File Management for HEP Data Grids

Caitriana Mairi Macleod Nicholson

Department of Physics and Astronomy
University of Glasgow

Thesis submitted for the degree of
Doctor of Philosophy

March 2006

Abstract

The next generation of high energy physics experiments, such as the Large Hadron Collider (LHC) at CERN, the European Organization for Nuclear Research, pose a challenge to current data handling methodologies, where data tends to be centralised in a single location. Data grids, including the LHC Computing Grid (LCG), are being developed to meet this challenge by unifying computing and storage resources from many sites worldwide and distributing data and computing tasks among them. This thesis describes the data management components of LCG and evaluates the performance of the LCG File Catalogue, showing it to be performant and scalable enough to meet the experiments' needs. File replication can be used to improve a grid's performance by placing copies of data at strategic locations around the grid. Dynamic file replication, where replicas are created and deleted automatically according to some strategy, may be especially useful and so the grid simulator OptorSim was developed to investigate different replication strategies. Simulation of several grid scenarios, including LCG, shows that relatively simple replication strategies can lead to significant reductions in data access times and improved usage of grid resources, while a more complex economic model may be useful in future.

Gu mo theaghlach -

anns a h-uile àite.

He reached down from on high and took hold of me;
he drew me out of deep waters.
He rescued me from my powerful enemy,
from my foes, who were too strong for me.
They confronted me in the day of my disaster,
but the LORD was my support.
He brought me out into a spacious place;
he rescued me because he delighted in me.

Psalm 18:16-19 (NIV)

Acknowledgments

There is a long list of people whose help in this work must be acknowledged, and without whom it would never have been completed. First, I would like to thank my supervisor, Prof. Tony Doyle, for his unstinting support and advice over the years and especially in the last six months. Thanks also to my second supervisor, Dr Rick St. Denis, to Prof. David Saxon for the opportunity to work in the Experimental Particle Physics group, and to PPARC for providing the funding.

Much of the work presented here was done in collaboration with the European DataGrid optimisation team, and each member deserves my thanks for all I learnt from them: David, Paul and Will at Glasgow and Kurt, Rubén and Floriano further afield. Special thanks to David for leading the way and showing how a thesis on grid technology should be written!

All my colleagues in the PPE group deserve mention. Chris, Helen, Tom and Steven were a tremendous help in so many ways, as well as being great office-mates, past and present. Stan helped me get the last bit of computing power I needed for my results, and Dan, Kenny, Christian and others generously let me plunder their machines too. Alison was a wonderful comrade-in-arms during the last few months of writing up, and encouraged me to the very end.

Thanks to all in the LCG Grid Deployment group at CERN for making me so welcome there: to Dr Ian Bird for allowing me to work in the group, and especially to James, Sophie and Jean-Philippe for their help, teaching and great patience. Thanks also to all the friends in Geneva who made my stay there what it was: Alison and Richard, Paul and Marion, Ole, Judit, Apollo, Leslie and Gertrude, Grace, Obi, Graeme and Meena.

My friends in Glasgow were always there, in good times and in bad. Special thanks to my past and current flatmates - Siew Chin, Alyson, Suba, Janet and Lin Lee - and to everyone at St Vincent Street Free Church. To Finlay and Christine especially I owe more than I can say, as well as to Peter and Marion, Evan and Alison and many others - and of course Szu Shien, who is still going through the same experience! Finally, thanks to all my family, for whom a mere "thank you" will never be adequate.

Ceud mìle taing dhuibh uile!

Author's Declaration

This thesis represents work performed from 2002 - 2005 in the Experimental Particle Physics group in the Department of Physics and Astronomy at the University of Glasgow. The data management tools for the LHC Computing Grid described in Chapter 3 were developed by the European DataGrid, LCG and EGEE projects. The performance testing framework described in Chapter 4 was developed by myself and I obtained the results presented using this framework. The OptorSim grid simulator presented in Chapter 5 was developed in collaboration with colleagues in Glasgow, ITC-irst (Trento) and CERN; I contributed a significant fraction of effort to OptorSim development, especially during the last two years, including coding, testing, debugging and documentation. All the results presented in Chapters 7 and 8 are my own work.

Preface

This thesis examines some of the issues concerned with file management for high energy physics (HEP) experiments in a data grid environment, especially file replication. Chapter 1 looks at the history of computing for HEP, together with the history of distributed computing, before examining the requirements of experiments for the Large Hadron Collider (LHC) at CERN, the European Organisation for Nuclear Research and showing how a data grid is being developed to meet these requirements. Chapter 2 discussed in more detail the concepts and architecture of grid systems, while Chapter 3 describes the real implementation of a grid in the LHC Computing Grid (LCG).

There is a special focus on those components of LCG which are used for file and replica management, showing how these have developed in response to experiments' requirements. Chapter 4 focuses on one of these components, the LCG File Catalogue (LFC) and presents a number of tests evaluating its performance and scalability.

Simulating a complex system is one way to investigate ideas which are not yet implemented in reality. Chapter 5 discusses the motivation for and issues involved in simulating a data grid and presents the architecture and implentation of the grid simulator OptorSim. OptorSim was designed to investigate grid optimisation by replication of files, and Chapter 6 presents several replication strategies which have been implemented in OptorSim, including an economic model for autonomously trading files between sites. Chapters 7 and 8 then present a series of results obtained using OptorSim, first for some simple grid topologies and then for a model of LCG as it is expected to be in 2008, the first full year of LHC data-taking. Chapter 9 then presents some conclusions.

A description of work carried out investigating the use of artificial neural networks to aid in particle discovery at the LHC, looking at the $HZ \rightarrow llb\bar{b}$ channel in particular, is given in Appendix A.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In December 1989, in an article entitled "Does HEP Still Hold Challenges for Computer Science?" [5], L. O. Hertzberger asserted that:

> When we pose the question, 'what are the technological challenges in HEP?', the answer is speed, complexity and scale. In recent years we have seen a development towards ever larger detectors ... resulting in experiments producing ever more data ...

Hertzberger was speaking on the eve of data-taking at LEP, the Large Electron-Positron Collider at CERN, the European Organization for Nuclear Research. At that time LEP was the world's most powerful particle accelerator, designed to collide electrons and positrons with a centre-of-mass energy of 90 GeV. His words remain true today, as the particle physics community prepares for the Large Hadron Collider (LHC) to start data-taking in 2007. The computing solutions employed for particle physics have evolved considerably, however, as accelerators reach ever-higher energies and produce ever-more data in the quest to deepen human understanding of the universe around us.

In this chapter, an introduction will be given to the development of computing for high energy physics (HEP), from the early years to the present day, using CERN as the main example. A short history of distributed computing and its current status will also be given, leading up to the introduction of grid computing. The computing requirements of different aspects of HEP experiments will be discussed, followed by a description of the LHC and the particular challenges which it poses for HEP computing. The solution which has been adopted by the LHC experiments - that of a worldwide data grid - will be presented, with a description of the resource requirements for the first few years of LHC data-taking. Finally, some mention will be made of other scientific disciplines which face similar challenges, and an outline of the structure of the following chapters will be given.

1.1 The Development of HEP Computing

Particle physicists were quick to realise the potential of general-purpose computers when they began to become commercially available in the post-war years, and have consistently been a driving force for computer technology since then. The first electronic computer at CERN was a vacuum-tube Ferranti Mercury with a processor speed of 16.6 kHz, taking two years to build and finally installed and running in October 1958 [6]. It was initially used for applications such as the analysis of paper tape produced by the Instruments for the Evaluation of Photographs (IEPs), which were used to scan and measure the photographic film from bubble chambers. By 1962 it was possible to read the paper tapes straight from the IEPs into the Mercury, then pass it onto a new machine, an IBM 709, for further analysis. Soon, machines like the IBM 709 were connected to the film measuring devices online, and through the rest of the 1960s the use of film declined as computers began direct acquisition of digital events.

At this time, all computers were large, complex, mainframe machines. It was impossible, however, to connect experimental devices directly to large central computers, which led to decentralisation and separation of functions for different machines. Coinciding as it did with the development of integrated circuits and microprocessors, this led to rapid growth in the use of smaller, cheaper computers. During the 1980s, this growth continued alongside the introduction of supercomputers like the Cray X-MP, which CERN installed to help handle the data from LEP. Eventually, mainframe computers were overtaken by the more scalable solution of distributed UNIX workstations, with the last mainframe at CERN turned off in 1996 [7]. This distributed computing model prevails today, and is now being taken to new levels with the development of grid computing.

1.2 The Development of Networked Computing

1.2.1 Network Technology

In tandem with, and essential to, the evolving status of computers in HEP was the development of computer networking and eventually the Internet and World Wide Web. The first proponent of a universal computer network was J. C. R. Licklider, in a series of memos at MIT discussing his "Galactic Network" concept, and in several influential papers [8] [9]. This led to ARPA, the US Department of Defense's Advanced Research Projects Agency, planning and deploying a computer communications network called ARPANET. It initially consisted of four nodes across the US, and its first message was transmitted in 1969.

At CERN, the 1960s saw the birth of its first local area network, with the connection of the Mercury to a sonic spark chamber at the Missing Mass Spectrometer requiring the construction of a 1 km data link. This eventually led to CERNET, a 2 Mbit/s network for fast file transfer between a number of mainframes and minicomputers. The development of such disparate local area networks (LANs), as well as packet radio networks, separate from ARPANET and from each other, gave rise to the need for an *internetwork*. The TCP/IP protocols were therefore developed as common protocols for all networks. At about the same time, the Domain Name System (DNS) was developed to resolve human-readable internet host names to their numeric IP addresses. ARPANET thus evolved into the first version of the Internet, with eventual commercialisation and privatisation leading to the Internet with which we are familiar today.

The most well-known internet application is the World-Wide Web, built on the two basic inventions of HyperText Markup Language (HTML) and Uniform Resource Locators (URLs) by Tim Berners-Lee at CERN in 1989. Web browsers, through which web pages can be accessed, are an example of network *clients*. A client is a system which accesses a service on a remote computer over a network. A *server*, on the other hand, is the application which makes the service available, such as a Web server making pages available for access by browsers. As well as the simple browsing of documents, Web applications now include email, retail sales, auctions, discussion boards, weblogs, and many more.

Web Services

The wealth of applications available via the web has led, in recent years, to the development of *Web services* for application-to-application communication. A Web service is an interface with a set of defined protocols and standards, which allows different applications to communicate regardless of the programming languages they are written in or the platform on which they are running. All the data exchanged, for example, are formatted in XML (eXtensible Markup Language), and the public interface to any Web service is defined in WSDL (Web Services Description Language). The Web Services Activity of the W3C (World Wide Web Consortium) [10] is concentrating on developing the technologies required to fully realise the potential of Web services. As Chapter 2 will show, many of the most recent developments in grid technology, such as the latest version of the Globus Toolkit [11], rely on Web services.

1.2.2 Distributed Computing

Distributed computing has already been mentioned in Section 1.1 as the successor to the central mainframe paradigm of computing for HEP. Encompassing a wide range of systems and architectures, the term covers any coordinated use of physically separated computing resources, from computer clusters to peer-to-peer systems to Web servers. Naturally, distributed computing relies heavily on the underlying network infrastructure described above.

Distributed computing is too broad a field to explore in depth here. The distributed systems currently used on a day-to-day basis by particle physicists are typically UNIX clusters or batch farms, where jobs are submitted through one or more front-end machines, which then use some load-balancing algorithm to choose a back-end node to which the job should be submitted. Alternatively, if high performance is required it can be submitted to a high-performance cluster where the job is split across many nodes. Some recent trends in distributed computing are, however, worth mentioning and these are now described.

Peer-to-Peer Systems

In a peer-to-peer (P2P) network, each individual node or *peer* can act as both client and server to its peers, rather than having a few dedicated servers. All nodes provide resources (bandwidth, storage space, computing power etc) to the system, so as nodes join the system the overall capacity increases, thus keeping the system scalable. The most common use of P2P systems is for internet file-sharing, particularly audio and video files, with the best-known P2P networks being Gnutella [12] and Freenet [13].

Public Computing

Public computing is the use of Internet-connected home computers as nodes in a distributed system, using spare CPU cycles to perform useful work. The most well-known such project is SETI@home [14], in which users run a client programme which analyses radio telescope data from a central server, in a search for extra-terrestrial intelligence. With over 5 million users to date, it is the world's most massively distributed system. Its success has inspired a number of similar projects ranging from climate prediction to protein folding, including the LHC@home project. LHC@home [15] examines the stability of the orbits of protons in the LHC beam, as a test of the usefulness of the public computing model for the LHC.

Grid Computing

Grid computing is perhaps the most recent of the major trends in distributed computing, although the concept has been around for some time. Its immediate conceptual ancestor is *metacomputing*, which aimed at combining the processing power of supercomputers, but as far back as 1965, the developers of the Multics operating system [16] had the vision of computing power as a utility, which is one of the underlying concepts of a grid.

There exist many definitions of what exactly constitutes a grid, and the most common of these will be discussed in Chapter 2, but the basic concept is of a service for sharing computing resources between geographically distributed users, allowing a user to access computing power or storage space without necessarily knowing or caring where it has come from, in much the same way that someone can use an electrical appliance without knowing where the electricity was generated. This analogy with the electrical power grid is indeed where the term “grid” comes from, although the analogy is somewhat simplistic in practice.

There are many grid projects, for various scientific disciplines; particle physics, biomedical science and other disciplines which require high data handling or processing capabilities are especially interested in using grids. A more detailed discussion of grid technology is therefore left until Chapter 2.

As these short histories of networked and distributed computing for HEP have now brought us up to the present day, the next section discusses the various aspects of a HEP experiment which present challenges for computing, then the LHC and its requirements are described in Section 1.4.

1.3 Elements of HEP Computing

A typical HEP experiment has a number of areas which require computing support. The most important of these are listed here with a brief description of their particular requirements.

Triggering

In a modern particle accelerator, collisions occur millions of times per second. Since very few of these collisions produce interactions which are interesting to the physicist, triggering is required. Triggering is the process of deciding, for each particle collision, whether a possibly interesting event has occurred or not. In most experiments, multi-level trigger systems are used. The level-1 trigger takes the detector data and uses custom-built fast electronics to decide whether the data should be passed for further filtering or rejected immediately. Higher levels of trigger (there may be two or three levels in total) perform more detailed analysis of event data to decide whether it should be stored for off-line processing and analysis. This is usually done on dedicated processor farms and requires high throughput rates. The highest trigger level may also be known as the on-line filter.

Data Acquisition

The data acquisition (DAQ) system is intimately related to the trigger system. It includes the readout of data from the detector components; the collating of the event fragments from different detector components into a single data buffer for the event (*event building*); and the collection of monitoring data to identify problems and ensure that useful events are not being rejected. The main issues in DAQ are fast networking and interconnecting of data.

Monitoring and Control

Monitoring of data from the experiment hardware - the detectors and infrastructure - must be performed for safety, for calibration of the physics data when analysing events, and for alignment of the detector components. This requires high fault-tolerance, as some detector subsystems may run continuously and thus require continuous monitoring. Practically, monitoring and control systems are very similar to DAQ systems.

Off-line Processing and Analysis

After event data have been stored and taken off-line, the reconstruction of the raw data into physics events must be done. Reconstruction is not only computationally expensive, but the first pass of the reconstruction must be done promptly. Reprocessing may be performed several times per year, as improvements are made to the reconstruction software. The reconstructed events can then be analysed to gain physics results. This involves many physicists, in different locations, running individual analysis programmes on different subsets of the overall dataset for the experiment. Apart from the computing power required to do the analysis, large quantities of data must be stored, catalogued and managed in such a way that the necessary data are available to the users when required.

Simulation

Monte Carlo simulation of events allows an understanding of detector efficiencies and signal resolutions, thus aiding the analysis process. The theoretical branching ratio for the production of some particle in a collision is calculated, for example, and used as a basis for simulation of a number of events. Simulation of the detector is then used to produce data in the same format as real detector data, which is then reconstructed using the same software and compared with the original event (the "truth" information) to understand the efficiencies. Simulation is also important in order to look for unexpected events or backgrounds which may need to be accounted for in the analysis.

Each of these aspects of a particle physics experiment has its own particular requirements, whether for processing power, storage or network bandwidth, and as accelerator and detector technology improves, these requirements become ever more demanding. To illustrate this, Figure 1.1 shows the event rates and sizes for a number of past, present and future HEP experiments. The bottom left corner of the diagram, coloured pink, covers the approximate area for which traditional computing models can be used. Those outside this area use or will use a distributed model for data, analysis or both. The LHC experiments in particular (coloured blue in the diagram) lie far outside the scope of a traditional computing solution, with large event rates, sizes or both. In the next section, the LHC and its computing requirements will be described in more detail, with the challenge that they present.

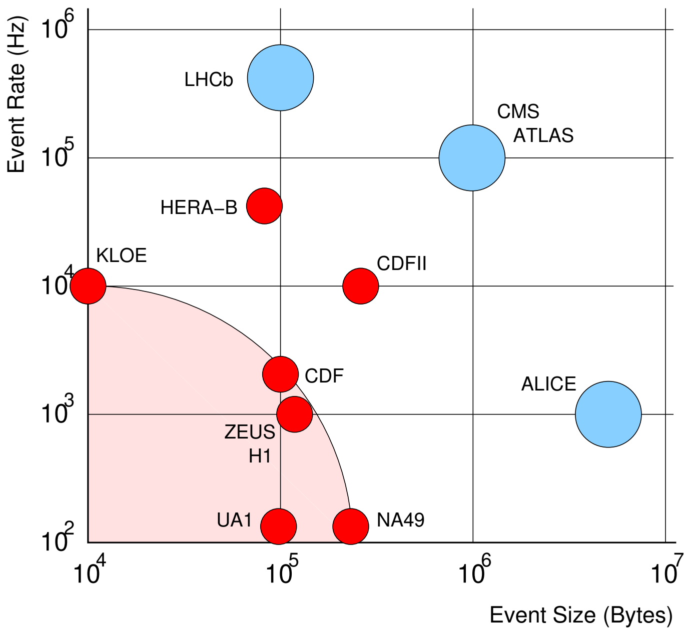

Figure 1.1: Approximate event rates and sizes for various HEP experiments, where the event rate is the rate after the Level-2 trigger. Adapted from [1].

1.4 The LHC Era

1.4.1 Introduction to the LHC

The Large Hadron Collider (LHC) is the newest particle accelerator to be developed at CERN. Using the 27 km circular tunnel from the LEP experiment, the LHC will act as a proton-proton collider for seven months of each year of running, with a centre-of-mass energy of 14 TeV, bunch crossing time of 25 ns, and collision rate of about 10^9 Hz. For another month each year, it will run as a heavy-ion collider (using lead ions in the first instance), with centre-of-mass energy of 1262 TeV (which is broken down to about 11.4 TeV per nucleon), bunch crossing time of 125 ns, and collision rate of about 410 kHz.

At different collision points around the LHC ring are the four particle detectors run by the four experiment collaborations: the two general experiments, ATLAS and CMS, and the two specialised experiments, LHCb and ALICE. Figure 1.2 is a cutaway diagram of the ATLAS experiment, showing the main detector components.

The beam pipe passes through the centre of the detector, with the particles produced

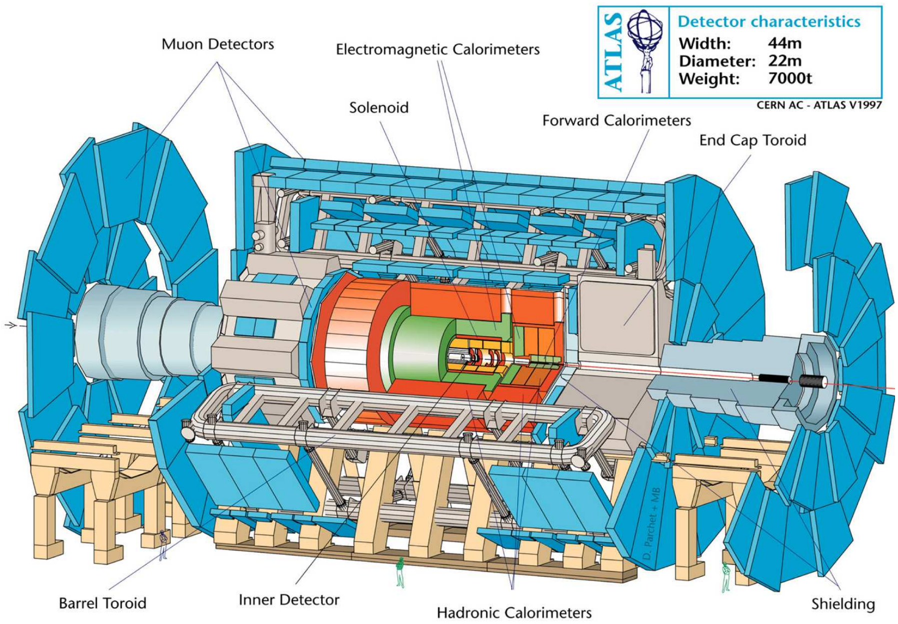

Figure 1.2: The ATLAS detector at the LHC.

in the collision propagating outward from the primary vertex. Depending on the type of particle, and the path it takes, signals are then produced in the different detector layers. An electron, for example, would give a track in the electromagnetic calorimeter whereas a muon would be the only particle to give a signal in the muon chambers. The detector signals are then read out by fast electronics to the trigger system, where the data for that collision are either passed on to the next trigger stage or discarded. The decision must be made and the readout electronics ready for the next collision within the bunch crossing time, which requires an extremely fast decision-making process even with buffering and pipelining of the data. The trigger system must reduce the data rate from the initial rate of 10^9 Hz (for proton-proton running) to 100-200 Hz, which is the rate at which the data can be written into files and sent for first-pass reconstruction.

Because the LHC is a hadron collider rather than a lepton collider like LEP, a single collision is a messy event, with large numbers of particles produced. Figure 1.3 shows a simulation of a Higgs event in the LHC, with a high multiplicity of particle tracks evident. This leads to a large event size for raw data, of the order of 1 MB per

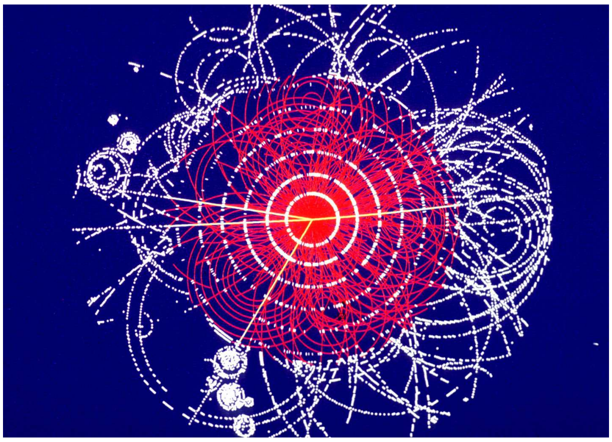

Figure 1.3: Simulated Higgs event in the ATLAS detector.

event for proton-proton running and of order 10 MB per event for heavy ion running. Combined with the trigger output rate, this gives an annual raw data production rate at the LHC of about 15 PB/year. This does not include the secondary data produced by reconstruction, analysis and simulation. The data must be available to each of the 5000 or so scientists who will be analysing it, in about 500 institutes around the world. All the data must also be available over the 15-year lifetime of the LHC.

This unprecedented data production rate, with the storage and computing resources required to handle it, has led to the adoption of data grids by each of the experiments as an integral part of their computing models. The traditional computing solution would be to store all the data, and site all the processing power, somewhere close to the experiments. This is indeed necessary for the online computing - the triggering, data acquisition, and monitoring and control systems.

For the offline processing and analysis, a central solution would perhaps be technically easier than a distributed solution, but a distributed solution has a number of advantages. Firstly, it is easier to handle the costs of upgrades and maintenance in a distributed system, where each local site can take responsibility for the resources at that site. Secondly, there is no single point of failure: if one site became unavailable, other sites could quickly take over its functions without major detriment to the users.

Thirdly, it allows users to receive the same service independent of their geographical location. The geographic distribution of sites also allows constant monitoring and support, as different time zones come into play. The next section outlines the main features of the experiment computing models and particularly the part that grid technology will play. From now on, the focus will rest solely on offline computing.

1.4.2 The Experiment Computing Models

Each of the LHC experiment collaborations has a well-developed computing model, expressed in [17] [18] [19] [20]. The reasons for adopting a distributed, grid-based solution have been outlined above, and thus the LHC Computing Grid (LCG) project has been established [21]. Its remit is to provide and maintain the data storage and analysis infrastructure for all the members of the LHC experiment collaborations. Before details of LCG are given, however, it is appropriate to describe more fully the data which it is expected to handle.

The different types of data file which are common to the LHC experiments, which will be produced and kept in the event store, are as follows:

- *RAW:* RAW data are the events as they come from the highest level trigger and are written to storage, before any reconstruction takes place.

- *Event Summary Data (ESD):* Known in ATLAS and ALICE as ESD but by CMS as RECO and LHCb as DST (Data Summary Tape), these are the event data produced by the reconstruction process. They include information on the tracks, vertices, jets, electrons, muons and so on, for each event. LHCb plans two forms of DST: *rDST*, a reduced form, which is written out at the first reconstruction pass with just enough information to allow physics pre-selection algorithms to select candidates for further analysis; and the full DST, which contains more information and is produced after the rDST has been passed by the pre-selection algorithm.

- *Analysis Object Data (AOD):* AOD is derived from the ESD level of event data and contains physics objects and other relevant information in a form suitable for analysis. LHCb does not plan to use AOD, performing analysis on a combination of RAW and DST data instead.

- *TAG:* TAG data are brief summaries of an event's characteristics, intended for fast identification of relevant events for analysis.

- *Simulated:* Simulated data are produced by all the experiments, and actually include a range of data types from the different stages of simulation.

As well as these, some of the experiments have other data formats such as *Derived Physics Data (DPD)* in ATLAS, which is an n-tuple style representation of the event data in a format suitable for histogramming and visualisation by end users, and *FEVT* in CMS, which is a term used for the combination of RAW and RECO data rather than a distinct format.

A summary of the estimated event sizes for each of the above data types in shown in Table 1.1, as well as trigger rates and the time to process an event [1] during reconstruction, simulation and analysis. For the ALICE experiment, the figures are given for heavy-ion running as well as proton-proton running, as the heavy ion programme is more significant for ALICE than for any of the other three experiments. The data are taken from the computing model documents which were described above, and from the LCG Technical Design Report [21].

The table shows that ATLAS, CMS and ALICE in its proton-proton phase have similar values for many of the main parameters, at least at an order-of-magnitude level. LHCb, on the other hand, has much smaller event sizes with a much higher trigger rate. This, as well as its planned use of LCG resources (described below) makes it closer in nature to the current generation of HEP experiments. ALICE in heavy-ion running, on the other hand, has very much larger event sizes than the other experiments and requires correspondingly longer times for processing and analysis. The next section shows how these different data types are handled in the LCG project, and what the data flow is within LCG from the point at which RAW data are output

[1] Processing times are given in units of SI2000-seconds; SPEC CINT 2000 or SI2000 benchmarks are a standard way of measuring CPU performance. A present-day Pentium IV PC has a value of about 0.5 kSI2000.

	ALICE (p-p)	ALICE (heavy ion)	ATLAS	CMS	LHCb
RAW size (MB)	1	12.5	1.6	1.5	0.025
ESD/RECO/DST size (MB)	0.2	2.5	0.5	0.25	0.075
AOD/rDST size (MB)	0.05	0.25	0.1	0.05	0.025
TAG size (MB)	0.01	0.01	0.001	0.01	0.001
SIM size (MB)	0.4	300	2	2	-
SIM ESD size (MB)	0.04	2.1	0.5	0.5	0.4
Trigger rate (Hz)	100	100	200	150	2000
Reco. time/event (kSI2000-s)	5.4	675	15	25	2.4
Sim. time/event (kSI2000-s)	35	15000	100	45	50
Analysis time/event (kSI2000-s)	3	350	0.5	0.25	0.3

Table 1.1: Summary of LHC experiment event sizes, trigger rates and processing times.

from the final trigger stage to analysis by the end user.

1.4.3 The Tiered LCG Architecture

LCG has adopted a three-tiered grid architecture, following the example of the MONARC project [22], which was initiated in 1998 to produce a suitable computing model for the LHC experiments. These three tiers (Tier-0 to Tier-2) are shown in Figure 1.4 along with Tier-3 (which was in the MONARC model but is not officially part of LCG), and their functions described below. Users' desktops at their institutions can be considered a fifth tier.

Tier-0

As CERN is the place where all the data are produced and where the experiments are situated, it is allocated a single central Tier-0 site. The original raw data are recorded and archived here. In addition, the first pass of reconstruction will take

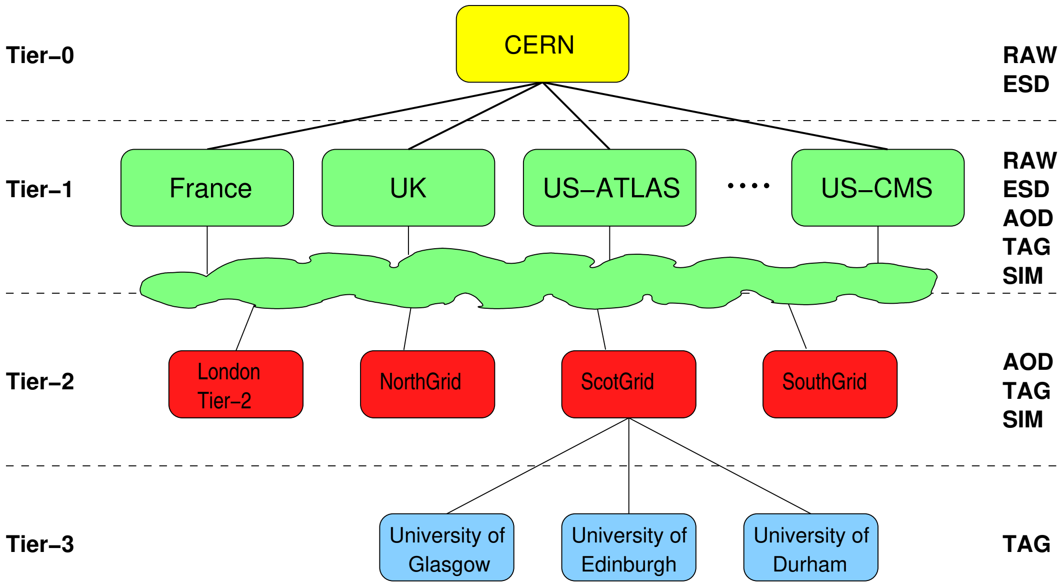

Figure 1.4: The tiers of the LCG architecture, with the type of data predominantly stored at each tier.

place at CERN and a copy of the reconstructed data will be stored there. The Tier-0 is also responsible for distributing a copy of the raw data for each experiment across the Tier-1 sites associated with that experiment, and copies of the reconstructed data depending on the experiment policy.

Tier-1

There will be approximately ten Tier-1 sites, which will act as Regional Centres for the country or region in which they are situated and each of which will serve some or all of the experiments. The Rutherford Appleton Laboratory (RAL) acts as the Regional Centre for the UK, for example, for all four experiments. Tier-1 sites are responsible for managing permanent storage of all the data (raw, reconstructed and simulated) which they have been allocated, and providing computational power for further reprocessing and for analyses which require access to large quantities of data. Tier-1s must also provide access to the data from the Tier-2 sites. The way the different experiments plan to use the Tier-1s varies, however. While ALICE, ATLAS and CMS plan to use only a fraction of Tier-1 capacities for analysis and simulation, LHCb plans to perform most of the distributed analysis on the Tier-1s and CERN.

Tier-2

Tier-2 sites will each serve some geographical area (perhaps a region within a country). Each has a "preferred" Tier-1 for data access, but this is not fixed, so that if the preferred Tier-1 is unavailable or if data would be more easily available elsewhere, a different Tier-1 could be used. This is depicted by the "cloud" of Tier-1s in Figure 1.4.

The role of Tier-2 sites is to provide computing resources and some storage capacity for Monte Carlo simulation, although the definitive simulated data will be stored at the Tier-1s. ALICE, ATLAS and CMS also plan to use Tier-2s for much of their end-user analysis. ATLAS and CMS may also use some Tier-2s to produce calibrations from the processing of raw data. Currently, over 100 sites are anticipated in the Memorandum of Understanding between CERN and the participating sites [23], divided into 39 Tier-2 centres, although not all are yet confirmed.

Tier-3

The Tier-3 layer consists of the computing facilities at universities and other LHC-related institutions, which will be used for processing and analysis but are not part of the LCG project. A university physics department cluster, for example, which does not qualify as a Tier-2 but is used locally for analysis, would be classed as a Tier-3. Depending on the experiments' policies, Tier-3s must be provided with access to the data they require.

It has already been mentioned that sites may act as Tier-1 or Tier-2 centres for more than one experiment; it is also true that sites may host several tiers. CERN, for example, will host a Tier-1 as well as the Tier-0, and several of the Tier-1s have attached Tier-2s. In addition, CERN will host the CERN Analysis Facility (CAF) to provide additional analysis capabilities for the experiments. Its proximity to the detectors gives it a vital role in calibration and code development, as well as being useful for user access to RAW data.

1.4.4 LCG Resource Requirements

Having described the experiments' event models and the basic LCG architecture, the experiment computing and storage requirements for the first few years of LHC running are now shown in Tables 1.2, 1.3 and 1.4.

	2007	2008	2009	2010
CERN Total	10.0	25.3	34.5	53.7
CERN Tier-0	6.9	17.5	22.4	32.8
CERN Tier-1/2	3.1	7.8	12.1	20.9
All external Tier-1s	19.2	55.9	85.2	142.0
All Tier-2s	23.6	61.3	90.4	136.6
Total	53	143	210	332

Table 1.2: Summary of LHC experiment CPU requirements for initial years of LHC running at CERN, Tier-1 and Tier-2 sites. All values are in MSI2000.

	2007	2008	2009	2010
CERN Total	2.2	6.6	9.2	12.6
CERN Tier-0	0.4	1.3	1.4	1.8
CERN Tier-1/2	1.8	5.3	7.8	10.8
All external Tier-1s	9.3	31.2	45.4	72.1
All Tier-2s	5.2	18.8	32.4	49.2
Total	17	57	87	134

Table 1.3: Summary of LHC experiment disk storage requirements for initial years of LHC running at CERN, Tier-1 and Tier-2 sites. All values are in PB.

These tables show a rapid increase in requirements for both computing power and storage as the LHC goes through the initial ramp-up phase and settles into normal operation, with 2008 being the first full year of data-taking. They also show that storage requirements increase twice as fast as computing requirements as the data

	2007	2008	2009	2010
CERN Total	4.9	18	31.1	45.6
CERN Tier-0	3.4	13.6	23.6	12
CERN Tier-1/2	1.5	4.4	7.5	11.1
All external Tier-1s	9.3	34.7	60.8	92.2
Total	14	53	92	138

Table 1.4: Summary of LHC experiment mass storage system requirements for initial years of LHC running at CERN, Tier-1 and Tier-2 sites. All values are in PB.

accumulates. Roughly speaking, the ratio of CPU in MSI2000 to storage in PB (disk plus mass storage) is approximately 1 for the Tier-1 sites and 5 for the Tier-2 sites, showing the role of the Tier-1s as large data centres. At the Tier-1s, there are approximately equal proportions of disk and mass storage. Tier-2 sites do not have mass storage, but have roughly the same processing power as the Tier-1s. Using a grid allows these requirements to be met by spreading the cost and infrastructure among the participating nations. The currently planned resources meet about 80% of the requirements, in general, for CERN and the Tier-1s but only about 20% of the Tier-2 requirements [21]; it is expected that the requirements will be met, however, as more Tier-2 sites continue to join.

This section has given a brief overview of the LHC, and in particular the experiment computing models and the role that the LHC Computing Grid plays. Further details of the implementation of the LCG project will be given in the next two chapters, but before summarising this chapter and outlining the rest of the thesis, it is worth mentioning some other disciplines which face similar data handling problems and are meeting their requirements by using grid technology.

1.5 Other Data-Intensive Disciplines

HEP is not the only science to face large quantities of data. In astronomy, terabytes of data are produced by a wide range of detectors, across all wavelengths from gamma rays to radio, and the total volume of data approximately doubles every year. An effort is being made to establish an international *Virtual Observatory* [24] to mine the pre-existing data, as well as new data which are added, for tasks such as deep field sky surveys and discovery of rare objects.

Climate modelling is best known for being highly computationally intensive, with advanced climate models requiring long durations of simulation on supercomputers. These simulations, however, will produce tens of petabytes of output in future and if this output is to be useful, it must be distributed to climate researchers at various institutions. The Earth System Grid (ESG) [25] is one project aiming to provide scientists with access to such distributed data in a grid environment.

In medicine, medical imaging produces large amounts of data which could be used for epidemiology, advanced image processing, radiographic education and perhaps even remote diagnosis, if scientists were able to access all the data rather than the limited subset available at their own hospital or university. Projects like MammoGrid [26] aim to redress this by federating databases in a grid. The EGEE [27] project, which is providing the middleware for LCG, also has an extremely active biomedical community, which is using the grid for projects such as data-intensive searches for anti-malarial drugs.

There are many other examples which could be drawn from biology, chemistry, engineering and earth science. Suffice it to say that science in general is facing a flood of data as technology develops, and that in many cases grids are seen as the solution to the problems this poses.

1.6 Summary

In this chapter, a short history of computing in high energy physics from the early days to the present day was given, showing the increasing decentralisation as computing and data requirements increased, alongside the advances in technology which made such decentralisation possible. Some of the areas of HEP experiments which require computing support were then discussed, with offline processing, analysis and simulation being those areas for which distributed computing is appropriate. The Large Hadron Collider at CERN was then presented and its computing requirements examined in some detail. The LHC Computing Grid was presented as the solution the experiments have adopted for these unprecedented data handling requirements, and finally some other fields with heavy data handling requirements were mentioned.

The rest of this thesis is structured as follows. Chapter 2 explores the definition, architecture and components of grids in more detail, again with an emphasis on the LCG project. Chapter 3 will then focus on the data management components of LCG and their implementation, then a performance analysis of the LCG File Catalogue will be presented in Chapter 4.

The focus will then move to grid optimisation, with the simulator OptorSim presented in Chapter 5 and the replication and job scheduling strategies it employs discussed in Chapter 6, drawing from previous work in these areas. The results of a series of experiments comparing these strategies in different grid scenarios will be shown in Chapters 7 and 8, then Chapter 9 will summarise and present overall conclusions.

Chapter 2

Grid Technology

While the last chapter gave a short introduction to grids as a development from distributed computing and showed how the LHC experiment computing models use a grid for their offline processing and analysis, little explanation was given of what constitutes a grid or how it is designed and built. This chapter covers the basic aspects of grid technology. First, a closer examination is made of what a grid actually is, before a generic grid architecture is then presented. The Globus Toolkit [11], which is a software kit for grid development, is then introduced as an example of how many components of grid architecture are implemented in real life, followed by discussion of grid standards and the role played by Web services. Finally, a number of grid projects are described to illustrate the different application areas of grid technology.

2.1 What is a Grid?

Grid technology is a relatively new and rapidly evolving field. As with any such field, definitions and standards have yet to settle into fixed forms, with many very diverse projects all claiming to be “grids”. It is therefore necessary to consider the fundamental concepts of a grid and come to some definition of what is, and what is not, a grid. As a starting point, this section will consider the simple definition of a grid as a computing utility - the aspiration of many computer scientists. Then, the technological prerequisites for the fulfilment of such an aspiration are described, before a well-known definition of a grid is adopted and adapted to data grids in particular. The last part of this section considers some of the challenges faced in building a grid that meets such a definition.

2.1.1 The Electricity Grid Model

The initial vision of a grid as a computing utility, like water or electricity, available to users on demand, has already been mentioned in Chapter 1. There are a number of points for which this analogy is useful in conveying what the characteristics of a computing grid should be.

First, when an electrical appliance is plugged into the mains electricity supply, the user expects the appliance to work regardless of whether it is a radio or a refrigerator. Similarly, in a computing grid the user should expect that regardless of the task they require to be performed, the grid will supply the computing power and storage capacity they need.

The electricity grid user needs no knowledge of the underlying infrastructure or of where the electricity they are using was generated; likewise, the computing grid user need not worry about where his task is being processed or where the data it needs is stored. Both electricity grids and computing grids consist of an infrastructure which links together diverse and distributed resources with their equally diverse and distributed users.

