[181]. This version included completely new Space Management, Permission and Directory functions as well as re-defining the previous Transfer and Status functions. A further version, 2.1.1[182], was finalised in March 2004 which contained an additional, optional parameter for a single function. For this reason the two protocol versions can be assumed to be all but the same. The SRMv2.x functionality can be split into five categories: *Transfer*, *Space Management*, *Directory Management*, *Permission* and *Status* functions. Appendix B has enumerated the SRM2.x functions.

Changes Since 1.x

In the development of the SRMv2.1.1 interface specification the previous version was modified and expanded. In addition to the new functions implemented existing functionality were refined or altered. Therefore version 1.x does not exist as a subset of the version 2.x specification. Many current SRM implementations use version 1.x as the interface and it is expected that as SRMs develop they will support both v1.x and v2.x. It is therefore important to appreciate the difference between the two specifications to avoid unpredictable behaviour when submitting requests to SRMs.

Additions

The addition of the Space, Directory and Permission Management functions are the main additions since SRMv1.x. These have allowed many of the important functionality desired of SRMs.

Alterations

The Get and Put call of the v1.1 have been updated to srmPrepareToGet and srmPrepareToPut to reflect the role of the SRM as separate from that of the file transfer service. Additional parameters were added to these commands and similarly with the new srmCopy function.

The ability to change the type of a file has been advanced from a single MkPermanent operation to support a changing of files to any type using srmChangeFileStorageType. The AdvisoryDelete function has been replaced by srmRemoveFile and the setFileStatus function used to update the SRM on the status of a Put operation has been replaced with srmPutDone.

The getRequestStatus function has been superseded by a number of new status requests: srmGetRequestSummary and srmStatusof{Get, Put, Copy}Request while getFileMetadata has been replaced by srmLs.

An explicit Unpin operation has been renamed srmRelease, but is often carried out automatically by the SRM.

Removals

Several operations have been removed from the specification completely.

The Pin operation is now carried out automatically by the SRM and the requirement to call the Pin operation to update a file lifetime has been placed with srmExtendedFileLifeTime. The getEstGetTime and getEstPutTime have been removed as the information required is now returned as part of srmStatusOfGetRequest and srmStatusOfPutRequest. The getProtocols function has been removed completely and is now expected to be obtained by a grid information service.

A.2.3 Additions in SRM Version 2.2

The definition of the SRM2.2 specification [76] was required because some clear use cases could not be clearly expressed in the previous nomenclature. The description of files as being permanent, durable or volatile says nothing about the medium on which they are stored. For the LHC experiments two additional properties of files were important: a measure of the time to access the file and the media on which a file is stored. The LHC experiments are the largest group of SRM users and their request lead to the definition of two additional file and space parameters: access latency and retention policy. The access latency defines a file as being either ONLINE, NEARLINE or OFFLINE. The retention policy is either REPLICA, OUTPUT or CUSTODIAL.

In principle, any combination of these properties can be supported, but in practise only three are used:

- ONLINE,REPLICA - the file is on disk and is not migrated to tape
- ONLINE,CUSTODIAL - the file is on disk and is migrated to tape
- NEARLINE,CUSTODIAL - the file is not on disk and is migrated to tape

A.3 SRM in LCG

The interface to storage in LCG is SRM. A reduced version of the SRM2.2 interface is supported by all of the SRM implementations deployed.

The rich schema of file and space types has not been exploited and all space reservations for the experiments are permanent. In addition, all files uploaded to the SRM are considered permanent. The workflows of the LHC experiments are well defined with specific storage requirements to support them. The resources are provided by the sites are associated directly to the experiments and practically may be used only by them. In this way the management of the space is in the hands of the experiments but removed the ability of the SRM to dynamically share and manage resources.

The concept of pinning is supported by the LCG SRM implementations, but has a reduced scope. The pin lifetime is used to instruct the storage element to retain the file on disk for the period of the pin. This pin lifetime is only relevant for files which are on custodial, tape storage as if they are only on disk they can not be removed by the SRM.

A.4 Summary

The original concept of a Storage Resource Manager was to allow dynamic management of storage resources on the Grid. To support this a number of space types and files types were defined that would allow the SRM to create space from data no longer being used. The SRMv2 interface defined a full set of methods for file, space, directory, permission management on top of the existing transfer and status functions. In real world usage, where storage resources are pledged explicitly to the clients, the rich functionality provided is not required and the SRM interface is primarily used for the file, directory, transfer and status functions.

Appendix B

SRM 2.x Commands

B.1 Data Transfer Functions

The main requirement of a SRM is to allow grid clients to utilise storage and to obtain files at a later point. There are ten functions in the data transfer category three of which will be discussed individually here: srmPrepareToGet, srmPrepareToPut and srmCopy. The other functions will also be discussed briefly.

To allow the transfer of files from a SRM to local system a client can use **srmPrepareToGet()**. The file must be *pulled* from the SRM, to avoid firewalls blocking the SRM from *pushing* the files into the local storage, and because the SRM does not know whether enough space is available to receive the file. The SRM prepares the files to be transferred, pinning them and providing the client with the relevant transfer URLs, so that the file transfer service can be invoked to perform the transfer. For this reason this function was renamed to 'PrepareToGet' since SRMv1.1. The function has been designed to be *asynchronous* and non-blocking. To obtain status information client can poll the SRM using a *request-token* supplied by the SRM. Once the SRM has pinned the requested files or directories it returns the transfer URL. Once the files have been copied the client should issue a release call to allow the space to be released.

Inversely to transfer files into an SRM space from a local system **srmPrepareToPut()** is used. The function performs space management op-

erations to prepare the space receiving the file. Similar firewall problems mean that the transfer of files is performed in *push* mode. The client supplies the *space-token* for the reserved space, file type, desired lifetime (if file is volatile/durable) and the size of file. The SRM returns the TURL as well as the future SURL of the file. Once each file transfer in the request is complete the SRM expects the client to issue a **srmPutDone()** function.

To allow clients to third party copy, the client can use **srmCopy()** with which the client must supply the source and target SURLs. These define whether the transfer is made in pull or push mode, so as to always give control to the SRM to which the request is made. For both types of request the client can flag whether the source files are to be removed on completion of the copy. On completion the SRM returns an array containing the status of the requested files/directories.

As well as transferring data it must be possible to control these requests. By supplying the request token to the SRM a client can abort a single file in a multi-file request using **srmAbortFiles()** or abort a complete request with **srmAbortRequest()**. The client can use the **srmSuspendRequest()** and **srmResumeRequest()** functions to pause and restart requests by supplying the request token to the SRM.

The **srmRemoveFiles()** function can be used by clients to remove files from an SRM by supplying the file SURLs and the request token issued when the files were obtained. If the files are not to be completely removed from the SRM, but only released then the **srmReleaseFiles()** function can be used. This releases all the files associated with a request token. If only a subset of the associated files are to be released the SURLs of these files can be optionally supplied with the function.

B.2 Space Management Functions

A SRM supporting the version 2.x functionality must support the ability to reserve and manage space. To reserve space, important for grid scheduling, the **srmReserveSpace()** function is used. This call should contain the type and size of the space desired. In response the SRM returns the type,

guaranteed size, total size and a space token.

The lifetime and size of a space can be requested to be changed using the **srmUpdateSpace()** function. It is also possible to request to change the type of storage for individual files or directories. This **srmChangeFileStorageType()** function takes the SURL path and the desired storage type for the file/directory. This operation may require the SRM to manage the user space allocations and is therefore deemed a space management function.

Space can also be released with **srmReleaseSpace()**. Which allows a Boolean *forceFileRelease* parameter to release pinned files present in the space. A client can also use a **srmCompactSpace()** function to remove released/unpinned files in their durable and permanent allocation to create space. An optional Boolean *doDynamicCompactFromNowOn* parameter can be defined so that in the future files are removed from the SRM as soon as they are released.

Information about space reservations can be obtained using the **srmGetSpaceMetaData()** call which returns all metadata pertaining to the space. In the event that a client loses the space token corresponding to a space they can use the **srmGetSpaceToken()** function using a space token description supplied with initial reservation. If no description is supplied the SRM returns all tokens associated with the client.

B.3 Permission Functions

The permission functions allow clients to view and change the access permissions for files stored at remote and local sites. The **srmCheckPermission()** function allows clients to check their access permissions for a set of supplied SURLs which can be local or remote. It is possible for the owner of a file or directory to pass ownership to another user with the **srmReassignToUser()** function supplying the SURL, the new user and the lifetime of assignment. The new user has the lifetime of the assignment to copy the file into their space before the file is removed.

An optional function in the 2.x protocol allows clients to add, remove or change file or directory permissions. In the **srmSetPermission()** function the client can supply the corresponding SURLs along with the UNIX type permission they wish to add, remove or change for individual users or groups.

B.4 Directory Functions

Directory manipulation of reserved space using various UNIX type functions is supported. Making directories, **srmMkdir()**, removing directories, **srmRmdir()** and removing files, **srmRm()**, by supplying the directory or file SURL is possible. The UNIX like **srmLs()** function allows the user to see the contents of a supplied SURL. Finally **srmMv()** function can be used to move files and directories by supplying the source and target SURLs but maybe subject to permissions in source and target spaces.

B.5 Status Functions

The final set of functions are the status functions that allow clients to check the status of their requests. The **srmStatusOfGetRequest()**, **srmStatusOfPutRequest()** and **srmStatusOfCopyRequest()** functions can be used with the request tokens to obtain information on all the files in a request. If only specific files are of interest the client can supply the SURLs of interest. The SRM returns an array containing the status of the requested files.

In addition to obtaining status information for files in a request it is possible to poll the SRM details of a submitted request using the **srmGetRequestSummary()** and supplying the request token. The SRM will then return all the metadata associated with the request type. In the event that the request token is lost it can be recovered by supplying a description of the request, supplied at the time of the initial request. If no descriptions are supplied with this **srmGetRequestID()** function all the request tokens associated with a client are returned. A final status function allows clients to reset the lifetime of their files with the **srmExtendFileLifeTime()** call.

Appendix C

RAW Upload Performance

The architecture employed for ensuring the integrity of RAW physics data was discussed Section 4.4.1. The first operation to be performed is the upload of the RAW data to the dedicated disk servers in Castor. The Computing Model outlines the requirement that during data taking RAW data is uploaded to Castor at a sustained rate of 60MB/s. To determine the configuration of the DIRAC services and agents that deliver the best performance a series of studies were conducted for various deployments. A single DIRAC instance was deployed on a single gateway machine[1], with access to the *online* and *offline* networks, and the number of transfer agents executing upload requests was varied. Each configuration was deployed for 24 hours and the read rate from the Online storage system was monitored at intervals of 2 seconds. The mean data read rate from the disk pools and the standard deviation are plotted against the number of executing agents in Figure C.1. The read rate from the disk pools increased with the number of clients until network saturation, approaching the 1Gb/s limit, occurred with four clients.

The online storage system was designed to allow concurrent high throughput reading and writing. Performance studies have shown it has surpassed its design goals [149]. To ensure the DIRAC deployment could also meet the 200MB/s peak rate required by LHC*b* a further test was performed with the deployment of an additional four executing agents on a further machine[2] with a dedicated 1Gb network connection to CERN. This configuration was able to saturate both dedicated network connections into Castor as seen in

[1] An eight core Intel Xeon 1.6GHz with 8GB of RAM.
[2] Another eight core Intel Xeon 1.6GHz with 8GB of RAM.

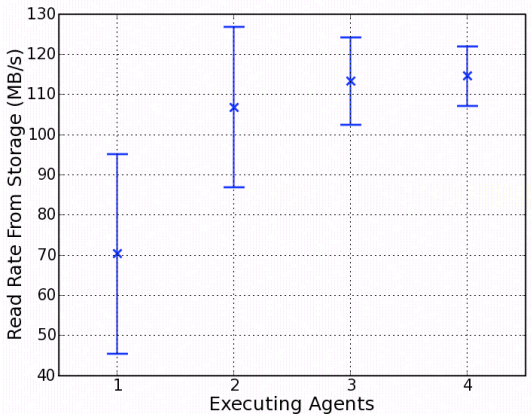


Figure C.1: Read rate from online storage with varying number of executing agents.