Lastly, we are all familiar with paying for electricity by the kilowatt-hour, and perhaps in future grid usage will be also be paid for on a per-unit basis, as grids move away from being research vehicles to commercial day-to-day use. Something similar happened in the development of the Web, as privatisation led to Internet Service Providers charging users for time spent online or, more recently, for data downloaded.

The current state of grid technology is still some way from this vision. Computing grids are considerably more complex than electricity grids; it is not as trivial an affair to enable a computer to access the grid as it is to plug a kettle into an electrical socket. Electrical appliances (at least within a single country) have a standard interface to the electricity grid in the form of a plug, while computing grid interfaces have yet to coalesce into such a single user-level interface. It is not wise to stretch the analogy too far, and so the next section looks at some of the essential concepts which are prerequisite to the development of real grid computing.

2.1.2 Basic Concepts

There are a number of areas within grid computing which are fundamental to the successful functioning of a grid. They are criteria which under-gird the vision of the grid as a unified global computational resource, and can be defined as follows.

Resource Sharing

At the heart of the grid is the sharing of computing and storage resources between users. The Web has made familiar the use of information sharing, but resource sharing goes beyond that to the access and control of remote computers, software, data and scientific instruments, owned by different people, to provide the necessary utility.

Security

Such resource sharing naturally leads to questions of trust and security. It is necessary for resource providers to know that their resources are being used in a proper manner, and for users to know that their applications or data have not been tampered with while running at a remote site.

Security can be split into three parts, each of which requires careful implementation to achieve a secure grid environment. These are:

- *Authentication*. There must be a mechanism for checking the identity of a user or of a resource host.
- *Authorisation*. There must be a mechanism for enabling access to and subsequently checking that users are allowed to use the resources to which they request access.
- *Access Policies*. Resource owners must define policies which state which kind of users may access their resources and under what conditions.

As well as the three aspects outlined above, the grid must have an accounting mechanism to keep track of which users have used which resources. If the grid reaches the stage of a true computing utility, this accounting mechanism will be essential to enforce pricing policies.

Resource Management

Given a collection of shared resources, with all the access policies defined and security infrastructure set up, mechanisms must be set in place so that these resources are used efficiently. There must be resource management systems which ensure that jobs are allocated to suitable resources, that loads are balanced and that data are available at appropriate locations. In principle, when a job is submitted to the grid it is possible to calculate the optimal allocation of resources for that job. In practice this is too complex a task to perform analytically, but it should be possible to implement heuristics which reach some almost-optimal state.

High-speed Networking

Grids require large quantities of data to be transferred, whether as input to a computing job running on a remote site or as output being returned to the user or to a third site. This means that grids only become a viable method of computing when network connections are sufficiently fast that the data transfer time does not negate the benefit of fast processing at a remote site. Today's high-speed research networks, such as the 10 Gbit/s SuperJANET backbone in the UK, are making this possible. The LHC experiments have high bandwidth requirements; CMS, for example, require their Tier-2 sites to have at least 1 Gbit/s connectivity [28]. Applications which require very low network latency, rather than high throughput, also need high-speed networks. These include applications such as remote conferencing, or real-time control of remote instruments.

Open Standards

Although the vision of "the grid" is of a unified worldwide entity, there are in fact many grids, which it is hoped will eventually become interconnected. If applications developed for one grid are to run on any other grid, there must be a set of common, open standards to which grids should adhere. Going back to the electricity grid analogy, appliances from the USA will not work if they are plugged directly into a European electricity supply, but adapters are available to enable this, if required.

Grid standards, while not yet in a fixed state, are being developed by the Global Grid Forum [29], which is a body of over 5000 grid developers and researchers working to define grid specifications and build an international grid community. The fact that most grid projects are developed using the Globus Toolkit (see Section 2.3) aids the agreement of standards in the basic technology.

Virtual Organisations

The concept of *Virtual Organisations* (VOs) is not essential to the running of a grid, but it is of high practical value. VOs were first defined in [30]. Put simply, a VO is a group of people with a common objective, who are using the grid to reach that objective. Examples could be the collaborators on one of the LHC experiments, or the design team working on a new aircraft, or epidemiologists searching for a rare cancer in an international collection of databases.

VOs are useful because, rather than having to define policies of resource use for many individual users, with new users continually joining and leaving, policies can be defined at the VO level. The VOs themselves can then handle membership at the individual level. VOs also enable members, who are perhaps personally unknown to each other, to collaborate in an atmosphere of mutual trust, as each has been certified as a member of that VO.

Having examined the fundamental ideas underlying grid technology, it is now appropriate to pause and present a working definition of a grid before moving on to the architecture and components of such a grid in Section 2.2.

2.1.3 A Definition

A three-point definition of a grid given by Ian Foster, the inventor of the term "grid", is:

> *A Grid is a system that co-ordinates resources that are not subject to centralized control, using standard, open, general-purpose protocols and interfaces, to deliver nontrivial qualities of service.* [31]

This definition is one which is generally accepted in the grid community, although individual grid projects may adopt slightly different or more specialised definitions.

The first point, that a grid does not involve centralised control, distinguishes it from local management systems such as PBS (Portable Batch System) [32], which perform resource management on computer clusters. These kinds of localised systems have complete knowledge and control of the state of the machines within their domain, whereas the grid concept is of sharing resources across different domains.

The reasons for the second point, that standard, open protocols should be used, have already been discussed above. Interoperation of today's many grid projects is essential if there is to be a single Grid analogous to the Web.

The third point in the definition is that a grid should "deliver nontrivial qualities of service". The different components of a grid should work together to give something which would otherwise be unachievable, whether it be resource availability, job turnaround time, or the combination of many resources in a single job. If the whole grid is not greater than the sum of its parts, there is no need for it to exist at all.

The simple concept of a grid which was presented at the beginning of this chapter is thus refined and made somewhat more stringent by the above definition. It is sufficiently general, however, not to distinguish between different varieties of grid. Differentiation can be made between *computational grids*, *data grids* and *service grids*. While in future, these may all be aspects of one underlying Grid, at present different grids are designed for different purposes. Broadly speaking, computational grids are designed for applications requiring large amounts of CPU time or memory. Data grids are for accessing, handling and processing large quantities of data. Service grids are for improved collaborative computing, such as interactions between scientists within a virtual shared space, requiring real-time control and low network latency.

Some grids, by their nature, are harder to categorise. LCG, for example, can be considered partly as a computational grid, in that there will be computationally intensive processes such as Monte Carlo simulation running at some sites. Its main remit, however, is the processing of the LHC data and this makes it primarily an example of a data grid, with the physics analyses performed on LCG requiring complex queries to be run over very large datasets. The focus in the chapters which follow, and particularly Chapters 5, 7 and 8, will be on data grids and so Foster's definition can be adapted to:

> *A data grid is a system that co-ordinates resources that are not subject to centralised control, using standard, open, general-purpose protocols and interfaces, to provide secure, efficient access to data which are distributed across a variety of heterogeneous storage resources.*

In other words, the "non-trivial quality of service" for a data grid is that it allows data to be handled in ways which would otherwise be impossible: allowing scattered collaborators to access each others' data, the mining of large distributed datasets, and so on. This is the working definition which will be adopted for the rest of this thesis.

2.1.4 Design Challenges

The nature of a grid, as expressed in the above definition, gives rise to a number of challenges for practical design and implementation. The first is *resource heterogeneity.* A local computer cluster will generally have a single type of machine, operating system and so on. As a grid, by definition, spans many resources which are not under central control, many of these resources will have different operating systems, machine architectures and software environments. The grid must be able to integrate each of these and handle any grid jobs which have specific requirements.

The second challenge, closely related to the first, is that of *multiple administrative domains.* This, the fact that many resources have different owners and therefore different access policies (as was discussed in Section 2.1.2), requires that the grid must adjust its overall resource usage policies to fit the many local usage policies.

The third challenge is that grids are highly *dynamic environments.* Resources may come online or go offline with little or no warning, due to the lack of central control, and so the overall resource map will fluctuate. The grid must be able to handle these fluctuations in resource availability and recover from sudden losses without losing work. If a job is running on a site which goes down, for example, the grid should be able to recover the job and finish running it at a different site, from the last checkpoint before the failure.

Lastly, a grid must be designed with a view to its *scalability.* Although grid projects may start with relatively few sites involved, as time passes more sites may join and more users may submit their jobs for running on the grid rather than at their local site. The scale of resources available at each site may also increase over time. With LCG, for example, many more users will start taking advantage of the grid as the LHC comes online and then as the total volume of data grows. The grid's resource management and security systems must be able to handle its growing scale and complexity.

2.2 Grid Architecture and Components

Having discussed the basic concepts, working definition and design challenges of a grid project, this section looks at how these come together in designing a grid architecture. A general grid architecture can be separated into a series of layers, each with various components which different grid projects can implement in their own way. Figure 2.1 shows one way of depicting such a generic layered architecture.

Starting from the bottom of the diagram, the most basic layer of a grid is the *fabric.* This consists of the physical infrastructure on which the grid runs: the network cables, the processors, the servers, the disks and mass storage systems. Any sensors and instruments which feed data into the grid, such as a remote telescope in an astronomy grid, would also belong to the fabric layer.

Just above the fabric layer comes the first layer of grid middleware, containing the *resource and connectivity protocols.* These handle the network transactions between

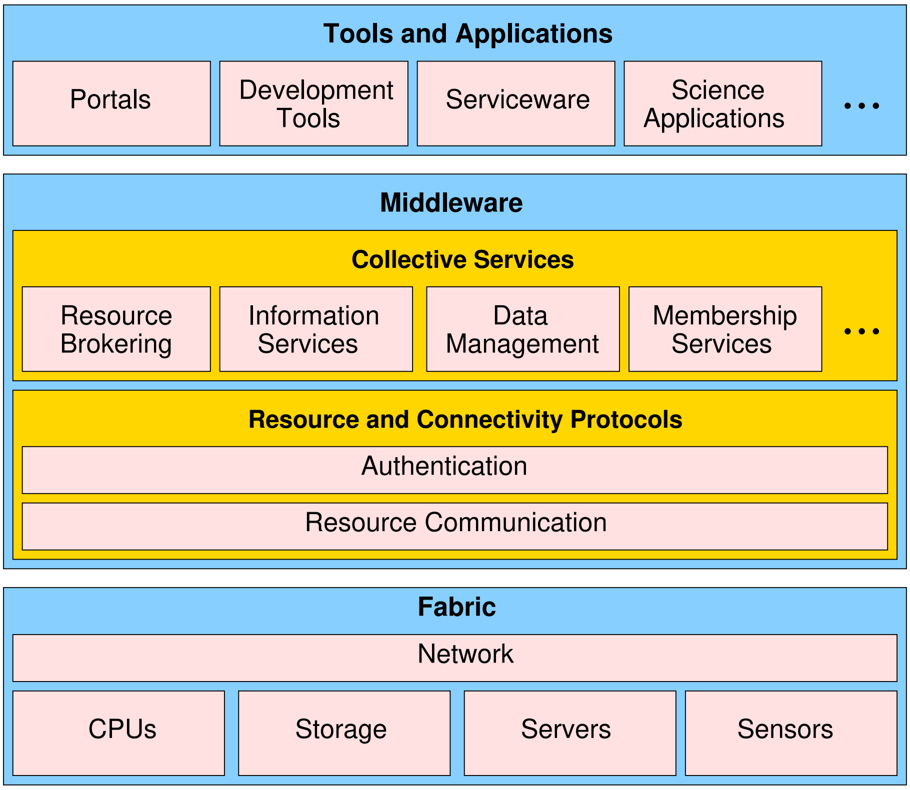

Figure 2.1: Generic grid architecture layers: fabric, middleware and user-level tools and applications.

the fabric-level components, allowing data to be exchanged between the resources. As well as these communication protocols, authentication protocols are required to enable secure verification of user and resource identities. The next layer of middleware, labelled *collective services* in the diagram, is concerned with higher-level functions such as information gathering and management systems. This would include monitoring the state of grid resources, replicating and cataloguing data, brokering resources for jobs, applying usage policies and so on. These two layers of middleware form the "glue" that holds the grid together, connecting the grid's many resources together and to the user-level tools and applications in the top layer.

This top layer consists of all components which will be visible to the users. As well as the scientific, engineering or other application programs written for specific problems, it includes portals by which users may access the grid, and development tools such as languages, compilers, libraries and debuggers for writing grid-enabled code. Also in this layer is the *serviceware*, which provides high-level management functions such as resource accountancy. In a commercially-run grid, this would include charging users for resources used, and paying resource owners. It is included in the top layer because, although it is not a user-written component, it is a user-visible component.

This scheme is just one way in which the different grid components can be separated into layers, but all architecture schemes tend towards a similar layered approach. What is clear, though, is that a grid is a complex structure with many components. Some real grid projects will be introduced in Section 2.5.

2.3 The Globus Toolkit

Many of the world's major grid projects are built using the Globus Toolkit (GT) [33], which is an open source project, originally begun for metacomputing environments, providing software for all the essential grid components such as resource management, data management and security. It is intended for grid developers and administrators, to set up a grid for their particular application or system, rather than for end users. Each of the main component areas in the latest version of the Toolkit, 4.0, is discussed below, with more information available on each of them at [11]. GT 4.0 is based heavily on web services, for reasons which will be discussed in Section 2.4.

Security

Globus security is centred on its *Grid Security Infrastructure* (GSI), which is based on public key cryptography. GSI uses the concept of *grid certificates* which are issued to each grid service and user as identification. Certificates contain the name of the user or service, their public key, and the name and digital signature of the Certificate Authority (CA) which issued the certificate. CAs have been set up in various regions, and before a certificate is issued, a representative of the CA manually verifies that the user is genuine. Certificates use the X.509 format, which is a standard format used in other public key software. The user's private key is stored in a file protected by a passphrase, so to use GSI the user must enter their passphrase every time their certificate is checked.

The GT 4.0 security components are:

- *Authentication and Authorisation.* Authentication and authorisation tools in GT 4.0 come in two varieties - Web services and pre-Web services. Both use grid certificates to verify identities, and access control lists to authorise users.

- *Community Authorization.* The Community Authorization Service (CAS) allows VOs to set policies on resources distributed across various sites.

- *Delegation.* Rather than have a user enter a passphrase many times while performing operations on the grid, a *proxy certificate* can be created, which is a time-limited certificate signed by the user. This is known as *delegation*, and the GT Delegation Service provides an interface for such delegation of credentials within a web service framework. Delegation is also necessary when agents are used to act on a user's behalf.

- *Credential Management.* There are two credential management components, MyProxy and SimpleCA. SimpleCA provides a simplified certificate authority which will issue grid certificates to users and services; certificates signed by a SimpleCA, however, are unlikely to be widely trusted. MyProxy is an online credential repository, where proxy certificates can be stored to avoid having to manually copy them to different machines, as the grid certificate must be present on the machine from which a grid request comes.

Data Management

Globus Toolkit data management tools consist of those for data movement and those for data replication. For data movement, these are:

- *GridFTP.* GridFTP [34] is a file transfer protocol for grids, defined by the GGF as an extension to the familiar FTP. It allows the secure, fast, efficient and robust transfer of data across a grid, particularly bulk data. The Globus Toolkit provides the most common implementation of GridFTP.

- *Reliable File Transfer Service* (RFT). The RFT service is a web service for the transfer of files in cases where long connections may not be desirable, or where client-side failure must be avoided. The user simply gives a list of source and destination URLs, and the RFT handles the transfer.

For data replication, GT provides a Replica Location Service (RLS). Users can query the RLS for a certain *logical file name* (LFN) and are returned all the *physical file names* or *storage URLs* (SURLs) of the replicas of that file which exist in the grid. The RLS consists of two types of catalogue. A Local Replica Catalogue (LRC) runs on every site and contains mappings of LFNs to all the SURLs of files at that site. The LRC regularly publishes a list of its LFNs to one or more Replica Location Indices (RLIs). RLI instances run on one or more sites, and contain mappings from LFNs to the LRCs which have the files.

GT 4.0 combines the RLS and RFT in a higher-level service called the Data Replication Service (DRS). It queries the RLS to find where the requested files are, copies them to their new location with the RFT, then registers the new replicas in the RLS.

Execution Management

Globus Toolkit job execution management is performed by GRAM, the Grid Resource Allocation and Management service, which is available as a Web services and as a pre-Web services implementation. GRAM is not itself a resource scheduler, but provides a uniform interface to a range of local resource schedulers. It allows management of jobs and the data they require, coordination of parallel jobs and monitoring of status and outputs.

Information Services

The information services, known collectively as the Monitoring and Discovery Service (MDS), allows users to monitor and discover resources on the grid, on a VO basis. The *Index Service* collects data from information providers and allows users to query or subscribe to that data. The *Trigger Service* is configured by users to perform some action based on the data taken from the Index Service. The *WebMDS* user interface allows user-friendly access to the Index Service.

Common Runtime

As well as all the tools which have been mentioned above, the Toolkit provides a set of common runtime components such as libraries and web services tools, to allow services to be platform-independent and to allow a higher layer of abstraction for developers to work from.

2.4 Grid Standards

The importance of defining open standards has already been emphasised in Section 2.1.2, and there is a definite push within the grid community to define and implement such standards, with the Global Grid Forum (GGF) being at the vanguard of these efforts. The GGF contains a number of working groups, each of which examines a current problem and produces specifications or recommendations for a solution. In seeking a solution for grid standards, it was decided to pursue a Web services approach, uniting Web services with grid technology to produce *Grid services*. The Open Grid Services Architecture (OGSA) working group was therefore convened to define requirements for a set of grid standards [35], while the Open Grid Services Infrastructure (OGSI) working group concentrated on the implementation of such standards [36].

In the OGSA model, each grid entity - computing resource, storage resource, network, database, application and so on - is represented by a service. In a service-oriented system, interoperability can be achieved if a standard interface for services and a set of protocols for interacting with them is defined. Then, the details of how a particular service is implemented is hidden from the user, allowing grids where the underlying infrastructure is quite different to be used together.

The OGSI implementation defined Grid services to be a subset of Web services, allowing controlled, fault-tolerant and secure management of *state*. State is the set of properties of a system, in this case a grid entity; in a grid environment, state is often distributed and long-lived. Since the initial OGSI-based implementation of the OGSA standards, a new implementation has been developed based on the Web Services Resource Framework (WSRF) [37], due to advances in the underlying Web service technology. In particular, WS-Addressing, which allows Web services to be addressed independent of protocol, and WS-MetaDataExchange, which provides ways for finding information about a published service, overlap with some of the functionality of OGSI.

Rather than keeping two versions of the same functionality, WSRF was designed to use these new Web service developments. Version 3 of the Globus Toolkit used the OGSI implementation, while GT 4.0 now uses the WSRF implementation. The widespread use of the Globus Toolkit, together with their close relation to Web services, which are already (particularly in industry) common, makes the OGSA and WSRF standards the likeliest candidates for overall grid standards.

2.5 Some Grid and Grid-like Projects

In the preceding sections, the concepts behind and architecture of a generic grid were discussed, the Globus Toolkit was introduced and the role of OGSA in defining grid standards was presented. In this section, examples of grid and grid-related projects, both past and present, drawn from various fields, are given. These are categorised here by their main scope as computational, data or service grids, although not all are fully-fledged grids in their own right - some are involved in developing grid tools or applications which will interact with other components to form a real grid. It should also be noted that this section cannot hope to include every grid project in existence, and so only a representative sample are included.

2.5.1 Computational Grids

A computational grid is one which is intended for high performance or high throughput computing. High performance computing often involves computationally expensive simulations, such as in climate modelling, astrophysics, economics or theoretical particle physics. High throughput computing is good for problems which can be broken down into many parallel tasks, such as Monte Carlo simulation of events in a particle detector. Computational grids are especially useful where the problem involves a combination of high performance and high throughput computing, such as a climate simulation where there are many parameters which can be varied, giving a large number of parallel computations, each of which requires high performance.

Condor and Condor-G

The Condor project [38] aims to develop and implement software tools for high throughput computing in a distributed environment. The Condor software is, in effect, a distributed batch computing system. It provides job management, scheduling, a priority scheme, and resource monitoring and management. A user submits their job to Condor, which chooses when and where to run the job.

Condor uses *opportunistic computing* to make full use of system resources. To do this, the group of machines or cluster - perhaps a set of departmental desktop workstations - on which Condor is installed forms a *Condor pool*. When one of the machines in the pool is idle, Condor can then start running a job on it, if it matches the job's requirements. Control remains with the machine owner, however, so Condor can be configured to stop running a job on a particular machine when it is no longer idle, checkpoint the job and move it to a different machine.

Condor-G [39] allows extension of the Condor concept to multiple administrative domains. Globus Toolkit components (particularly GRAM, the MDS and GSI) are used to allow jobs submitted to Condor-G to be run on Globus-managed grid resources. More details of the way Condor and Condor-G schedule jobs are given in Chapter 6.

Legion / Avaki

The Legion project [40] was started to produce middleware which connects computing resources using autonomous agents, giving the illusion of a single virtual computer. Legion is an integrated architecture rather than a "toolkit" like Globus middleware, and sits on top of a host's operating system. As with Condor, resource owners retain control of their resources and decide how much to contribute to the system. Legion software is now developed and distributed commercially by Avaki [41].

NetSolve / GridSolve

NetSolve [42] is another middleware project, aiming to bridge the gap between the standard interfaces used by scientists and computing resources available on the grid. A NetSolve system consists of clients, servers and agents, with the agent keeping a list of all available servers and matching clients with servers when a client makes a request. It is particularly aimed at computationally intense problems, so there are clients available in problem-solving environments such as MATLAB and Mathematica, as well as in traditional languages like C and Fortran. Interaction with Globus resources is possible via a client proxy.

The GridSolve [43] project is now building on NetSolve for a more integrated approach to other grid projects such as Globus and Condor, focusing on the development of GridRPC, a standard for remote procedure calls over the grid.

Gridbus

The Gridbus [44] project takes a slightly different approach, aiming to develop an economy-based grid architecture, where problems of scheduling and resource management would be solved by a computational economy, with resource providers being paid for some agreed quality of service. Gridbus provides a toolkit of higher-level tools (for monitoring, scheduling and so on), while the Globus Toolkit is used for lower-level functions like security, execution management and access to storage. The Gridbus architecture is currently implemented in the World-Wide Grid testbed, which contains over 200 nodes in Australia, Asia, Europe and the Americas.

2.5.2 Data Grids

A definition of a data grid has been given earlier in this chapter, and in Chapter 1, mention was made of various scientific disciplines which require data grid facilities to handle their data. This section describes briefly some major data grid projects.

Earth System Grid

The Earth System Grid (ESG) [25], which was mentioned in the previous chapter, is an example of a data grid for a particular scientific application - that of climate modelling. ESG uses some components of the Globus Toolkit, namely the RLS and GridFTP, together with the Network Weather Service (NWS) [45] (which gives information on network performance) and various other components. Via a specialised analysis tool, users interactively specify the characteristics of the data in which they are interested and the analysis they wish to perform.

European DataGrid, LCG and EGEE

The European DataGrid (EDG) [46] was a European Union-funded project, led by CERN, to provide access to distributed computing and storage resources at different institutions, for the three data-intensive disciplines of particle physics, earth observation and biomedical science. EDG built on top of the Globus Toolkit to develop a set of middleware more directly suited to the needs of the LHC experiments. The middleware suite which was developed was then used as the basis for the first release of LCG software.

The EGEE (Enabling Grids for E-SciencE) [27] project, which is the successor to EDG, is now building on EDG and LCG middleware to provide a permanent production grid service for many scientific communities, starting with particle physics and extending out to bioinformatics, chemistry, astronomy, geophysics and so on. With LCG as the main driving force, EGEE is re-engineering the EDG middleware in order to have this production infrastructure ready when the LHC starts data-taking.

Other Particle Physics Grid Projects

There are a number of other grid projects around the world catering for high energy physics as part of their main remit. These include:

- **DataTAG**: With the many grid projects on both sides of the Atlantic, DataTAG [47] is a European-led project focusing on the advanced networking and interoperability issues involved in a large-scale transatlantic grid.

- **GridPP**: The UK Grid for Particle Physics, GridPP [48] [49], is a collaboration between particle physicists and computer scientists in the UK to build a grid for the UK institutes which are participating in the LHC. A testbed has been set up over 17 institutions, with GridPP developers also contributing middleware and application development to LCG and EGEE.

- **GriPhyN**: The GriPhyN (Grid Physics Network) project [50] is developing grid technology for astronomy, particle physics and gravitational wave experiments, using the concept of *virtual data*. Much scientific data is not raw data, but is derived from raw data using well-known procedures. If these procedures and relationships between different datasets are documented, data can be generated on demand - this is the virtual data. GriPhyN, together with PPDG and iVDGL (see below) have developed the *Virtual Data Toolkit* (VDT) for handling such data, using Condor-G and Globus for the basic grid services. The US collaborators in CMS and ATLAS are among the contributors to GriPhyN and the VDT.

- **iVDGL**: The International Virtual Data Grid Laboratory [51] also uses the VDT, aiming to establish a single grid system using resources in Europe, the US and Asia for interdisciplinary experimentation in scientific computing, and to bring it from development to everyday use. Among the applications are the CMS and ATLAS experiments, as well as astronomical applications, gravitational wave searches and bioinformatics.

- **Open Science Grid**: The Open Science Grid (OSG) [52] is a US-based grid infrastructure project, driven by the US participants in the LHC, and built by collaboration between GriPhyN, iVDGL and PPDG.

- **PPDG**: The Particle Physics Data Grid (PPDG) [53] is a third US-based project, with close links to both GriPhyN and iVDGL. Apart from ALICE, ATLAS and CMS, it caters for the CDF and D0 experiments at the Fermi National Accelerator Laboratory (FNAL), the BaBar experiment at the Stanford Linear Accelerator Center (SLAC), the PHENIX and STAR experiments at

Brookhaven National Laboratory (BNL) and the Jefferson Lab (JLab).

While the above list of projects is not exhaustive, it does indicate the extent and interconnectedness of grid development activity within the particle physics community, and the importance of data rather than computing power as the driving force for these projects.

2.5.3 Service Grids

Service grids, also known as *collaborative grids* or *community grids*, are concerned with the enhancement of collaborations and interactions between people, often by using a virtual shared space. The main challenges are the real-time requirements of the users, for visualisation, communication, decision-making and so on, and there are several grid projects specifically addressing these challenges. Here, three examples are described.

Access Grid

The Access Grid [54] is a project which supports interaction between groups which are geographically separated, such as distributed meetings, seminars, lectures and tutorials. It uses a semi-immersive visualisation system, in which each collaborating institution has a purpose-built room called the Access Grid Node. This includes a display on which participants can see the remote participants in their rooms, with video and audio so that remote participants can in turn see and hear them. It differs from conventional video conferencing systems in that it is intended for group rather than individual use.

BIRN

As a final example, the Biomedical Informatics Research Network (BIRN) [55] consists of three testbeds, investigating brain function in schizophrenia, brain morphometry in illnesses such as Alzheimer's Disease, and animal models of disease. BIRN allows scientists from the different participating institutions to collaborate in the sharing and visualisation of different kinds of data, with data at the different sites being presented as a single, unified dataset.

CrossGrid

The CrossGrid project [56] was developed for scientific applications which require real-time interaction. In particular, applications were developed for medicine, flood management, particle physics and meteorological and pollution modelling. The Virtual Arteries application, for example, was developed to help cardiovascular surgeons visualise a patient's arteries and predict the blood flow, from medical scanners, data and computing resources in different geographical or administrative domains. Resource selection and scheduling in CrossGrid is based on Condor-G, with grid access and monitoring tools developed independently.

It is clear from this section that there are currently a plethora of grids and related projects developing middleware and tools for various kinds of grid and scientific application. Although the examples which have been presented have been divided into computational, data and service grids, it is also clear that these boundaries are not clear-cut, with many projects encompassing aspects of more than one type of grid.

2.6 Summary

This chapter has given an overview of grids, starting from the analogy of a utility something like electrical power, going on to set out some concepts and criteria which are basic to grid technology before coming to the generally accepted definition. A particular working definition of a data grid was then adopted, and some challenges to the design and implementation of such a grid presented.

A generic grid architecture with its fundamental components was then presented, and the Globus Toolkit introduced, followed by the Grid Services approach to building a grid. Finally, a number of different grid projects from various application areas were discussed, with particular emphasis on those which are being used for particle physics.

Chapter 3

The LHC Computing Grid

In this chapter, an overview of the architecture and implementation of LCG is given, focusing on the middleware components. First is an explanation of the history of the LCG middleware and how it is related to the EGEE project, followed by a description of the various middleware components. The second part of the chapter then concentrates on the data management middleware. The original data management tools from the European DataGrid are described, then the problems discovered when they were run in a production environment are shown to be the driving force for a new set of LCG data management tools, with particular emphasis on the file catalogues.

3.1 Evolution of LCG Middleware

The role of the LCG project is to prepare, deploy and operate the computing environment needed by the LHC experiments. The role of EGEE (the Enabling Grids for E-science project [27]) is to build on the recent developments in grid technology to produce a reliable, stable grid infrastructure for scientists in Europe. The two projects are closely connected, with both being led from CERN. LCG is not intended as a development project, but aims to take the required middleware from other providers, modifying or fixing it if required.

The first set of LCG middleware (LCG-1) was provided by EGEE's precursor, the European DataGrid (EDG). After further development to provide more functionality and stability, this was deployed in September 2003 as LCG-2. The LCG-2 testbed then became the first production service for EGEE, while further modification of components has continued.

The EGEE Middleware Activity is responsible for providing middleware components for the core grid services which were described in Chapter 2, such that they can be deployed on a number of platforms and operating systems. The EGEE middleware suite is known as *gLite* [57] and is based on a number of components from EDG, the Virtual Data Toolkit (VDT) and AliEn [58] (the ALICE Environment, developed by the ALICE experiment for their own computing needs). Figure 3.1 shows the relationship between these projects in contributing to EGEE. AliEn is shown with a dotted line rather than a solid line, as the code is being deprecated from gLite while retaining many of the ideas. Currently, gLite is undergoing a series of prototyping and development cycles, existing alongside LCG-2, until it is ready to replace or merge with LCG-2 to give a single service.

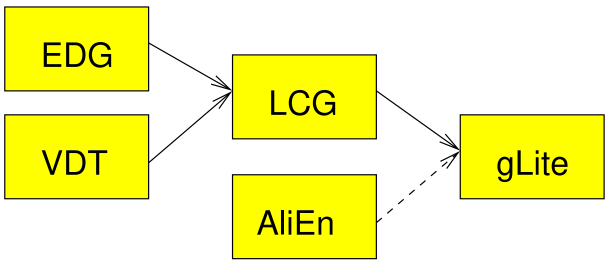

Figure 3.1: History of contributions to the gLite middleware.

The gLite services use a *Service Oriented Architecture*, in which each middleware component is represented by a service, to help interoperability between grids and compliance with the OGSA / WSRF standards which were discussed in the last chapter. The role of other grids, such as Open Science Grid (OSG) [52] in the USA and NorduGrid [59] in Scandinavia, in collaborating with LCG for computing support for the LHC experiments makes interoperability even more important.

Another important influence on the gLite middleware is the ARDA (A Realisation of Distributed Analysis, originally known as Architectural Roadmap towards Distributed Analysis [60]) group. The ARDA group are responsible for coordinating activities for the LHC experiments to use a grid for their distributed analysis, and so act as an interface between gLite and LCG and the experiments.

3.2 LCG and EGEE Middleware

Given the situation at the time of writing, in which the LCG and EGEE middleware strands have not yet converged, reference will be made to both in the sections which follow. The figures, however, will reflect the LCG-2 implementation. The middleware described here can be separated into two types - *site services*, which are those which operate at the level of a single site, and *global* or *VO services*, which are those which operate at the level of the whole grid or of a VO - and these will be discussed in turn. An overview of the main components and their interactions is shown in Figure 3.2. Users interact with the grid via a *User Interface* (UI) node, from where

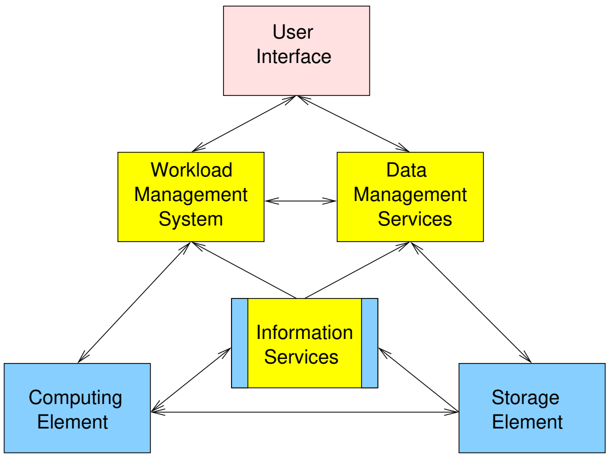

Figure 3.2: The main LCG / EGEE middleware components and their interactions. Site services are in blue and global services in yellow, with the User Interface in pink.

they can submit jobs to the Workload Management System (WMS) or access the Data Management Services. For Computing Elements (CEs) which operate in *push mode*, the WMS submits jobs to the CEs, using the Information Services and Data Management Services in its decision making. For CEs which operate in *pull mode*, the CEs request jobs from the WMS. The Data Management Services are used to move and catalogue the grid's data files, while the Information Services keep track of the state of the grid. The Computing Elements (CEs) accept jobs from the WMS, pass computing resource information to the Information Services (and, if in pull mode, query the Information Services) and access data held by the Storage Elements. The Storage Elements provide data to the CEs and Data Management Services, and pass storage resource information to the Information Services.

3.2.1 Site Services

Security

All the middleware services rely on the Globus Grid Security Infrastructure (GSI), which was introduced in Chapter 2. Users must apply for a grid certificate to an accredited Certificate Authority (CA), which is typically responsible for grid users from a particular country or region, rather than the basic SimpleCA. The UK CA, for example, is hosted by RAL. Proxy certificates are then generated from the user's grid certificate and used for authentication and authorisation. Information about the user's VO is added by the Virtual Organisation Membership Service (see Section 3.2.2, so that the site policies for different VOs can be applied. Local policies are controlled by the Local Centre Authorization Service (LCAS) for C/C++ services and the Java Authorization Framework for Java services. Users are mapped to local accounts (user and group IDs) by the Local Credential Mapping Service (LCMAPS), using the Distinguished Name (DN) information from their grid certificate. If a proxy certificate has too short a lifetime for a particular job, the MyProxy service provided in GSI can be used to renew the proxy. Local sites also keep Certificate Revocation Lists (CRLs) to prevent users whose certificates have been revoked by their CA from accessing the site.

Computing Element

The Computing Element (CE) is the node at a site which provides grid access to the local resources. There are currently two types of CE, the LCG-2 CE, which is deployed on the production service, and the gLite CE, which is deployed on the pre-production service. Both handle the submission of jobs to a site, job cancellation, suspension and resumption, status inquiries and notification to the user that a job is finished.

The LCG-2 CE works only in push mode. When it receives a job, the Globus gatekeeper (which handles access requests to a site), LCAS and LMAPS are used to authorise the user and Globus GRAM is used to submit the job to the local resource management system (LRMS), which allocates it to one of the local computing nodes. The LCG-2 CE also has an interface to the Logging and Bookkeeping services (see Section 3.2.2) to monitor job status. The components of the LCG-2 CE and their interactions when submitting a job are shown in Figure 3.3.

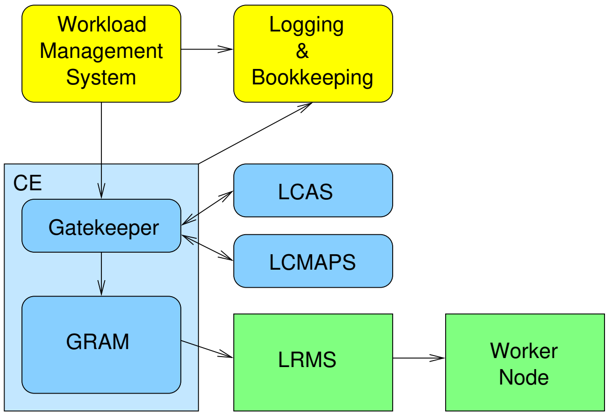

Figure 3.3: The LCG-2 CE components and interactions at job submission.

The gLite CE is largely similar to the LCG-2 CE, but it works in pull mode, where a CE asks the WMS for jobs, as well as push mode. This is one of the concepts adapted from AliEn. As well as the Globus gatekeeper, GSI and LCAS/LMAPS components, the gLite CE also uses Condor-C, which is a Condor component to allow jobs in one machine's job queue to be moved to another machine's job queue if necessary. As with the LCG-2 CE, the Logging and Bookkeeping services are used to track jobs.

Storage Element

The Storage Element (SE) is the component at a site which provides grid access to the local storage resources, allowing other services to access storage in a uniform way without having to know the details of the local system. It needs to have some interface to the storage system and methods to manipulate the data.

The local storage may be a mass storage system (MSS) or a disk pool. An MSS is usually accessed via a Storage Resource Manager (SRM) such as the CERN Advanced Storage System (CASTOR) or the dCache [61] system developed by FNAL and DESY. The SRM protocol [62] itself was developed by several laboratories, led by Lawrence Berkeley National Laboratory; "SRM" can refer either to the protocol or to an SE which supports it. LCG has also developed the Disk Pool Manager (DPM) as a simple SRM for smaller sites which may not have mass storage or which cannot afford the resources or manpower for a more advanced storage system. More information about the DPM is given in Section 3.3.3.

The LCG-2 SE can currently be either a "classic" SE or an SRM SE. The classic SE uses GridFTP to access disk storage systems, without an SRM interface, whereas the SRM SE is used for MSSs. As an SRM interface is now provided by the DPM, however, the classic SE can be phased out to give more uniform access to storage.

LCG also developed the Grid File Access Library (GFAL) for the file manipulation requirements of an SE. GFAL allows access to files in a POSIX-like way (allowing a user to open, read, write and close a file) via their Logical File Name (LFN). The LFN is a user-friendly filename such as `lfn:grid/atlas/data/run1234/2005-09-02`.

There may be several replicas of this file, each with a different location and hence a different Storage File Name (SFN) or Storage URL (SURL), but file catalogues are employed (detailed in Section 3.2.2) to map from the LFNs to the SURLs, so that the user need only know the LFN.

The gLite SE uses an SRM for access to storage, with a gridFTP server for moving the data. The gLite I/O service performs a similar role to GFAL in the LCG SE. Both these file access services have interfaces to the various grid file catalogues; further details of GFAL in relation to the rest of the LCG data management tools will be given in Section 3.3.

Monitoring and Accounting Services

The monitoring and accounting services are responsible for gathering and publishing information on the grid services at each site, including their usage data. The LCG-2 monitoring system is based on the Globus Monitoring and Discovery System (MDS). Each resource at a site (CE, SE, etc) report via their Grid Resource Information Server (GRIS) to the site's Grid Index Information Server (GIIS), as shown in Figure 3.4. Each site may have several CEs or SEs, each of which has its GRIS. Each

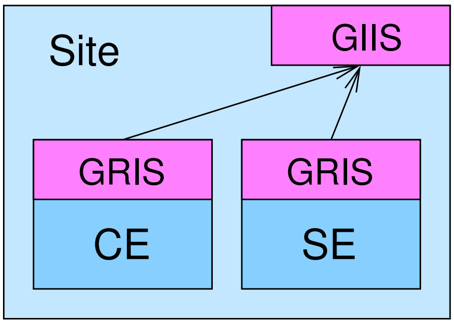

Figure 3.4: Monitoring components within a grid site.

GIIS then publishes its information, using the Lightweight Directory Access Protocol (LDAP) to a BDII (Berkeley Database Information Index), which thus contains status information on the whole grid and can be queried by users and other grid services, such as the WMS. Each site also has an Accounting Processor for Event Logs (APEL) system. This gathers accounting information related to individual jobs, such as time taken, resources used and ownership, and publishes its data into the site's Relational Grid Monitoring Architecture (R-GMA) system (below). This information is used to compile usage records for the VOs.

R-GMA, the Relational Grid Monitoring Architecture [63], initially developed by EDG and now continued in EGEE, is a grid-wide information and monitoring system and is described in Section 3.2.2. It is the main component of the gLite monitoring system, where it is used to discover services as well as information. For accounting, gLite are developing the Data Grid Accounting System (DGAS). DGAS gathers information about resource usage by individuals and groups (such as VOs), which can then be used in the application of quotas and, in future, billing.

All data in the monitoring, accounting and information services conform to the GLUE (Grid Laboratory for a Uniform Environment) schema [64], of which LDAP is in fact an implementation. The GLUE schema aims to specify a uniform model for resource discovery and monitoring, and does so by defining the attributes and binding information for CEs and SEs, and for the network resources.

3.2.2 Global Services

Apart from the site-level components which were described above, there are a number of components which act across the whole grid, perhaps within the bounds of a particular VO.

Virtual Organisation Membership Service

The Virtual Organisation Membership Service (VOMS) was developed as part of the EDG project and is used by both LCG-2 and gLite. VOMS adds information on a user's VO to their proxy certificate, so that other components - such as the Workload Management System and file catalogues - know what their access privileges are.

Workload Management

The Workload Management System (WMS) is responsible for the overall management of jobs on the grid. It allows users to submit jobs to the grid, then to cancel, suspend and resume them, as well as querying their status. The LCG-2 and gLite WMSs are similar, with the LCG-2 WMS being the basis for the gLite version.

The components and interactions of the WMS are shown in Figure 3.5. When the WMS receives a job from a client, the Workload Manager (WM) or Resource Broker (RB) matches the job requirements with available resources and sends it to an appropriate CE. Job requirements are specified using an extension of Condor's ClassAd language. Information about the CE resources is gathered using the BDII. If data location is one of the matchmaking criteria specified by the user, the data management components are queried to find sites which have the data stored.

File Catalogues

The file catalogues are responsible for keeping track of file locations and providing access to the files. This is necessary because *replicas* of files may exist in a number

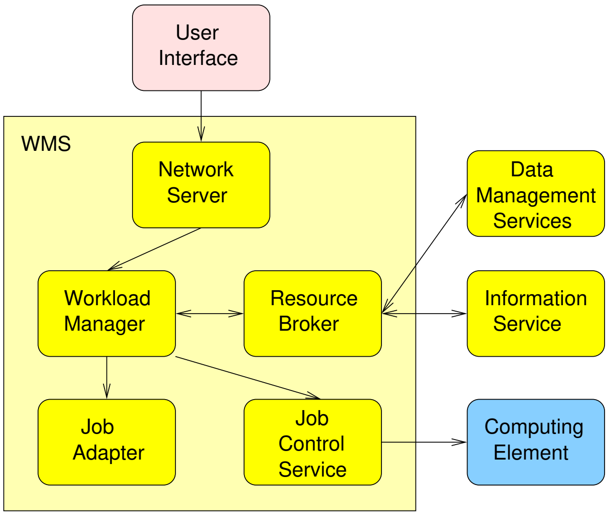

Figure 3.5: The LCG-2 Workload Management System components and interactions at job submission.

of grid sites. A file may also be known by several LFNs, as different users may give it different names. Each file therefore has a Globally Unique Identifier (GUID) to ensure it is uniquely identifiable across the grid. A GUID is a hexadecimally-grouped 128-bit number such as `cc4ff5d0-1a27-11da-8cd6-0800200c9a66`. GUIDs are not a user-friendly format of file name, so for human users it is important to allow mapping from LFNs to GUIDs. The catalogues must therefore handle the relationships between LFNs, GUIDs and SURLs, which are shown in Figure 3.6. The user should only need to know one of the LFNs in order to access the data.

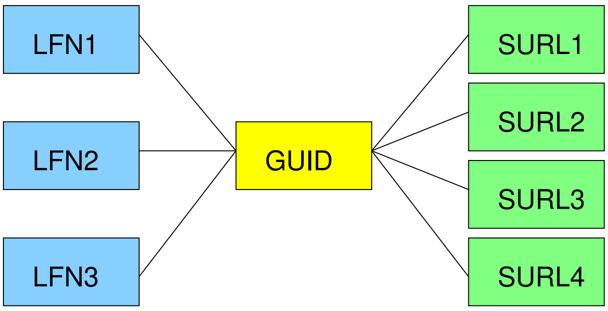

Figure 3.6: Relationship between LFNs, SURLs and GUIDs.

There are a number of different catalogue implementations currently available, including the EDG Replica Location Service, the Globus Replica Location Service, the LCG File Catalogue and the gLite FiReMan catalogue. The details of these will be given in Section 3.3.

Information Services

Some components of the information services were already mentioned in Section 3.2.1, namely the BDII and R-GMA. The BDII is in fact a more scalable version of the GIIS at each site, allowing it to hold global grid information. Figure 3.7 shows the overall structure of the LCG Information System, with the GRIS at each local resource supplying information to the GIIS at that site, which in turn feeds one or more BDIIs. Each VO may have its own BDII.

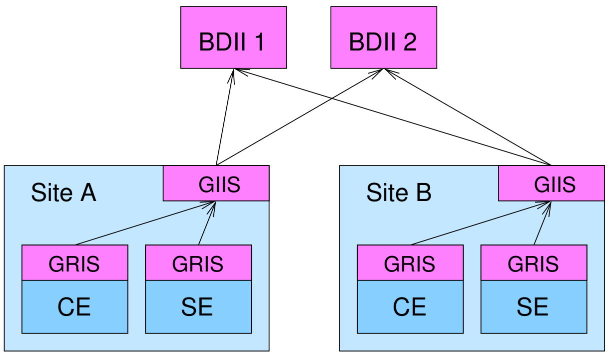

Figure 3.7: The LCG Information System.

While both the BDII and R-GMA are currently used by LCG-2 and gLite, the main information services for gLite are based on R-GMA. R-GMA is an implementation of the Grid Monitoring Architecture (GMA) specified by the GGF. The GMA design consists of three components: producers, consumers, and a directory or registry, as shown in Figure 3.8. Producers register themselves with the registry and describe the type of data they wish to make available. When a consumer is then searching for information, they query the registry for producers which provide that type of information. The consumer then contacts the producer directly for the data. R-GMA, as well as implementing this architecture, makes the whole information and

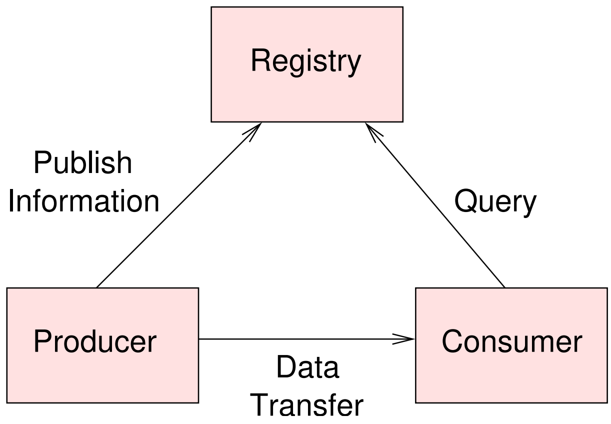

Figure 3.8: The Grid Monitoring Architecture (GMA).

monitoring system look like a single relational database which can be queried as such, with the registry hidden from the user.

The Logging and Bookkeeping Services are used to track job information, such as submission time, start time and execution time. Local logs are kept at the sites and the information then passed on to bookkeeping servers. Related to Logging and Bookkeeping are the Job Provenance Services. These are currently only at the prototype stage, but are responsible for keeping job information (such as the execution environment) on a longer timescale, so that job results can be verified, problems diagnosed and jobs re-run if necessary.

File Transfer Services

Finally, a service is needed to actually move data around the grid. In LCG-2, there was originally no specific file transfer service, with the onus on the user to do the data movement “by hand”. A set of tools, known as Radiant, has now been developed for robust data transfer between CERN and Tier-1 sites. gLite does have a transfer service, known as the File Placement Service (FPS) when it is used in conjunction with the FiReMan catalogue (see Section 3.3.3) and as the File Transfer Service (FTS) when used alone. Users submit data transfer requests to the FPS, which schedules them according to defined policies. A queue of transfers is used, allowing resending of any failed transfers, such as in a network failure. These file transfer services could be considered part of the data management middleware, but as their function is somewhat different from the replica management services discussed in the next section, they are not examined in depth.

3.3 Data Management Middleware

The previous section dealt with the LCG and EGEE middleware components in general. In this section, the data management middleware, much of which was mentioned previously, is examined in more detail. First, however, the EDG data management components will be described, as they were the initial basis for LCG and for the design of the OptorSim architecture which is presented in Chapter 5. The reasons for the change in LCG to the present components will then be discussed and the components themselves presented.

One of the most important concepts in the management of a data grid is *replication* of data. Replication is the process of copying data files to different locations on the grid, keeping track of all the replicas of a file so that they can be kept consistent with the original. The benefits of replication are faster access to data, and higher reliability as no site is a single point of failure. Many of the components described below are therefore used for the management of replicas.

3.3.1 Data Management in EDG

The EDG data management middleware uses Web services, with the code written in Java. There are three services: the *Replica Location Service*, the *Replica Metadata Catalogue*, and the *Replica Optimisation Service*. In addition, the *Replica Manager* is used on the client side to access these services. Each of them is now described briefly; further details may be found in [2].

Replica Manager

The Replica Manager (RM) client is the central access point to the data management tools, and its interactions with the various services are shown in Figure 3.9. As well as interacting with the data management services on behalf of the user, the RM must

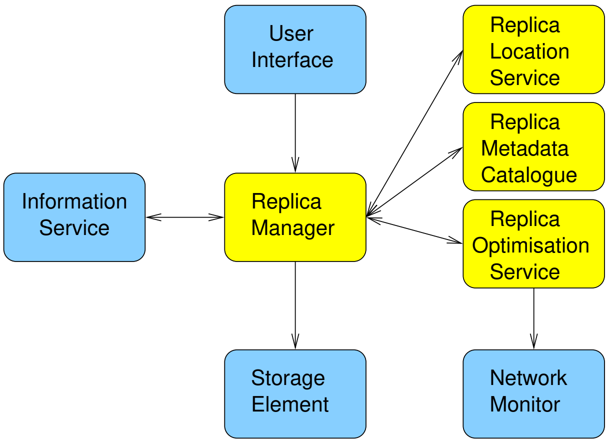

Figure 3.9: The EDG Replica Manager and its interactions with the data management services. From [2].

interact with several other grid components, including the Information Service, to discover the state of grid resources and the SEs, to access and manipulate data.

Replica Location Service

The file catalogue role in EDG was played by the Replica Location Service (RLS). When there are a number of replicas of a file in various locations, it is important to know where they are, and the RLS is responsible for keeping these mappings of logical to physical file names. The RLS framework [65] was developed jointly by EDG and Globus, and subsequently implemented separately by the two projects. The main differences are, firstly, that the EDG RLS was written in Java and the Globus version in C, and secondly, that the Globus RLS maps directly from LFNs to SURLs rather than going through the GUID as the EDG RLS does (see below).

Recalling from Section 2.3 that an RLS consists of two components, the Local Replica Catalogue (LRC) and the Replica Location Index (RLI), a possible RLS configuration is illustrated in Figure 3.10. In this configuration, not all the RLIs have information from all the LRCs. The pattern of subscriptions between LRCs and RLIs would be determined according to the usage patterns and requirements of the sites. It would also be possible to implement a hierarchical RLS, with an extra level

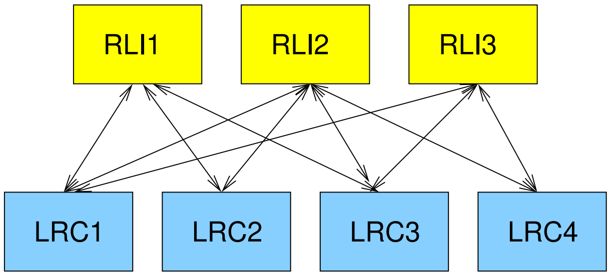

Figure 3.10: One possible configuration of the Replica Location Service. After [3].

of RLIs to which the first layer of RLIs subscribe. It is also possible to deploy the LRC as a central, standalone replica catalogue for the whole grid, and this is in fact the way in which it was deployed in the EDG testbed.

Replica Metadata Catalogue

While the RLS holds mappings from GUIDs to SURLs, the mapping from LFNs to GUIDs in EDG was done by the Replica Metadata Catalogue (RMC). Metadata can be loosely defined as "data about data"; the RMC held some items of system metadata, such as file size and ownership information, and about ten items of user-defined metadata. Thus, the RMC and RLS together provide the full catalogue service for mapping from LFNs to SURLs, as was discussed in Section 3.2.2.

Replica Optimisation Service

When a query is made for a particular file, whether to access it from its current location or to copy or replicate it elsewhere, there may be several replicas from which to choose. For efficient use of grid resources, and a quick response time to the user, it is important to choose the best replica, and this is the responsibility of the Replica Optimisation Service (ROS).

The implemented version of the ROS used the network monitoring services to find the network latencies between the requesting site and the candidate replicas. These were used to calculate the expected transfer times, and the replica with shortest access time chosen. The ROS could also be queried by the WMS to find which sites would be able to access the job's data files most quickly. In a future data grid, a ROS could use more advanced algorithms to automatically replicate files around the grid to give an optimal distribution of the data, and hence optimise use of grid resources as a whole.

3.3.2 Consequences of the LHC Data Challenges

The LHC experiments are running a series of "data challenges" to test their computing models and infrastructure in preparation for LHC start-up. The 2004 set of data challenges were the first to use the LCG-2 middleware in a realistic environment. The CMS collaboration, for example, aimed to reach a data-taking rate of about 25% of that predicted for the LHC, for a period of one month. ATLAS aimed to produce 30 TB of simulated events, which would then be reconstructed and distributed to Tier-1 and Tier-2 sites, followed by a test of the distributed analysis system; ALICE aimed to produce and analyse about 10% of a standard year's data sample; LHCb aimed to use LCG resources to provide at least 50% of the total production capacity for the data challenge. A number of problems and limitations in the data management middleware were thus exposed, as well as differences between the expected and actual usage patterns.

Replica Manager

The EDG Replica Manager faced significant performance problems, mainly because of the Java implementation. This meant that the Java Virtual Machine (JVM) was started up with each call to the RM, giving slow start-up times; starting the JVM took about 1 second each time. In addition, there was no support for bulk operations; that is, performing a large number of similar operations in the same procedure call. These two factors meant that while it was useful for interactive usage, it was too slow for production usage.

RLS and RMC

It also became apparent that the RLS and RMC were too slow for both inserts and queries [66], with operations involving both catalogues being particularly slow [67]. Although a C++ API was available for the catalogues, the command-line tools were available only in Java, again leading to performance loss from starting the JVM with each call. Also, the Web services approach was not used in the way which developers had anticipated. They were implemented such that a remote procedure call (RPC) was performed for each low-level operation; users, however, wanted to send high-level or multiple commands in a single RPC. As this was not available, the cumulative overheads from a large number of low-level RPCs led to the performance loss which was seen. Multiple queries by GUID took of the order of seconds per file, for example.

In addition, the pattern of queries was different from that expected by the developers. Whereas the two-catalogue structure of the RLS and RMC had been adopted with the expectation that users would perform some pre-selection using the RMC to get the GUID, then allow the Replica Manager or Resource Broker to look up the physical location at run-time, queries for the SURL were made based on metadata attributes such as creation date. This required queries to span both catalogues, and thus led to performance loss; CMS, for example, found it could take several hours to query a dataset collection in this way.

Other problems were due to missing functionality. First, as for the Replica Manager, there was lack of support for bulk operations. There was also lack of support for user-controllable database transactions, which meant that if an operation involving multiple files failed part of the way through, the user had to manually undo the changes from the database. Another design problem was that the EDG catalogues did not have a hierarchical file namespace. This meant that to search for a file, a full table scan was required of the database table which held the LFNs, which does not scale to large catalogue sizes. Similarly, because there was no support for database cursors (which allow traversal of records in a results set), queries which returned many results required re-running the same query on the database many times.

Storage Element

The EDG "classic" SE was already mentioned in Section 3.2.1 as using GridFTP for data access at a site. In fact, each SE consisted of a single GridFTP server, with different mount points for the file system defined for different VOs. This meant that all file requests to an SE went through a single server, which led to problems with scalability.

In response to the experiment feedback from the data challenges, highlighting the above problems, a number of new data management components were developed by LCG as an immediate replacement for the EDG tools. These are described in the next section.

3.3.3 Data Management in LCG

GFAL and lcg_utils

The Grid File Access Library (GFAL) has already been introduced in Section 3.2.1 as a low-level interface to storage systems for interactions with the underlying file system. It was also designed with interfaces to the EDG catalogues and Information System, in order to be accessible by the rest of the grid storage infrastructure. When the problems with the EDG Replica Manager were discovered, GFAL then became a good foundation on which to develop a replacement.

This replacement for the Replica Manager consists of a set of command-line tools, known as *lcg_utils*, which provide the same functions and accept the same command-line arguments as the Replica Manager. Both GFAL and lcg_utils are written in C and lcg_utils also provides a C API which the experiments can use to access the replica management functions directly from their applications. Apart from the original Replica Manager functions, a number of extra features were added at the experiments' request. There is no Replica Optimisation Service equivalent in LCG at present, while the focus is on getting a grid service running before the LHC starts, but replica optimisation will be an important issue in future.

Disk Pool Manager

The Disk Pool Manager (DPM), also mentioned in Section 3.2.1, was developed as a storage solution for the smaller Tier-2 sites in LCG, with "small" being defined as between 1 and 10 TB of disk. It fulfils the criteria of the SRM specification, with support for different types of file space - *volatile* space, which is for short-term work near the worker nodes, and *permanent* space, which is for long-term storage. It also supports multiple replicas of a file within a disk pool, to avoid overloading disks which hold popular replicas.

The DPM share the same code base as the LCG File Catalogue (below), with both based on the CASTOR name server code. This means that in future, the DPM could act as the local part (equivalent to the LRC in the RLS architecture) of a distributed grid catalogue, with the LCG File Catalogue playing the role of the RLI.

LCG File Catalogue

The LCG File Catalogue (LFC) has a completely different architecture from the RLS framework which was described above. Rather than two different catalogues, it is a single catalogue which stores both logical and physical mappings for the file in the same database. Rather than a flat namespace, there is a hierarchical namespace of LFNs which are mapped to GUIDs, which in turn are mapped to SURLs. It is designed to mimic a UNIX filesystem, with a directory structure and function calls with which the user is familiar, such as `creat`, `mkdir` and `chown`. Figure 3.11 shows the architecture of the LFC and Figure 3.12 shows how these entities are related in the schema.

The LFN is the main key for the database. Symbolic links (symlinks) are used to add extra LFNs or *aliases* to the GUID. One field of user metadata is defined on the LFN, while system metadata about the replicas (such as creation time, last access time, file size and checksum) are stored as attributes on the LFN.

Addressing those features which were absent in the EDG catalogues, the LFC supports database transactions and cursors. In database terminology, a *transaction* is an atomic unit of work containing one or more SQL statements. A transaction begins

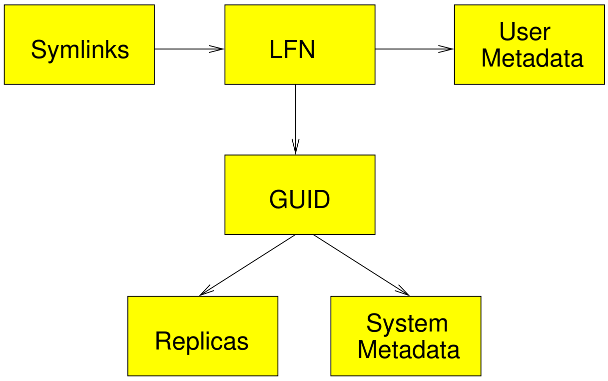

Figure 3.11: Architecture of the LCG File Catalogue.

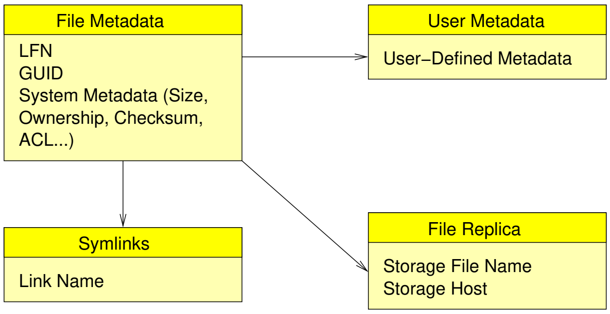

Figure 3.12: Schema diagram for the LCG File Catalogue. From [4].

with the first executable SQL statement and ends when it is *committed* (applied to the database) or *rolled back* (undone from the database). In the LFC, the transaction methods are exposed in the API so that the user can explicitly start a transaction, perform a number of operations then commit to the database (or abort and roll back if there is an error). Otherwise, the database auto-commits immediately after each operation finishes. As there is only one catalogue, transactions involving both LFN and SURL operations are possible, which was impossible with the EDG catalogues.

Timeouts and retries are implemented, to handle loss of network connectivity to remote sites, and support for bulk operations is currently underway. Authentication is by GSI, with the distinguished name (DN) taken from the user's grid certificate and mapped to a local uid/gid pair which is then used for authorisation. VOMS is also being integrated, with VOMS roles mapped to a list of group IDs in the LFC. The uid/gid pairs are used for authorisation by means of the file ownership information stored by the LFC, with both standard UNIX permissions and POSIX-compliant Access Control Lists (ACLs) on each catalogue entry.

The LFC implementation is entirely in C. There is not currently a Web services implementation, but this may be added in future. It consists of a server and client; the server is a multi-threaded daemon with a relational database backend, and the client may also be multi-threaded. Oracle and MySQL are the supported databases. Clients exist for GFAL, POOL and lcg_utils; POOL [67] is the Pool Of persistent Objects for LCG, a framework for the LHC experiments to navigate distributed data without knowing details of the underlying storage technology. Initial results of the performance of the LFC compared to the EDG catalogues were presented in [68], but the full results of performance testing are presented in the next chapter.

The gLite FiReMan Catalogue

In addition, gLite have also produced a file catalogue, the File and Replica Manager (FiReMan) [69], which provides similar functionality to the LFC while being architecturally different. It uses a Web services approach, with clients communicating with the server via the SOAP protocol. It also provides a hierarchical, UNIX-like namespace. Bulk operations are supported, as is GSI authentication and authorisation. A performance comparison of the LFC and FiReMan is presented in [70], showing that while the LFC is faster for single operations, FiReMan is faster when bulk operations are used. Both catalogues have currently been deployed for testing by the LHC experiments, as it is not yet clear what their final requirements will be. They are by no means mutually exclusive, and it may be that some experiments will choose one implementation and some experiments the other, so both are being made available.

3.4 Summary

This chapter has given an overview of the present set of LCG middleware, collectively known as LCG-2, and its relationship to the EGEE middleware, gLite. It is in some sense a snapshot of the tools at this moment in time, as the two projects are converging to give a production grid for LHC computing. As the LHC experiments refine their requirements and decide on their preferred solutions, there will doubtless be some evolution of the middleware over the next two years.

While the main components were described briefly, more attention was given to the data management tools and in particular those for replica cataloguing. The effects of the LHC data challenges on the EDG replica management tools were described, and the replacement set of tools which were developed; in particular, the LCG File Catalogue was described.

Chapter 4

Performance of the LCG File Catalogue

This chapter presents a series of performance tests of the LCG File Catalogue (LFC), which was developed to replace the Replica Location Service (RLS) in LCG. First, the test methodology and setup is given, then the results of testing with a single LFC client, covering the basic operations of adding entries, querying and deleting as well as more advanced operations. Tests with multiple LFC clients are then presented, followed by an examination of the effects of security and then an evaluation with respect to the the Globus and EDG RLS implementations.

4.1 Test Methodology and Setup

Clients can interact with the LFC either by calling methods from its API (Application Programming Interface) or by using one of its interfaces. For the performance tests, programs were written in C which called the API directly. The general format of these tests was as follows: command-line arguments were parsed and any necessary structure set up in the catalogue. A timer was then started and as many threads as were specified in the command-line forked off. Each thread executed the operation under test a specified number of times before returning; when all threads were finished, the timer was stopped and the overall rate or mean time (depending on the particular test) was output. Some tests were more finely instrumented so that timings of individual operations and of individual threads could be output. Each set of tests was run from a Perl script, which would typically call the C program several times, with several thousand operations each time, and take the overall mean of the operation times.

In the following sections, the results of the performance tests conducted on the LFC are presented, first with a single client machine and then with many clients connecting to the same server. Results with the secure version of the LFC are then presented. In all cases the server was a dual 1 GHz Pentium III processor with 512 MB of RAM, connected to an Oracle 10*g* database which was running on a dual 2.4 GHz Intel Xeon. The server operating system was RedHat Linux 7.3 and for most of the tests described below it was configured to run with 20 threads. Note that all references to "server" in this chapter refer to the LFC server and not the Oracle database server.

For the single client tests (Section 4.2), the client machine was a Pentium III 853 MHz processor with 128 MB of memory, running Red Hat Linux 7.3. Server and client machines were connected by a 100 Mbit/s local area network.

For the multiple client tests (Section 4.3), 10 identical 1 GHz Pentium IIIs were used. Each had 512 MB memory and were running CERN Scientific Linux 3.0.3. The server was as for the single client tests, and all the machines were on the same 100 Mbit/s local area network.

4.2 Tests with a Single Client

Using a single client connecting to the server as described in Section 4.1, each of the most important catalogue operations (insert, query, delete) were tested in some detail, as well as other important aspects such as reading directories, finding replicas and using symbolic links. In this section, the results of these tests are presented, categorised into the three broad areas of inserting entries, deleting entries and querying for information from the catalogue.

4.2.1 Inserting Entries into the Catalogue

Insert time. First, the mean time to insert a single entry into the LFC was measured as the number of entries in the catalogue was increased, with a single client thread. Starting from a low number of entries in the LFC ($\mathcal{O}(100)$), new entries were inserted, with each insert time recorded, until there were just over 40 million entries. The mean of every 1000 entries was then taken and plotted as shown in Figure 4.1.

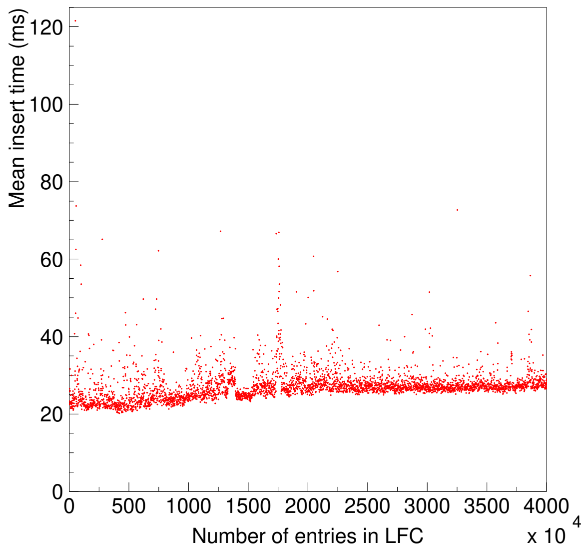

Figure 4.1: Mean insert time with increasing catalogue size.

The results show that there is very little dependence of insert time on the number of entries in the catalogue. This is what one would expect at the database level, as inserting a row should not have much dependence on database size, unlike querying for a row. Up to about 5 million entries, the mean insert time is about 22 ms, after which it increases slightly to a plateau of about 27 ms, with no further rise as far the limit of the test.

The results also show a number of large spikes; in other words, the distribution of insert times is long-tailed. This could be due to delays on the network, in database access or both, and is a systematic effect which affects the rest of the plots in this chapter. The plots which follow show the calculated statistical error, but in many cases this is small, so the systematic effect should be kept in mind while viewing the results.

Transactions. The role of user-controllable database transactions was explained in Chapter 3. To test the performance when transactions are used, the number of operations inside a single transaction was varied and the mean insert time measured for different numbers of client threads, again with 20 server threads. Figure 4.2 shows the results.

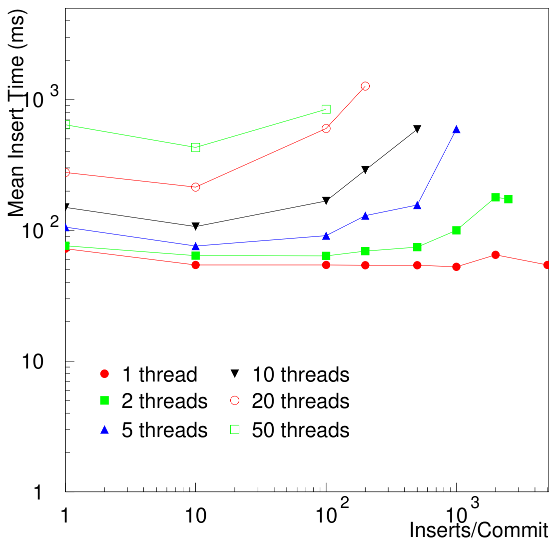

Figure 4.2: Mean insert time when using transactions, with increasing number of inserts per transaction.

This shows that explicitly using transactions increases the insert time by at least a factor of two; with a single client thread at the optimal number of operations per transaction, the mean insert time is about 54 ms whereas without transactions the time is about 27 ms (as shown in Figure 4.1). These results also show that the optimal number of operations per transaction is the range 10 - 100, depending on the number of client threads. The number of data points is lower for higher numbers of threads because of the structure of the test, in which the total number of inserts remained fixed. This could have implications for the LHC experiments in the future. It is not yet known whether they will use single or bulk inserts, but if bulk inserts (e.g. of all the files in a dataset) are used, these results indicate that the number of inserts in a bulk operation should be kept within the optimal range.

To understand the reasons for the performance loss when transactions are used, the times for each part of the transaction - waiting to start, running the operations and finishing - were measured and the above experiment repeated. The results from this are shown in Figure 4.3.

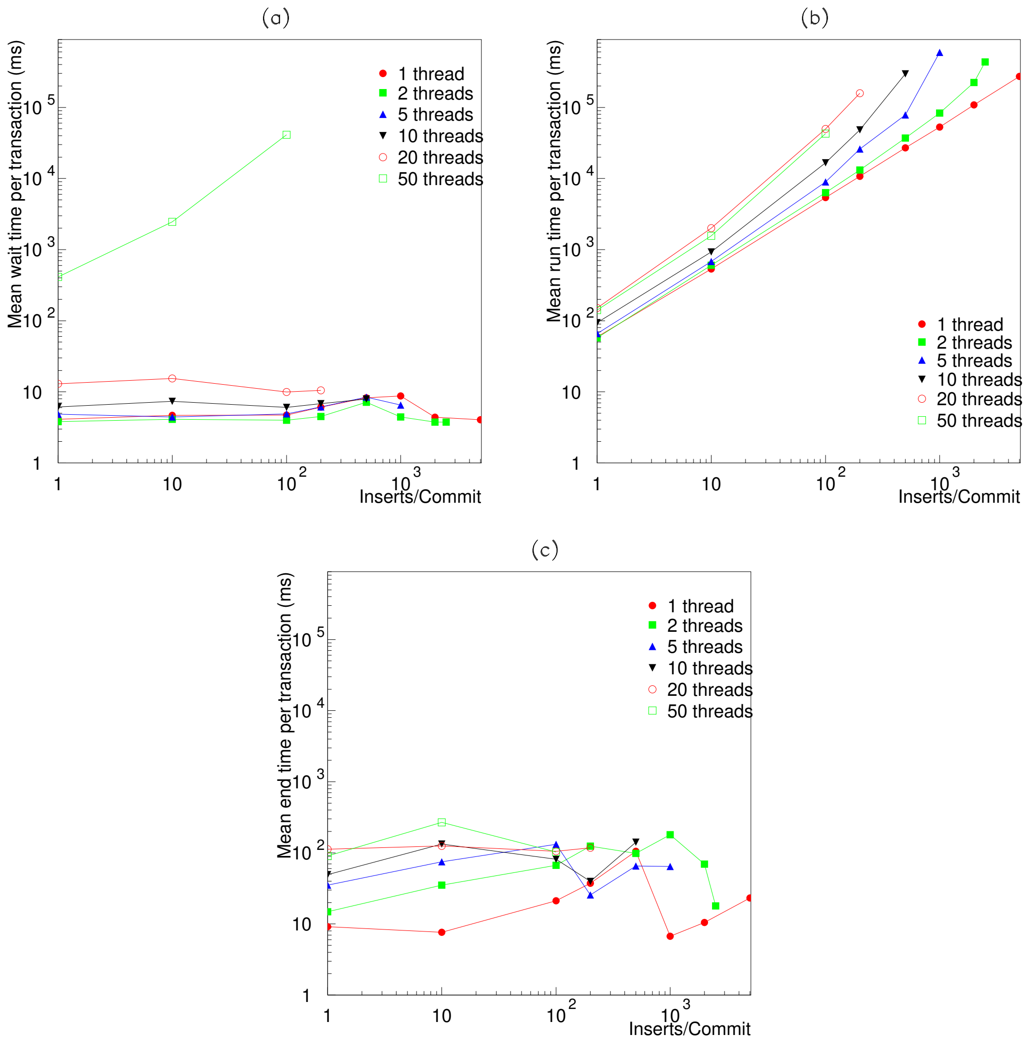

Figure 4.3: Mean time spent by transactions in (a) waiting, (b) running and (c) finishing.

This shows that while the number of client threads is less than or equal to the number of server threads, the time each thread needs to wait before starting a transaction is between 5 and 10 ms. When there are more client threads than there are server threads, the wait time increases to upward of 50 ms per transaction. The time spent running is almost directly proportional to the number of operations in the transaction, as might be expected. The time spent ending the transaction, however, varies randomly between 10 and 100 ms and when this is combined with the wait time, it accounts for much of the performance loss. This also indicates that the bottleneck is in the Oracle database, perhaps in the choice of database parameters, rather than in the LFC code.

Changing into the working directory. Another of the features of the LFC is the ability to change directories using a `chdir` operation, and in particular to change into the current working directory. When performing a large number of operations within the same directory, it should be faster to change into that directory before performing all the operations, as permissions would only be checked once (when entering the directory) rather than for every operation. The next test aimed to verify this and quantify the performance gained by changing directory.

The mean of 5000 inserts was measured for increasing numbers of subdirectories in the path of the LFN to be inserted, with and without performing the directory change, and for various numbers of client threads. Again, the server was running with 20 threads and there were about 250,000 entries in the catalogue. Having more subdirectories in the LFN path length should increase the difference between performance with and without changing directory, as permissions are checked for every directory in the path.

The results, shown in Figure 4.4, indicate that the `chdir` is working as intended. When the absolute path name is used, i.e. `chdir` is not used, the mean insert time increases fairly linearly with the number of subdirectories in the path. When `chdir` is used, the mean insert time is independent of the number of subdirectories. The results show that the performance gain from using `chdir` is significant; with a single client thread and a path containing ten subdirectories, changing directory more than

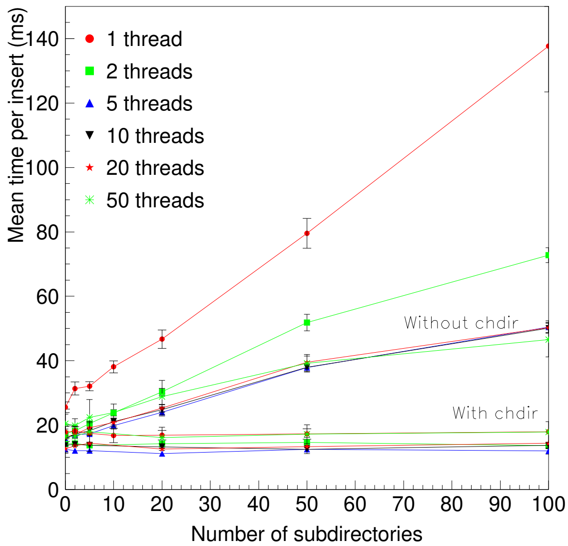

Figure 4.4: Mean insert time with and without changing into the working directory, with increasing number of subdirectories in the path.

halves the time for an insert. If a large number of operations were to be performed in the same directory, it would therefore be highly beneficial to use `chdir`. All measurements of insert and delete rates shown henceforward were made using such a change of directory.

Insert rate. Operation rate is a better measure of throughput than mean time, so the insert rate was measured for varying numbers of client threads. For this test, there were approximately 1 million entries in the LFC and the server was running with 20 threads. Figure 4.5 shows that with one client thread, the rate is about 75 inserts per second; as more client threads are added, the rate increases to a maximum of about 220 inserts per second until the number of client threads is equal to the number of server threads. Although not shown in this graph, after this point the rate gradually drops again.

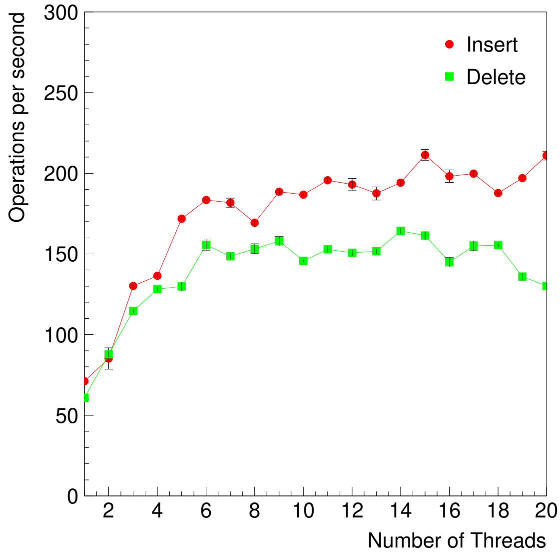

Figure 4.5: Insert and delete rates for increasing number of client threads, LFC with 1 million entries.

4.2.2 Deleting Entries from the Catalogue

Next, the performance of the LFC in removing entries from the catalogue was examined. As LHC experiment users are far less likely to delete than to insert or query entries, this was examined in considerably less detail. The delete rate was measured as the number of client threads was varied, with 20 server threads and about 1 million entries in the catalogue, and these results are also shown in Figure 4.5.

This shows that the delete rate follows a very similar pattern to the insert rate, with an initial increase as more client threads are added and saturation of the server at about 160 deletes per second. The rates achieved are 70-80 % of those for insertion. Measuring the time for an individual delete operation gave a mean for a single client thread of 22 ms per delete.

4.2.3 Querying Entries in the Catalogue

Querying the catalogue for entries will be the most common operation after the initial production phase at the LHC and the query performance is thus very important. In the following tests, a query is defined as finding a given LFN in the catalogue and performing a `stat()` operation on it. This returns all the associated metadata such as file size, checksum, last modification time and so on, as well as checking permissions, and is illustrated by the pseudo-code in Figure 4.6.

```
foreach ( component of LFN pathname )
   get file metadata
   check entry permissions
if ( GUID is supplied )
   check that GUIDs match
return LFN and file metadata
```

Figure 4.6: Pseudo-code for a `stat()` operation

The query rate was measured with an increasing number of client threads. As in the insert rate and delete rate experiments, the server was running with 20 threads and there were about 1 million entries in the LFC. Figure 4.7 shows the results, with the insert and delete rates from Figure 4.5 for comparison. The pattern of results is similar, but a maximum rate of about 275 queries per second is reached, which is slightly higher than the insertion and deletion rates.

The time for an individual query was also measured, with different numbers of client threads, for increasing catalogue size from about 50 entries to 1 million entries. This is shown in Figure 4.8, and it is clear from this that the query time is independent of the size of the catalogue, up to the limit of the test. With a single client thread, for example, the mean query time is about 12 ms. It is also clear that the more client threads there are, the longer an individual query response time is, although the overall rate of query processing is the same. The large increase in time between 20 and 50 threads is due to the server being configured with 20 threads, so there can be long wait times before a server thread is available.

Reading directories. Reading through files in a directory is another important use case for users querying for information from the catalogue. The total time to

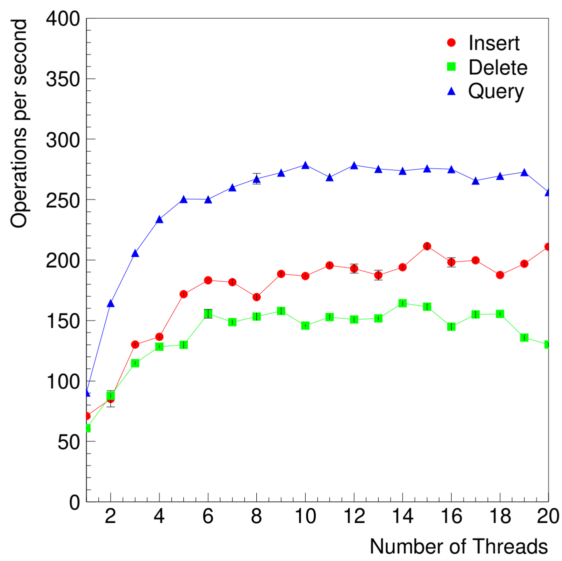

Figure 4.7: Query, insert and delete rates for increasing number of client threads, LFC with 1 million entries.