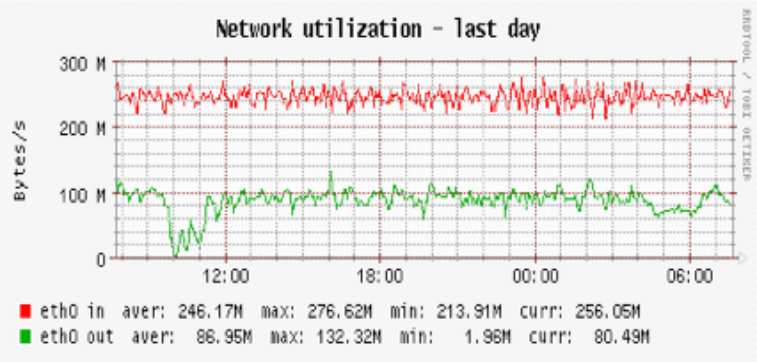


Figure C.2: Lemon network monitoring of Castor pools accessed by the DIRAC agents deployed at the gateway. The red line shows transfer activity into the pools from the online storage and the green line shows the transfers out to the tape servers.

Figure C.2.

During a preparatory computing exercise to mimic real data taking, conducted over the period of a month, the Online system fed 2GB files to DIRAC every 30s. This was performed for six hours followed by a period of six hours without data to simulate the duty period of the LHC. The transfer activity from the Online system over a period of three days is shown in Figure C.3. Over the duration of this exercise a total of 91,447 files, corresponding to 182TB, were uploaded to Castor.

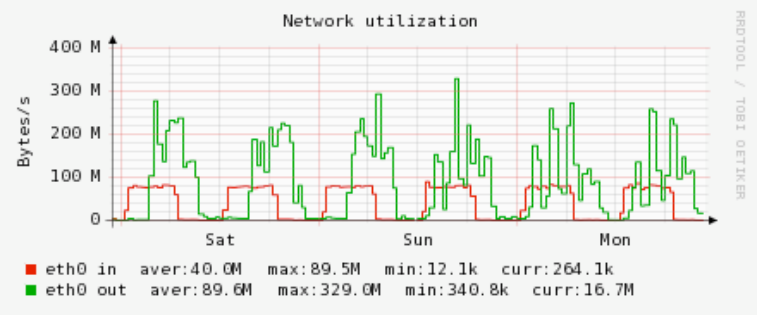


Figure C.3: Lemon network monitoring of Castor pools accessed by the DIRAC agents. The red line shows transfer activity into the pools for six hours followed by a six hour break and the green line shows the transfers out of the disk pools to the tape servers and the Tier-1s.

The integrity of all LHC*b* RAW files must be ensured before any processing or replication takes place. To ensure that the file can always be recovered it must be migrated to tape and the tape copy checksum verified against the checksum calculated at the first write by the online system. Therefore, the migration time dictates the timetable for the reconstruction of the RAW data. The file migration times are given in Figure C.4. After 130 minutes 50% of files have been migrated and by 6 hours 90% have been migrated. During the exercise several small outages of the network connecting the online system to Castor occurred causing backlog of files to accumulate. Once connectivity was repaired high sustained rates of transfers were observed resulting in a increased migration queue within Castor and the long tail observed in the histogram. In these cases all files were eventually migrated within a day.[3]

[3]During the entire period of the exercise no file failed checksum matching, suggesting such a strict integrity requirements may not be necessary. While the DIRAC Data Management System can ensure integrity of the file in Castor it knows nothing of the physics quality of the files. The real-time processing of data will not occur until the physics quality has been verified and therefore the decision was made to retain the thorough integrity checking mechanism.

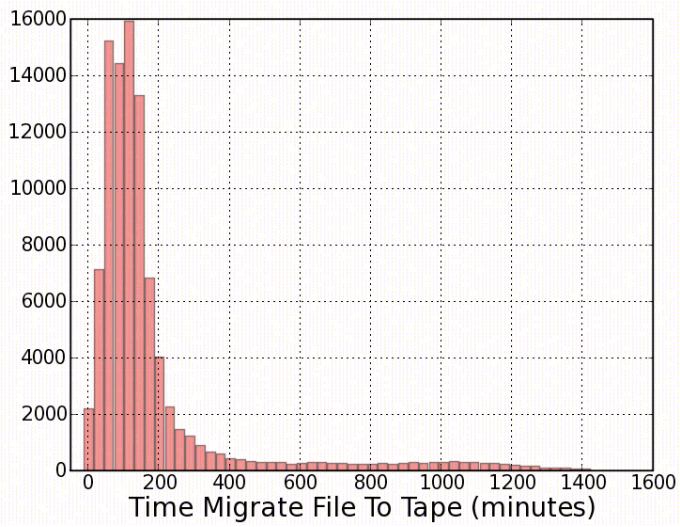


Figure C.4: Histogram of the time for files to be migrated to tape after being uploaded to Castor disk pools.

Appendix D

SRM Performance Results

This appendix presents the tabulated results obtained during the SRM and GFAL testing performed in Section 5.2. Four GFAL methods were tested (gfal_ls, gfal_prestage, gfal_turlsfromsurls and gfal_deletesurls) against seven storage elements at CERN, CNAF, GRIDKA, IN2P3, NIKHEF, PIC and RAL. Two Castor instances were tested (CERN, RAL), four dCache (GRIDKA, IN2P3, NIKHEF and PIC) and one StoRM (CNAF). The results for each of the tests are given in the following sections.

D.1 Retrieving File Metadata

The results in this section correspond the those presented in Section 5.2.1.

D.2 Issuing Prestage Requests

The results in this section correspond the those presented in Section 5.2.2.

D.3 Retrieving Transport URLs

The results in this section correspond the those presented in Section 5.2.3.

D.4 Removing Files

The results in this section correspond the those presented in Section 5.2.4.

Files	Metric	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	Mean (s)	0.28	7.48	2.41	6.03	0.96	5.58	0.56
	StdDev	0.48	46.94	2.62	40.58	2.38	36.24	0.56
	Median	0.20	0.28	1.65	0.47	0.63	0.74	0.42
	Suc. Rate	1.00	0.99	1.00	0.97	1.00	0.98	1.00
2	Mean (s)	0.38	4.94	3.80	2.56	1.29	4.47	0.52
	StdDev	1.16	23.45	4.00	10.46	2.64	25.15	0.64
	Median	0.26	1.30	2.50	0.63	1.00	0.93	0.41
	Suc. Rate	1.00	0.99	1.00	0.98	1.00	0.98	1.00
5	Mean (s)	0.48	7.14	8.67	3.66	2.39	6.20	0.62
	StdDev	0.22	27.79	9.32	13.16	2.14	32.21	0.55
	Median	0.44	2.24	6.60	1.33	2.18	1.93	0.51
	Suc. Rate	1.00	0.99	1.00	0.98	1.00	0.98	1.00
10	Mean (s)	1.09	12.91	16.11	5.84	4.27	6.79	0.95
	StdDev	2.22	41.67	19.56	20.54	1.03	24.81	1.11
	Median	0.75	3.78	11.69	2.51	4.15	3.50	0.72
	Suc. Rate	0.99	0.99	1.00	0.99	1.00	0.98	1.00
20	Mean (s)	1.54	12.11	29.47	8.17	8.27	8.81	1.46
	StdDev	0.88	25.50	26.86	24.02	2.88	22.66	1.49
	Median	1.35	6.48	22.52	4.59	7.99	6.57	1.12
	Suc. Rate	1.00	0.98	1.00	0.99	1.00	0.98	1.00
50	Mean (s)	3.67	21.20	69.92	14.32	19.94	16.22	2.53
	StdDev	3.19	35.54	63.86	11.59	2.57	16.83	1.52
	Median	3.20	14.17	56.04	11.56	19.65	15.13	2.23
	Suc. Rate	0.46	0.98	1.00	0.96	0.94	0.96	0.95
100	Mean (s)	11.93	35.61	117.31	29.45	39.62	30.05	4.79
	StdDev	32.49	29.51	74.94	30.32	5.74	17.26	3.09
	Median	6.24	33.49	101.67	22.40	38.85	29.69	4.10
	Suc. Rate	0.01	0.99	0.88	1.00	0.67	0.87	0.52

Table D.1: Mean, standard deviation and median of the retrieval time and the success rate for file metadata queries from site SRMs for a varying number of files using GFAL.

Files	Metric	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	Mean (s)	0.77	1.94	0.68	0.57	1.02	1.14	1.10
	StdDev	0.04	4.60	1.91	0.18	1.69	0.26	0.19
	Median	0.76	0.62	0.44	0.54	0.86	1.10	1.08
	Suc. Rate	1.00	0.99	1.00	1.00	1.00	1.00	1.00
2	Mean (s)	0.80	1.85	0.58	0.55	1.11	1.15	1.15
	StdDev	0.05	5.30	0.93	0.11	1.90	0.32	0.34
	Median	0.79	0.68	0.44	0.54	0.87	1.10	1.10
	Suc. Rate	1.00	1.00	1.00	1.00	0.99	1.00	1.00
5	Mean (s)	0.89	2.72	0.91	0.56	1.25	1.12	1.16
	StdDev	0.07	7.31	6.17	0.30	2.77	0.10	0.20
	Median	0.89	0.88	0.45	0.53	0.88	1.10	1.13
	Suc. Rate	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	Mean (s)	1.05	3.21	0.60	0.56	1.32	1.16	1.23
	StdDev	0.07	7.64	1.16	0.07	3.80	0.28	0.24
	Median	1.04	1.19	0.45	0.56	0.90	1.11	1.20
	Suc. Rate	1.00	1.00	1.00	0.99	0.99	1.00	1.00
20	Mean (s)	1.36	6.15	0.77	0.59	1.73	1.20	1.40
	StdDev	0.12	11.39	2.37	0.11	6.99	0.25	0.27
	Median	1.38	1.83	0.47	0.58	0.95	1.16	1.35
	Suc. Rate	1.00	1.00	1.00	0.99	0.99	0.99	1.00
50	Mean (s)	2.31	9.01	0.69	0.62	1.88	1.39	1.85
	StdDev	0.37	12.79	0.95	0.10	4.18	1.30	0.41
	Median	2.36	5.17	0.52	0.62	1.16	1.29	1.78
	Suc. Rate	0.00	0.00	0.94	1.00	0.82	0.97	0.84
100	Mean (s)	4.12	9.16	0.71	0.67	3.09	1.98	2.62
	StdDev	0.98	7.32	0.82	0.30	10.74	9.69	0.91
	Median	4.32	6.88	0.55	0.63	1.39	1.34	2.49
	Suc. Rate	0.96	1.00	1.00	1.00	0.99	1.00	1.00

Table D.2: Mean, standard deviation and median of the operation time and the success rate for pre-stage request to site SRMs for a varying number of files using GFAL.

Files	Metric	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	Mean (s)	9.25	19.48	38.80	28.82	2027.92	667.23	2.96
	StdDev	28.19	76.78	307.74	173.74	4807.79	3046.06	1.37
	Median	2.03	12.35	1.77	2.18	2.03	2.59	2.57
	Suc. Rate	0.98	0.84	0.97	0.98	0.82	0.96	0.99
2	Mean (s)	16.10	9.80	44.76	31.33	2903.27	406.71	8.22
	StdDev	117.50	22.43	447.07	128.38	5559.52	2239.28	81.16
	Median	1.91	3.58	1.62	12.07	1.91	2.44	2.54
	Suc. Rate	0.98	0.86	0.96	0.97	0.75	0.96	0.98
5	Mean (s)	9.21	11.99	7.56	78.30	3606.62	224.18	3.83
	StdDev	28.78	40.64	13.41	689.90	5888.32	1093.15	3.86
	Median	2.17	12.33	2.54	12.17	2.02	2.46	2.71
	Suc. Rate	0.98	0.89	0.95	0.97	0.66	0.93	0.98
10	Mean (s)	10.90	10.85	7.90	56.46	3444.36	397.07	4.28
	StdDev	30.08	17.30	13.32	364.23	5925.72	1772.79	4.35
	Median	2.67	12.43	11.70	12.24	2.00	2.48	3.01
	Suc. Rate	0.98	0.87	0.96	0.97	0.65	0.88	1.00
20	Mean (s)	24.98	12.49	19.48	65.69	2770.15	252.00	20.02
	StdDev	146.80	22.52	176.99	503.93	5271.50	1784.22	231.98
	Median	6.34	12.51	11.75	12.30	2.12	2.55	3.63
	Suc. Rate	0.98	0.88	0.93	0.96	0.66	0.88	0.98
50	Mean (s)	3.28	14.22	11.84	151.57	2642.61	55.46	5.33
	StdDev	17.84	23.86	55.39	1059.93	5188.65	454.13	3.90
	Median	1.49	12.69	11.87	22.46	2.32	2.71	5.23
	Suc. Rate	0.00	0.89	0.87	0.97	0.64	0.83	0.69
100	Mean (s)	25.49	18.21	13.56	218.25	1943.07	124.38	18.51
	StdDev	24.18	68.90	29.53	1498.82	4574.24	741.99	5.01
	Median	21.16	12.83	12.00	22.64	12.81	3.25	18.56
	Suc. Rate	0.98	0.90	0.93	0.97	0.67	0.85	0.99

Table D.3: Mean, standard deviation and median of the retrieval time and the success rate for transport URL queries from site SRMs for a varying number of files using GFAL.