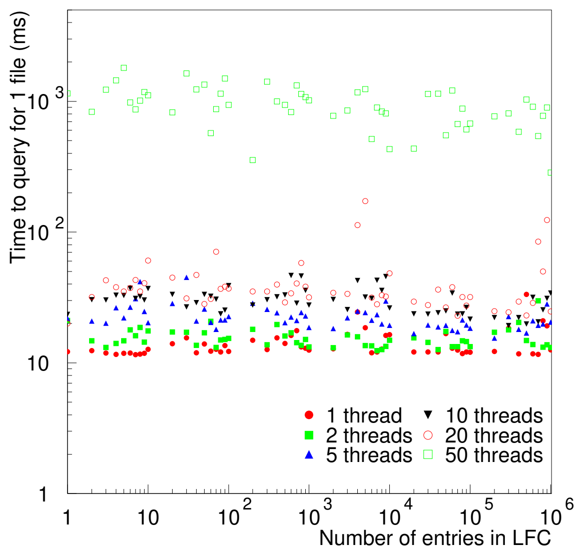

Figure 4.8: Query time for increasing number of LFC entries and client threads.

perform such a `readdir()` operation was therefore measured for increasing numbers of files in a directory, from 1 to 10000, with the results shown in Figure 4.9. The

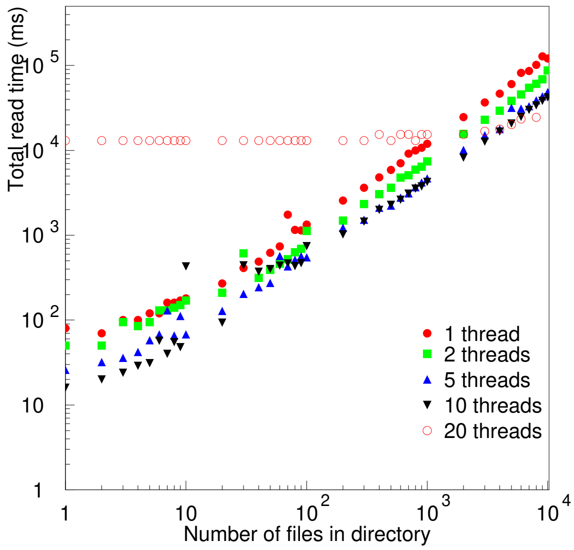

Figure 4.9: Total read time with increasing directory size, for varying number of client threads, 20 server threads.

time taken to read the whole directory is directly proportional to the number of files in the directory, until the number of client threads matches the number of server threads. At this point, the time spent by a thread waiting for another thread to finish dominates, until the directory is sufficiently large (about 2000 files) for the read time to dominate. The contribution of an indvidual entry to the read time is about 15 ms.

Symlinks. Symbolic links, or symlinks, are used in the LFC as aliases of an LFN. It is therefore important to investigate the performance when symlinks are used. As for the `readdir` test described above, the total time to list and perform a `stat()` operation on each symlink in a directory was measured for increasing numbers of symlinks in the directory. For this test the server was running with only 6 threads, and there were about 50,000 entries in the catalogue. The results are shown in Figure 4.10. Again, the time taken is proportional to the size of the directory, as long as the number of client threads is lower than the number of server threads.

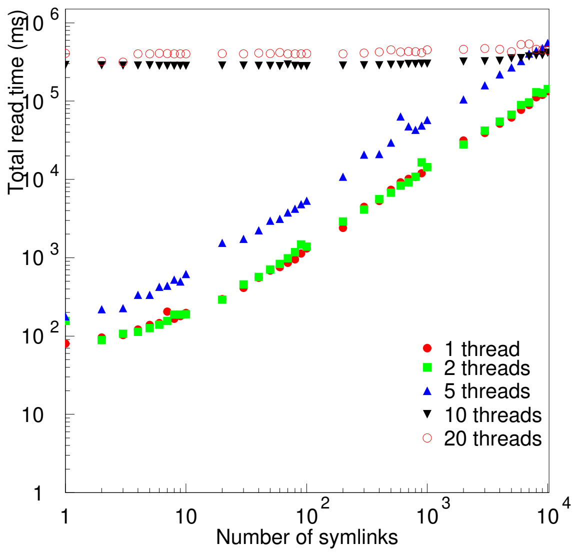

Figure 4.10: Total time to list and `stat()` all symlinks in a directory, for varying number of client threads, 6 server threads.

Next, a test was made of the time to get to the original file (the LFN) by traversing a chain of symlinks. In other words, the LFN is referenced by a symlink, which is referenced by another symlink, and so on. The total time to reach the LFN for increasing length of symlink chains is shown in Figure 4.11. As for the previous test, there were about 50,000 entries in the LFC but in this case the server was again running with 20 threads.

It can be seen that the time taken to reach the LFN varies linearly with the length of the symlink chain. With one client thread, for example, each additional symlink adds about 15 ms to the total time to reach the original file. There is therefore a performance loss associated with the use of symlinks as aliases for an LFN. As it is unlikely that long chains of symlinks would be used in reality, however (there may be many aliases of an LFN, but they are more likely to refer directly to the LFN than to other aliases), this should only be a small effect.

As an example, consider a query which wants all replicas of a particular file, where

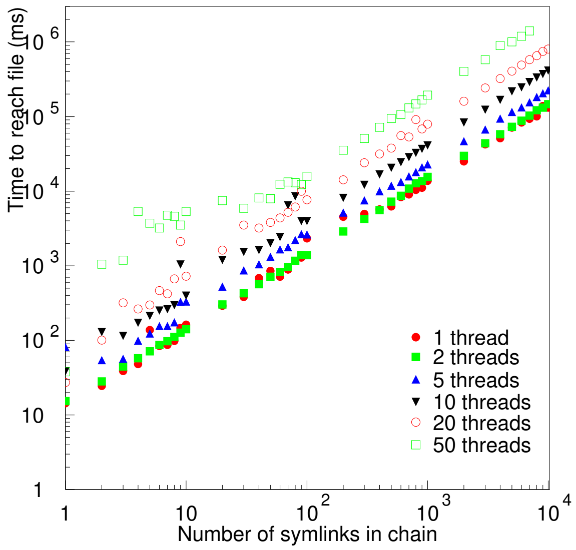

Figure 4.11: Total time to traverse a chain of symlinks, for increasing number of symlinks and varying number of client threads, 20 server threads.

the filename is a symlink to the LFN. Suppose also that there exist ten replicas of the LFN. Then the breakdown of times taken for the whole query would be approximately as shown in Table 4.1, and it can be seen that resolving the symlink would account for only 7% of the total query time.

Stage of query	Time (ms)
Finding symlink	12
Resolving symlink to LFN	15
Listing 10 replicas and returning `stat()` information	190
Total	217

Table 4.1: Approximate times for different stages of a query using symlinks.

Replicas. When finding replicas of an LFN, the time taken to list all the possible replicas is obviously important. A test was performed of the time to list all replicas of a file, for an increasing number of replicas. Figure 4.12 shows that the total time is directly proportional to the number of replicas, up to 10,000 replicas (which is much higher than would ever be likely in reality). With 20 client threads, the time taken remains constant up to about 1000 replicas, and in fact is lower than the time taken with 1-10 threads above 600 replicas; the reason for this anomaly is unknown, but may be due to the underlying non-stochastic network and database processes mentioned earlier. The average time contributed by each replica is about 19 ms, and the LFC is therefore scalable with respect to the number of replicas of a single LFN while the number of client threads is lower than the number of server threads.

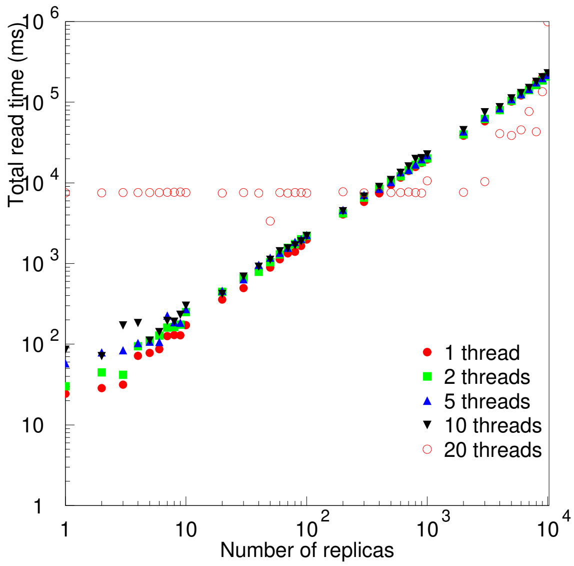

Figure 4.12: Total time to list and `stat()` all replicas of a file, for varying number of client threads, 20 server threads.

Figures 4.9- 4.12 all show a similar pattern of results. In each, the total time scales linearly with the size of the operation being performed, until the limit of threads on the server is reached. At this point, the time taken is more or less constant until the size of the operation is large enough to dominate over the time spent waiting for a server thread to become available. Thus, the LFC's behaviour is shown to be both self-consistent and consistent with expectations.

Tuning the Query Rate. For all the tests described above, the size of the LFC reply buffer was set at 4 KB, thus limiting the volume of data which could be transferred at any one time. To investigate this effect, the buffer size was varied and the time to perform a `readdir()` on a directory containing 100,000 entries was measured. A large directory size was chosen to ensure that a large volume of data would be returned from the server to the client.

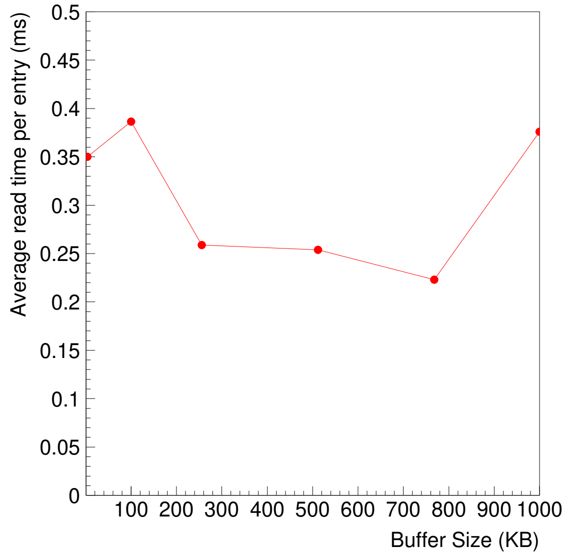

Figure 4.13: Average read time per entry for a directory with 100,000 entries, with varying reply buffer size.

Figure 4.13 shows the results. This test was only performed once, and so the error on these points is unknown. It is clear, though, that a larger buffer size gives better results, with the optimum in this case being about 800 KB and giving about 40% reduction in the total time taken. Further tuning of this and other parameters, depending on the particular system, are likely to improve the query performance even more.

4.3 Tests with Many Clients

Having run the tests in Section 4.2 on a single client, a subset of them were selected to be run simultaneously on more than one client, to check the scalability of the LFC with respect to the number of clients connecting to the server. The machines used are those described in Secion 4.1.

First, the insert and query rates were measured for increasing numbers of clients, without transactions. The test scripts were the same as those used in the single client tests and each client was running with 10 threads. The measurement of rate, however, was taken from the server log rather than the client side, taking care to start measuring only when all clients had started running.

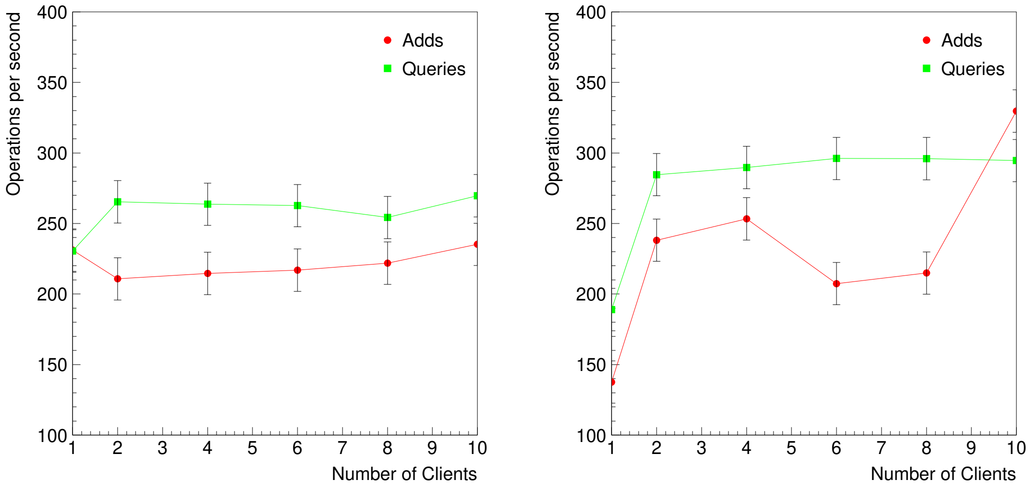

Figure 4.14: Operation rates with increasing number of clients, (a) without transactions and (b) with transactions, 100 operations per transaction.

Figure 4.14(a) shows that without transactions, operation rates are essentially independent of the number of clients, up to the limit of 10 clients measured by the test. The standard error on the data points is about 10-20 operations per second, but is not available for all the data points; the error bars shown are therefore an estimated average of 15 operations per second.

The tests were then repeated using transactions, with 100 operations being performed in each transaction. The results for this are shown in Figure 4.14(b). This shows a similar pattern to the situation without transactions, with little significant change in rate with the number of clients, with the exception of the measurement with 10 clients. The greater the number of clients, however, the more difficult the measurement was to make, and so the error on this point is larger than for the lower numbers of clients. The increase in rate between 1 and 2 clients is more significant, and occurs because with 1 client, the server is not fully utilised and thus not running at its maximum rate. With more than one client running 10 threads each, the server is running at maximum load and the rate, as seen from the server side, is constant, regardless of what the total number of client threads is or whether transactions are being used. By looking at the rates on the client side, it is seen that the load is being shared equally among the clients. 260-270 queries per second without transactions, for example, is consistent with the 275 queries per second which were measured from the client side in the single client test.

To summarise this section, the LFC scales well with an increasing number of clients, both with and without transactions. In fact, up to the limits of the tests presented here, the server performance is independent of the the number of clients, while each client is getting an equal share of the server's resources.

4.4 Performance with Security

The previous two sections presented performance measurements for the LFC when running insecurely. In this section, a subset of these tests are repeated with security activated, to measure the overhead introduced by security and to determine the best mode of operation for a secure LFC.

First, the results are presented for the insert rate. Figure 4.15 compares the rates with and without security, both when transactions are not explicitly used and when transactions are used with 100 inserts per transaction.

It is clear from Figure 4.15(a) that security introduces a very large overhead when transactions are not used. The insert rate is reduced to under 10 inserts per second, due to the credentials being checked for every individual operation.

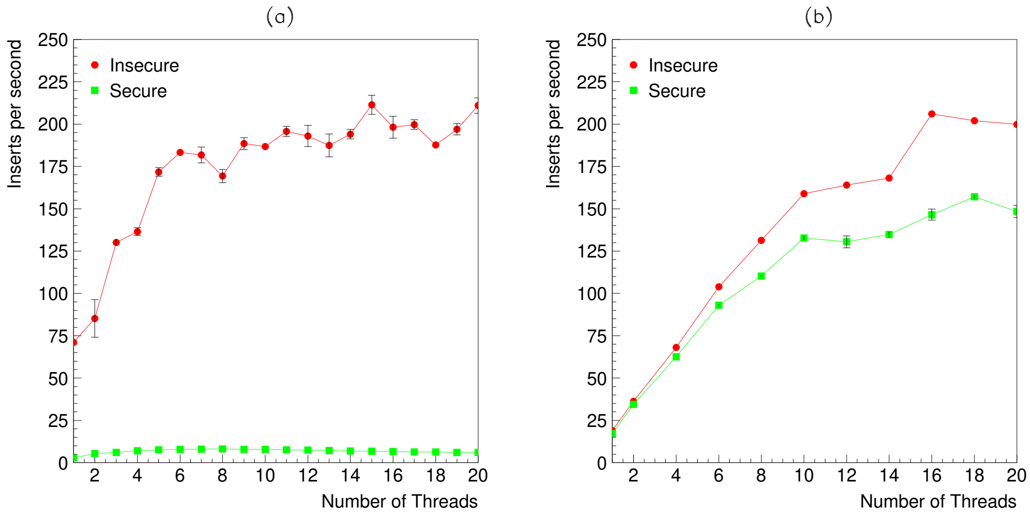

Figure 4.15: Insert rates with increasing number of client threads, with and without security, (a) without transactions and (b) with transactions, 100 operations per transaction.

When transactions are used, however, the security overhead is much reduced. Figure 4.15(b) shows that the insert rate with security is about 80% of the insecure rate, when 100 inserts are performed per transaction.

The effects of security on the query rate are very similar, as Figure 4.16 shows. Again, if transactions are not used, the query rate is reduced to below 10 queries per second whereas with 100 queries per transaction, the query rate is reduced by only 20-30%.

These results suggest that for a secure LFC to be used efficiently, it should either be used with transactions (and with an optimal number of operations per transaction) or sessions should be implemented, which would allow a user to perform many operations with only a single checking of the credentials. Transactions would be preferred for a relatively small number of operations, while sessions would perform better for a large number of operations.

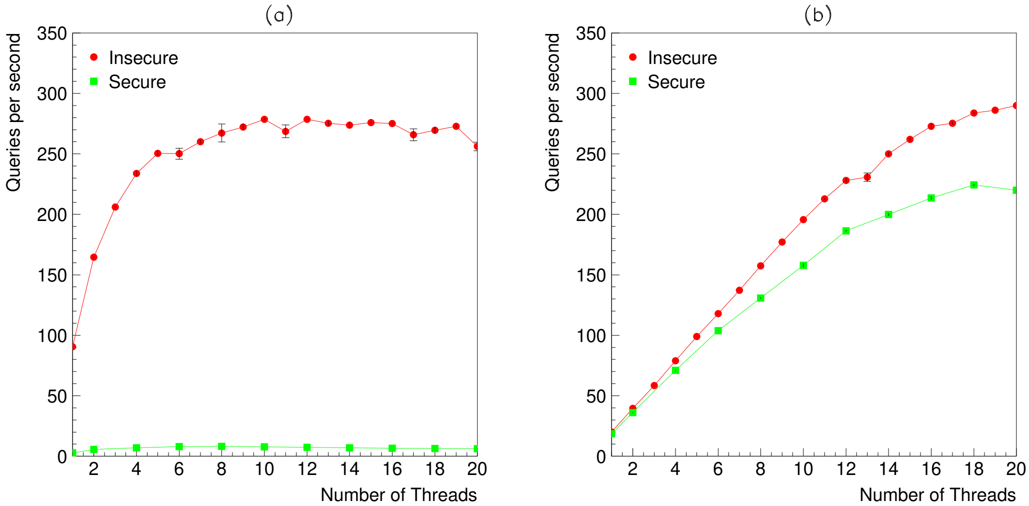

Figure 4.16: Query rates with increasing number of client threads, with and without security, (a) without transactions and (b) with transactions, 100 operations per transaction.

4.5 Comparison with EDG and Globus RLSs

4.5.1 The RLS Framework

The Replica Location Service (RLS) framework was designed jointly by the EDG project and the Globus Alliance [65]. The two teams then went on to develop their own implentations of this framework.

There are two basic components: the Local Replica Catalog (LRC) and the Replica Location Index (RLI). The LRC at a site contains mappings from logical names to replicas at that site. The RLI is a higher-level catalogue which contains mappings from logical names to the LRCs in which they can be found. The RLI is updated with information from its LRCs using a *soft state* update mechanism; that is, the state is periodically refreshed, and any information which is not updated is expired. In the Globus RLS, the logical names correspond to the LFNs in the LFC; the EDG RLS, however, stores GUIDs and thus requires another catalogue, the Replica Metadata Catalog, to define LFNs and map from them to the correct GUID.

The Globus RLS is written in C whereas the EDG RLS is written in Java. For both RLS implementations, security is by Grid Security Infrastructure (GSI) authentication as described in Chapter 2.

4.5.2 Comparison of Hardware Used

Although it would have been desirable to run tests for all three catalogues on the same hardware, this was not practicable. Instead, the published results for the Globus RLS, from [3], and for the EDG RLS, from [2], were compared with the LFC results which have been presented in this chapter, at every point where such a comparison was applicable. The Globus results quoted use Globus Toolkit 3.0; however, there have been no substantial changes to the RLS for Globus Toolkit 4.0, so the performance results should be unchanged.

A comparison of the hardware used for the LFC tests with that used in the EDG and Globus RLS tests, in terms of the SPEC CINT2000 values [71] of the machines used, is shown in Table 4.2. Dual CPUs are indicated by a 2× in front of the CINT 2000 value for a single CPU. When comparing the results for the different catalogues, the difference in values for the server in particular should be kept in mind. As a rule of thumb, these could be applied as a scaling factor, scaling the LFC and EDG results by a factor of 2 to compensate for the more powerful Globus RLS server.

	LFC	Globus RLS	EDG RLS
Server	2 × 420	2 ×810	2 × 420
Single Client	420	2 × 220	2 × 420
Multiple Clients	400	2 × 220	2 × 420

Table 4.2: Approximate SPEC CINT2000 values for test machines.

4.5.3 Insert Performance

Globus RLS. Chervenak et al report in [3] that with 1 million entries in their Local Replica Catalog (LRC), they achieve rates of above 700 inserts per second when database flush is disabled, and about 84 inserts per second when database flush in enabled, for a single client with multiple threads. Database flush is the process whereby the database backend immediately writes transactions to the physical disk. Disabling this gives a risk of database corruption and is not normally considered acceptable for high energy physics experiments. Comparing the case where database flush is enabled, then, with the results for the LFC which were presented in Section 4.2.1, it is seen that the maximum rate with the LFC is about 220 inserts per second. This is more than twice (and if the scaling factor is applied, more than four times) the rate achieved with the Globus RLS.

EDG RLS. The results presented for the EDG RLS by Cameron in [2] are in terms of times rather than rates. There, the time per insert is plotted as a function of the number of entries in the LRC, which can be compared directly with Figure 4.1. With the EDG Java client, the average time per insert was about 18 ms until there were about 200,000 entries in the LRC, after which it increased, reaching 40 ms per insert by the time there were 500,000 entries. The EDG C++ client was more stable, with a constant insert time of about 17 ms up to 500,000 entries, but it is not known how it would behave at higher orders of magnitude of the catalogue size. The LFC, on the other hand, had a stable insert time of about 27 ms per insert up to 40 million entries, at which point the test was stopped. The LFC measurement was also made without using the `chdir()`, which would reduce the insert time further.

4.5.4 Delete Performance

With the Globus RLS, no directly comparable measurements of the delete time or rate are available. With the EDG RLS, the only available measurement is for the case with 1 client thread, where with both Java and C++ clients the delete time is 25-29 ms. With the LFC, the mean delete time with a single thread is about 22 ms, which is only slightly faster.

4.5.5 Query Performance

Globus RLS. As Figure 4.7 shows, the maximum query rate achieved by the LFC in these tests was about 275 queries per second. Applying the rough scaling factor puts this up to about 550 queries per second. The Globus RLS results are considerably higher, reaching over 2000 queries per second; however, in this instance it is hard to compare the numbers directly. Firstly, the Globus RLS query was for doing a simple lookup of LFN to SURL whereas the LFC performs a `stat()` operation on the LFN and returns items of metadata as has already been discussed. Observation of actual usage patterns seen in the LHC experiment data challenges [67] suggests that users rarely perform only a lookup, but tend to request items of metadata with the SURL. The checking of permissions also has a significant effect. Secondly, tuning of parameters to optimise the query rate would give significant improvements, as was shown in Section 4.2.3.

EDG RLS. The results reported for the EDG RLS give a query time for a single client thread of about 16 ms for both Java and C++ clients. With the LFC, the query time is about 10 ms with a single thread, which is significantly faster. It should also be considered that the LFC performs an SQL `ORDER BY` statement when listing the entries in a directory, allowing the implementation of safe iterators and cursors. With large directories, this naturally has an impact on performance, but iterators and cursors are user-requested features which were not present in the EDG RLS and this is therefore the price which must be paid.

4.5.6 Security

It was shown in [2] that security caused large overheads for the EDG RLS, with operations taking 10 to 20 times longer than without security. Results have not been published for the Globus RLS with security, but it is unlikely that they would differ greatly from this as the security mechanism used is the same. This overhead is also similar to that experienced with the LFC, as was shown in Section 4.4. With the RLSs, however, there is no way to avoid this whereas with the LFC, using transactions or (should they be implemented) sessions greatly reduces the security overhead.

In summary, comparing the performance of the LFC with that of the two RLS implementations, from Globus and from the EDG, shows that the LFC gives comparable or better performance than them in all the key catalogue operations - insertion, deletion and queries - especially when the hardware used is taken into account. This is shown particularly when security is used. This, together with the extra features the LFC offers, gives it significant advantages as a grid catalogue.

4.6 Summary

In this chapter, the results of performance testing of the LCG File Catalogue have been presented. It has been shown to be scalable and stable up to tens of millions of entries and hundreds of client threads, which was the limit of testing. Operation rates scale well with the number of client threads until the number of server threads is reached, at which point performance degrades as seen from the client side. This, and the server saturation seen in Section 4.3, suggests that it would be best to deploy the LFC with as many threads as possible and to use a server with as high a specification as possible. The maximum number of threads depends on the operating system, but is likely to be of the order of a few hundred.

While absolute value of operation rates will depend on the hardware, operating system and database parameters used, Table 4.3 shows a summary of the operation times which were measured in this chapter. The numbers shown are for a single client, running insecurely, with 1 million entries in the catalogue.

Operation	Time (ms)
Insert	22
Delete	22
Query	12
Read entry	15
Resolve symlink	15
List replica	19

Table 4.3: Summary of average times for basic LFC operations.

Some of the ways in which optimal performance can be reached have been presented: making use of the `chdir()` functionality, choosing a suitable number of operations (10-100) per transaction if transactions are to be used, and the tuning of parameters such as the reply buffer size. If security is to be used, it has been shown that a secure catalogue can be achieved without unacceptable consequences for performance if many operations are performed within the same authentication session.

A comparison has also been made with the EDG and Globus Replica Location Services, showing that the LFC can outperform both of these. As it has been developed in response to user feedback, it also contains features required by users but unavailable in the RLS implementations. It has been used continously under heavy load for extended periods of time without problems. Thus, it has been shown to be a mature and viable candidate for the replacement of the EDG RLS as part of the LHC Computing Grid. Currently, it is used in production by three groups as a central catalogue: the ZEUS experiment at DESY, Germany; the SEE-GRID (South-Eastern European Grid) project; and TW Grid (Taiwan Grid). It is also being tested by all the LHC experiments as a pre-production service at CERN, and by all Tier-1 and some Tier-2 sites.

Chapter 5

Grid Simulation with OptorSim

In this chapter, the rationale behind the development of a grid simulator is discussed, then the features required for a successful simulator are described. The grid simulator OptorSim is then presented with a description of its design, features and implementation, with an emphasis on the methods employed to make it as realistic as possible. Finally, a comparison is made with a number of other grid simulators.

5.1 Introduction to Grid Simulation

5.1.1 Motivation

Data grids are highly complex environments. The many components, users and sites, with their interactions, mean that it is impossible to predict behaviours from first principles. Before time and effort is spent on developing a new and experimental component, however, it is desirable to know its effects. The only way this can be done without implementing a prototype and deploying it on a testbed is by simulation.

In addition, the LHC Computing Grid (LCG), in common with other grid projects, is still in its development phase at the time of writing. There are an increasing number of real users, but these are not enough to show what grid usage will be like when the LHC is in operation. Even if novel components were developed and tested on the real grid, then, it would still not be possible to gain a true idea of how they would perform under the conditions of a grid in its production phase. Simulation is again necessary, therefore, to indicate how the components would perform under future conditions.

For these reasons, when the Data Management work package of the European DataGrid (EDG) project (which was the basis of the first LCG release) were designing the replica management services, it was decided to perform simulations to evaluate possible replication strategies. In particular, an economic model of file replication (details of which are given in Chapter 6) had been suggested as a way of achieving distributed, autonomous and optimal replication. Such an experimental strategy could not be implemented in the real Replica Optimisation Service at such an early stage in the development of grid middleware, but if simulations showed it to be effective, it could be an exciting and innovative field of future development. A number of existing simulators (which are discussed more fully in Section 5.7) were investigated but the conclusion was that none fulfilled all the necessary requirements. A new simulator, OptorSim, was then developed specifically for the simulation of data replication.

5.1.2 Necessary Components of a Grid Simulator

In any simulation, it is important to model the system components at the right level of abstraction. For a grid simulator, it is therefore necessary to examine a real grid system and decide which components are necessary for the simulation and at what level of detail. For a data management simulator such as OptorSim, the important components are:

- jobs or applications which run on the grid;
- computing resources to which jobs can be sent;
- storage resources where data can be kept;
- network infrastructure to connect the resources;
- a scheduler to decide where jobs should be sent;
- a transfer system to move data around the grid;
- and a component to perform replica management.

It should be possible to input different grid topologies and workloads and investigate different scheduling and replication strategies, and the simulation must be able to output sufficient information to evaluate the performance of these strategies.

5.1.3 Validation and Verification

In a predictive simulation, it is necessary to make sure the simulation model and its results are sufficiently "correct". In the simulation community, this is achieved through the twin processes of *validation* and *verification*. Essentially, validation is making sure the model of the components and inputs to the simulation is sufficiently accurate, and verification is ensuring that the actual implementation of this model is correct.

In [72], which is a standard reference for simulation modelling at an introductory level, a variety of techniques for validation and verification are outlined, some of which are relevant to the case of grid simulation and some of which are not. Those used in the development of OptorSim are as follows (using the terminology in [72]):

- Degenerate Tests: appropriate values of the simulation's input and internal parameters are selected and the output is examined to see whether the simulation's behaviour is what would be expected given these parameters. If the rate of job submission, for example, is much higher than the rate of job processing at a site, one would expect the queue size at a site to increase with respect to time until all jobs had been submitted.

- Fixed Values: a subset of Degenerate Tests. Fixed values of the simulation parameters are chosen such that the expected results can be calculated analytically. A very simple grid with a few jobs, for example, could be used as input and the expected job times calculated, then compared with the actual results.

- Internal Validity: a number of runs of the simulation are performed, with the same parameters (including a fixed random seed), and the variability of the output is observed. If there is a lack of consistency in the results, i.e. high variability, the results may not be trustworthy.

- Face Validity: people who are "experts" in the system being simulated check whether the model and its behaviour are reasonable. This is useful at the start of the modelling process for determining if the logic in the model is correct, and also for determining whether the output is reasonable for a given input.

Also of great importance to simulation accuracy is the validity of the inputs. In the case of a grid simulation, these will be the characteristics of the jobs and files being simulated; storage and computing capacity at grid sites; network capacity and so on. Many of the points made in this section are common sense; nevertheless, it is useful to categorise them in this way for evaluation purposes.

5.2 OptorSim Architecture and Design

5.2.1 General Architecture

As OptorSim was initially developed specifically for simulating the Replica Optimisation Service of the EDG, its architecture is based closely on the EDG model and the data management components (which were described in Chapter 3) in particular. The conceptual model of the architecture used for OptorSim is shown in Figure 5.1.

In this model, the grid consists of a number of grid sites, connected by network links in a certain topology. A grid site may have a Computing Element (CE), a Storage Element (SE) or both. Each site also has a Replica Optimiser (RO) which makes decisions on replications to that site. A Resource Broker (RB) handles the scheduling of jobs to sites, where they run on the CEs. Jobs process files, which are stored in the SEs and can be replicated between sites according to the decisions made by the RO. A Replica Catalogue holds mappings of logical filenames to physical filenames and a Replica Manager handles replications and registers them in the Catalogue. Note that the central Replica Manager and Catalogue in this model would constitute a single point of failure in a real grid, and thus a distributed solution would be preferred. In the EDG implementation, there was a Replica Manager at each site, of which the RO was a component, and cataloguing was performed by the Replica Location Service (RLS) which is described in Chapter 3. For the simulation, however, it is easier to

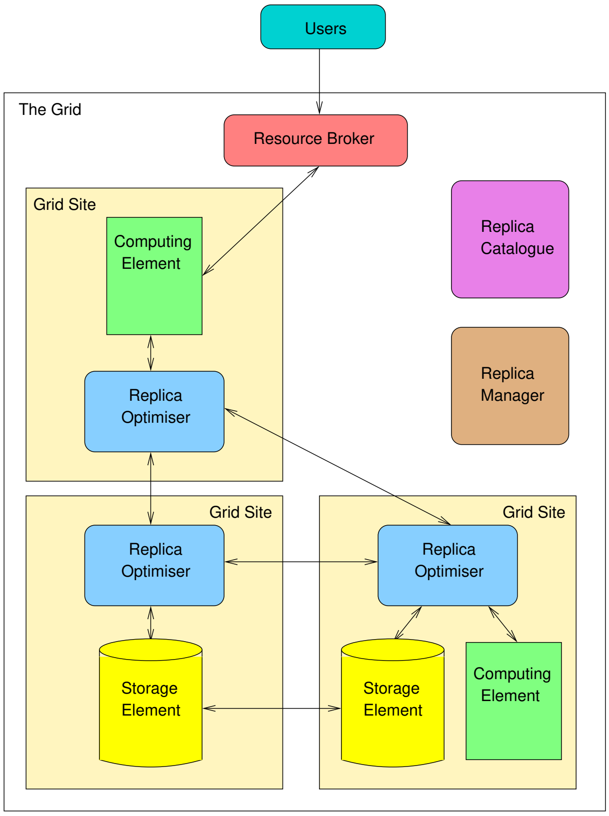

Figure 5.1: Grid architecture used in OptorSim, based on EDG data management components.

abstract out the Replica Manager and Catalogue as centralised services. This does not have any implications for the running of the simulation.

This architecture clearly fulfils the requirements laid out in Section 5.1.2. Its basis on the EDG architecture, as well as the fact that the model was developed by the developers of the EDG data management services, fulfils the criterion of Face Validity, while it is general enough to be applied to any grid.

5.2.2 Simulation Inputs

As well as the basic architecture, the detailed conceptual model of the grid applications and fabric components is crucial in achieving a valid simulation. These are components which must be easily configurable for each simulation and so are classified together here as simulation inputs.

To achieve an accurate idea of how these should be modelled, a close examination was made of both current systems and planned future systems. In particular, the Computing Model documents from the LHC experiments [17] [20] [18] [19] were used, as these detail the experiments' assumptions on dataset and file sizes, data distribution, processing times and computational and storage requirements as well as the kind of grid structure they envisage. These simulation inputs are now described in turn.

Grid Topology

The topology of the grid - the number of sites and the way they are connected - has a strong impact on grid behaviour. It is therefore important that OptorSim take any grid topology as input, i.e. that the user can specify the storage capacity and computing power at each site, and the capacity and layout of the network links between each. SEs are defined to have a certain capacity, in MB, and CEs to have a certain number of "worker nodes" with a given processing power. Sites which have neither a CE nor an SE act as routers on the network. A description is given in Chapter 8 of how the national research network maps were used to achieve the simulated topology for LCG in 2008.

Jobs and Files

In physics analysis, a job usually processes a certain number of files. This is simulated in OptorSim by defining a list of jobs and the files that they need, with their sizes. The list of files that a job may need is defined to be its *dataset*. A job, when sent to a CE, will process some or all of the files in its dataset, according to the *access pattern* which has been chosen. The time a file takes to process depends on its size and on the number and processing power of worker nodes at the CE. It is assumed that the output files from a physics analysis would be small enough to ignore compared to the input datasets, and also that these are likely to be stored at local sites rather than on the grid, and so no simulation of output files is required.

Site Policies

Different grid sites are likely to prioritise different kinds of job. A university with strong involvement in the ATLAS collaboration, for example, may prefer to accept ATLAS jobs, whereas a regional Tier 2 centre may be contracted to serve all experiments. Sites are also likely to apply quotas, with different experiments being given a different size of quota according to the site's priorities.

Network Bandwidth Variation

Apart from the layout of sites on the grid, it is important to model the underlying behaviour of the network over which files are transferred. Networking is a highly complex field and it would be possible to simulate the network to a high level of detail; indeed, there exist many specialised network simulators which do this. For OptorSim, however, it is not necessary to go down to the level of individual packets or different protocols. It is necessary only to know how the available bandwidth on a link varies over time and adjust file transfers accordingly. The network model used allocates each link a maximum capacity, and adjusts this through the running of the simulation to give a "diurnal" variation[1]. On top of the diurnal variation is a randomness given by a long-tailed distribution, to give the long-tailed transfer time effect which is commonly observed in networks. This model was developed empirically by monitoring of real networks, which is discussed further in Section 5.6. The implementation of each of these inputs, as opposed to the conceptual model, is described in Section 5.3.

5.2.3 Simulation Parameters

There are a number of parameters, other than the simulation inputs, which are of interest when performing simulations of a data grid. The most important are outlined here; a full description of all the parameters is given in [73]. These are:

[1] "Diurnal" according to the internal time of the simulation, not real time. See Section 5.3 for details of the timing model.

- Initial File Distribution: The distribution of files around the grid at the start of the simulation can affect the behaviour of the replica optimisation strategies. A set of the master files is therefore placed at designated sites, and the option is also given to fill up the sites with replicas of these files before the simulation gets under way.

- Access Patterns: Different kinds of job may access the files in the dataset in a different way. To simulate this, the files in the dataset are ordered. Some jobs may process each file in sequence; others may miss some files or read them in a different order.

- Users: The pattern and rate of job submission by users may affect the behaviour of job queues at sites and hence the behaviour of the Resource Broker and Replica Optimisers.

- Number of Jobs: Optimisation strategies may perform differently under conditions when the grid is under-utilised (few jobs) or congested (many jobs) so it is important to vary the number of jobs and measure this effect.

- Job Scheduler: The algorithm used by the Resource Broker for its scheduling decisions will clearly have an important effect on the running of the jobs and performance of the grid.

- Optimiser: The algorithm used by the Replica Optimisers for the file replication strategy, which is the original research focus of OptorSim.

To sum up this section on the architecture and design of OptorSim, it has been shown that the conceptual model used has captured the main components of a data grid, while removing unnecessary levels of complexity.

5.3 Implementation

OptorSim is written in Java, for two main reasons. The first is that the object-oriented approach fits well with modelling a grid, where there are many distinct components which interact via well-defined methods in their interfaces. The second is the ability to run many concurrent threads, which allows simulation of the many independent entities in a grid. Threads are allocated to each of the CEs and to the RB, and a single thread to the Users. Threads are also used in the timing and for certain components of the economic model. OptorSim is not numerically intensive and so using a more powerful language like C++ would offer no advantages.

Java is also portable, allowing the simulation to be distributed easily without having to recompile the code for different operating systems. The object-oriented model makes it easily extensible, and a number of other researchers around the world have adapted it to their own requirements.

The code is structured into several packages, each of which deals with a different part of the simulation. The list of packages and their main remit is shown in Table 5.1. Details of most of the classes within these packages is not given here, but can be found in the Javadoc API which comes with the OptorSim release [74].

Package	Scope
optorsim.time	Control of timing
optorsim.infrastructure	Grid fabric (sites, network etc) Reading of configuration files Mathematics support and statistics gathering
optorsim.reptorsim	Replica management and cataloguing
optorsim.auctions	Auction functionality for economic models
optorsim.optor	Replica optimisation
optorsim	Overall simulation control Resource Broker Users Simulation output and Graphical User Interface

Table 5.1: List of packages in OptorSim, with their main functionality, ordered from lowest to highest level.

5.3.1 Run-time Processes

The simulation inputs are read in at the start of a simulation run from a set of configuration files, one of which describes the grid topology, one which describes the jobs, files and site policies, one which contains information on the network bandwidth variation and one which contains all the simulation parameters. When the sites and grid have been instantiated with the given inputs, the Computing Element and Resource Broker threads are started, along with the simulation timing thread and followed by the Users thread.