Files	Metric	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	Mean (s)	1.28	1.39	1.65	1.97	0.87	0.19	1.27
	StdDev	0.58	1.17	1.49	12.17	1.05	0.22	1.13
	Median	1.38	1.11	1.24	0.15	0.40	0.13	0.86
	Suc. Rate	0.99	0.99	1.00	0.94	1.00	0.91	0.99
2	Mean (s)	1.34	1.56	2.56	0.73	0.95	0.16	1.92
	StdDev	0.57	1.21	2.31	5.54	0.98	0.28	1.84
	Median	1.37	1.37	1.67	0.16	0.47	0.11	1.15
	Suc. Rate	0.96	0.96	0.98	0.90	0.91	0.91	0.98
5	Mean (s)	1.46	3.06	4.60	0.36	1.39	0.31	2.38
	StdDev	0.64	1.44	2.96	0.49	1.30	0.37	1.80
	Median	1.53	3.01	4.08	0.20	0.71	0.14	1.88
	Suc. Rate	0.95	0.93	0.98	0.77	1.00	0.93	0.96
10	Mean (s)	1.72	3.84	4.91	1.06	6.79	0.44	3.21
	StdDev	0.57	1.64	3.07	6.83	7.08	0.55	2.24
	Median	1.71	4.04	4.39	0.26	3.64	0.20	2.61
	Suc. Rate	0.99	0.91	0.99	0.69	0.44	0.98	0.99
20	Mean (s)	2.17	4.79	13.52	2.89	10.99	0.55	4.26
	StdDev	0.56	1.93	6.29	14.17	9.13	0.56	2.20
	Median	2.15	5.07	13.68	0.58	7.90	0.30	4.17
	Suc. Rate	0.98	0.80	0.99	0.43	0.95	0.93	0.99

Table D.4: Mean, standard deviation and median of the removal time and the success rate for 1 byte file removal requests from site SRMs for a varying number of files using GFAL.

Files	Metric	CERN	CNAF	GRIDKA	IN2P3	NIKHEF	PIC	RAL
1	Mean (GB/s)	1.04	6.52	3.23	7.39	5.03	4.89	4.87
	StdDev	0.46	3.05	2.55	2.23	1.04	3.32	0.75
	Median	1.01	7.77	2.31	7.51	5.15	4.70	5.00
	Suc. Rate	0.97	0.94	1.00	0.99	1.00	0.99	1.00
2	Mean (GB/s)	2.09	10.01	5.44	16.94	10.17	10.73	8.80
	StdDev	0.80	5.83	4.53	4.64	2.05	3.86	2.02
	Median	2.05	11.31	3.81	17.92	10.34	10.97	9.44
	Suc. Rate	0.98	0.95	1.00	0.99	1.00	0.99	1.00
5	Mean (GB/s)	5.58	4.21	8.90	39.05	22.65	25.45	18.76
	StdDev	2.59	2.29	7.27	11.07	5.80	14.44	2.84
	Median	5.27	4.69	6.46	41.92	23.70	25.79	19.32
	Suc. Rate	0.99	0.95	1.00	0.99	1.00	0.98	1.00
10	Mean (GB/s)	11.94	5.56	10.96	63.11	39.31	45.54	27.91
	StdDev	5.01	3.62	6.30	21.20	12.48	36.45	5.07
	Median	11.00	5.93	9.34	69.41	43.66	45.43	29.43
	Suc. Rate	0.98	0.96	1.00	0.99	1.00	0.98	1.00
20	Mean (GB/s)	18.57	7.23	10.47	84.38	50.17	70.33	40.23
	StdDev	6.90	5.91	5.46	30.29	15.95	60.64	6.41
	Median	17.42	7.26	9.43	94.07	56.40	71.09	41.96
	Suc. Rate	0.99	0.94	1.00	0.98	1.00	0.97	1.00

Table D.5: Mean, standard deviation and median of the removal rate and the success rate for 2GB file removal requests from site SRMs for a varying number of files using GFAL.

Bibliography

[1] LHCb Collaboration. LHCb Technical Proposal. Technical Report CERN/LHCC-98-004, CERN, (1998)

[2] LHCb Collaboration. LHCb Magnet TDR. Technical Report CERN/LHCC-00-007, CERN, (2000)

[3] LHCb Collaboration. LHCb VELO TDR. Technical Report CERN/LHCC-01-011, CERN, (2001)

[4] LHCb Collaboration. LHCb Outer Tracker TDR. Technical Report CERN/LHCC-01-024, CERN, (2001)

[5] LHCb Collaboration. LHCb Inner Tracker TDR. Technical Report CERN/LHCC-02-029, CERN, (2002)

[6] LHCb Collaboration. LHCb RICH TDR. Technical Report CERN/LHCC-00-037, CERN, (2000)

[7] LHCb Collaboration. LHCb Calorimeter TDR. Technical Report CERN/LHCC-00-036, CERN, (2000)

[8] LHCb Collaboration. LHCb Muon System TDR. Technical Report CERN/LHCC-01-010, CERN, (2001)

[9] LHCb Collaboration. LHCb Trigger TDR. Technical Report CERN/LHCC-03-031, CERN, (2003)

[10] LHCb Collaboration. LHCb Online System TDR. Technical Report CERN/LHCC-01-040, CERN, (2001)

[11] First telegraphic message, S.Morse, The Samuel F. B. Morse Papers at the Library of Congress, (1844)

[12] Toward A Cooperative Network Of Time-Shared Computers, Fall Joint Computer Conf, AFIPS Conf., T. Marill and L. Roberts, (1966).

[13] Multiple Computer Networks and Intercomputer Communication, ACM Conference, L. Roberts, (1967)

[14] Computer network development to achieve resource sharing, Spring Joint Computer Conf., AFIPS Conf., L.G. Roberts, B.D. Wessler, (1970)

[15] BBN Report 1822: Specification for the Interconnection of a host and and IMP, Bolt Beranek and Newman, Inc., (1969)

[16] ARPANET Completion Report, F. Heart et al., (1978)

[17] A File Transfer Protocol, Internet Engineering Task Force RFC 114, Abhay Bhushan, (1971)

[18] A Protocol for Packet Network Interconnection, IEEE Trans on Comm, R. Kahn and V. Cerf, (1974)

[19] The Internet Protocol, Internet Engineering Task Force RFC 791, (1981)

[20] Transmission Control Protocol, Internet Engineering Task Force RFC 793, (1981)

[21] File Transfer Protocol (FTP), Internet Engineering Task Force RFC 959, J. Postel and J. Reynolds, (1985)

[22] Domain Names - Concepts and Facilities, Internet Engineering Task Force RFC 882, P. Mockapetris, (1983)

[23] Domain Names - Implementation and Specification, Internet Engineering Task Force RFC 883, P. Mockapetris, (1983)

[24] Domain Names - Concepts and Facilities, Internet Engineering Task Force RFC 1034, P. Mockapetris, (1987)

[25] Domain Names - Implementation and Specification, Internet Engineering Task Force RFC 1035, P. Mockapetris, (1987)

[26] Information Management: A Proposal, CERN Doc, T. Berners-Lee, (1989)

[27] Hypertext Markup Language - 2.0, Internet Engineering Task Force RFC 1866, T. Berners-Lee and D. Connolly, (1995)

[28] Universal Resource Identifiers in WWW, Internet Engineering Task Force RFC 1945, T. Berners-Lee et. al, (1994)

[29] Hypertext Transfer Protocol – HTTP/1.0, Internet Engineering Task Force RFC 1945, T. Berners-Lee et. al, (1996)

[30] Hypertext Transfer Protocol – HTTP/1.1, Internet Engineering Task Force RFC 2616, T. Berners-Lee et. al, (1999)

[31] World Wide Web Consortium, http://www.w3.org

[32] W3C Web Service Glossary, http://www.w3.org/TR/ws-gloss

[33] Extensible Markup Language (XML) 1.0, W3C Recommendation, T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau,(2006)

[34] SOAP Version 1.2, W3C Recommendation, M. Gudgin, M. Hadley,N. Mendelsohn, J.J. Moreau, H.F. Nielsen, A. Karmarkar and Y. Lafon, (2007)

[35] Web Services Description Language (WSDL) 1.1, W3C Note, E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, (2001)

[36] FAFNER, http://www.lehigh.edu/~bad0/fafner.html

[37] Entropia: Architecture and Performance of an Enterprise Desktop Grid System, A. Chien et al., Journal of Parallel and Distributed Computing, (2003)

[38] SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM, D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, (2002)

[39] BOINC: A System for Public-Resource Computing and Storage, 5th International Workshop on Grid Computing, D. P. Anderson, (2004)

[40] Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment, IEEE Symposium on High Performance Distributed Computing, I. Foster, J.Geisler, J. Nickless, W. Smith and S. Tuecke, (1997)

[41] The Grid : Blueprint for a New Computing Infrastructure, Morgan Kaufmannm, I. Foster and C. Kesselman, (1998)

[42] Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, I. Foster and C. Kesselman, (1997)

[43] The Globus Alliance. http://www.globus.org.

[44] The Anatomy of the Grid: Enabling Scalable Virtual Organization, The International Journal of High Performance Computing Applications, I. Foster, C. Kesselman, S. Tuecke, (2001)

[45] Global Grid Forum: http://www.ggf.org

[46] Enterprise Grid Alliance: www.gridalliance.org

[47] Open Grid Forum: http://www.ogf.org

[48] What is a Grid? A Three Point Checklist, GRIDToday, I. Foster, (2002)

[49] The Grid 2 : Blueprint for a New Computing Infrastructure, Morgan Kaufmannm, I. Foster and C. Kesselman, (2004)

[50] A Different Perspective on the Question of What is a Grid?, GRIDToday, W.E. Johnson, (2002)

[51] Weaving Computational Grids: How Analogous Are They with Electrical Grids?, Computing in Science and Engineering, M. Chetty and R. Buyya, (2002)

[52] The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Global Grid Forum, I. Foster, C. Kesselman, S. Tuecke, (2002)

[53] Characterizing Grids: Attributes, Definitions, and Formalisms, Journal of Grid Computing, Z. Nemeth and V. Sunderam, (2003)

[54] Open Grid Services Infrastructure (OGSI) Version 1.0, S. Tuecke, K. Czajkowski, I. Foster, et al., (2003)

[55] Modeling Stateful Resources with Web Services, Whitepaper, I. Foster, J. Frey, S. Graham, S. Tuecke, K. Cza jkowski, D. Ferguson, F. Leymann, M. Nally, L. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, (2004)

[56] From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Evolution, Whitepaper, K. Cza jkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, (2004).

[57] Response to Ian Fosters "What is the Grid?, GRIDToday, W. Gentzsch, (2002)

[58] Storage Resource Broker - Managing Distributed Data in a Grid, Computer Society of India Journal, Special Issue on SAN, A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, SY Chen, R. Olschanowsky (2003)

[59] Amazon Simple Storage Service - http://aws.amazon.com/s3

[60] Blueprint for the Future of High Performance Networking, Communications of the ACM, M. Brown (2003)

[61] Job scheduling under the Portable Batch System, Job Scheduling Strategies for Parallel Processing, R. L. Henderson, (1995)

[62] TORQUE, http://www.clusterresources.com/products/torque

[63] Distributed Computing in Practice: the Condor Experience, Concurrency and Computation: Practice and Experience, D. Thain (2005)

[64] Platform LSF, http://www.platform.com/Products/platform-lsf

[65] Sun Grid Engine, http://gridengine.sunsource.net

[66] Fibre Channel Protocol for SCSI, The T10 Technical Committee, (2006)

[67] The InfiniBand Architecture Specification, The InfiniBand Trade Association, (2004)

[68] NFS: Network File System Protocol Specification, Internet Engineering Task Force RFC 1094, Sun Microsystems, (1989)

[69] NFS Version 3 Protocol Specification, Internet Engineering Task Force RFC 1813, B. Callaghan et al., (1995)

[70] Network File System (NFS) version 4 Protocol, Internet Engineering Task Force RFC 3530, B. Callaghan et al., (2003)

[71] Scalable, Secure, and Highly Available Distributed File Access, IEEE Computer, M. Satyanarayanan, (1990)