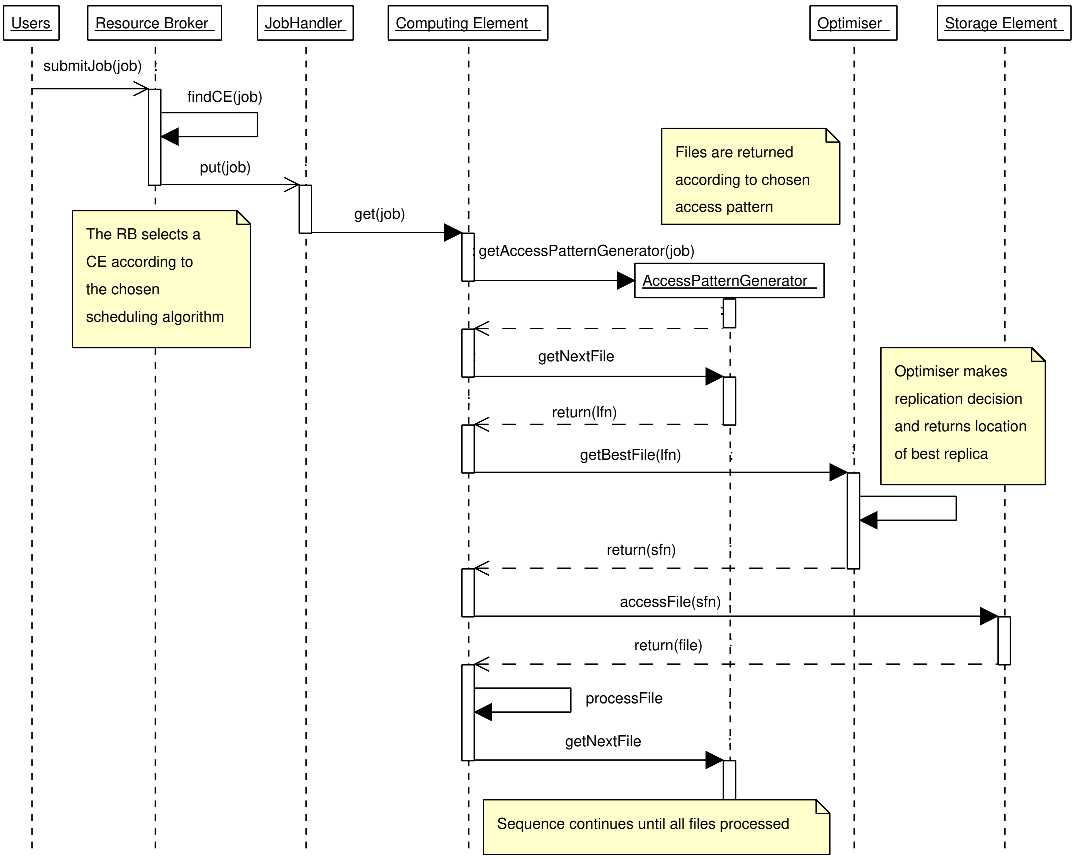

Figure 5.2: UML diagram of interactions between Users, Resource Broker and Computing Element when simulating a grid job.

Figure 5.2 shows the interactions of the Users, Resource Broker and Computing Element threads during the lifetime of a grid job. It is an example of a UML[2] sequence

[2]Unified Modelling Language

diagram, in which time increases from top to bottom and interactions between objects are shown by arrows. Method calls are shown by solid arrows and method returns are shown by broken arrows.

First of all, the Users thread submits a job to the Resource Broker (RB). The RB then searches the list of CEs in the grid which will accept this job type, and selects one according to the chosen scheduling algorithm. The RB then puts the job in the queue of that CE, via its Job Handler. When the CE is ready to process the job, it takes it from the Job Handler and begins processing by creating a new AccessPatternGenerator object for the job. This returns logical file names (LFNs) from the job's dataset according to the access pattern chosen in the parameters file. When the CE has the LFN, it calls the Optimiser to find the best replica of this LFN. The Optimiser returns the storage file name (SFN) of the file, i.e. its location on the grid. The CE then reads that file and processes it, before calling for the next file, and so on until all the files for that job have been processed. When all jobs have finished, the CEs shut down and the RB then shuts down the simulation.

The file processing time is simulated as

$$t[s] = \frac{f_{lat} + f_{lin} S[MB]}{N_{wn} P_{wn}[kSI2000]} \tag{5.1}$$

where f_{lat} is a latency factor with units of kSI2000-s, f_{lin} is a linear processing factor with units of kSI2000-s/MB, S is the file size in MB, N_{wn} is the number of worker nodes in that CE and P_{wn} is their power in kSI2000. For simplicity, worker nodes are not currently implemented as separate objects and so a CE only processes one job at a time. The higher the number of worker nodes and the more powerful they are, the faster the job is processed.

Figure 5.2 gives a view of the simulation which is centred on the CE activity. The crucial point from a data management point of view, however, is when the Optimiser is making the replication decision, and this will be shown in detail in Chapter 6.

Underlying these processes is the timing. OptorSim does not necessarily proceed in real time, as it uses its own internal time system. This can be set to either *simple* grid time or *event-driven* grid time. Simple grid time is, in effect, real time; if a CE takes 10 seconds of simulation time to process a job, it will take 10 seconds of wall clock time plus calculation time, such as that associated with finding the best replicas.

Event-driven grid time, on the other hand, is completely independent of real time. A thread is used to keep track of which of the simulation threads (CEs, RB etc) are active and which are waiting for some process to happen. When no other threads are active, i.e. all other threads are waiting for various events to occur before they can proceed, the timing thread advances time to the point at which the next event occurs. The appropriate thread is then activated and the simulation proceeds as usual until the next point at which all the threads are waiting. This saves a considerable amount of time in running the simulation, as the real time taken is only a fraction of the simulated time, but gives the same results. For most purposes, therefore, event-driven grid time will be used, but simple grid time can still be useful for demonstration purposes and hence either can be chosen using the parameters file.

5.3.2 Inputs and Parameters

A list of the main simulation inputs and parameters was given in Section 5.2. In this section, further explanation of their implementation is given.

Grid Topology

Each element of the grid - sites, CEs and SEs - is represented by an object which is instantiated when this information is read in from the grid configuration file at the start of the simulation. In addition, there is a GridContainer object which holds a list of all grid sites, and maps the network routes between sites. The shortest routes from a site to all other sites are found using Dijkstra's algorithm when the simulation is initialised. Dijkstra's algorithm [75] is a well-known algorithm for solving single-source shortest-path problems and is used in internet routing in the Open Shortest Path First (OSPF) routing protocol [76]. It works by building a shortest-path tree from a given starting vertex to every other vertex on a graph, using "weights" allocated to each edge; in this case, the vertices are sites and the weights are the bandwidth available between each site.

Jobs and Files

All information on the jobs and files that they need is stored in the job configuration file, which is also read in at the start of the simulation. Each job type is given a certain probability of running, so when the Users thread submits a new job, one is chosen accordingly to its probability. Each job is represented by a GridJob object, which keeps a list of the files it requires. This is either the whole dataset which has been given in the job configuration file, or a subset thereof. The fraction of its dataset which is required by a job type is also given in the configuration file. If only a subset is required this is chosen from a random point in the dataset, thus giving each individual job a different fileset from the same overall dataset. This represents the normal situation in particle physics analysis, where an individual physicist will run some analysis on only a small part of the whole experiment dataset.

Each file in the simulation is given a unique identification by way of an index number. Files in the same dataset have similar index numbers while files in different datasets have widely separated index numbers. This is to aid the file value prediction algorithms of the economic model, where it is assumed that if a file from a particular dataset has been accessed, it is likely that another file from the same dataset will be accessed in future.

Site Policies

Each CE stores a list of the job types it will run. In the current implementation, a site will either run a job of a given type or it will not; there is no fine-grained prioritisation or application of quotas within a site.

Network Bandwidth Variation

The implementation of the model of network bandwidth variation is as follows. First, information on the diurnal variation in bandwidth on a particular link is input to OptorSim by a configuration file. This contains, for each link, the name of a file which contains the mean available bandwidth (as a percentage of the maximum) for each half-hour period of the day. This information is read in and stored at the start of the simulation. When performing a file transfer, the simulation time is checked and the appropriate bandwidth found. Next, a random number is chosen from a Landau distribution about that bandwidth to give the necessary random jitter, and this number is used as the available bandwidth for that particular transfer. The reasons for using a Landau distribution are given in Section 5.6.

The use of this variation in available bandwidth is optional and can be turned off using the parameters file.

Access Patterns

There are five different types of file access pattern implemented in OptorSim, to represent different possible kinds of job. These are:

- Sequential: files are accessed in the order in which they are listed in the job configuration file.
- Random: files are accessed from the dataset randomly.
- Unitary Random Walk: starting from a random file in the dataset, the next file chosen has a file index which is 1 lower or higher than the previous file, with both directions having equal probability.
- Gaussian Random Walk: similar to the Unitary Random Walk, but in this case the next file index is chosen at random from a Gaussian distribution about the previous file index. The width of the distribution is set to be half the number of files in the dataset.
- Zipf: files are accessed at random from a Zipf-like distribution over the dataset. A Zipf-like distribution is defined as $P_i \propto i^{-\alpha}$, where P_i is the frequency of occurrence of the i^{th} ranked item and $\alpha \leqslant 1$ (a pure Zipf distribution would have $\alpha = 1$). In other words, if the files in a dataset are ranked, those with a high ranking will be accessed frequently whereas those with a low ranking will be accessed very infrequently. In the simulation, the file ranking for this access pattern is aligned with the file indices, so if a job has a list of 100 files, with indices from 150 to 250, file 150 will have the highest rank (1) and file 250 will

have the lowest rank (100). α is configurable in the parameters file, but should be set between 0.7 and 1 for a realistic distribution. Zipf-like distributions are observed in a number of real situations, notably in patterns of web page access [77].

Until the LHC is actually running, it will be impossible to predict exactly what physicists' access patterns will look like. However, some general observations can be made. Event reconstruction jobs, where physics events are reconstructed from detector data, are likely to have a sequential access pattern, as there will be a list of data files, each of which must be processed to give the reconstructed event. This would also apply to each of the data reduction steps from Event Summary Data to Analysis Object Data to TAG data (see Chapter 3 for a description of the different kinds of data file).

During analysis, the access pattern will be less predictable, as individual physicists will have different kinds of analysis job. Some are likely to be sequential, such as when the user knows which files are needed and gives a list of them with the job. On the scale of a whole collaboration, however, there will be some items of data which are much more interesting than others and which will therefore be used in analysis many more times. This suggests that a Zipf-type access pattern may be likely.

The access pattern will also depend on the kind of *streaming* model adopted by the experiments. Streaming is the separation of different types of event, such as all events involving B physics, into different datasets, and is another way of helping to optimise data access. If data are heavily streamed, the files within each stream may be accessed more sequentially than otherwise. Even with streaming, however, some events may appear in more than one stream, thus contributing a Zipf-like component.

The other access patterns may be relevant to other grid applications and are included for completeness.

Users

Job submission is controlled by the Users thread, of which there are three different implementations currently available. A *job delay* parameter, d, is set in the parameters file and is used for the first two types of User: *Fixed Interval* and *Random Interval*. With *Fixed Interval* job submission, the Users submit jobs to the Resource Broker at regular intervals of size d until all the jobs have been submitted. With *Random Interval* submission, the interval between one job and the next is a random length between zero and $2d$.

The third category of user is *CMS DC04*, and is based on observations of job submission times during the 2004 CMS Data Challenge, when the CMS collaboration were testing their computing framework using simulated data. From inspection of data available in [78], it was seen that users tended to submit most jobs between 0900 and 2100 local time, in many cases in a roughly normal distribution with a maximum at around 1500. The maximum number of jobs submitted in an hour was around 100. This was modelled by having the Users submit a variable number of jobs per hour, depending on the time of day in simulation time. A normal distribution was used, centred on 1500, with variance of 7.5 and modulated by the maximum job rate of 100.

In summary, each of the main inputs to OptorSim has a selection of choices implemented, and the modular nature of the code makes this easily extensible should more choices be required.

5.3.3 Outputs

There are two ways in which OptorSim can output useful information to the user: the terminal and the Graphical User Interface (GUI). In each case, a number of statistics are gathered at the levels of CE and SE, sites and whole grid. When the simulation finishes, these are output to the terminal according to the level of detail stipulated by the user in the parameters file.

If the GUI is being used, more detailed monitoring of the state of the grid is performed and output to the GUI in real time as the simulation progresses. This includes continuous calculation of job times, SE usage, network usage and so on. Figure 5.3 shows a screenshot of the GUI during a simulation run.

The GUI window is divided into three panes. In the top left is a hierarchical view of the grid being simulated, which allows each site node to be expanded to show its

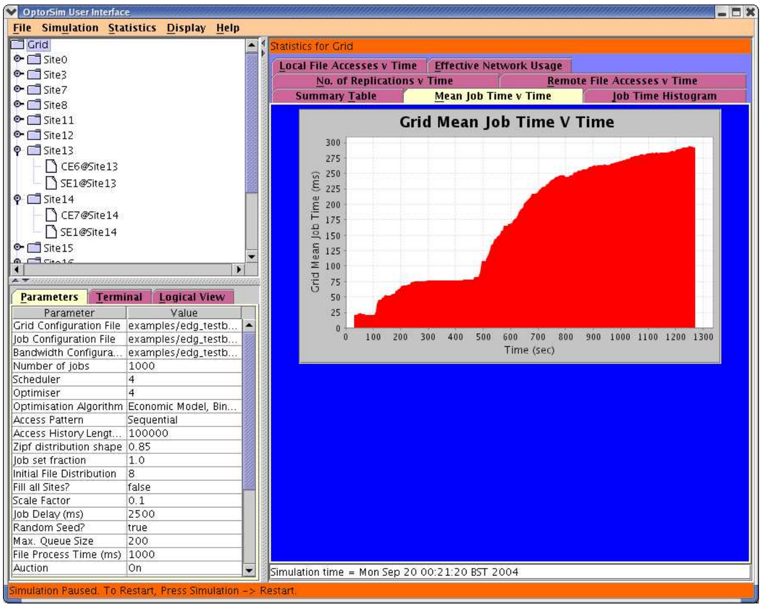

Figure 5.3: Screenshot of OptorSim GUI during a simulation.

CEs and SEs. Clicking on a component in the hierarchy pane brings up the statistics for that component in the statistics pane, which is on the right. This contains a number of tabs with graphs of accumulated statistics for that component, plus a table with a summary of the current status. The statistics pane is refreshed regularly at a periodicity chosen by the user. On the bottom left is the information pane. This has three tabs; one lists the current values of the simulation parameters; one shows the output as it would appear on the terminal; and one gives a logical view of the grid with all current file transfers between sites. Clicking on a node in the logical view also brings up the relevant statistics in the statistics pane. Using the GUI, an HTML page can be saved at any point during the simulation with a copy of each of the available graphs. This can also be saved at the end of the simulation.

5.4 Limitations

Any simulation of such a complex system as a data grid will naturally be limited in some way, and these limitations should be remembered when evaluating the results generated. The chief limitations of the OptorSim model are as follows.

Computing Elements. In the current implementation, worker nodes within a site do not act as independent entities, and a CE processes only one job at a time. In effect, a farm of processors behaves like one superprocessor. This means that queues will build up more quickly. It also means that instead of many jobs requesting and replicating files simultaneously, only one job's files will be replicated to a site at any given time.

Site Resource Quotas. As mentioned previously, site policies are applied in a coarse-grained manner. If quotas were applied to the resources available to each experiment at each site, this could affect the performance of both the replication and scheduling algorithms.

Resource Availability. It is assumed in OptorSim that grid resources are always available. In reality, resource availability would fluctuate, perhaps with sites suddenly losing their connection to the grid. This would require re-scheduling of jobs which were running on the unavailable site as well as reducing the overall resource capacity of the grid.

Of these, implementation of a more realistic CE should have the highest priority for future work; meanwhile, the use of a suitable scheduling strategy and appropriate values for the queue and worker node parameters can mitigate the effects. Limitations in the implementation of the replica optimisation strategies presented in the next chapter are discussed there.

5.5 Testing

Having discussed the design and implementation of OptorSim, this is an appropriate point at which to refer back to the points which were made about validation and verification in Section 5.1.3. The role of Face Validity in approving the conceptual model and architecture of the simulation has already been presented; the acceptance of the OptorSim model by experts at conferences and in peer-reviewed journals [79] [80] adds to this. The other techniques which were mentioned - Degenerate Tests, Fixed Values and Internal Validity - are now discussed.

Degenerate tests were applied by inspection during the development process. The simulation would be run with a high volume of output information, and if anomalous or unreasonable behaviour was observed, the problem would be searched for and solved. This was sometimes due to implementation bugs and sometimes due to shortcomings in some aspect of the model, such as the way of estimating future values of files.

The Fixed Value tests were administered via a suite of functional tests, which also verified the correctness of the implementation during development. The functional tests covered basic network behaviour, schedulers, access patterns, replication times and optimisation algorithms. These used very simple grid configurations with only a few sites, where the correct results could be calculated and used to check the simulation results, to show that the implementation was indeed correct. Some simple configurations were also tested in [2], which also showed that OptorSim behaves as expected.

For the Internal Validity tests, the parameters of the simulation were chosen to give as little randomness as possible (such as in the scheduling of jobs to different sites, or in the jobs which were run). OptorSim was then run with the same parameter set 200 times, using the same CPU type each time and using a fixed random seed to give internal consistency. The mean job time was calculated each time and the distribution of times plotted as shown in Figure 5.4. Although at first glance this looks like a large variance, with an RMS value of 55.95, when compared to the absolute values (which are of order 10^4) it is clear that the variation is in fact only about 0.5%. What variation does exist is due to the fact that in certain situations, when there is a choice between two equally valid options, one will be chosen at random. If the Replica Optimiser at a site, for example, wants to get a certain file in the shortest time possible, and there are two sites which could deliver the file in exactly the same time, one of those sites will be chosen at random. Rather than being due to the random seed for the simulation changing, this is due to the way certain elements, such as hashtables, are implemented in Java. This is therefore an irreducible variation, but it is small enough that we can confidently state that the simulation is internally valid at the required level.

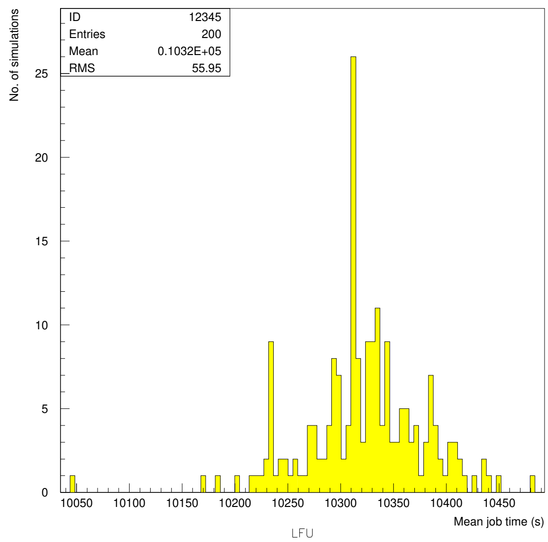

Figure 5.4: Variation in mean job time over 200 simulation runs, using the LFU replication algorithm.

Finally, tests were conducted to ensure that the simulation results are independent of the type of CPU on which they are run. With the simple grid time, there will be a dependence on processor speed, as the processing time is included; with event-driven grid time, however, it should be possible to run on any CPU and get the same results. Table 5.2 summarises the results of running OptorSim with the same parameters set on a variety of different machines. Each result is the mean of 200 simulation runs. In the table, ScotGrid refers to the IBM batch farm run by the Glasgow Particle Physics (Experimental) (PPE) group; ppepc65 and ppepc87 are workstations on the Glasgow PPE network; and lxplus043 was a CERN Linux Public Interactive Logon Service node.

Combining these results gives an overall mean of 10235±22 s, showing that the standard error from variations in the machine behaviour is less than 1%. There may be some slight residual dependence on CPU speed, but this is small enough to be negligible within the range of CPU speeds available today. It is therefore clear that using OptorSim with the event-driven grid time model makes it independent of the

Machine	CPU Speed	Memory	Operating System	Mean job time (s)	RMS (s)
ScotGrid (Dual Pentium III)	1 GHz	2 GB	RedHat 7.2	10190	74
ScotGrid (Dual Xeon blade with hyperthreading)	2.4 GHz	1.5 GB	RedHat 9	10320	56
ppepc65 (Pentium 4)	1.7 GHz	1 GB	RedHat 7.2	10340	48
ppepc87 (Pentium 4)	2.8 GHz	0.5 GB	RedHat 9	10350	58
lxplus043 (Pentium III)	1 GHz	1 GB	RedHat 7.3	10120	34

Table 5.2: Results of running OptorSim on different CPUs.

machine used, within the limits of the internal variance.

In summary, the range of tests presented in this section verify OptorSim's implementation and demonstrate that it is a valid grid simulator.

5.6 Developing the Network Bandwidth Variation Model

To develop the model of network bandwidth variation due to non-grid traffic which was mentioned in Section 5.2.2, network monitoring data from several sources were used. These were: the monitoring sites at SLAC[81] and FNAL[82] for U.S.A. and transatlantic connections; EDG network monitoring for Europe; and the UK e-Science Grid Network Monitoring Programme[83] and the GridNM tool[84] for UK links.

These gave measurements of the instantaneous available bandwidth at the time of measuring, with the European and UK sites taking data half-hourly and the USA sites every two hours. The measurement tool used was Iperf [85], which measures the maximum end-to-end TCP bandwidth between two points on a network. A script was set up to gather these data daily, with the permission of the groups conducting the monitoring, and this ran for several months, from September to December 2003.

Dividing the day into 48 half-hour bins, the bandwidth measurements for each link were split into the different bins and the mean taken for each. This gave a set of profile histograms of how the available bandwidth for each link varied through the course of a day, on average. Some examples are shown in Figure 5.5.

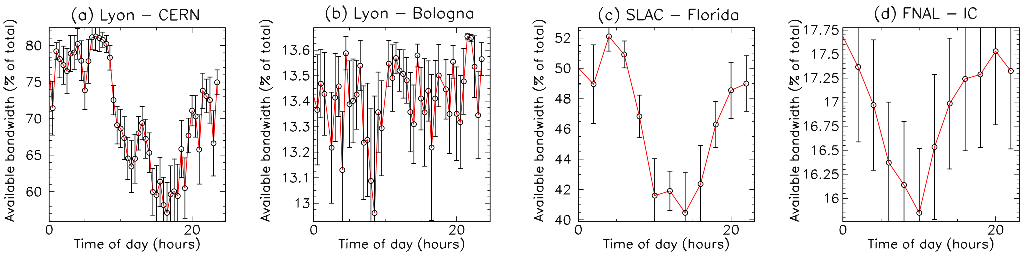

Figure 5.5: Mean available bandwidth on a selection of monitored links as a function of time of day.

Many, though not all, of the links show a distinct diurnal variation in the mean. The random variation about the mean, however, generally has a long-tailed distribution, as Figure 5.6 shows. This is a well-known feature of networks which is related to their self-similar nature [86] [87], the reasons for which are still an active area of research.

To model this distribution for the purposes of the simulation, three kinds of function - Gaussian, log normal and reverse Landau - were fitted to the data and the χ^2 per degree of freedom for each function calculated for each link. A comparison was then made of the number of links which were best fitted by each function, for which a fit with a $\chi^2/ndf < 2$ was available. This is shown in Table 5.3.

Gaussian best fits	Landau best fits	log-normal best fits
19	30	9

Table 5.3: Number of "best fits" per function, with $\chi^2/ndf < 2$.

It is generally accepted that there is no single function which can describe every observed distribution [88], and this is borne out by the above results. As the Landau distribution gave the best fit for the highest number of distributions, it was decided to use this to model the variation in bandwidth in OptorSim. Other functions, such

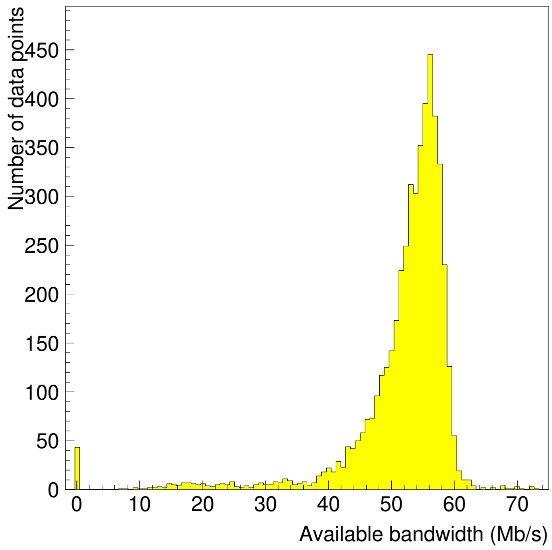

Figure 5.6: Distribution of available bandwidth between Newcastle and Belfast.

as the Weibull or Pareto distributions [86], could also have been investigated, but would have been more difficult to implement in OptorSim with no significant gain in the realism of the simulation.

Combining the diurnal variation in the mean with a long-tailed Landau-distributed random variation about the mean, then, gives the model finally used in OptorSim.

5.7 Comparison with Other Simulators

There exist several other grid simulators, each with a different focus and design. This section briefly discusses each of them and how they differ from OptorSim.

MONARC

The *Models of Networked Analysis at Regional Centres for LHC Experiments*, or MONARC, project [89] was initiated in 1998 to explore which computing models would be feasible for the LHC projects, given the projected network and data handling resources at LHC start-up and running time. As part of this project, a simulator was developed to provide a realistic simulation of distributed computing systems, customised for physics processes, with which different data processing architectures could be evaluated [90].

Like OptorSim, the MONARC simulation is a process-oriented, discrete-event simulation written in Java, to make use of Java's concurrent thread facility. It models fabric objects such as databases, disks, mass storage systems and CPUs as well as sizes and processing requirements of all different types of LHC data file. Its main difference from OptorSim, however, lies in the lack of infrastructure for automated replication and replica optimisation. The level of detail of fabric elements would also make it more difficult to measure the effect of different optimisation strategies, should these have been added to the MONARC simulation.

GridSim

GridSim [91] is a grid simulation toolkit developed to investigate resource allocation techniques and in particular, a computational economy. Again, it is a discrete-event simulator based on Java. It supports modelling of heterogeneous computing resources from individual PCs to clusters, and various application domains from biomedical science to high energy physics. The focus is very much on scheduling and resource brokering with very detailed implementation of application resource requirements, budgets and local policies and influences at grid sites, down to the effects of public holidays and weekends. There is not, however, capability for data management such as would be required for investigation of replica optimisation strategies.

Bricks Grid

Bricks Grid [92] is another Java-based discrete event simulator, based on the Grid Datafarm (Gfarm) [93] architecture. In this architecture, each grid site is designated as a Gfarm filesystem node, acting as both storage and computing node with a single virtual filesystem. At the time at which development of OptorSim was begun, Bricks Grid's focus was exclusively on scheduling; it has since been extended to include replica management. Its replica management components, however, are centralised rather than the distributed Replica Optimiser architecture of OptorSim.

GridNet

GridNet [94] is written in C++ as a layer on top of the NS [95] network simulator. It aims to investigate dynamic data replication strategies, with a focus on network behaviour. Its architecture is simple, containing Storage Elements, Replica Managers and the network components. It thus gives a complementary approach to that used in OptorSim.

ChicSim

ChicSim [96] is a discrete event simulator built on top of a C-based simulation language called Parsec [97], and designed to investigate scheduling strategies in conjuction with data location. Its architecture includes a configurable number of schedulers rather than one Resource Broker, for example. It also allows for data replication but with a "push" model in which, when a site contains a popular data file, it will replicate it to remote sites, rather than the "pull" model used in OptorSim. It is also, therefore, complementary to OptorSim.

Others

The simulators which have been mentioned above are those which are most relevant for comparison with OptorSim, and illustrate how each has its own particular scope for research. Other simulators also exist, such as EDGSim [98], which models job and data flows around the EDG framework; GridMate [99], which analyses resource management and load-balancing strategies; and MicroGrid [100], which is an emulator for Globus applications.

At the time at which OptorSim development began, none existed which could have been easily modified for replica optimisation studies and hence a decision was made to develop one specifically to fit these requirements. Rather than competing, the breadth of simulators available allow different research groups to investigate different aspects of computing and data grids, and OptorSim now occupies a significant niche among them.

5.8 Summary

At the beginning of this chapter, the *raison d'être* for grid simulations was introduced, followed by discussion of the necessary components of a grid simulator and ways of validation and verification. The architecture and design of OptorSim with its main components were then described, showing how it fulfilled the necessary requirements for a data grid simulation with autonomous file replication. The way in which the conceptual model was implemented was then presented, looking in particular at the different options which have been included for simulation inputs such as file access patterns and job scheduling algorithms.

Following the architecture and implementation, a description was given of the ways in which OptorSim has been tested and shown to be reliable. A description was then given of the way in which the model of underlying network bandwidth variation, used in the simulation, had been developed. Finally, a number of other grid simulation projects were discussed and it was shown both how they and OptorSim differ, and in what ways they are complementary.

Chapter 6

Optimisation Algorithms

In the last chapter, a description of OptorSim was given without discussing the optimisation strategies which have been implemented, or even what an optimisation strategy should look like. This chapter rectifies this by first discussing grid optimisation in general, then job scheduling, before concentrating on replica optimisation and the algorithms which have been implemented in OptorSim. The task of dynamic replication can be split into three stages - replication decision, replica selection and file replacement - which a replication strategy must perform. An economic model of file replication, which was briefly mentioned in the last chapter, will be presented as well as the simpler strategies of Least Recently Used (LRU) and Least Frequently Used (LFU).

6.1 Grid Optimisation

In the complex environment of a grid, there are an enormous number of variables which determine its overall performance. Many of these are controlled by different people who may have different goals. The computing resources at a site, for example, may have policies determined by a government funding body, a batch system set up by a local site administrator and software written by different LHC experiments. A network may be run by a consortium of national funding councils but the cables leased from a commercial company. In addition, there are factors which are largely uncontrollable, such as variations in network bandwidth due to non-grid traffic. It is clearly impossible to harmonise all these variables into one optimal configuration for the whole grid, especially if it traverses both national boundaries and those between academia and industry.

It is possible, however, to optimise those variables which are a part of the grid middleware itself. For a data grid, the most important area to optimise will then be the data management middleware, and two aspects of this in particular: the way in which the jobs are scheduled to sites and the way in which the replica management is done. In this chapter, these are discussed in turn, designated as Job Scheduling Optimisation and Replica Optimisation respectively.

It is also important to consider the scale on which optimisation is performed. Optimisation can be short-term, or local, with the aim of maximising the immediate performance for an individual user; it can also be long-term, or global, with the aim of maximising future performance across the whole grid.

6.2 Job Scheduling Optimisation

Scheduling can be defined as the process of selecting a resource on which to run a task, given that task's requirements, and a time at which the task should be run. There are many levels at which scheduling occurs: in the operating system of a single machine, in the batch system of a local PC cluster or farm, and the choice of site on which to run a job in a distributed system. Scheduling in a grid occurs on all these levels, but here only the latter will be considered, considering the scheduler to be that part of the grid's resource management system with responsibility for choosing, given a number of possible sites, the site to which the job is sent. This is the subject of much active research [101], complicated by the fact that grids operate over different administrative domains and heterogeneous resources. Some different approaches to scheduling in grid systems are now discussed, including the leading grid scheduling systems, before describing the algorithms which have been implemented in OptorSim.

6.2.1 Approaches to Grid Scheduling

Different types of grid - computing, data or service grids - may have different requirements when scheduling jobs. Most grid projects to date have focused on computational grids, and this is reflected in many of the scheduling policies which have been implemented. We shall therefore categorise schedulers by the type of grid for which they are designed rather than by their degree of centralisation, which is the usual way of categorising schedulers (such as in the taxonomy presented in [101]).

Schedulers for Computational Grids

One example of a centralised scheduler (on which the EDG Resource Broker was based) is Condor [38], which uses a central matchmaking service to match resource providers and consumers using its ClassAds mechanism. This allows providers to advertise what they have available for use, and consumers to advertise what their requirements are, by defining them in a ClassAd. ClassAds include both requirements, which must be satisfied, and ranking attributes, which indicate preferences. The matchmaker scans the list of ClassAds and creates provider-consumer pairs which satisfy each others' requirements. When there are a number of possible matches, resource providers are ranked according to the ranking attributes in the consumer's ClassAd (e.g. processor speed) and the one with the highest rank is chosen.

The Community Scheduler Framework [102] is a reference implementation of a scheduler based on the Globus Toolkit's GRAM (Globus Resource Allocation Mechanism) package. It assumes a more hierarchical model than Condor, with each VO running its own scheduler, and has a default round-robin scheduling policy. More complex policies can be implemented by the VOs as required.

Legion [40] is a metacomputing environment, similar to Condor but taking a more distributed approach in that scheduling is by negotiation between autonomous agents rather than through a central service. Some simple random schedulers are included with Legion, but the expectation is that application developers would write their own specific scheduling agents if those provided are too basic. AppLeS [103], the Application Level Scheduler project, takes the application-specific approach further by concentrating solely on developing scheduling agents, one for each application, which can then interact with systems such as Globus or Legion. An AppLeS agent prioritises resources with respect to a valuation function specific to the application, plans a number of possible schedules, estimates performance for each then chooses the best schedule. The GrADS [104] (Grid Application Development Software) project takes a similar approach to AppLeS, but includes a metascheduler which takes the candidate schedules from the scheduling agents and applies policies to balance the interests of different applications.

NetSolve [42] and Ninf [105] are both designed for scientific problem-solving environments, and are in fact interoperable. In NetSolve, for example, a client contacts a central NetSolve agent, which examines the list of available servers and lists them in order of completion time for the job, using knowledge of their CPU speeds, workload, bandwidth and the latency between client and server to estimate the completion times. In Ninf, a set of metaservers are distributed through the system, each responsible for a group of servers, rather than a central agent.

Finally, the Nimrod-G [106] resource broker implements a computational economy, where resource users specify time and budget constraints for their applications using "Grid credits" or tokens as payment. The scheduler component is responsible for finding available computing resources which meet the user's deadline and budget, as well as optimising either for time or for cost.

Schedulers for Service Grids

Although Nimrod-G was listed above as a resource broker for computational grids, it could just as well be categorised as for service grids, as there is a strong emphasis on provided the user with a defined quality of service (QoS). As its main design aim was as a tool for computationally-intensive parameter-sweep applications, however, it was included in the preceding section rather than this one.

Darwin [107], however, is designed for service grids in the sense in which they were defined in Chapter 2, for applications such as video/audio transcoding or distributed simulation. Its resource broker component, Xena, finds resources which will satisfy all the user's constraints then optimises according to application-specific criteria, such as maximum quality or minimum cost.

Schedulers for Data Grids

As initial research efforts were focused on computational grids, data grid schedulers are often extensions of computational grid schedulers. The Gridbus [108] broker, for example, is an extension of Nimrod-G to data grids. It sends jobs to computing resources which are close to the source of the data, taking into account the capability and performance of the resource, available bandwidth between computing and storage resources and the cost of data transfer to try and minimise the job completion time.

The EDG Resource Broker [109] is a component of the Workload Management System (WMS) and is based on Condor's matchmaking system. As for Condor, matchmaking is performed in two stages - requirements check and ranking - but if the job has data access requirements then Computing Elements (CEs) are classified by the number of input files which are in a Storage Element close to that CE. The CEs with the highest number of nearby input files are then ranked by the user-specified ranking attributes (such as CPU speed or memory) and the best CE chosen. If, however, the user has specified the Access Cost as a ranking attribute, classification by number of nearby input files is not done and CEs are instead ranked by the "cost" (i.e., time) to access all the required files.

Although there are many differences between the schedulers listed here, some general observations can be made before looking at the policies adopted in OptorSim. Each scheduler first examines the grid's resources and makes a shortlist of those sites which fulfil the job's requirements. The "best" site is then chosen either according to the user's preferences, whether that be for fast completion time or high quality of service, or according to an inbuilt metric. In data grids, however, the focus tends to be on getting the jobs to run on sites close to the data, without ignoring the effects of the sites' computing capabilities.

6.2.2 Scheduling Algorithms in OptorSim

Given that OptorSim was developed as a data grid simulator, scheduling algorithms were chosen which would reflect this and allow comparison of how non-data grid schedulers would behave in a data grid. Those which have been implemented for use by the Resource Broker are as follows; note that the schedulers only consider CEs which will accept the job type in question.

- *Random*: Schedules randomly to any CE which will accept the job, like the default scheduler for Legion.
- *Queue Length*: Schedules to the CE with the shortest queue. This represents a computational grid scheduler, accounting for load at the CEs but not for data location.
- *Access Cost*: The time taken to access the files required by the job is estimated for each CE. The job is then sent to the CE where this time is lowest, similar to the Access Cost ranking attribute in the EDG Resource Broker. No account is taken of computational load.
- *Queue Access Cost*: As for *Access Cost*, the time to access all the job's files is estimated for each CE. In addition, the time to access the files for all the jobs in the queue at the candidate CE is estimated. The job is sent to the CE where this combined access time is lowest, thus combining information on CE load as well as data location.

Although many other and more sophisticated algorithms could have been implemented, these were not included, as OptorSim's main aim was replica optimisation rather than scheduling optimisation. Nevertheless, those which have been implemented allow broad investigation of schedulers which consider CE load, data location, or both, or neither, in a data grid context.

6.3 Replica Optimisation

Replication of data is the process of placing copies of data objects at different points on a network, and replica optimisation is the process of distributing these replicas in an optimal fashion. A distinction can be made between "static" file replication, in which replicas are made to various sites in a planned fashion, and kept at these sites, and "dynamic" replication which is performed by the Replica Optimisation Service while jobs are running. In OptorSim, static replication would correspond to detailing which sites contained replicas at the beginning of the simulation, while dynamic replication would be performed during the simulation. The emphasis in the replication strategies described below is therefore on dynamic replication.

Applied to data grids, replication is perhaps the most recent example of the *file assignment problem*, which is the well-known NP-complete problem of how to optimally assign files to nodes on a network [110]. The complexity of the problem is such that it is impossible to solve analytically in an efficient way and thus some heuristic must be adopted; the strategies discussed in Sections 6.3.4 and 6.4 are examples of such heuristics. First, however, some approaches to and motivations for replication, both outwith and within the field of grid technology, are discussed.

6.3.1 Approaches to Data Replication

Distributed database technology is one area in which there has been much research into dynamic data replication [111] [112]. Replication schemes for distributed databases depend heavily on whether there are more reads or writes on these databases. If the system is read-intensive, wide replication is advocated, as reading data locally is faster than reading over the network. If the system is write-intensive, however, it is better to have less replication, as the cost of keeping the replicas consistent with each other increases with the number of replicas. Data analysis for HEP is an example of a read-intensive system, as when data files have been created they will be read-only.

Peer-to-peer (P2P) networks are another type of system in which replication is important. To improve performance of a P2P system implies minimising the number of hosts which have to be contacted before a query is resolved, and this suggests replication of the data to a number of hosts on the network. The Gnutella network [12] does not replicate pro-actively, but when a search is successful, the object requested is stored at the node which requested it. Freenet [13], on the other hand, replicates the requested object to every node along the path of the query which requests it. This would clearly cause problems with site policies and storage in a HEP grid situation.

In general, replication can be motivated by two issues: availability of data, and system performance. In the early days of distributed systems, when network bandwidths were low, performance was the main motivation but as bandwidths increased, communication costs became less of an issue. Fault tolerance, or availability of data even if one site became unavailable, then became more important. Availability is still the main motivation for replication in P2P networks and standard distributed databases, although performance is once again becoming important for databases in mobile computing environments.

In a data grid, the high level of reliability of the main data storage sites makes fault tolerance less of an issue, while the large file sizes increase the file access times of grid jobs. Performance thus becomes the main motivation for replication in data grids. If several replicas of a data file exist at different locations on the grid, a user application will be able to access the one which is "nearest" (nearness may be in terms of network bandwidth rather than geographical location, depending on the state of the network) and thus get the data faster, as well as there being little likelihood of all the replica sites being simultaneously unavailable. Replication in a data grid differs from other systems in that the primary data stores are unlikely to fail suddenly (unlike P2P nodes), and also that in a particle physics scenario, data is likely to be written once and read many times. Consistency is not, therefore, such an important issue as it is in distributed databases and so it is not necessary to restrict replication to keep consistency.

It is clear, however, that there cannot be uncontrolled replication. Sites have only limited storage space and cannot accommodate replicas of every data file on the grid; networks have only limited capacity for transferring them. A grid must therefore have a replica management system which follows some replication strategy in order to optimise the location of replicas and thus improve the performance of the grid.