[72] An Introduction to GPFS Version 3.2, IBM White Paper, (2007)

[73] OGF GLUE Working Group, http://www.ggf.org

[74] GLUE Specification v2.0, OGF Public Comments Document, S. Andreozzi et al., (2008)

[75] OGF Grid Storage Management Working Group, http://www.ggf.org

[76] The Storage Resource Manager Interface Specification Version 2.2, OGF Grid Final Document 129, A. Sim, A. Shoshani et al., (2008)

[77] OGF GridFTP Working Group, http://www.ggf.org

[78] GridFTP: Protocol Extensions to FTP for the Grid, OGF Grid Final Document 20, W. Allcock, (2003)

[79] GridFTP v2 Protocol Description, OGF Grid Final Document 47, I. Mandrichenko, W. Allcock, T.Perelmutov, (2005)

[80] OGF OGSA ByteIO Working Group, http://www.ggf.org

[81] ByteIO Specification 1.0, OGF Grid Final Document 87, M. Morgan, (2007)

[82] OGF Grid File System Working Group, http://www.ggf.org

[83] The GGF Grid File System Architecture Workbook, OGF Grid Final Document 61, A. Jagatheesan, (2006)

[84] OGF OGSA Basic Execution Services Working Group, http://www.ggf.org

[85] OGSA Basic Execution Service Version 1.0, OGF Grid Final Document 108, I. Foster et al., (2007)

[86] Job Submission Description Language (JSDL) Specification, Version 1.0, OGF Grid Final Document 136, A. Anjomshoaa et al., (2008)

[87] Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, Internet Engineering Task Force RFC 3820, S. Tucke et al, (2004)

[88] The UK e-Science Program and the Grid, T. Hey, Proceedings of the 4th International Symposium on High Performance Computing, (2002)

[89] UNICORE A Grid Computing Environment, Concurrency and Computation: Practice and Experience, D.W. Erwin, (2001)

[90] Research data storage available to researchers throughout the U.S. via the TeraGrid, D. McCaulay et al.,Proceedings of the 34th annual ACM SIGUCCS conference on User services, (2006)

[91] The European HPC Infrastructure DEISA, ISC08, H. Lederer, (2008)

[92] CNGrid: a test-bed for Grid technologies in China, IEEE International Workshop on Future Trends of Distributed Computing Systems, Q. Depei, (2004)

[93] CNGrid Software 2: Service Oriented Approach to Grid Computing, Proceedings of EPSRC AHM, X. Xie, N. Xiao, Z. Xu, L. Zha, W. Li, H. Yu, (2005)

[94] Compute Canada, http://computecanada.org

[95] India's National Grid Computing Initiative (Garuda), http://www.garudaindia.in

[96] Open Science Grid, http://www.opensciencegrid.org

[97] Enable Grids for E-sciencE, http://www.eu-egee.org

[98] GriPhyN, http://www.griphyn.org

[99] PPDG, http://www.ppdg.net

[100] VDT, http://vdt.cs.wisc.edu

[101] Condor-G: A Computation Management Agent for Multi-Institutional Grids, Cluster Computing, M. Livny et al., (2002)

[102] Virtual Organization Management Across Middleware Boundaries, IEEE International Conference on e-Science and Grid Computing, V. Venturi et al, (2007)

[103] GridPP UK Computing for Particle Physics, http://www.gridpp.ac.uk

[104] INFN Grid, http://grid.infn.it

[105] Globus Toolkit Version 4: Software for Service-Oriented Systems, I. Foster, IFIP International Conference on Network and Parallel Computing, (2005)

[106] A National-Scale Authentication Infrastructure, IEEE Computer, R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, (2000)

[107] An Online Credential Repository for the Grid: MyProxy, International Symposium on High Performance Distributed Computing, J. Novotny, S. Tuecke, V. Welch, (2001)

[108] Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2, Proceedings of the International Workshop on Middleware Performance, X. Zhang and J.M. Schopf, (2004)

[109] GT4 GRAM: A Functionality and Performance Study, TeraGrid 2007, M. Feller, I. Foster, and S. Martin, (2007)

[110] The Globus Striped GridFTP Framework and Server, ACM/IEEE SC Conference, W. Allcock, J.Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Foster, (2005)

[111] Standardization Processes of the UNICORE Grid System, Austrian Grid Symposium, M. Riedel, D. Mallmann, (2005)

[112] Design and Implementation of NAREGI Problem Solving Environment for Large-Scale Science Grid, IEEE International Conference on e-Science and Grid Computing, H. Kanazawa, Y.Itou, M. Yamada, Y. Miyahara, Y. Hayase, S. Kawata, H. Usami, (2006)

[113] Worldwide Fast File Replication on Grid Datafarm, Y. Morita, S. Matsuoka, N. Soda, S. Sekiguchi and O. Tatebe, CHEP, (2003)

[114] Data Management on Grid Filesystem for Data-Intensive Computing, International Symposium on Applications and the Internet Workshops, H. Sato and S. Matsuoka, (2007)

[115] Advanced Resource Connector middleware for lightweight computational Grids, Future Generation Computer Systems, M. Ellert et al., (2007)

[116] Resource management and organization in CROWN grid, International Conference on Scalable Information Systems, J.Huai, T. Wo and Y. Liu, (2006)

[117] Open Middleware Infrastructure Institute, http://omii-europe.org

[118] NextGRID: Architecture for Next Generation Grids, http://www.nextgrid.org

[119] An Introduction to the NextGRID Vision and Achievements, NextGRID Architecture White Paper, (2008)

[120] Scalability and Performance Analysis of the EGEE Information System, L. Field et al., CHEP07, (2007)

[121] Distributed Policy Management and Comprehension with Classified Advertisements, University of Wisconsin (Madison) Computer Sciences Department Technical Report, M. Livny et al., (2003)

[122] The gLite Workload Management System, P. Andreettoo et al., CHEP07, (2007)

[123] Job Submission and Management Through Web Service: the experience witht he CREAM service, C. Aiftimiei et al., CHEP07, (2007)

[124] Recent Developments in LFC, S. Lemaitre et al., CHEP07, (2007)

[125] LHCb experience with LFC database replication, B. Martelli et al., CHEP07, (2007)

[126] DPM Status and Next Steps, J.P. Baud et al., CHEP07, (2007)

[127] GFAL and LCG-Util, R. Mollon et al., CHEP07, (2007)

[128] Building the WLCG file transfer service, G. McCance et al., CHEP07 (2007)