Most data grid replication strategies to date have been investigated only in simulations. In [113], a number of replication strategies were investigated, including some similar to those used by Gnutella and Freenet, and found that although there was no best strategy for all scenarios, the path-based Freenet-style replication gave consistently good results. In [114], a "Dataset Scheduler" at each site kept track of the popularity of its datasets. When their popularity exceeded a certain threshold, they would then be replicated to some other grid site, either at random or to a site with low computational load. In [115], a cost-based model is used to decide when the gain from having an extra replica to read from outweighs the loss from keeping consistency. The cost model used requires knowledge of the whole system, however, unlike the OptorSim economic model which is presented below.

Finally, an example of a real grid which performed automated replication was the European DataGrid (EDG). When a job requested a file, the Replica Location Service was contacted to find a list of all existing replicas of that file. The Replica Optimisation Service (ROS), which was described in Chapter 3, was then used to find the best replica to access. If this was not on the local site, the one which could be copied to the local site most quickly was chosen and the replication performed.

6.3.2 Design of a Replica Optimisation Service

Having examined briefly some methods of data replication in various fields, the main design decisions for the OptorSim replication model are presented before discussing the strategies which have been implemented.

First, the choice between a centralised, hierarchical or distributed service must be made. The distributed nature of the EDG ROS was already shown in Chapter 5. If each site has its own Replica Optimiser (RO), it is able to manage its replica content autonomously and there is no single ROS server, which could come under very heavy load and would be a single point of failure. It also removes the need for an individual RO to know the state of the whole grid, which would be impractical for a system on the scale of a data grid.

The decision must then be made on whether to use a push or a pull model. In a push model, the site already containing a particular data file would decide when to replicate it and where to. In a pull model, a site which did not initially have the file would decide when to replicate it to itself, and where from. The push model was used in [114] and [113]; the problem with this is that in a real grid, sites would be unlikely to accept spontaneous replication of data to their SE from some other site, although it depends on individual sites' policies. To respect sites' autonomy, a pull model is therefore favoured. Each RO should be responsible for optimisation only at its own site; this is what separates the OptorSim model from the grid replication models mentioned in the previous section.

Finally, a replication trigger must be chosen: some condition which will cause the RO to consider replicating some file. When the SE is requested for a file which it does not have, for example, this could trigger the first stage of the replication strategy for that RO, as described below. Another possible trigger could be a file on some other SE reaching a certain level of popularity. This would require monitoring of all file popularities, perhaps in a central database or by a publish/subscribe method (one SE could subscribe to another SE to receive regular updates of its top ten most popular files, for example). The simplest, however, is to trigger on a file request, and so this is what has been implemented in OptorSim.

The overall aim of such a distributed replication model would be to achieve global optimisation as a result of local optimisation by each RO. Each RO therefore has two goals:

- Minimisation of individual job execution cost: each user wants their job at as low a cost as possible. The RO therefore tries to minimise the execution cost of each job which runs on its site, defining cost as the total time taken for the job to run.

- Maximisation of usefulness of locally stored files: if the RO aims to keep on its site those files which will be most useful either at that site or at neighbouring sites, good overall distribution of data should emerge. The average data access costs and hence running times of jobs should then decrease.

A good replication strategy will be one which fulfils these goals, achieving a good trade-off between individual running times and overall resource utilisation.

6.3.3 Stages of a Replication Strategy

Replication can be logically separated into three stages through which any replication strategy must proceed. These are delineated as follows, with each stage depending on the success of the preceding stage.

1. *Replication Decision:* First, given the trigger condition, the RO at a site must decide whether or not to replicate the file to its local Storage Element. If it decides not to replicate, the file must be read remotely.

2. *Replica Selection:* Next, if the RO has decided to replicate the file, it must choose which existing replica to copy.

3. *File Replacement:* Lastly, if the local SE does not have sufficient space to store the new replica, one or more existing files must be deleted until there is enough space.

These three stages will each be discussed for the replication strategies which follow.

6.3.4 Simple Replication Strategies

As well as the economic model of file replication which is described below in Section 6.4, some simpler replication strategies have been implemented in OptorSim. The problem of choosing which file(s) to delete from a full SE to make room for new replicas is analogous to the well-researched problem of cache replacement. Caches are used in CPUs, disk buffers, file systems and, perhaps the most relevant to grid systems, in web browsers and proxy servers. It would therefore be instructive to examine some cache replacement algorithms and apply these to replica management.

Two of the most basic cache replacement algorithms are *Least Recently Used (LRU)* and *Least Frequently Used (LFU)*. When the LRU algorithm is applied to a cache, the data object (file, web page, data block or whatever is relevant to the cache in question) which was accessed least recently is deleted to make room for the incoming object. This is simple to implement for small caches, although it may be computationally expensive for large ones. Its main drawback is that it does not take account of access frequencies, and so may delete an object with a high access frequency if it has been idle for some time.

The LFU algorithm, on the other hand, uses the history of accesses to predict future popularity by deleting the data which were accessed least frequently in the past. This can, however, lead to pollution of the cache by previously popular but now unused data which are not deleted because of very high access counts. This is usually counteracted by putting an ageing policy on inactive data so that eventually they are flushed from the cache.

There are many variations of these algorithms which aim to combine the best features of both. The Segmented LRU (SLRU) algorithm [116] divides the cache into two segments, a *protected* and a *probationary* segment, each ordered from most recently to least recently used. Data requested more than once is put in the protected segment, from which the least recently used data can be transferred back to the probationary segment, while the least recently used data from the probationary segment are deleted from the cache. The LRU-K algorithm [117] uses the last K accesses of a cached object to estimate the inter-arrival time of requests and the object with longest inter-arrival time is deleted. LRU-1 corresponds to the traditional LRU algorithm.

The advent of the World Wide Web has given new impetus to cache management research, with the recognition that these traditional algorithms may not be effective with the variable sizes of objects and larger timescales of web caches. This has led to the development of new algorithms such as "GreedyDual-Size" [118], which accounts for network costs and file sizes so that the deleted document is that with the lowest ratio of cost to size, and the Least Relative Value (LRV) algorithm [119], which selects the document with lowest relative probability of the document being accessed again in future.

As it is not yet known what the access patterns of production-level data grids will be like, it is not known which of these families of algorithms would be most effective for file replacement on SEs. As the LRU and LFU are "standard" algorithms, however, it was decided to include them in OptorSim, rather than the more sophisticated algorithms, for comparison with the economic model presented below.

The two simple replication strategies which have been developed, then, are named LRU and LFU respectively, after the algorithms used for the File Replacement stage. Each SE keeps a history of its file accesses, of a length of time specified by the user in the parameters file. When a file needs to be deleted, the candidate is chosen by applying the LRU or LFU algorithm, depending on which strategy has been chosen, to the access history. The Replication Decision and Replica Selection stages for the two strategies are exactly the same: the decision to replicate is always taken, again imitating caching systems, and the replica which will take the shortest time to access, given the current state of the network, is chosen as the source. The results of using the LRU and LFU strategies and their comparison with the economic model are shown in Chapters 7 and 8.

6.4 An Economic Model for Data Replication

Although replication is a well-known method of improving access to data in a number of fields, as was discussed above, grid computing poses some new problems. First, a grid may consist of heterogeneous distributed resources ranging from desktop PCs to supercomputers. Second, grids consist of autonomous sites. Sites have different owners, who may have different goals and set different policies. Third, it is a dynamic system. Sites may suddenly become unavailable, network capacities fluctuate, CEs can rapidly change from idle to busy. Lastly, there are a large number of users, spread across different time zones, whose usage patterns may be complex and time-varying.

The complexity of such a system, in which there are a number of autonomous entities with their own goals, yet which must reach a stable equilibrium if it is to function well, naturally brings to mind an economic system or marketplace. Such a marketplace is completely decentralised, being controlled by supply and demand. It is able to reach an equilibrium by the economic interactions between its entities, which would hopefully correspond to some optimal distribution of goods or services. It is also responsive to sudden changes in the system, with the ability to regain its equilibrium when disturbed.

An economic model also has the advantage that cost management is naturally incorporated. In future, grids may need payment or accounting mechanisms for the use of resources. In commercial grids, the necessity of this is clear: resource owners allow their resources to be exposed on the grid solely that they may profit from them. In research grids, such as for particle physics or biomedical science, this is not the driving motive. Nevertheless, funding bodies or investors may wish to see how resources are being used; different experiment groups may have higher priority or a larger share of the resources. An economic system which used "tokens", where users could be allocated a certain number of tokens according to their priority, would then be useful in generating accountability.

Applying this to replication, grid sites become sellers of the replicas which they have stored in their SEs. These are bought by other sites, either because they are needed by the CE for a particular job, or because it would be profitable for them to keep it on the SE for future use. As replicas are bought and sold, eventually the individual optimisation of files at each site leads to the optimisation of replica location across the grid as a whole.

The use of economic models in distributed systems is by no means a new idea. The use of a computational economy for scheduling with the Nimrod-G resource broker has already been mentioned. Spawn [120] uses slices of CPU time on different types of workstation as commodities in a computational marketplace, with users selling time on their resources to the highest bidder. POPCORN [121] allows buyers and sellers to trade CPU time via the Internet. Mariposa [122] uses an economy for managing data objects and query processing in distributed databases, and in particular allows sites to buy and sell replicas of database table fragments. In the Mariposa economy, each site manages a certain volume of storage, and competes with other sites to allocate its CPU, I/O and storage resources in such a way that its income (based on an artificial currency) is maximised. At the start of a query, a budget is allocated for that query. Clients find likely sites, ask for bids then choose the winning site to run their query. Queries may be broken into subqueries which run at different sites, and to run a query or subquery, sites must purchase the database fragments which are relevant to that query. The situation has parallels with the grid replication case and with the OptorSim economic model presented below.

These are just a few of the most relevant examples; however, they show that the majority of work has been on trading of computing power, with the exception of Mariposa. This gives even more motivation to consider the use of economic models for data replication in grid systems.

6.4.1 Common Economic Models

In economics, there are many market models which could be applied to grid systems, and several of these are listed here.

Commodity Markets

In a commodity market, resource owners specify a price per unit, which can be fixed or can vary according to supply and demand. Users then pay according to how much of the resource they use.

Posted Price

A posted price model is similar to a commodity market, but as well as the specified regular prices, the resource owners may post "special offers" to tempt new customers or sell more of an under-utilised resource.

Bargaining

In this model, buyers and sellers start with their "ideal" price, which may be very low for the buyer and very high for the seller, and negotiate until they reach a mutually agreeable price or until one of them is unable to meet their objectives by negotiating any further.

Auctions

In auctioning, a seller offers the resource to many buyers and sells it to the highest bidder. Alternatively, in a *reverse* or *procurement* auction the buyer calls for bids from many sellers, and buys from the lowest bidder. Theoretically, ordinary and reverse auctions are exactly the same and henceforward no distinction will be made between them.

There are several different types of auction which may be held, the main ones being:

- *English (second-price open-bid)*: Bidders must increase the current bid by some increment, starting from a minimum price set by the auctioneer, until only one interested bidder is left, who then wins the auction and pays the amount of the current bid. This is a second-price auction, however, because the winner is the bidder who would be willing to pay more than the current bid; in other words, out of the private valuations of all the bidders, bidding stops at roughly the level of the second-highest valuation. The price paid need only be higher than the second-highest bid by the minimum increment.

- *Vickrey (second-price sealed-bid)*: Bidders submit sealed bids, at the same time, and the winner is the bidder with the highest bid. He pays, however, the amount bid by the second-highest bidder. A Vickrey auction is strategically equivalent to an English auction, with bidders' best strategy being to bid at their true valuation [123].

- *Dutch (first-price open-bid)*: The auctioneer starts the auction with a very high price, which is incrementally reduced until one of the bidders indicates their willingness to pay that price.

- *First-price sealed-bid*: Identical to the Vickrey auction, but the winning bidder pays the price he has bid rather than the second-highest price. This is strategically equivalent to the Dutch auction, with bidders tending to bid less than their true valuation.

It was shown by Vickrey in [123], in which he introduced Vickrey auctions and showed their logical equivalence to English auctions, that second-price auctions have a higher probability of leading to optimal allocation of resources than first-price auctions.

Tendering

This is based on the way in which businesses or government organisations contract companies to perform services by issuing a call for tenders. The buyer specifies its requirements and sellers which can fulfil these requirements reply with a sealed bid. The bids are then evaluated and a winner selected. Sealed-bid auctions can be considered a special case of this kind of tendering, in that the evaluation is performed on only one criterion, price. With tenders, however, price can be weighed up against other considerations such as speed and quality of service.

Bartering

Rather than exchanging "money" (whether real or token), in a barter economy goods or services are sold in exchange for a certain quantity of other goods or services. This was traditionally used in economies with no monetary system or where the currency was unstable.

Further details of these, and of some other economic models which can be applied to grid computing (and to job scheduling in particular) are given in [124]. Applying them now to replication, the author's contention is that the simplest and most effective model to use is auctioning. A commodity market is suitable when there are clearly defined buyers and sellers, with the sellers having a stock of some commodity. This could work for CPU time, for example. With file replicas, however, sites can be both buyers and sellers and no site would have a stock of one particular replica. It would also be very difficult for sites to keep prices at a suitable level in such a distributed and highly dynamic environment. The commodity market is therefore unsuitable for replica optimisation. By corollary, the posted price model is also unsuitable.

The bargaining model, while suffering to some extent from the same problems as the commodity market in that sites would need to know the true market price of the replica, would also incur large overheads from the message passing associated with the bargaining process. Tendering for replicas would be equivalent to auctioning, as replication is a discrete process to which the other considerations, such as quality of service, do not apply. Bartering, although the concept may appeal in the interests of simplicity (a monetary system for the grid would not have to be introduced), would require each buyer and seller to have something that the other wanted.

Auctioning, therefore, stands as the most viable candidate. Replicas are *"a small number of discrete, indivisible units"* [123] and therefore highly suitable for auctioning. It allows sites to act as both buyers and sellers. Auctioning allows quick reaction to the changing market, as prices are set according to the bidders' valuations. Although open-bid auctions on a grid would cause heavy overheads from message-passing, sealed-bid auctions, where bidders bid only once, would reduce this. Given that second-price auctions have been shown to lead to more optimal resource allocation, it was therefore decided that OptorSim should implement a Vickrey auction mechanism for the economic model.

6.4.2 Architecture of the Economic Model

Consideration is now given to how this Vickrey auction-based economic model has been applied as a replication strategy in OptorSim. Recalling the three stages of a replication strategy from Section 6.3.3, the first stage is deciding whether or not to replicate. In the economic model, this is decided by the relative value of the potential replica compared to those already in the SE. In particular, the *future* value of the files is estimated and, if the potential new replica is valuable enough, it is replicated. If there is still space on the SE, replication occurs by default. The method of estimating future file values is described in Section 6.4.4.

The second stage, that of selecting the best replica, is performed by the auction mechanism. This requires several new components to be added to the large-scale architecture of OptorSim as it was shown in Figure 5.1. An *Access Mediator* (AM) is needed to take file requests from the CE and start an auction for the replica. A *Peer-to-Peer Mediator* (P2PM) is needed for sites to communicate with one another and pass the auction messages. Finally, a *Storage Broker* (SB) is needed to handle auctions on behalf of the SE, bidding if the SE contains the file and perhaps starting a secondary auction to get the file if the SE does not already contain it. The exact mix of components in any given site's RO depends on whether the site has a CE, SE or both, as Figure 6.1 shows.

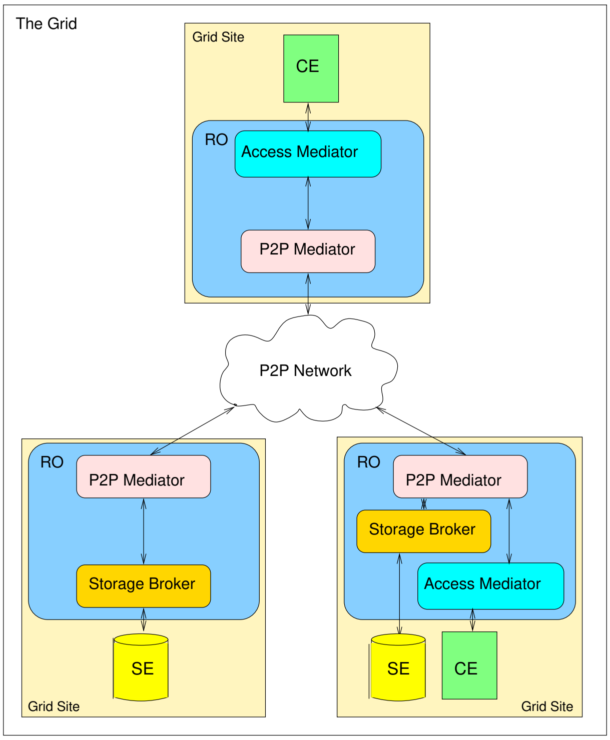

Figure 6.1: Architecture of economic model components.

Finally, for the third stage of the replication strategy, that of choosing which file(s) to delete if deletion is necessary, the least valuable file in the SE (as estimated by the prediction function) is deleted until there is sufficient space for the new replica.

6.4.3 Implementation of the Auction Model

The implementation of the auction model again makes use of the Java threading facilities, representing each auction by a thread. A pool of Auction threads is kept, and when a CE requests a file, the RO at that site calls its AM. The AM, acting as auctioneer, starts a new Auction thread and sends a call for bids to all the SBs via the P2P network. Those SBs which have the file in their SE send a bid based on the cost of transferring the file to the requesting site. If the SE at a site does not have the file, this triggers the replication decision at that site: the SB considers whether it is worth replicating the file for itself, and if so, begins its own auction and sends a bid to the parent auction. This is known as a *nested auction* and is discussed further below. After waiting a certain time to collect the bids, the AM selects the one with the lowest price and sends an announcement of the winner to all the SBs. The location of this best replica is then passed back to the CE, which accesses the file and begins to process it.

The UML diagram in Figure 6.2 illustrates the auction process for a site where the local SB wins the auction. Note that when the AM sends the call for bids, this

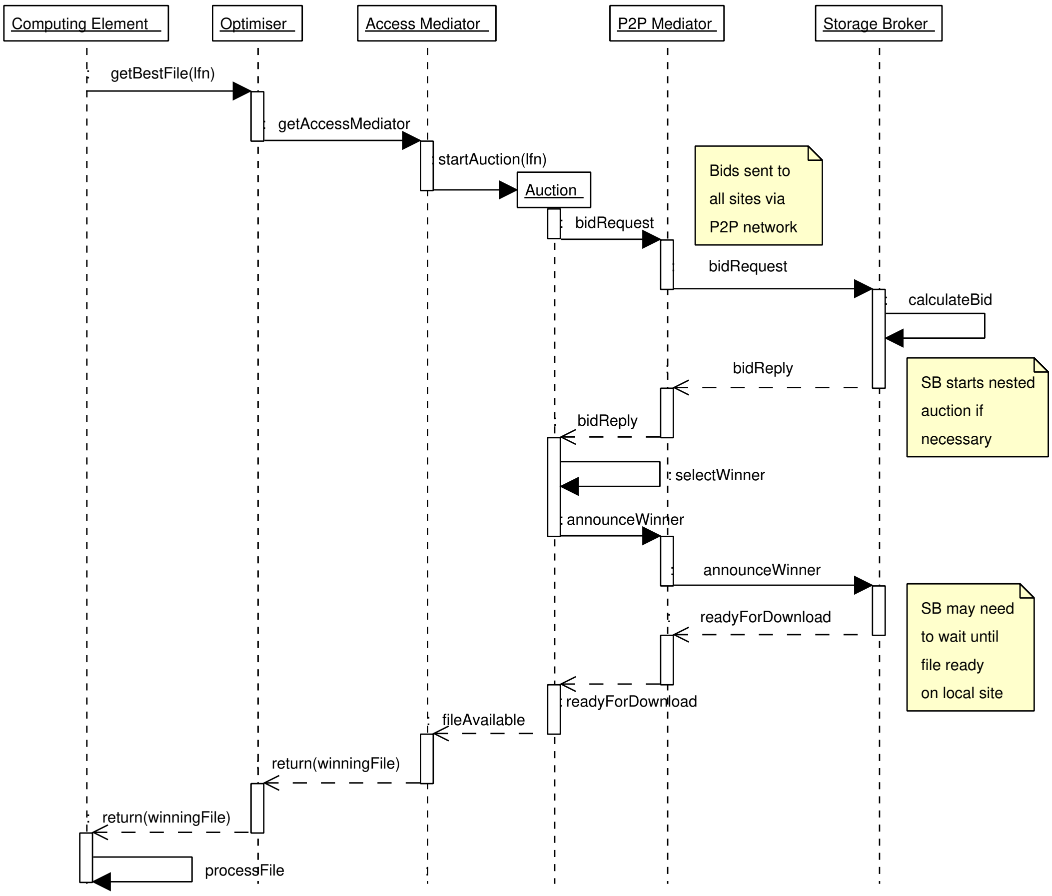

Figure 6.2: UML sequence diagram of interactions between site components during an auction.

includes the SB on the local site. Although the bid request message is propagated throughout the grid (up to a limit defined in the parameters file), and bid replies are received by the Auction from all SBs which have or are getting the file, only the messages for one of the SBs (in this case, the local one) are shown here, for the sake of simplicity.

If the file is already present on the site, the local SB can send a bid with a cost of zero (given the assumption of infinite bandwidth within a site) and hence will always win the auction. If the file is not already on the site at the time the AM issues its call for bids, the SB will evaluate the worth of the file and decide whether or not to start a nested auction. If it does, and the file is replicated to the local site, then the SB will again be able to bid in the parent auction with a bid of zero and thus win the auction. The nested auction is not explicitly shown in the diagram, but would take place during the `calculateBid` process shown for the SB. This implementation of the auction process thus combines the first and second stages of the replication strategy, although their logical separation is as was shown in Section 6.4.2.

While auctioning by the AM in response to a file request from a job results in a new replica at the requesting site (if it is economically optimal for the replication to take place), nested auctions on other sites lead to third-party replication. In other words, a site replicates the file even though it does not need it, as there is a high possibility that the replicated file will be needed again, either by that site or some other site, which would lead to increased revenue for that SB. This could lead to "hot-spots" of data at optimal sites on the grid - perhaps well-connected sites neighbouring smaller sites which cannot afford to keep many replicas themselves. This would help avoid bottlenecks on the network and thus contribute to increased performance of the grid as a whole.

The main limitation of the auction protocol is the overhead in sending messages to sites and waiting for replies. While this overhead is much lower than it would be using a bargaining model, for example, it puts the economic model at a disadvantage with respect to the simpler replication strategies.

6.4.4 File Value Prediction

To evaluate the worth of the files currently in its SE, and that of a possible new replica, it is necessary for the RO at a site to estimate their future usefulness. In the simple replication strategies, this was done when deleting files simply by examining the access history and defining their value by how recently, or how frequently they had been used. For the economic model, the estimates of future value are slightly more sophisticated.

The future value of a file can be defined as the income the SB receives from requests for that file over a future time period T. Formally,

$$V(i,k,n) = \sum_{j=k}^{k+n} \delta(i, i_j) \tag{6.1}$$

where the value function $V(i,k,n)$ calculates the income from file F with ID i starting at time t_k, over the next n file requests. The δ in the sum is the Kronecker delta and, for simplicity, file prices are considered unitary, so the problem is reduced to estimating the number of times a file will be requested in the next n file requests; or, finding an analytical form for an estimation function $E[V(i,k,n),r]$, where r is the number of file requests made before t_k.

Correlation of File Requests

To predict the future pattern of file requests, some assumptions have to be made. A reasonable assumption which can be made, especially in a high-energy physics context, is that if some file is requested from the SB, then the next file to be requested will probably have similar content to the first. In other words, the assumption is of *sequential correlation* between the files which are requested.

To model this, the file space F is defined to be the set of all possible files: all the files defined by the user in the job configuration file. The file ID space $I \subset \mathbb{N} \cup \{0\}$ is then defined as the set of file identifiers, which are also defined, in the job configuration file. Each file $f \in F$ maps to one ID $i \in I$. Similarity of file content is then defined by closeness of file IDs; for two file IDs i_1 and i_2, the smaller the value of $|i_2 - i_1|$ the more similar the corresponding files f_1 and f_2.

The history of file requests can then be represented by a random walk in the file space F. Starting from a file ID i_0, a random step s can be taken anywhere in the interval $[-S, S]$, where $2S$ is the maximum difference between the IDs of any two successively requested files. To get an explicit form for $E[V(i,k,n),r]$, then, requires the choice of a probability distribution for a generic random step s. Two prediction functions have been implemented in OptorSim, a binomial-based function and a Zipf-based function, and these are now presented in turn.

Binomial-based Prediction Function

For the binomial-based prediction function, each step s is assumed to be an discrete random variable with a binomial distribution of width S. The starting point of the random walk, i_0, can be approximated to the centre of the distribution, $\bar{i}$. To define what n, S and $\bar{i}$ are, it is necessary first to fix a time interval T' in the past, as the basis of the estimation, and a time interval T in the future, for which the prediction is to be made. Then r is the number of requests received in T'. If the arrival rate of requests remains constant, n is then

$$n = r\frac{T}{T'} \tag{6.2}$$

and for $T = T'$, $n = r$.

To estimate the distribution width S, the past history can also be used. Each file ID i_j represents the file requested at the j^{th} step and is obtained by starting at the initial file ID i_0 and following the random walk through the first j steps. That is,

$$i_j = i_0 + \sum_{k=1}^{k=j} s_k \tag{6.3}$$

and since each s_k is an independent random variable with binomial probability distribution, each i is then also an independent random variable with the same distribution and with variance $\sigma^2 = \frac{jS}{2}$. If the random walk is done in reverse, from t_0 to t_{-r}, then for each j from 1 to r, i_{-j} should have a variance of $\frac{jS}{2}$. The best estimate of the variance is given by $(i_0 - i_{-j})^2$, so calculating a weighted average (in favour of recently requested files) gives

$$S = \frac{2}{r}\sum_{j=1}^{r}\frac{(i_0 - i_{-j})^2}{j} \tag{6.4}$$

Finally, to fix a value of $\bar{i}$, it is simplest to calculate the weighted average of the values of i in the last r file requests, again weighted towards the most recent requests:

$$\bar{i} = \sum_{j=1}^{r} \frac{i_{-j}}{j} \tag{6.5}$$

Combining the results from (6.2), (6.4) and (6.5), the future value estimation $E[V(i,k,n),r]$ can now be calculated. The binomial-based prediction function, then, is implemented in OptorSim by keeping an access history of the file accesses on each SE, with T' defined by the user in the parameters file, and using the values of n, S and $\bar{i}$ as defined above to estimate the future value V of the file.

Zipf-based Prediction Function

The Zipf distribution and its possible application to grid access patterns was discussed in Chapter 5. An economic prediction function has also been developed in OptorSim based on a Zipf-like distribution of file requests.

First, the files in the access history at a site are ranked according to the number of requests received for each in the history. The more popular the file, the higher the rank, with 1 being the highest ranking. The future popularity is then predicted by finding the position of the requested file in the rankings. Formally,

$$V(i,k,n) = \frac{T}{T'} j^{-\alpha} \tag{6.6}$$

where T is the future time interval for which the popularity is predicted, and T' is the past time interval from which it is predicted. The file ranking is j, and α is the Zipf parameter, which is found for the distribution of the rankings in T' and assumed to be the same in T.

It would of course be possible to investigate many other prediction functions, but the binomial- and Zipf-based are sufficiently different to show the effects of the prediction function on the economic model, and to compare this basic economic model with the simple replication strategies which were discussed earlier. These two prediction functions, therefore, are the only two investigated here, with the results presented in Chapters 7 and 8. The OptorSim economic model is not yet as sophisticated as that implemented in Mariposa, in that jobs are not allocated budgets which must be satisfied. Imposition of budgets and an artificial currency would prove an interesting avenue for future work.

6.5 Summary

This chapter has discussed the problem of data grid optimisation both in general terms and specifically in relation to job scheduling and data replication strategies. Some of the different existing approaches to job scheduling were discussed, for different kinds of grid environment, and general principles drawn which were then used as a basis for the scheduling algorithms implemented for the OptorSim Resource Broker: *Random*, *Queue Length*, *Access Cost* and *Queue Access Cost*.

The focus was then moved to data replication, with an examination of replication in the fields of distributed databases, peer-to-peer networks and prior work in data grid replication. Lessons drawn from these were then used in the design of the OptorSim Replica Optimisation Service, and three logical stages were identified through which any replication strategy must proceed. Some simple strategies, LRU and LFU, were presented, borrowing from cache management strategies.

An economic model for replication was then presented, using a Vickrey auction model, before showing how this has been implemented in OptorSim. The two file value prediction functions which have been implemented were shown, giving two flavours of the economic model: binomial-based and Zipf-based. Table 6.1 shows how each of

Strategy	Replication Decision	Replica Selection	File Replacement
LRU	Always replicate	Fastest transfer	Least Recently Used
LFU	Always replicate	Fastest transfer	Least Frequently Used
Economic model (binomial-based)	Binomial prediction function	Auction	Binomial prediction function
Economic model (Zipf-based)	Zipf prediction function	Auction	Zipf prediction function

Table 6.1: Summary of replication strategies implemented in OptorSim.

the replication strategies implement the three stages of a replication strategy.

In the highly distributed, dynamic and heterogeneous environment of a data grid, it remains to be seen whether the traditional or economic models would be most scalable and effective. None have so far been deployed on any grid; although EDG did have a replica optimisation component, it did not perform file replacement. The evaluation of these different replication strategies in OptorSim, including realistic models of analysis performed on LCG during LHC data-taking, is given in the next two chapters.

Chapter 7

Simulation Results: Basic Topologies

Following the description of OptorSim in Chapter 5 and the discussion of optimisation strategies in the last chapter, this chapter now presents some results of running the simulation with these strategies, with a variety of simple grid topologies.

First, some of the most important variables in a grid topology are explained, followed by some of the metrics by which a simulation can be evaluated. Two basic network structures - a tree and a ring topology - are then simulated and the results discussed, followed by combinations of the tree and ring into simple hybrid topologies. In [2], the scheduling and optimisation algorithms were tested for a number of very simple grid configurations involving two or three sites, showing that the simulation behaves as expected in these situations. These results build on and complement that foundation. Some more complex hybrids are then simulated, culminating in an "LCG-like" hybrid which will serve as a prototype for the simulation of the LHC Computing Grid.

7.1 Grid Characterisation

The inputs to OptorSim can be many and complex, so some characterisation is necessary if comparisons between different simulations are to be made. The grid configuration used and the set of jobs submitted are the two main inputs to a grid simulation. The grid configuration includes the number, composition and layout of sites on the grid. The job configuration includes the dataset for each job, site policies, probability of a job being submitted and execution times. The main attributes of a particular grid topology can be summarised in the following three characterisation metrics, which were also used in [2]:

- The *storage metric*, D: the ratio of the average SE size to the total dataset size, where the total dataset consists of all the files defined for all the jobs. That is,

  $$D = \frac{< SE\ capacity >}{Total\ dataset\ size}$$

  The size of D then indicates the likelihood of replication occurring. If $D > 1$, an average SE has more than enough capacity to hold all the files that jobs could require, and so there will never be any deletion of files required. This means that the replication strategy chosen will have little effect. If $D < 1$, the replication strategy becomes more important as the SE is not capable of holding all the files, and so deletion must take place. For $D << 1$ due to a large dataset, however, replication will begin to lose its advantage compared to remote access, as each new job is then more likely to request files which have not been requested before.

- The *network metric*, C: the average connectivity of a grid site. This is defined as

  $$C = \frac{\sum_{sites} Site\ connection\ bandwidth}{Number\ of\ sites}\ [Mbps]$$

  The value of C thus reflects the speed at which files can be transferred around the grid. If C is high, files can be transferred quickly and jobs can thus finish faster. If C is low, there may be delays in jobs being processed, leading to queues at sites. Different scheduling strategies may thus be appropriate for grids with different values of C. It also has implications for the replication strategy chosen, as a strategy which makes poor use of the network will perform badly in a grid with low connectivity.

- The *compute metric*, P: the average processing power at a site,

$$P = \frac{\sum_{sites} Site\ processing\ power}{Number\ of\ sites}\ [kSI2000]$$

  A configuration with a high value of P will process files faster, after data transfer has taken place. A low value of P will cause not only longer processing times for jobs but also longer queueing times, as a backlog of work builds up at sites. The relative importance of P and C depends on the type of applications run on the grid. For a data-intensive application, the network metric C is likely to be more important, while for a compute-intensive application P will naturally be more important.

D, C and P will be used in the future sections of this chapter when comparing different grid topologies. They can be applied not only at the level of the whole grid, but also for subsets of the grid. If only some sites are running jobs, for example, a value of D could be calculated using only those sites.

Some other important parameters which can be varied include the number of jobs submitted to the grid, the length of file access history kept, the level of non-grid traffic on the network, and the initial distribution of files in the grid. While this chapter cannot cover the whole parameter space associated with the simulation, some of the most interesting of these parameters will be examined.

7.2 Evaluation Metrics

As well as the characterisation metrics described above, a number of *evaluation metrics* can be defined with which the results of simulations can be compared. In a real grid, different people may have different criteria by which they evaluate the success of the grid. An ordinary user of a data or computational grid will most likely be interested in the time a job takes to complete (the user of a service grid will be more interested in achieving good quality of service, but this is not considered here). The owners of grid resources, on the other hand, will want to see their resources being used efficiently. These two goals are not always mutually attainable, but use of the following three evaluation metrics can help quantify particular grids and optimisation strategies with respect to these goals.

- ***Mean job time***: This is defined as the average time a job takes to run, from the moment it is scheduled to a CE by the Resource Broker to the moment when it has finished processing all the files it requires. To calculate this, the total time taken by each job is recorded, summed and divided by the total number of jobs. Ordinary grid users will want the mean job time to be minimised, and it is considered here to be the primary evaluation metric. Thus, in the plots of results which follow, those of mean job time are presented in bold.

- *CE Usage*: At the level of a single CE, the CE usage is the percentage of time during the simulation that it is active. The CE usage of each individual CE is aggregated to give a value for CE usage across the whole grid. CE usage is a metric that could be of interest to resource owners, as high CE usage would mean that the scheduling policy was effective at balancing workload across the grid. Low CE usage, on the other hand, would mean that some CEs had long queues while others were underused.

- *Effective Network Usage (ENU)*: The optimisation strategy chosen must make effective use of the network resources. While replication is important in reducing access times and increasing availability, too much replication will result in inefficient use of the network. A good replication strategy will replicate only in the interests of reducing network traffic in the future.

  This can be quantified by the ENU, which is defined as:

  $$ENU = \frac{N_{remote\ file\ accesses} + N_{replications}}{N_{remote\ file\ accesses} + N_{local\ file\ accesses}}$$

  This sets the ENU as the ratio of file requests which use network resources to the total number of file requests, and so the lower the ENU the more efficient the use of the network.

- *Hit Rate*: The hit rate is the number of times a file request by a job is satisfied by a file which is already present on that site's SE. It indicates the success, or otherwise, of the replication strategies in making as many files as possible available locally.

Armed with these characterisation and evaluation metrics, various simulated topologies can now be investigated. For each of the plots given in this and the next chapter, the data points are generally the mean of at least 3 simulation runs and both the standard deviation and standard error are indicated. This is done by showing error bars on the data points, corresponding to the size of the standard deviation but with tick marks giving the standard error. The standard deviation is useful to indicate the variability of the simulation for the given parameters. A scheduling or replication strategy with low variability would be more predictable, and thus perhaps more desirable, than one with a large variability.

7.3 Simulation Aims

Before describing the details of the simulation setup and results, it is worth pausing to consider the aims of simulating these simple topologies. There are a number of questions which can be addressed with these smaller grids which cannot be answered with a larger, more complex grid. First, the way the underlying topology affects the behaviour of the scheduling and replication strategies can be investigated. This includes not only the basic structure but also the importance of interconnections between sites, the location of the data source, and how close other sites are to the data source. Second, the effects of a grid's scale and complexity can be observed, as the number of grid sites is increased. Third, observations can be made about the behaviour of the algorithms in regimes which it may not be possible to reach with a very large simulation.

7.4 Simulation Setup

Some simple configurations of jobs, files and site resources were defined. Four job types were defined, each of which had an associated dataset of 2,500 files. Each file was defined to be 1 GB in size, making the total dataset size 10 TB. Any one job, when running, was configured to require 100 of the files from its dataset, which is 4% of the total. The choice of values for these, while partly arbitrary, was also driven by real grid scenarios. Files of raw LHC data, for example, are expected to be about 2 GB in size, and experiment analysis datasets approximately $\mathcal{O}(100)$ TB, although this is clearly variable. The proportion of a dataset required by an individual job was chosen such that it would allow the access histories to generate good statistics and thus allow the replication algorithms to perform at their best. The aim was to have a basic setup but with reasonable values of parameters.

The site resources were set up as follows. Except where indicated below, a single site initially held all the data files and accordingly had an SE size of 10 TB. This "master site" had no CE, representing a data source. The other sites each had an SE of size 1 TB, which could accommodate the files for ten jobs. This allowed file deletion to begin after a short time, thus giving scope for the file replacement algorithms of each replication strategy to act.

The processing time required per file was set at 5 MSI2000-s; that is, the linear processing factor f_{lin} from Equation 5.1 was set at 5 kSI2000-s per MB, which is in line with the LHC experiments' predicted processing time per event; the latency factors were set at zero for simplicity. Each CE was allocated 100 worker nodes each of processing power 0.5 kSI2000 (approximately the processing power of a 1.5 GHz Pentium IV PC). All sites apart from the master site could accept all job types, and all network links were set at 1 Gbps.

The simulation parameters were set as follows (a fuller explanation of each was given in Chapter 5). Each simulation ran 1000 jobs, submitted to the sites at intervals of 2.5 seconds. Each site could hold up to 200 jobs in its queue, and a job accessed its files using the sequential access pattern. For simplicity, no background traffic was included. For those optimisation algorithms which made use of the access history, the access history length was set at 10^9 ms. This is about ten times the average job time, which was shown in [2] to be the length of access history at which the algorithms begin to perform optimally.

7.5 Simple Topologies

7.5.1 Tree Topology

The first topology to be examined is a simple hierarchical or tree model, as shown in Figure 7.1, with a total of 15 sites. The characterisation metrics for this configuration are shown in Table 7.1. This shows a fairly high value for the network metric C, while the storage metric D is small (< 1) but not extremely small ($<< 1$) and should thus give good scope for the replication algorithms to perform.

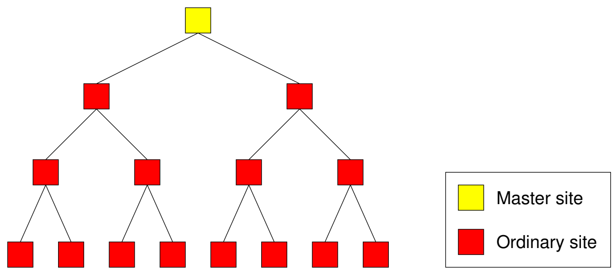

Figure 7.1: Tree topology with 15 sites.

Metric	Value
D (all sites)	0.16
D (ordinary sites)	0.1
C (Gbps)	1.9
P (kSI2000)	46.7

Table 7.1: Characterisation metrics for the tree configuration.

The four scheduling algorithms described in Chapter 6, together with the four optimisation strategies - Least Recently Used (LRU), Least Frequently Used (LFU), Economic Binomial (Eco Bin) and Economic Zipf (Eco Zipf) - were examined, together with the control case where no replication takes place.

Figure 7.2 shows the mean job time, CE usage, ENU and hit rate respectively for each combination of optimisation and scheduling algorithm. It is immediately obvious that using any of the replication strategies greatly improves the performance, with

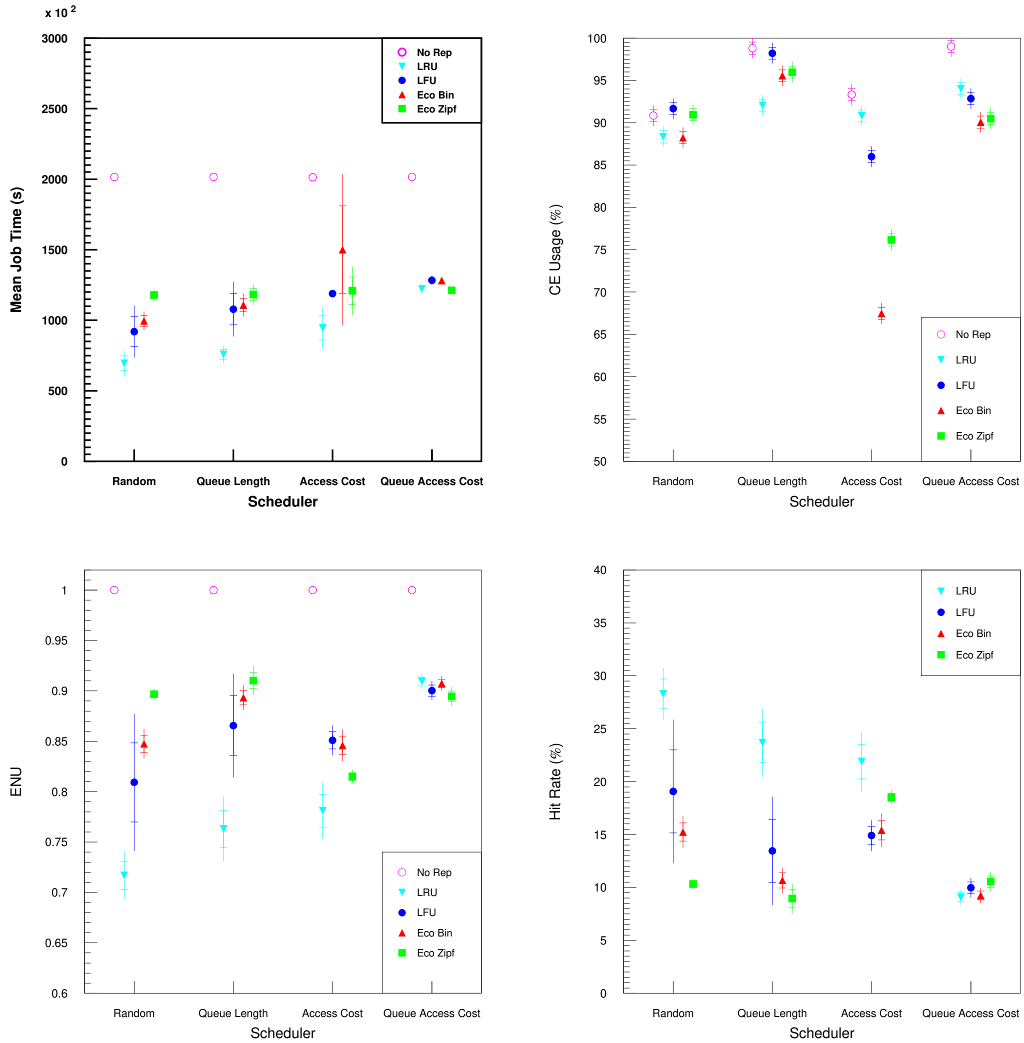

Figure 7.2: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different scheduling and replication algorithms, using the tree topology.

mean job time reduced by about 50%. The LRU is the fastest, followed by the LFU, with the economic models being slightly slower. Examining the effect of the different scheduling algorithms, there is not a large variation in mean job time. The Random and Queue Length schedulers are, however, slightly faster than those involving the access cost. This is because at this scale of grid, with fairly high bandwidth links, data location is less important than even distribution of jobs around the grid.

Looking at the other metrics, Queue Length and Queue Access Cost give the highest CE usage, as they use the CE status in allocating jobs. Access Cost generally gives the lowest, particularly for the economic models, as it makes no use of queue information when scheduling. CE usage with replication is slightly lower than without replication. ENU is consistently lower, showing that replication does make better use of the network, with LRU having the lowest ENU. The LRU also has the highest hit rate, reaching 28% with the Random scheduler. For this simple tree topology, therefore, the LRU strategy performs best.

7.5.2 Ring Topology

The second simple topology examined is that of a ring. This was tested with different degrees of connectivity, ranging from each site being connected to 2 others (dual connectivity) to 4 (quadruple connectivity) to 8 other sites (octuple connectivity). These are shown in Figure 7.3.

As the only characterisation metric which varies between these configurations is C, Table 7.2 shows the metrics for all three configurations. The storage and compute metrics are the same as with the tree topology, while the network metric ranges from approximately the same (for the dual connectivity ring) to several times higher.

First, the full set of results for the dual connectivity ring, in the same format as for the tree topology, are shown in Figure 7.4. Looking at the mean job time, the absolute time taken without replication is longer than the tree topology, taking up to about 300,000 seconds rather than 200,000. This is because with the dual connection ring, a file being transferred to a relatively distant site must go through all the intermediate sites, following the same route as concurrent transfers to other sites. Congestion is therefore more likely and so if there is no replication there is a

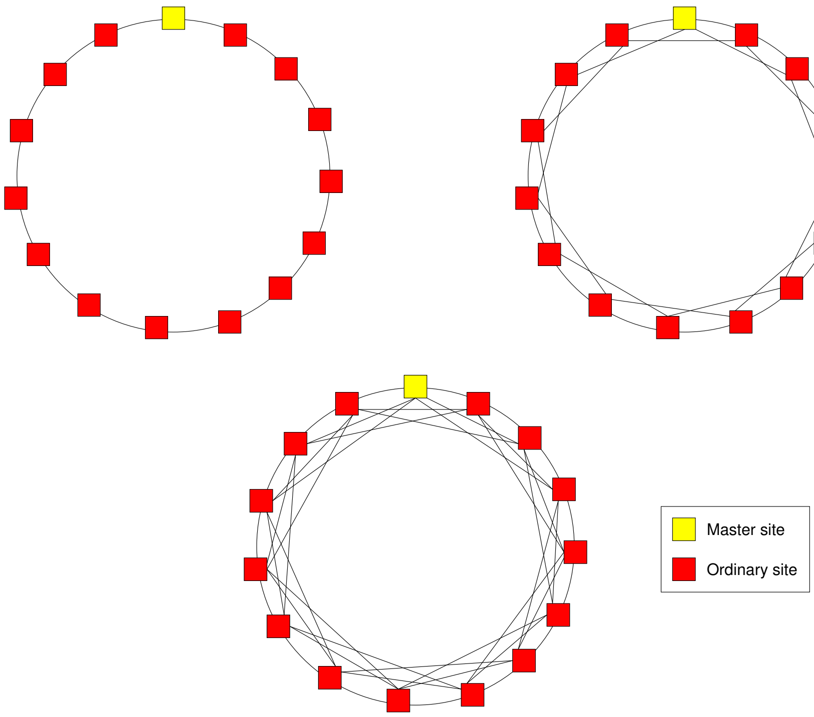

Figure 7.3: Ring topology with (clockwise from top left) dual, quadruple and octuple connectivity.

Metric	Value
D (all sites)	0.16
D (ordinary sites)	0.1
C (dual) (Gbps)	2
C (quad) (Gbps)	4
C (octuple) (Gbps)	8
P (kSI2000)	46.7

Table 7.2: Characterisation metrics for the ring configurations.

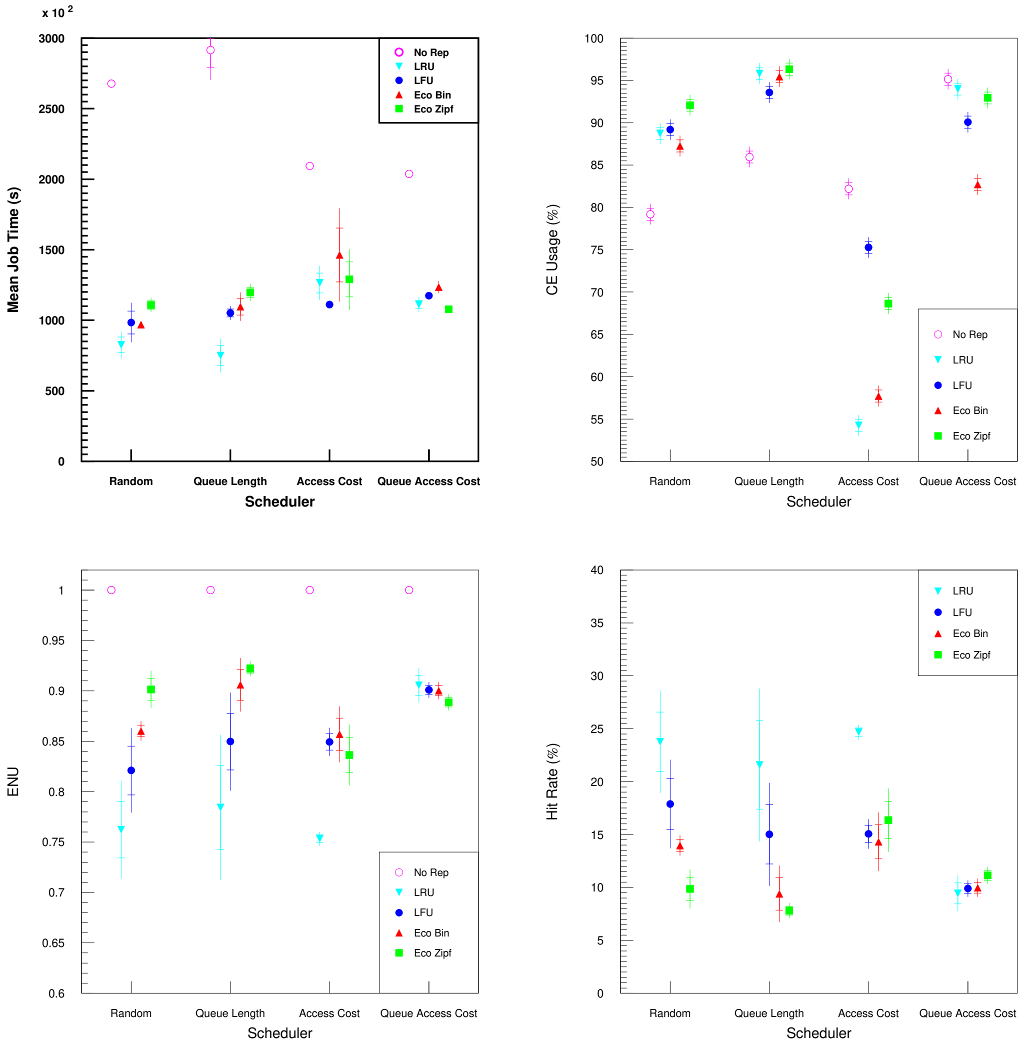

Figure 7.4: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different scheduling and replication algorithms, using the ring topology with dual connectivity.

repeatedly longer transfer time. The times with replication are very similar to the tree topology, with slightly longer times for the Access Cost scheduler and slightly lower times for Queue Access Cost.

Comparing the different schedulers, the effect on the mean job time without replication is quite pronounced, with the schedulers which account for data location (Access Cost and Queue Access Cost) up to 30% faster than the others. Again, this is due to the network structure, as the importance of scheduling jobs near to the data has increased. With replication, there is less difference between the schedulers.

Differences appear in the CE usage, with replication giving higher CE usage in this case. The difference is due not to the replication algorithms themselves giving higher CE usage, however, but to the non-replication case giving lower CE usage than in the tree topology. This is for the same reason as the increased job time; greater congestion means that some sites finish their jobs faster than others. This leads to some sites being very busy in the latter part of the simulation while others are idle. ENU and hit rate numbers are almost identical, with LRU again giving lower ENU and higher hit rate. The effect of this on the job time, though, is reduced. Overall, while the LRU strategy gives the best performance, it is less clear-cut than in the tree topology.

Rather than present the full result set for the quadruple and octuple connectivity rings, the variation of the metrics with the increasing connectivity is presented. The Queue Access Cost scheduler is selected for the comparison and the performance of the different replication algorithms examined; the pattern is similar for all the scheduling algorithms. The results are shown in Figure 7.5.

As might be expected, the benefits of replication are much greater at low connectivity. While the replication strategies do give very similar performance, the Zipf-based economic model is slightly faster than the LFU and LRU, while the binomial economic model is slightly slower, at the lowest connectivity. At high connectivity there is no difference between any of the algorithms in terms of job time, although all the replication algorithms are still about 30% faster than no replication. While there is little significant change in ENU and hit rate as the connectivity increases, there is a drop in CE usage, both with and without replication. This is because, as

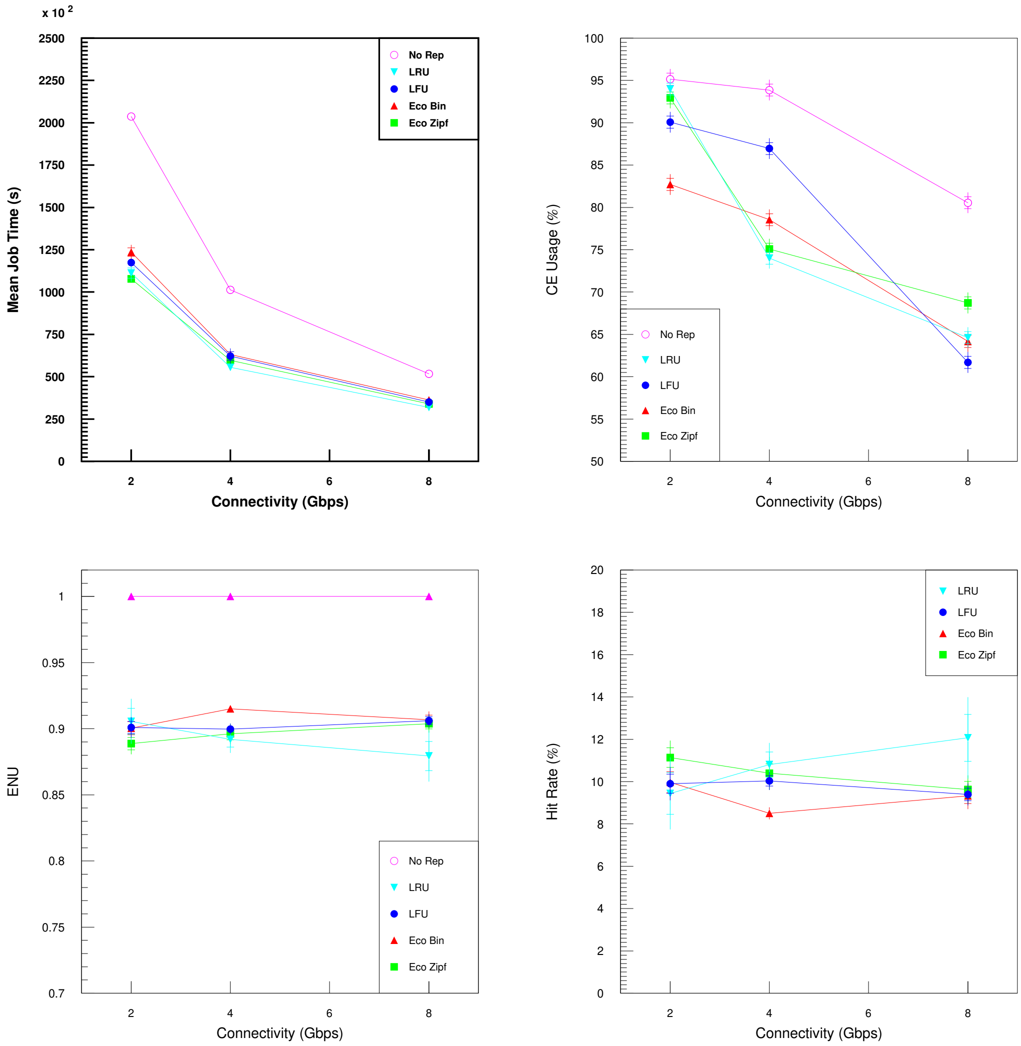

Figure 7.5: Variation in (a) mean job time, (b) CE usage, (c) ENU and (d) hit rate with increasing connectivity, for ring topology with Queue Access Cost scheduler.

the connectivity increases, there are more sites which have a direct connection to the master site. This means that the schedulers tend to send jobs to these sites, resulting in a more uneven distribution of jobs and hence lower CE usage.

7.5.3 Tree/Ring Hybrids

Having examined quite simple tree and ring topologies in the previous sections, this section combines the two topologies to give "hybrid" topologies, building up towards configurations which are more like real internet topologies. Figure 7.6 shows two such hybrids, denoted Hybrid 1 and Hybrid 2. Hybrid 1 is simply the tree topology

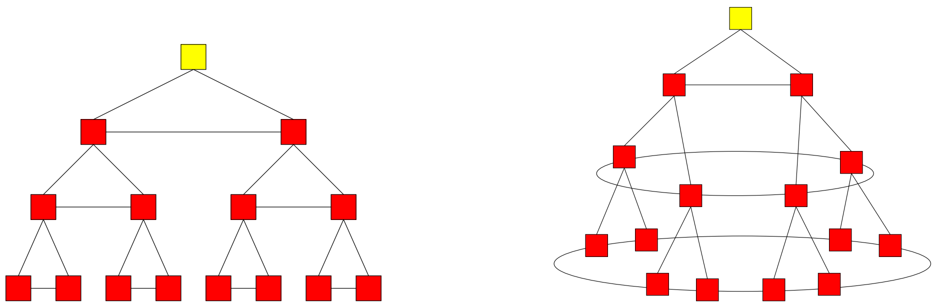

Figure 7.6: Hybrid topologies with 15 sites: Hybrid 1 (left) and Hybrid 2 (right).

with the addition of an extra connection between each immediate sibling. Hybrid 2 extends Hybrid 1 by adding links between all sites which are at the same distance from the master site. This means that each level of the hierarchical tree model now consists of a complete ring. The characterisation metrics for these hybrids are given in Table 7.3. The storage and compute metrics are the same as before, while the network metric C is now higher than both the tree and dual connectivity ring, but lower than than the quadruple and octuple connectivity rings.

Rather than present the full results set for each topology, Figure 7.7 shows the results of the different replication algorithms with the Queue Access Cost scheduler, with the tree and dual connectivity results for comparison. This shows first, that while the ring topology allows jobs to run a little faster than the tree topology with this scheduler, the hybrid topologies are faster than either. Naturally, given the higher connectivity, Hybrid 2 is faster than Hybrid 1. These results show, in fact,

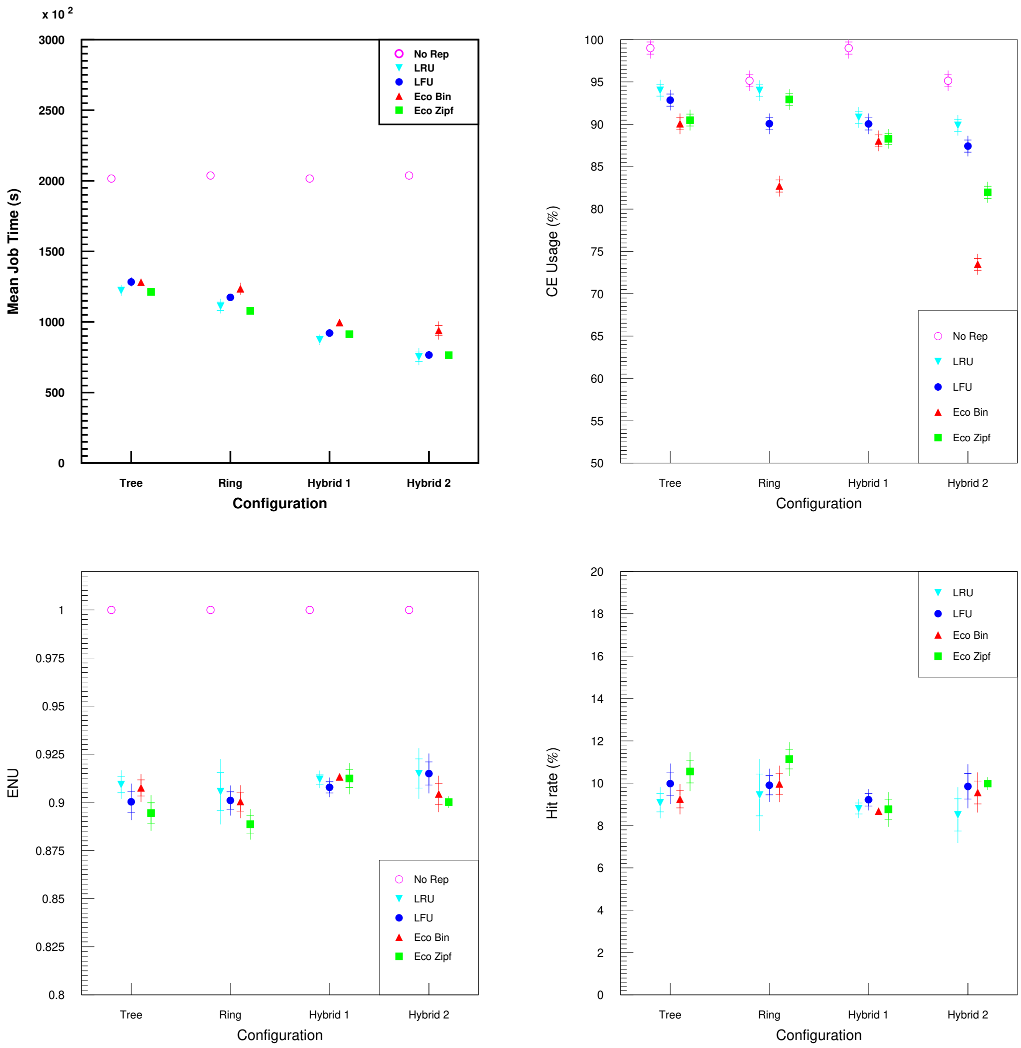

Figure 7.7: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, using the simple hybrid topologies and Queue Access Cost scheduler.

Metric	Value
D (all sites)	0.16
D (ordinary sites)	0.1
C (Hybrid 1) (Gbps)	2.8
C (Hybrid 2) (Gbps)	3.6
P (kSI2000)	46.7

Table 7.3: Characterisation metrics for the simple hybrid configurations.

that as the value of the network metric C increases, the job time falls, and that the fall is in agreement with that seen in the ring topology with increasing connectivity (Figure 7.5). This suggests that the underlying topology is less important than average connectivity.

The exception is when no replication takes place; in this case, topology has almost no effect with the Queue Access Cost scheduler, as Figure 7.7 shows, but increasing the connectivity of the ring had a large effect. It may be recalled that without replication, schedulers which do not account for data location were considerably slower with the ring topology. This implies that without replication, it is the connectivity of the master site which is most important.

7.6 More Complex Hybrids

The previous hybrid topologies presented were fairly simple, with only 15 sites running jobs. This section introduces a slightly more complex topology, as a step towards real grid systems. It may be recalled that the tiered LCG model which was introduced in Chapter 1 consisted of a single Tier-0 site (CERN), from which data were shipped to about 10 Tier-1 sites. These Tier-1s could then be accessed by various Tier-2 sites where most of the analysis is performed. This forms the basis for the topologies presented in this section, although it is still only a loose basis.

7.6.1 Hybrid 3

The first of the topologies based on the 3-tier model is shown in Figure 7.8. There is a single master "Tier-0" site, as there was in the previous topologies, ten "Tier-1" sites which have an SE but no CE, and each of which is connected to five "Tier-2" sites which have both SE and CE. Links between the Tier-0 and Tier-1s are 1 Gbps, while links from Tier-1s to Tier-2s are 155 Mbps. It can be seen that the overall topology is that of a tree with the master site at the apex, with the Tier-1s connected in a ring. SE and CE sizes are as before. The characterisation metrics are shown in Table 7.4; the storage metric D has the same value as the simple topologies for those sites which run jobs, and the compute metric is slightly lower. The value of C is lower, being less than half that for the tree topology.

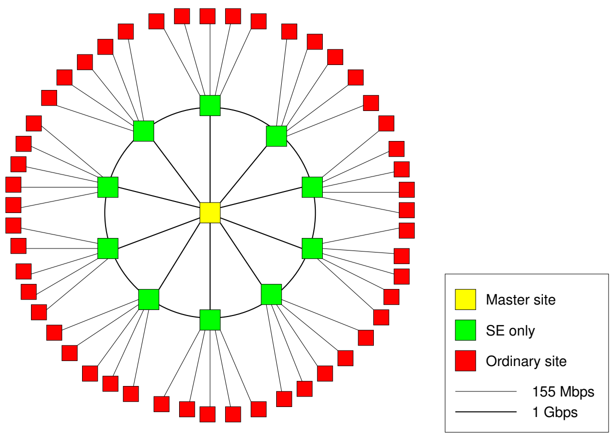

Figure 7.8: Topology of Hybrid 3.

Metric	Value
D (all sites)	0.12
D (Tier-2)	0.1
C (Gbps)	0.91
P (kSI2000)	41.0

Table 7.4: Characterisation metrics for the Hybrid 3 configuration.

Running this configuration with all the scheduling and replication algorithms gave the results shown in Figure 7.9. Looking first at the mean job times, the absolute

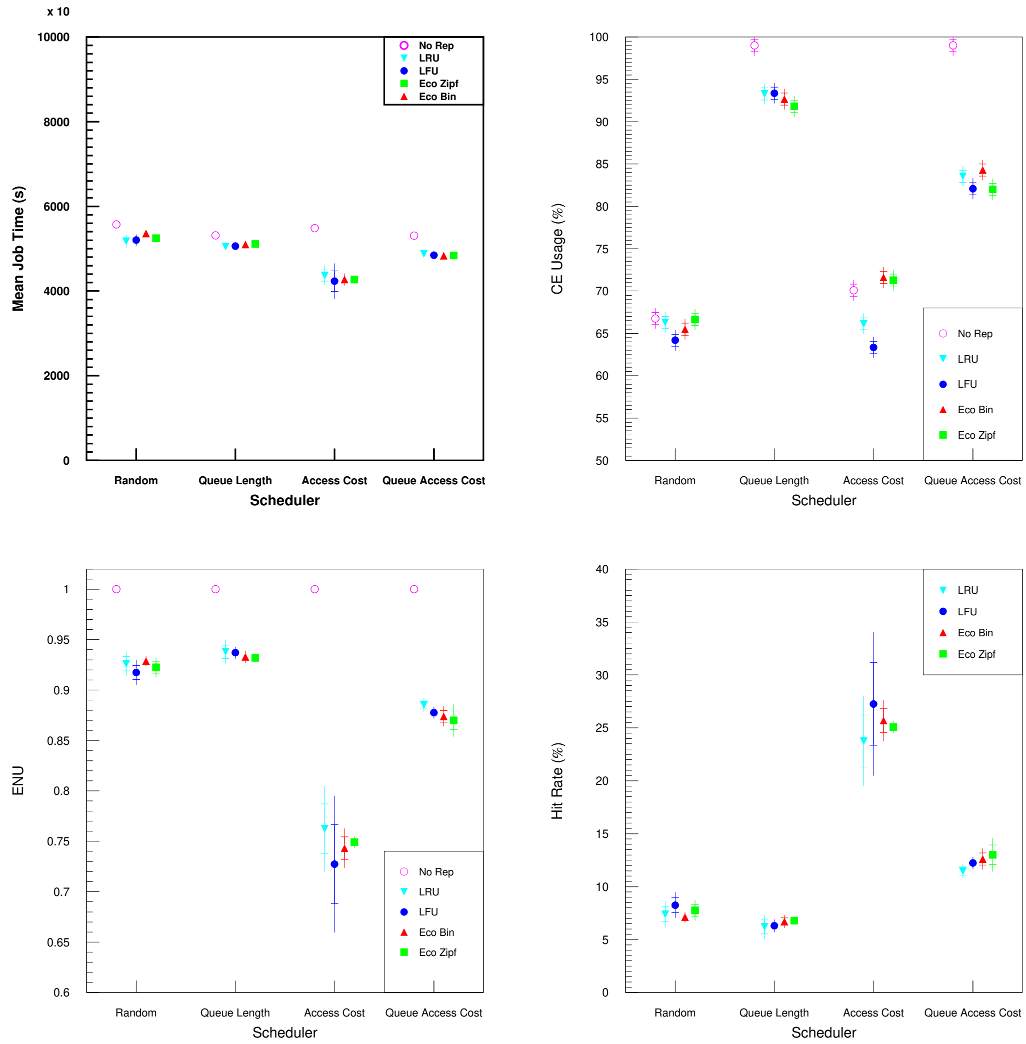

Figure 7.9: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different scheduling and replication algorithms, using the Hybrid 3 topology.

time without replication is increased by a factor of 2 when compared to the Hybrid 2 topology. The time taken with replication is increased much more, roughly by a factor of 5, although it is still lower than non-replication. This is due to the lower connectivity of the Tier-2 sites. The random scheduler is clearly the slowest here, showing that the "real" schedulers have the desired effect when there is a higher degree of complexity. The Access Cost scheduler is the fastest with all the replication strategies here. This is because as it tends to send jobs to sites as near the required data as possible, data "hot-spots" form at some sites, allowing jobs at these sites to run very quickly. This did not occur with the earlier topologies because of the higher bandwidth, which allowed jobs to be scheduled further from the data without loss of performance. The Queue Access Cost shows a similar, but weaker effect, as the formation of hot-spots is mitigated by the balancing of CE loads. There is, however, little discernible difference between any of the replication strategies.

There is also a clearer difference between the schedulers when the CE usage is examined. Those schedulers which account for the status of CEs when scheduling - Queue Length and Queue Access Cost - give 15 - 25% higher CE usage than the Random scheduler. Access Cost gives a low CE usage, because those sites which are not hot-spots receive fewer jobs, leading to uneven CE usage across the grid. The Queue Access Cost scheduler therefore gives a good trade-off between job time and CE usage. The Access Cost scheduler gives the lowest ENU, showing that in this case, sending jobs to the data makes the best use of network resources. The fact that the Access Cost scheduler also gives the highest hit rate simply reflects the fact that, where possible, it sends jobs to sites already containing the data. Of the four replication strategies, the economic models give a slightly higher hit rate than the LRU and LFU, with the Zipf version slightly higher than the binomial economic model.

7.6.2 Hybrid 4

The final hybrid investigated in this section is a variant of Hybrid 3, where instead of the Tier-0 being central to the topology, with direct connections to the Tier-1 sites and from there to the Tier-2 sites, a network backbone is introduced through which the sites are connected. This is shown in Figure 7.10. The Tier-1 sites here all have SEs of size 10 TB, which will hold all the files in the dataset, and are connected to the backbone by a 1 Gbps link. The Tier-2 sites have SEs of size 1 TB, which will hold enough files for 10 jobs, and are connected by 155 Mbps links. Initially, all the files were at a single master site. From the point of view of jobs running on Tier-2s

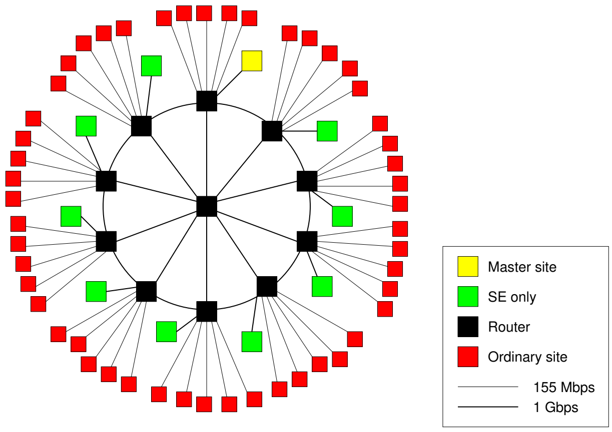

Figure 7.10: Topology of Hybrid 4.

in the LCG, however, they will be able to access a number of Tier-1 sites containing the data rather than a single site. The number of master sites was therefore varied from 1 to 10 and the behaviour of the algorithms examined. Table 7.5 gives the characterisation metrics of this topology. Again, values of D and P are similar to the previous configurations, but C is considerably lower.

Metric	Value
D (all sites)	0.25
D (Tier-2)	0.1
C (Gbps)	0.296
P (kSI2000)	41.7

Table 7.5: Characterisation metrics for the Hybrid 4 configuration.

Figure 7.11 gives the full result set for the case with only one master site. Overall, this looks very similar to the previous results sets. There is a large (more than sevenfold) increase in the time taken when replication is not used. This can be attributed directly to the fact that all traffic from the master site must now pass through a single network link. When replication is used, the mean job time is only about a factor of two higher than with Hybrid 3, showing the clear advantage of replication when such bottlenecks are present. As in previous configurations, however,

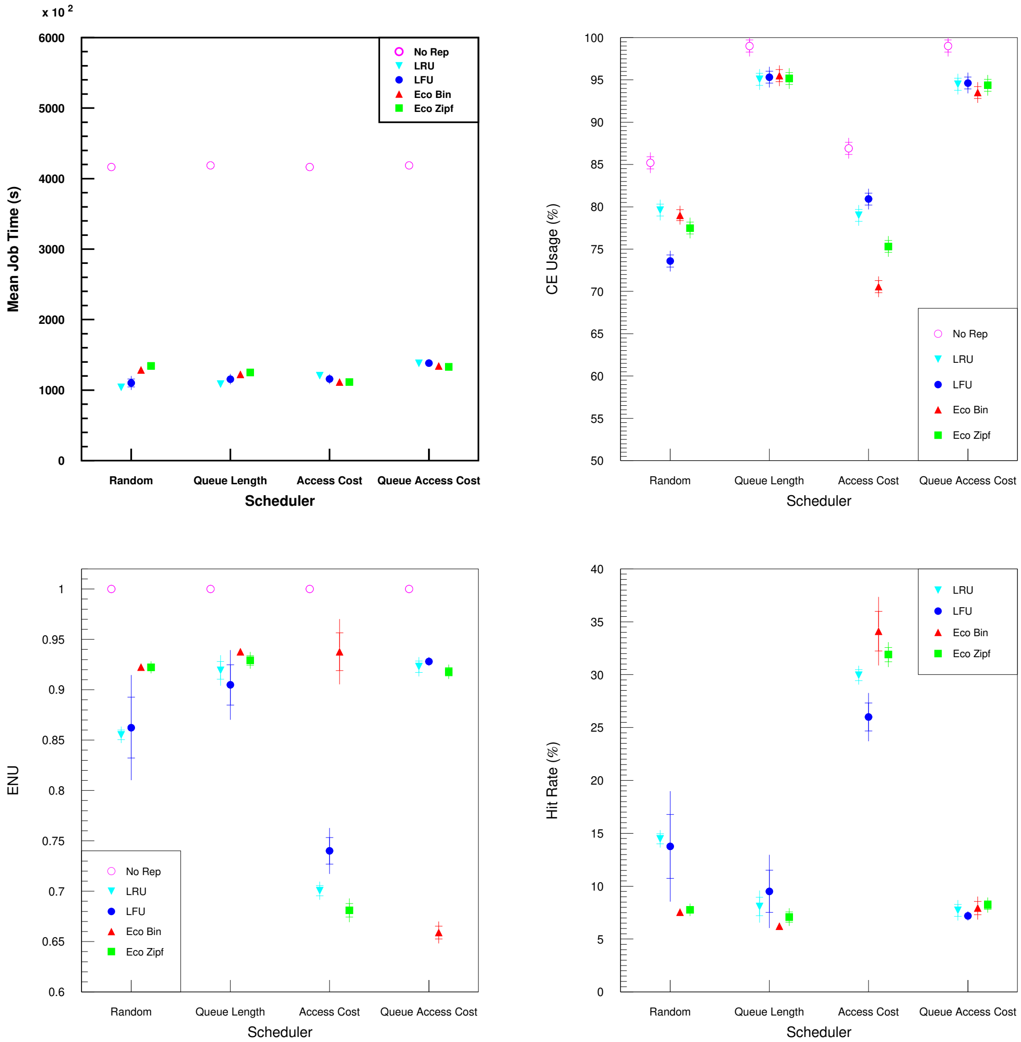

Figure 7.11: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different scheduling and replication algorithms, using the Hybrid 4 topology with a single master site.

it is unclear that any one replication strategy is faster than another, although the economic models are perhaps marginally faster here. The economic models also have a slightly higher hit rate than the LRU and LFU in this configuration. In terms of job time, there is little difference between the different schedulers, although Access Cost is perhaps slightly faster. The CE usage indicates that the Queue Length and Queue Access Cost algorithms make better use of compute resources.

Next, consideration is given to the variation of these metrics as the number of master sites - in other words, the number of replicas existing in the grid at the start of the simulation - is increased. The case where all the Tier-1 sites are filled with replicas at the start is the closest to the LCG topology and hence can be considered a prototype of the LCG configuration presented in the next chapter.

First, the variation in mean job time and CE usage for the different schedulers, using the LFU replication algorithm, is shown in Figure 7.12. This shows that, while

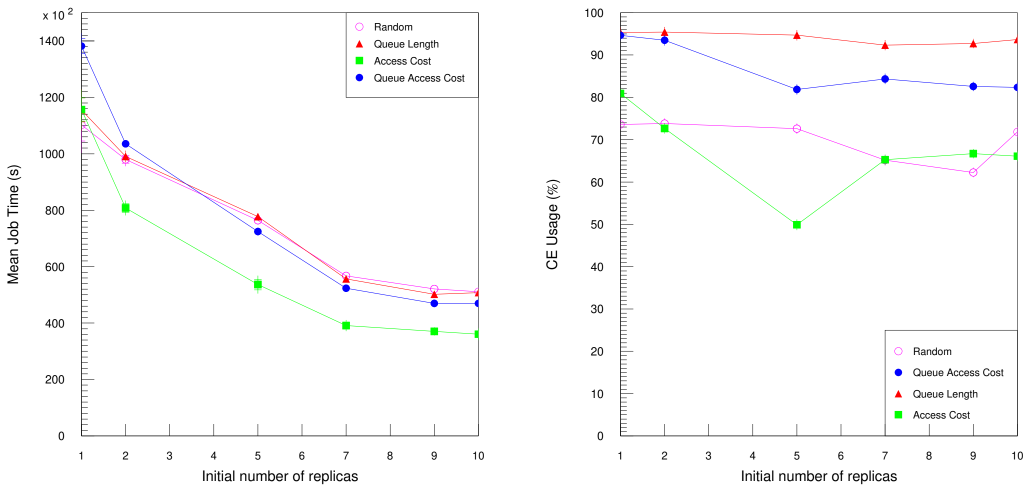

Figure 7.12: (a) Mean job time and (b) CE usage for different scheduling algorithms, with LFU replication, for Hybrid 4 topology with varying number of initial replicas.

the mean job time for all the schedulers decreases as the number of replicas increases, simply due to the greater availability of replicas, the Access Cost scheduler is consistently faster than the others. Both Access Cost and Queue Access Cost decrease faster than the Random and Queue Length schedulers, showing their effectiveness in sending jobs close to the data. The Queue Length scheduler gives the highest CE usage throughout, while the other algorithms give a slight decrease. Queue Access Cost therefore gives a good balance between job time and CE usage, especially when there are few replicas initially available.

Figure 7.13 shows the variation in mean job time for the different replication algorithms, using the Queue Access Cost scheduler. This shows, first, that the benefit

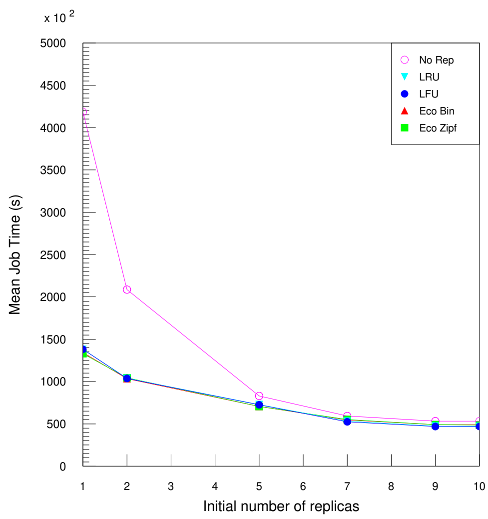

Figure 7.13: Mean job time for different replication algorithms, with the Queue Access Cost scheduler, for Hybrid 4 topology with varying number of initial replicas.

of dynamic replication decreases as the number of existing replicas increases; the rate of decrease in the mean job time is higher without replication than with replication. Secondly, it shows that the LFU algorithm is the fastest when there is a high number of existing replicas; the economic models become slightly faster when there is a low number of existing replicas.