[129] Diagnostics and Prognostics on the Grid: the Distributed Aircraft Maintenance Environment project (DAME), Proceedings of EPSRC AHM, J. Austin, T. Jackson, M. Jessop, (2004)

[130] Medical Data Federation: The Biomedical Informatics Research Network, Chapter 8; The GRID2 Blueprint for a New Computing Infrastructure, M. Ellisman, S. Peltier, (2004)

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

[132] ATLAS Collaboration. ATLAS computing: Technical Design Report. Technical Report CERN-LHCC-2005-022, CERN, (2005)

[133] CMS Collaboration. CMS computing : Technical Design Report. Technical Report CERN-LHCC-2005-023, CERN, (2005)

[134] ALICE Collaboration. ALICE computing : Technical Design Report. Technical Report CERN-LHCC-2005-018, CERN, (2005)

[135] LHCb Collaboration. LHCb Computing TDR. Technical Report CERN/LHCC-05-119, CERN, (2005)

[136] AliEn - ALICE environment on the GRID, Nuclear Instrumemts and Methods, P. Saiz et al., (2003)

[137] Panda: Production and Distributed Analysis System for ATLAS, D. Kaushik et al., CHEP06, (2006)

[138] DDM Design and Implementation, CERN Technical Report, M. Branco et al., (2006)

[139] DDM Scenarios and Principles, CERN Technical Report, M. Branco et al., (2006)

[140] PhEDEx high-throughput data transfer management system, L. Tuura et al., CHEP06, (2006)

[141] LHCb software architecture group. GAUDI LHCb Data Processing Applications Framework. Architecture Design Document LHCb 98-064, CERN, (1998)

[142] Simulation Application for the LHCb Experiment, I. Belyaev et al. CERN Technical Report physics/0306035, (2003)

[143] The Tuning of Pythia for the LHCb, using Gauss, K. Lessnoff, LHCb-2007-125, (2007)

[144] EvtGen package homepage, http://www.slac.stanford.edu/*sim*lange/EvtGen

[145] Geant4 - A Simulation Toolkit, S. Agostinelli et al., Nuclear Instruments and Methods A, (2003)

[146] BaBar Collaboration, http://www.slac.stanford.edu/BFROOT

[147] Software for the LHCb Experiment, , IEEE Trans. Nucl. Sci., (2006)

[148] DaVinci for Busy People. Generic selection algorithms : a user guide, P. Koppenburg et al., CERN/LHCb-2005-016, CERN, (2005)

[149] S. Cherukuwada et al., A High-Performance Storage System for the LHCb Experiment. IEEE NPSS Real Time (2007)

[150] LHCb Collaboration. LHCb Computing TDR. Technical Report CERN/LHCC-05-119, CERN, (2005)

[151] 2009 Pledged Resources, http://lcg.web.cern.ch/lcg/planning/phase2_resources, Verified 17 November 2008.

[152] DIRAC, Distributed Infrastructure with Remote Agent Control, A. Tsaregorodtsev et al., CHEP03, (2003)

[153] Architectural Roadmap Towards Distributed Analysis - Final Report, P. Buncic et al., Technical report CERN-LCG-2003-033, CERN, (2003)

[154] Results of the LHCb Data Challenge 2004, J. Closier et al., CHEP04, (2004)

[155] DIRAC: Workload Management System, A. Tsaregorodtsev et al., CHEP04, (2004)

[156] DIRAC Review Report, J. P. Baud et al., Technical Report LHCb-06-004, CERN, (2006)

[157] The LHCb Computing Data Challenge DC06, R. Nandakumar et al., CHEP07, (2007)

[158] LHCb Data Replication During SC3, A. C. Smith et al., CHEP06, (2006)

[159] DIRAC, the LHCb data production and distributed analysis system, A. Tsaregorodtsev et al., CHEP06, (2006)

[160] DIRAC Production Manager Tools, G. Kuznetsov et al., CHEP06, (2006)

[161] Experience with distributed analysis in LHCb, S. Paterson et al., CHEP06, (2006)

[162] DIRAC Infrastructure for Distributed Analysis, S. Paterson et al., CHEP07 (2007)

[163] DIRAC Security Infrastructure, A. Casajus Ramo et al., CHEP06, (2006)

[164] Extension of the DIRAC workload-management system to allow use of distributed Windows resources, Y. Y. Li et al., CHEP07, (2007)

[165] DIRAC Framework for Distributed Computing, R. Graciani et al., CHEP07, (2007)

[166] DIRAC Agents and Services, A. Casajus Ramo et al., CHEP07, (2007)

[167] DIRAC Optimized Workload Management, S. Paterson et al., CHEP07, (2007)

[168] Conclusions from SC4 workshop Data Management, SC4 WLCG Service Workshop, (2006)

[169] R. Stoica, A. C. Smith et al., Data Handling and Transfer in the LHCb Experiment, IEEE Real Time, (2007)

[170] O. Barring et al., CASTOR2: design and development of a scalable architecture for a hierarchical storage system at CERN. CHEP07, (2007)

[171] A Universally Unique IDentifier (UUID) URN Namespace, Internet Engineering Task Force RFC 4122, P. Leach et al., (2005)

[172] Protocol Independent Multicast (PIM), http://tools.ietf.org/html/rfc4601

[173] Rapid Spanning Tree Protocol (RSTP), IEEE 802.1D-2004, (2004)

[174] V. Jarník, O jistém problému minimálním, Práce Moravské Přírodovědecké Společnosti (1930)

[175] R. Prim, Shortest connection networks and some generalizations, Bell Syst. Tech. J, (1957)

[176] J.Zurawski et. al., Logistical Multicast for Data Distribution, IEEE International Symposium on Cluster Computing and the Grid, CCGrid'05, (2005).

[177] SRM Joint Functionality and Interface Design, SRM Version 2.1, Arie Shoshani et al., (2002)

[178] Common Storage Resource Manager Operations, Ian Bird et al. (2001)

[179] SRM Joint Functionality Design, Summary of Recommendations, Ian Bird et al., (2002)

[180] The Storage Resource Manager Interface Specification Version 2.0, Ian Bird et al., (2002)

[181] The Storage Resource Manager Interface Specification Version 2.1 Final, Arie Shoshani et al., (2003)

[182] The Storage Resource Manager Interface Specification Version 2.1.1 Final, Arie Shoshani et al., (2004)