The other evaluation metrics, shown in Figure 7.14, show relatively little variation as the number of replicas changes. There is a small reduction in the CE usage for all the replication algorithms. This may be due to the data location aspect of the scheduler becoming less effective as the number of data sources increases. There is also a slight reduction in the ENU for all the replication strategies, as the higher number of sources puts less strain on the network. The hit rate, on the other hand, increases,

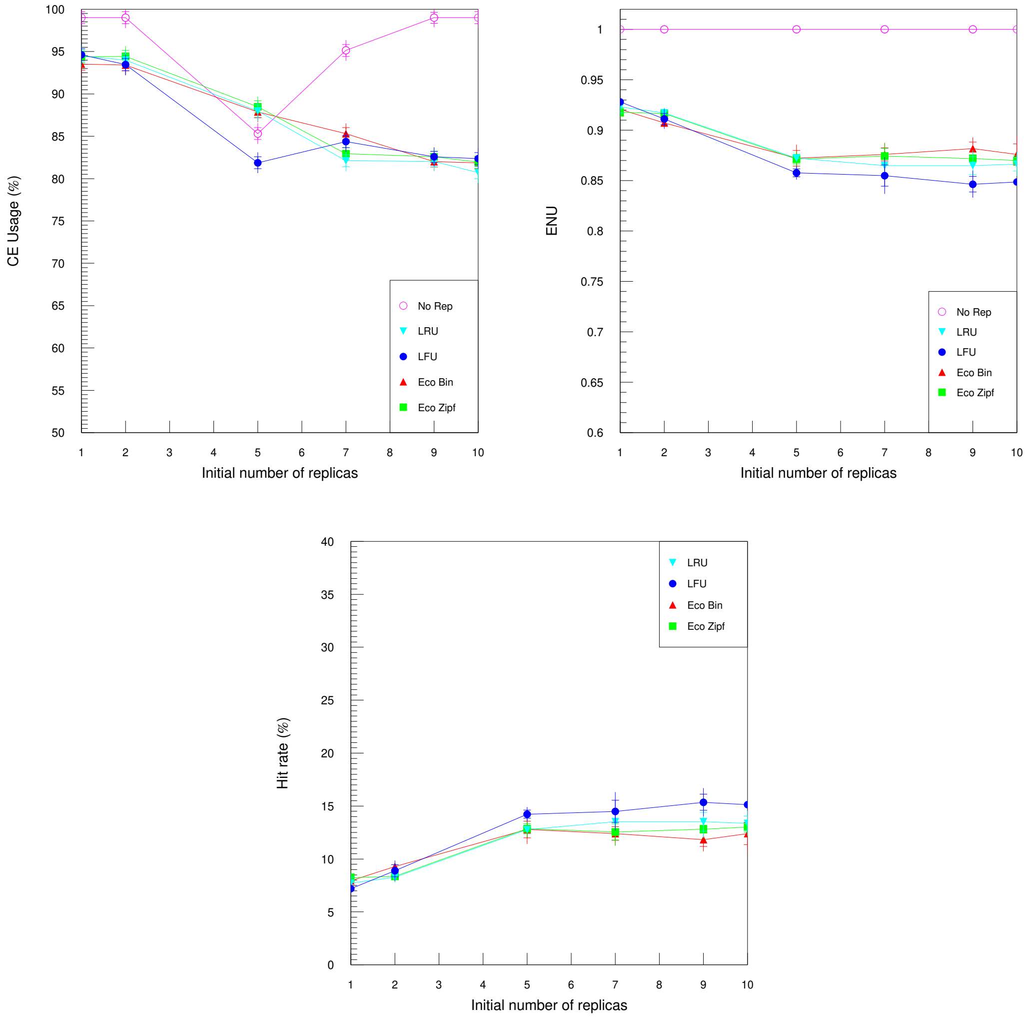

Figure 7.14: (a) CE usage, (b) ENU and (c) hit rate for different replication algorithms, with the Queue Access Cost scheduler, for Hybrid 4 topology with varying number of initial replicas.

with the LFU and LRU hit rates increasing more than those of the economic models. This indicates increasing effectiveness of these replication strategies at removing the least useful files.

7.6.3 Increasing Number of Jobs

As the Hybrid 4 configuration with 10 initial sets of replicas can be considered a proto-LCG configuration, it is useful to examine the effects of increasing numbers of jobs on the grid. This is an area of parameter space which is somewhat constrained for the full-scale LCG simulation, and so the results from this section may be used to indicate the likely results for the full simulation.

The number of jobs submitted was therefore varied from 500 to 50,000 and the performance of the replication algorithms measured, using the Queue Access Cost scheduler. The Eco Zipf strategy was omitted, however, because of its similarity to the Eco Bin results for this topology and the long simulation time when many jobs are simulated. The results are shown in Figure 7.15. Note that the data points for LFU, LRU and Eco Bin at 50,000 jobs are the result of a single simulation run. This shows, most importantly, that the mean job time increases linearly with the number of jobs. The rate of increase, however, is different for the different replication strategies. Fitting straight lines to the data, excluding those points for which only one measurement was available, gave the gradients shown in Table 7.6. The values of

Replication Strategy	Gradient (seconds/job)	Difference from No Rep. (%)
No Replication	50.46200 ± 0.00001	0
Eco Bin	43.52 ± 0.05	13.8 ± 0.1
LRU	36.70 ± 0.11	27.3 ± 0.2
LFU	30.69 ± 0.04	39.2 ± 0.1

Table 7.6: Gradients of best-fit lines for increasing number of jobs.

χ^2 per degree of freedom are large because of the relatively small errors on the points compared to the very large job times, and it was not possible to generate sufficient simulations to give a better estimate of these errors. Nevertheless, it is clear that

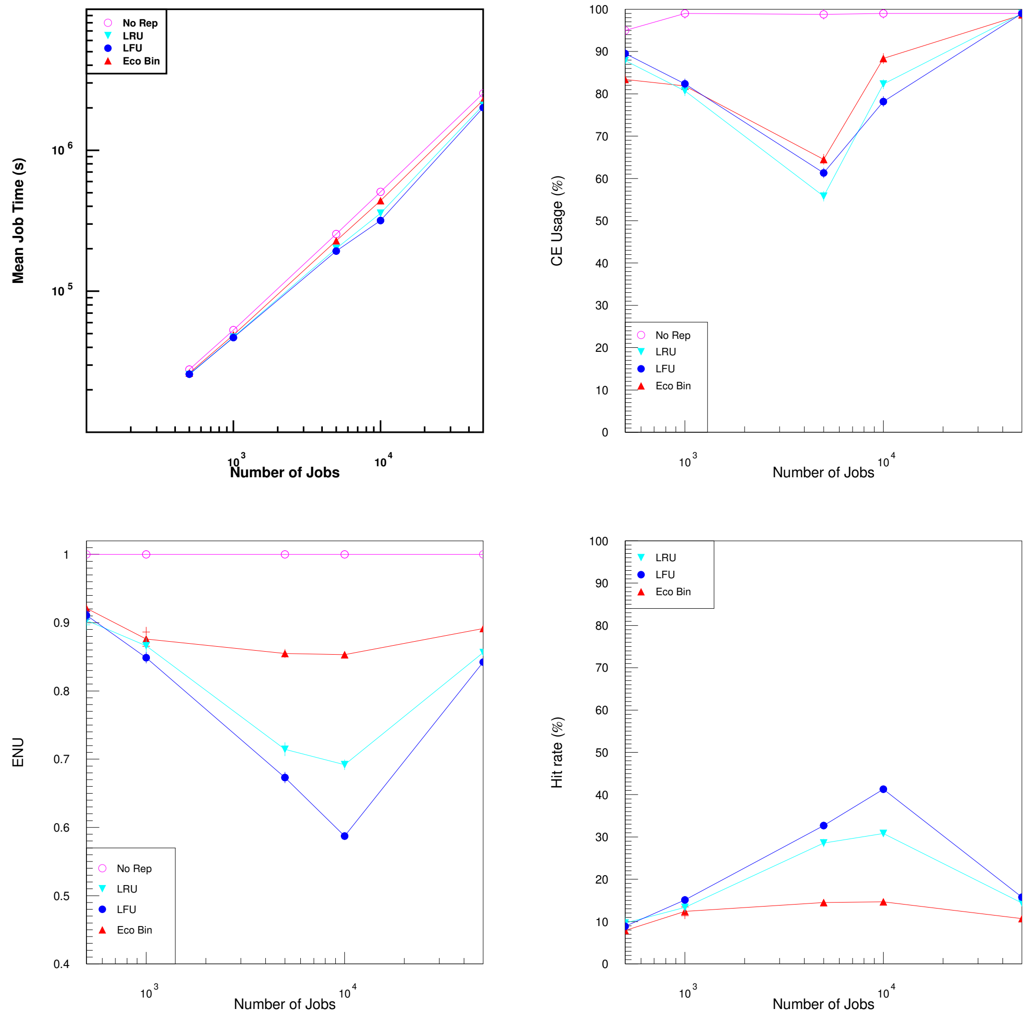

Figure 7.15: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, with increasing number of jobs on the prototype LCG topology.

with replication, the rate of change is lower, and so the effectiveness of replication increases as the number of jobs increases. The LFU is the fastest, followed by LRU and then Eco Bin, with differences between the algorithms being magnified at higher numbers of jobs.

The CE usage with replication drops off initially then rises again after the job queues at the sites have become saturated. The ENU falls rapidly for all the replication strategies, while the hit rate increases, up to about 10,000 jobs on the grid, after which the ENU rises and hit rate drops. This may be explained by the length of time which jobs now spend queueing affecting the performance of the scheduling algorithm. If the mean job time is $\mathcal{O}(10^6)$ seconds, for example, the running time is only $\mathcal{O}(10^3)$ seconds. As jobs are allocated according to data location at the time of scheduling, by the time the jobs come to run the data available locally may have changed, thus giving the lower hit rate and increased ENU. With a high number of jobs, then, dynamic rescheduling may have to be implemented to account for changes in the grid's state. Dynamic rescheduling is in fact used in many real resource management systems such as AppLeS. This is beyond the scope of the current version of OptorSim and is left for future work. Among the results presented in this and the next chapter, the disproportionately large ratio of total job time to running time is restricted to this particular scenario, however. Until this point is reached, the ENU and hit rate behaviour show the increasing effectiveness of the replication algorithms as time passes.

Figure 7.16, which shows the mean time that jobs are actually running (i.e. discounting the queueing time), corroborates this: the running time drops as the number of jobs increases, particularly with the LFU strategy, then rises again. There is a large intrinsic variance in these results because any individual job's running time depends on factors unique to that job, such as how many files it needs and where these files are. It should be noted that the processing time in this simulation is constant, as each job has the same number of files, and at 10 seconds per job is much lower than the data access time. Figure 7.16 can therefore also be viewed as the mean data access time per job. This seems disproportionately large in comparison to the processing time, but is due to the fact that few files are available locally in this scenario.

Although in reality this ratio is unlikely to be so large, analysis jobs are likely to require access to files which are non-local and thus be limited by the file transfer times. The scenario for computationally intensive jobs such as event reconstruction would be quite different.

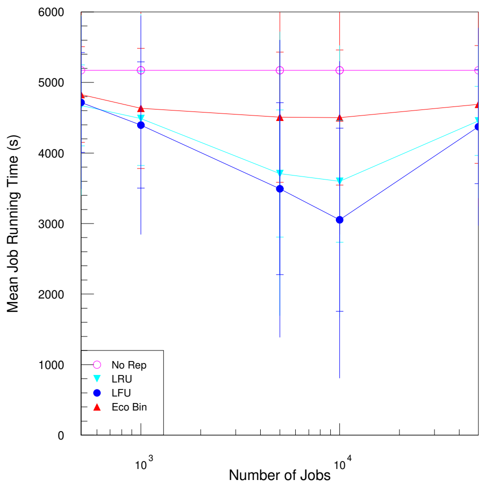

Figure 7.16: Mean job running time for different replication algorithms, with increasing number of jobs on the prototype LCG topology.

This degradation of performance for individual jobs does not greatly affect the overall performance so overall, these results indicate that replication, and a good choice of replication algorithm, become increasingly important for heavily loaded grids.

7.7 Summary

The investigation into various network topologies which has been performed in this section has yielded a number of insights into the behaviour of the different scheduling and replication algorithms.

First, it has shown that replication is important for reducing job times and improving resource usage. This is especially true where access to the data source is limited by bandwidth. Differences between the replication algorithms were less clear-cut. In general, the LFU and in particular the LRU performed slightly better than the economic models. The more highly connected the grid, the less effective replication became in comparison with no replication. It was found that while the underlying topology had a subtle effect on the way the scheduling and replication strategies performed, topology is much less important than connectivity.

With the more complex hybrids, it was found that with high connectivity to the master site (as in Hybrid 3), the effect of replication was less strong than when the master site was poorly-connected (Hybrid 4). In general, while the Queue Access Cost scheduler gave a good compromise between job time and CE usage, differences between the replication algorithms were less well-defined. Performances were very similar, with the LFU performing slightly better than the others, especially when there were a number of master replicas already present. Data location and availability is thus of greatest importance, especially when the connectivity is low.

The performance as a function of the number of jobs was also investigated for the most LCG-like of the hybrid topologies, showing that a small difference in performance for a small number of jobs led to a much larger difference in performance for a large number of jobs. A 10% saving in job time with 1000 jobs, for example, went up to a 20-40% saving with 10000 jobs. The LFU strategy in particular gave better performance than any other algorithm for a high number of jobs on the grid. Simple replication strategies are therefore more performant for complex grid scenarios and should be adopted while economic models undergo further investigation.

Chapter 8

Simulation Results: LCG 2008

This chapter presents a series of simulation studies based on the LCG resources predicted for 2008, the first full year of LHC data-taking. First, an outline of the resources and computing model assumed is given, explaining the simplifications which have been made within the simulation. While complete accuracy in every detail is impossible to achieve, the aim is to have a simulation which yields useful information about grid behaviour. A series of tests with their results are presented, examining scheduling algorithms, access patterns, job submission patterns, network traffic, and the characterisation metrics described in the previous chapter, before drawing some conclusions about the efficacy of the replication strategies in different situations.

8.1 Simulation Setup

8.1.1 Analysis Jobs and Files

The experiment computing model documents [17] [18] [19] [20] describe, for each experiment, their planned analysis model and the roles played by the different tiers within that model. The differences in the way the tiers are used by the four experiments have already been outlined in Chapter 1. Regarding the analysis jobs themselves, there are generally two types of analysis activity. Scheduled analysis is the process of analysing the ESD to extract new TAG or AOD samples, done in a centrally coordinated fashion by experiment working groups. The development of such analysis jobs would involve running over small sub-samples in a chaotic fashion by individual physicists at Tier-2 sites, before being approved for running over the whole dataset. Chaotic analysis is that done by individuals, with the AOD (or in the case of LHCb, the DST) as the primary source. A typical scenario would involve a physicist running a query or algorithm over a dataset, generating some output data which could then be used for further analysis. All the experiments except LHCb plan to do most of the analysis at Tier-2s, with some capacity at Tier-1s and also at the CERN Analysis Facility (CAF), which will effectively be a large Tier-2. LHCb plans to use Tier-2s mainly for Monte Carlo generation, using the Tier-1s and the CAF for analysis. Different types of analysis activity may be represented in OptorSim by different file access patterns, and so the experiments which follow include a comparison of sequential and Zipf-like file access.

The above outline of the experiment analysis models has been modelled for simulation in OptorSim as follows. Each experiment is assigned a dataset, corresponding to the full AOD or DST, which is placed at each Tier-1 site and at CERN at the beginning of the simulation. Six job types were defined: `atlas`, `alice-pp`, `alice-hi`, `cms`, `lhcb-big` and `lhcb-small`. The ALICE and LHCb jobs are divided into these two types as ALICE will have large event sizes during the heavy-ion runs, which means the heavy-ion analysis jobs will be correspondingly larger, while LHCb estimate that about 20% of their jobs will run over large event samples. Assuming that a "typical" analysis job runs over 10^6 events (with the `lhcb-big` jobs running over 10^7 events), dividing each dataset into 2 GB files, and taking the AOD event sizes for each experiment as they were presented in Table 1.1, this gives the parameters for each job type as presented in Table 8.1. The total size of the dataset for each job type is the approximate size of a single copy of the AOD/DST for a year's worth of data taking. The simulated jobs process their given subset of files from the dataset, selected from a random point within that dataset. When a job is running on a site, it retrieves the files it requires (according to the chosen access pattern) and processes them according to the computing resources available at that site.

The time taken to process a file is calculated for each job according to the expected processing time per event during analysis, in kSI2000-seconds, and the processing

Job	AOD event size (kB)	Dataset size (TB)	Total number of files	Number of files per job
`alice-pp`	50	50	25000	25
`alice-hi`	250	25	12500	125
`atlas`	100	200	100000	50
`cms`	50	75	37500	25
`lhcb-small`	75	75	37500	38
`lhcb-big`	75	75	37500	375

Table 8.1: Job configuration parameters used in the LCG 2008 configuration.

Job type	Time per event (kSI2000-sec)	Time per file (MSI2000-sec)
`alice-pp`	3	120
`alice-hi`	350	2800
`atlas`	0.5	10
`cms`	0.25	10
`lhcb-small` `lhcb-big`	0.3	8

Table 8.2: Processing times for the defined job types.

times for the four experiments are shown in Table 8.2. These were used to calculate the linear factors for Equation 5.1; as for the simple topologies, the latency factor was left at zero. The probability of a particular job being run on the grid is modelled by the relative number of expected users for the different experiments. Estimates of the numbers of active users for each experiment are given in the computing models and are shown in Table 8.3. The approximate number of ALICE users was not published and hence was estimated at about 500 users. The split of probabilities between `alice-pp` and `alice-hi` jobs is 87.5% to 12.5%, while `lhcb-small` and `lhcb-big` are given 80% and 20% respectively of the LHCb share.

Experiment	Number of users
ALICE	500
ATLAS	600
CMS	1000
LHCb	140

Table 8.3: Estimated number of active users for the LHC experiments.

To understand these parameters in context, consider the example of an ATLAS job in this model. The probability of any particular simulated job being from ATLAS is about 27%. When running at a site, this job requires 50 files, or 0.05% of its total AOD dataset, corresponding to 10^6 events. Running at a standard Tier-2 site (see below), each file would take $\sim$15 seconds to process and so the job would take 775 seconds of processing time, plus the much larger time required to fetch the files.

8.1.2 Site Resources

The basic compute and storage requirements for LCG in the first few years of LHC data-taking were presented in Chapter 1, being drawn from the LCG Technical Design Report [21]. The requirements for 2008 are summarised in Table 8.4. The Tier-1 and

	Disk (PB)	Tape (PB)	CPU (MSI2000)
CERN Total	6.6	18	25.3
CERN Tier-0	1.3	13.6	17.5
CERN Tier-1/2	5.3	4.4	7.8
All external Tier-1s	31.2	34.7	55.9
All Tier-2s	18.8	-	61.3
Total	57	53	143

Table 8.4: LCG required resources for 2008.

Tier-2 sites participating in LCG at that time were taken from the LCG Memorandum of Understanding [23]. As this is still in negotiation with some sites, the list of Tier-2 sites in particular is not confirmed at the time of writing. For the experiments below, the September 2005 version was used. Using this list of sites, the analysis model (described above) was used to allocate appropriate resources to each site as follows.

Storage Resources

The Tier-0 (CERN) and Tier-1 sites were designated as the "master sites" for the simulation, as each will hold a copy of the current AOD sample for each experiment, which is the primary data source for end-user analysis. These sites were given SEs according to their planned capacities, as presented in Table 8.5. As OptorSim does not differentiate between types of storage, the tape and disk capacities were summed to give a total capacity for each site. Table 8.5 also shows the experiments preferentially served by each site.

Site	Storage capacity (PB)	Experiments served
CERN Tier-0	12.5	All
CERN Analysis Facility	6.4	All
Canada, TRIUMF	1.5	ATLAS
France, IN2P3	7.7	All
Germany, GridKa	4.0	All
Italy, CNAF	7.5	All
Netherlands, NIKHEF/SARA	5.2	ALICE, ATLAS, LHCb
Nordic	2.8	ALICE, ATLAS, CMS
Spain, PIC	3.5	ATLAS, CMS, LHCb
Taiwan, ASCC	2.5	ATLAS, CMS
UK, RAL	3.6	All
USA, BNL	5.1	ATLAS
USA, FNAL	5.2	CMS

Table 8.5: LCG Tier-0 and Tier-1 storage resources for 2008.

Detailed resource estimates are not available for all the Tier-2s, and so each Tier-2 site was given a canonical value, simply averaging the total Tier-2 resource requirements over the total number of Tier-2 sites. This gave an average SE size of 197 TB. The number of sites in the LCG simulation, however, put some limitations on the number of jobs which could be simulated, due to the physical limitation of available memory when running the simulation. This meant that the simulations were restricted to the order of 1000 jobs. To reach a state in the simulation where the SEs are full and file deletion is occurring would require about 200,000 jobs if the SEs were kept at 197 TB, and so the Tier-2 SE sizes were scaled down to 500 GB. These then hold 250 files, allowing file replacement to start when at most 10 jobs have been submitted to a site. This has the disadvantage that the storage metric D is then very small, so the file prediction algorithms will not perform to their best advantage. The effect of changing D by changing the size of the dataset, however, is among the tests presented.

Computing Resources

In the analysis model, most jobs run at Tier-2 sites, downloading the required files from Tier-1 sites if they are not available locally. The Tier-1 sites in the simulation are therefore not given CEs, except those which run LHCb jobs and which are given a CE equal to those at the Tier-2s. In reality, of course, the Tier-1s have large computing resources, but as the focus here is on analysis, they are assumed to be reserved for reconstruction and thus unavailable for analysis. The CERN Analysis Facility (CAF) is a special case, and was allocated a CE of 7840 kSI2000 as well as its SE. The Tier-2s were given an averaged CE compute power of 645 kSI2000.

Site policies were also applied according to the Memorandum of Understanding. While some sites will accept any job type, many sites plan to restrict usage to a subset of the experiments, according to which collaborations the physicists at that site are involved in. This was reflected in which jobs were accepted by the sites in the simulation; the Canadian sites would only accept ATLAS jobs, for example, while the UK sites would accept any job type.

8.1.3 Network Topology

The network topology between the sites was developed using the published topologies of the main research networks, such as GÉANT [125] in Europe, Abilene [126] in the USA and JANET [127] in the UK. These were used, with some simplifications, for the network backbone. Sites were connected to their closest router node, with the published bandwidths used if these were available and a default of 155 Mbps used if they were not. As the simulation was geared towards the user analysis view of the grid, where resources are available via the standard research networks rather than the dedicated paths which will be available for initially transporting data from CERN to Tier-1 sites, this is not inappropriate. Sites with both a Tier-1 and a Tier-2 facility, such as RAL, had the Tier-2 attached directly to the Tier-1 by a 1 Gbps link. The resulting topology is shown in Figure 8.1.

The characterisation metrics for the LCG 2008 configuration are shown in Table 8.6. Note the large difference between D for the grid as a whole and for the Tier-2 sites;

Metric	Value
D (all sites)	1.38
D (Tier-2)	0.0012
C	0.42 Gbps
P	710 kSI2000

Table 8.6: Characterisation metrics for the LCG 2008 configuration.

the value of C is also lower than might be expected, given the high-speed backbone, because C accounts only for the site connectivities. The value of the compute metric P is substantially higher than was used in the simple topologies in the previous chapter. Together, the analysis model, site resources and network topology described in this section give as realistic a simulation model as possible with OptorSim.

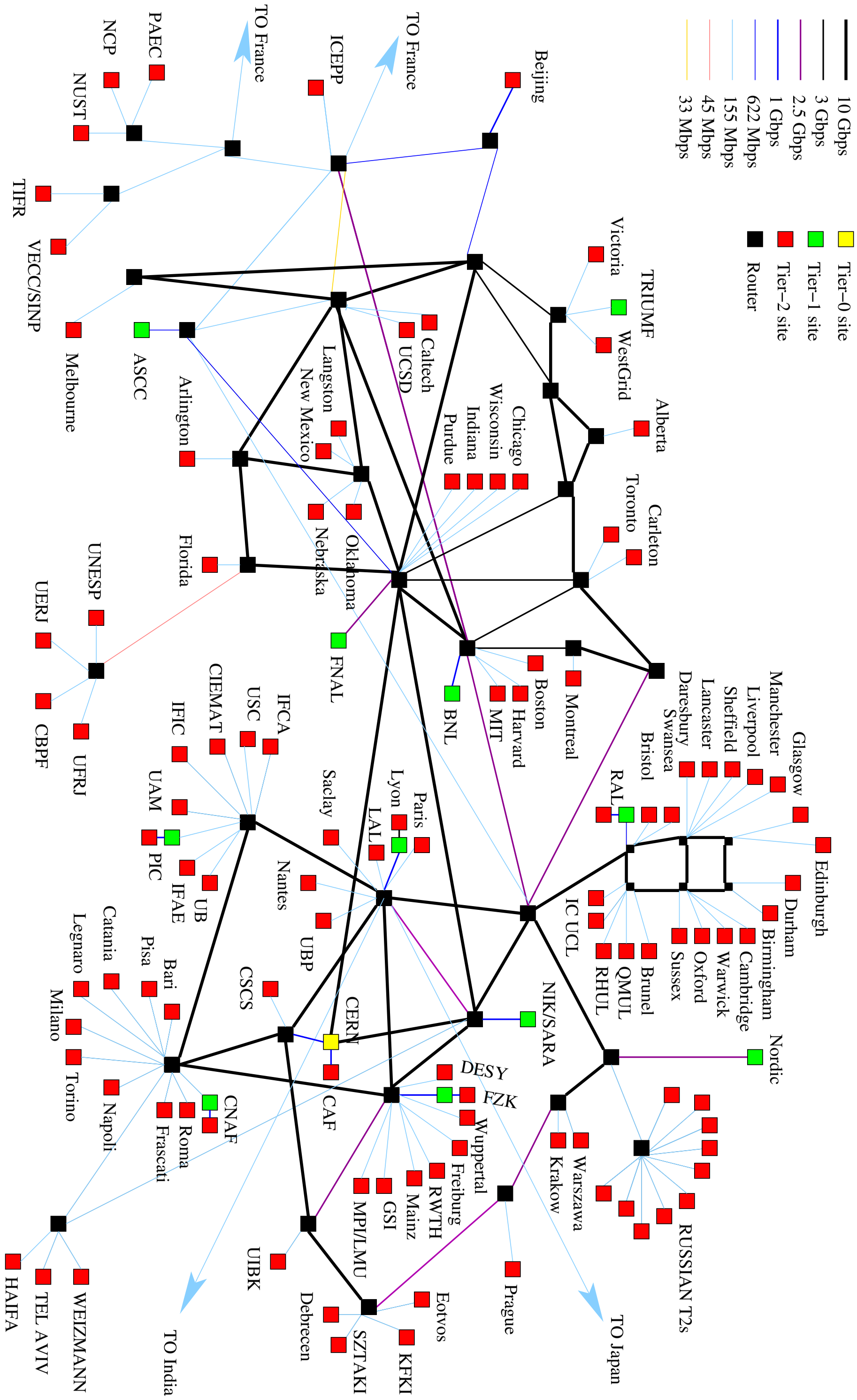

Figure 8.1: Simulated topology of the LCG 2008 grid. CERN, as the Tier-0 site, is shown in yellow, while Tier-1 sites are green, Tier-2s are red and router nodes are black. Network links have the values shown in the key.

8.1.4 Input Parameters

For most of the tests, 1000 jobs were submitted to the grid using the Random users. Although in a localised grid, user activity patterns are likely to follow the daily variation modelled by the DC04 users class, in a worldwide grid such as LCG this variation will be less apparent, due to the different time zones of participating sites, and so the Random class was chosen. Files were initially distributed to CERN and all the Tier-1 sites, and each job accessed its required files sequentially. The access history length, dt, was set at 10^9 ms, which is sufficiently large to allow the replication algorithms to perform well.

For the background network traffic, a representative variation (shown in Figure 8.2) was applied. This was based on measurements of available bandwidth, in this case between Lyon and CERN, as described in Chapter 5, and assuming that these typical patterns of network usage will not change significantly over the next few years. Using this profile, then, a link would have between 50% and 80% of its bandwidth available for grid traffic at any given local time.

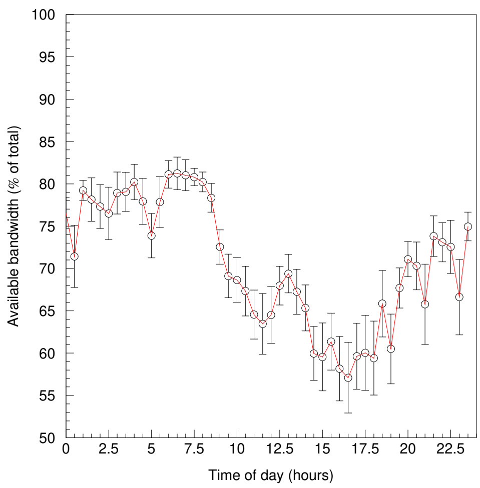

Figure 8.2: Profile of bandwidth variation used in LCG 2008 simulation. The time of day on the bottom axis is the local time at the receiving site.

8.2 Scheduling Algorithms

The first set of tests investigates the comparative performance of the four scheduling algorithms. The topology, jobs, files and parameters described in the previous section were used. The simulation was run with each of the scheduling algorithms, measuring the mean job time, CE usage, ENU and hit rate for each of the replication algorithms, and the results are shown in Figure 8.3. While there are some similarities between these results and those for the earlier, simpler grids, there are significant differences.

First, the choice of scheduling algorithm has a much greater impact here. It is clearly seen that the Random scheduler is the slowest, with the highest ENU and lowest hit rate. Of the remaining algorithms, both Queue Length and Queue Access Cost are faster than Access Cost, unlike the simple topologies where Access Cost tended to be fastest. This indicates that here, balancing the load between the sites is more important than locating jobs close to the data. The fact that Queue Access Cost is about 45% faster than Queue Length, however, shows that combining the queue information with the data location information gives the best performance. Queue Access Cost also gives significantly higher levels of CE usage than the other schedulers, at about 45% compared to under 20% for Queue Length. Both Access Cost and Queue Access Cost gave similar levels of ENU and hit rate. For all the following experiments, Queue Access Cost was therefore used as the scheduling algorithm.

The other significant difference between these results and those for the simpler topologies is the comparative ineffectiveness of the replication algorithms. Although with the Random scheduling algorithm, replication gives a lower mean job time than not replicating, with the other schedulers there is no discernible difference. This is true for all the evaluation metrics, and (as explained previously) is due to the very small value of D for the Tier-2 sites. The size of the whole dataset is so large that each new job which runs at a site is likely to request files which have not been requested before and which are therefore not in the access history for that site. There are then nearly as many file transfers occurring as when there is no replication, with fresh replications for every job. To address this, the performance of the replication strategies with varying D is studied in Section 8.4.

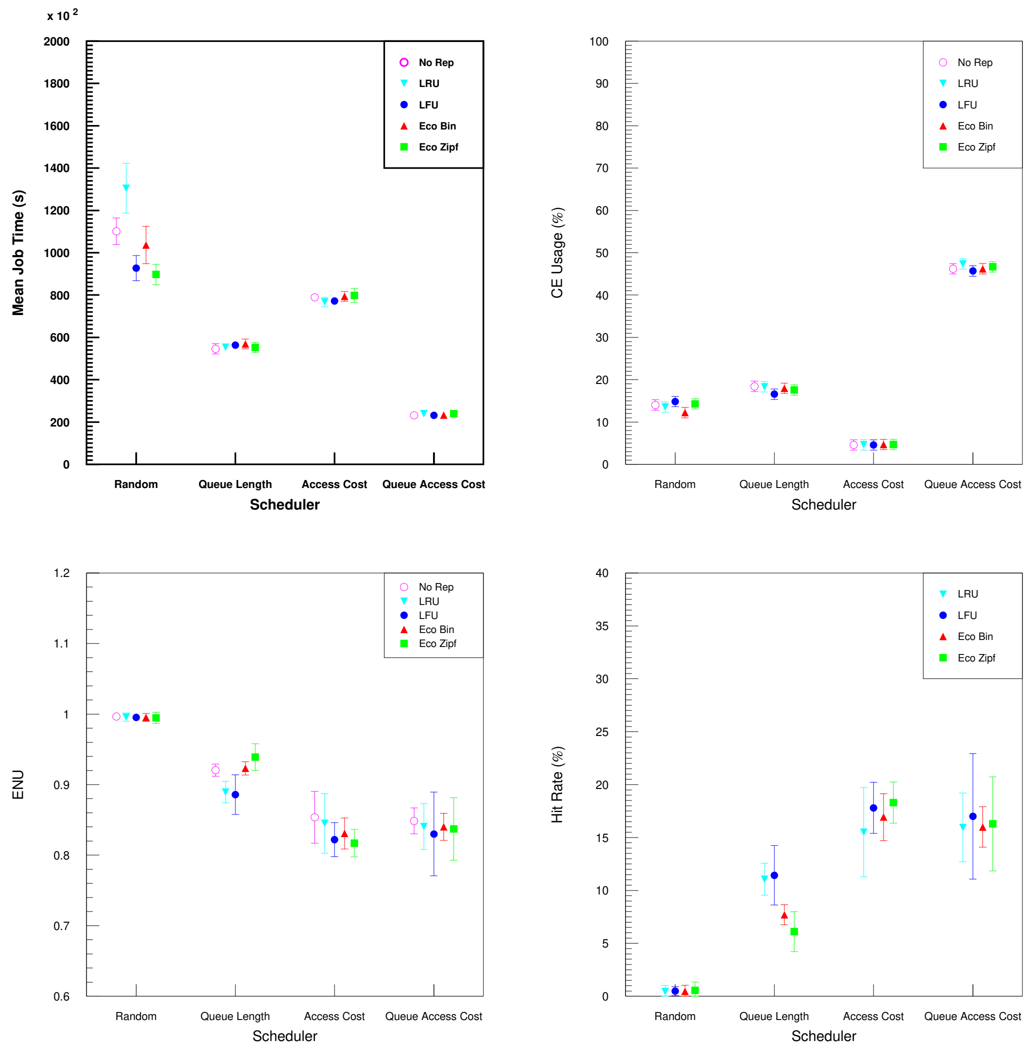

Figure 8.3: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different scheduling and replication algorithms, using the LCG topology.

8.3 Varying Number of Jobs

A real HEP data grid of the scale of LCG is likely to have $\mathcal{O}(10000)$ jobs running simultaneously. It is therefore important to understand how the replication algorithms perform with increasing numbers of jobs on the grid. Using the Queue Access Cost scheduler, with the other parameters as described in Section 8.1.4, the number of jobs submitted was varied from 500 to 4000 and the performance of the different replication algorithms measured. Higher numbers of jobs were not possible with the hardware available. The results are shown in Figure 8.4.

This shows a linear increase in the mean job time as the number of jobs on the grid increases, in agreement with the result from the prototype LCG topology in the previous chapter. This is because, as more jobs are submitted, the queues at the sites increase. If the job submission rate is higher than the grid's job processing rate, this build-up of queues is inevitable, and it is likely that this would also occur in a real grid. It is not possible to state, with these results, whether any replication strategy is better than any other, or indeed whether replication is effective at all. Beyond the range measured here, however, the linearity of the increase both here and in the prototype LCG topology makes it likely that with higher numbers of jobs this ambiguity would be resolved, and also that any differences in performance in the following sections, which all use 1000 jobs, would be magnified.

If the differences in gradient measured for the rate of change of job time with number of jobs in the prototype LCG topology are tentatively applied to this topology, one would expect the economic models to be approximately 14%, the LRU about 30% and the LFU about 40% faster than no replication. While this is not clear from Figure 8.4, looking ahead to the measurements with a more realistic value of the storage metric D in the next section shows that, as $D \to 0.1$, the economic models are about 18% faster and the LRU and LFU about 30% faster than no replication. This is in rough agreement with the values from the prototype topology. Extrapolating the values from Figure 8.5 from 1000 to 10000 jobs using the differences in gradient from Table 7.6 gives the values shown in Table 8.7.

It can thus be stated with a reasonable degree of confidence that the measurements

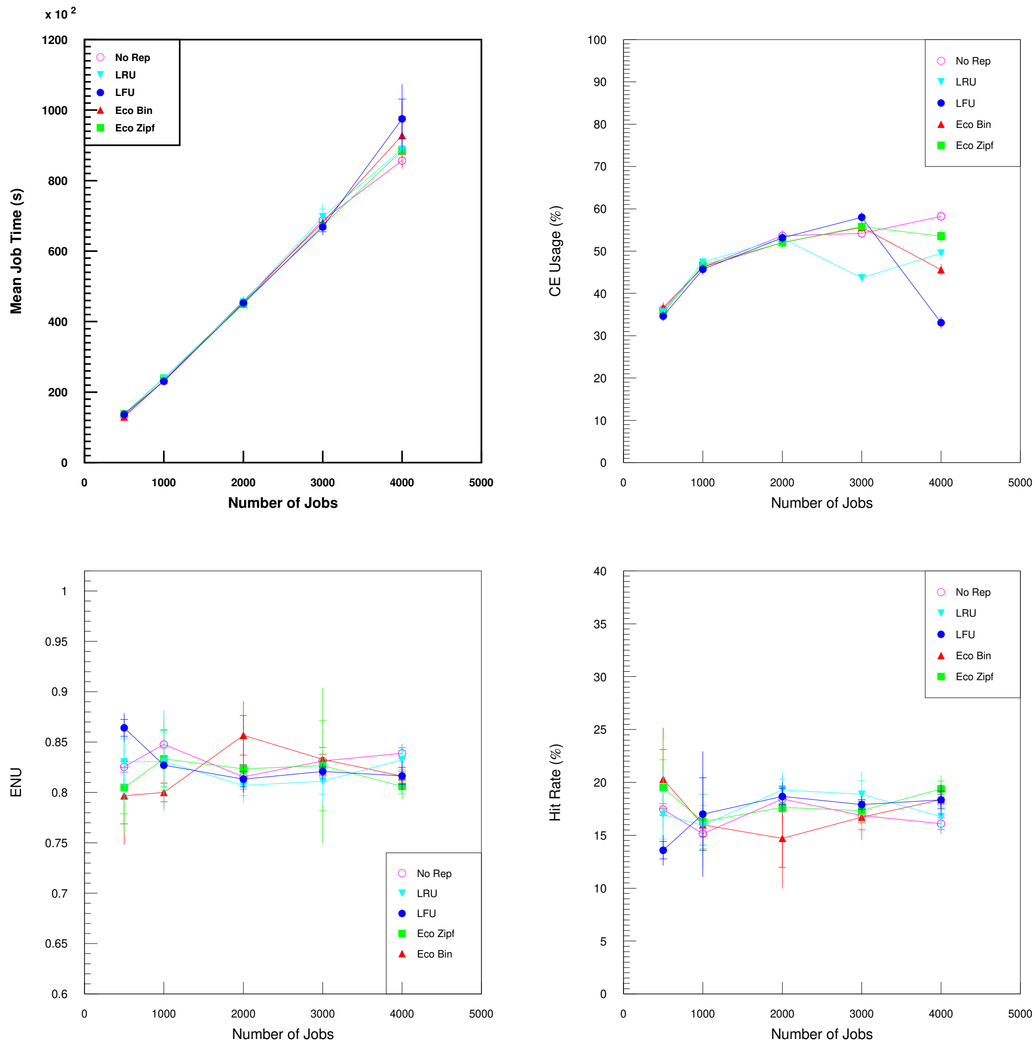

Figure 8.4: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, with varying number of jobs, using Queue Access Cost scheduler.

Replication Strategy	Mean Job Time (1000 jobs) (seconds)	Mean Job Time (10000 jobs) (extrapolated) (seconds)
No Replication	23120 ± 190	221700 ± 1200
Eco Bin	20740 ± 280	191900 ± 2400
LRU	19100 ± 390	163400 ± 2100
LFU	19900 ± 140	140600 ± 1000

Table 8.7: Extrapolation of mean job times from 1000 to 10000 jobs.

of mean job time with 1000 jobs can be extrapolated to $\mathcal{O}(10000)$ jobs and beyond, with the relationships between the different replication algorithms holding. The LFU is the fastest strategy, giving a projected saving in mean job time of up to 40%.

Looking at the other metrics, it can be seen that the CE usage generally rises as the number of jobs increases, reflecting the heavier workload. With the LFU, however, there is a sharp drop between 3000 and 4000 jobs. This is because with the higher number of jobs, the scheduling algorithm is sending most of the extra jobs to a few sites from where the data are easily accessible, leading to more uneven distribution of jobs around the grid. The same trend, although less marked, can also be seen with the other algorithms. The ENU is largely constant, for all the replication strategies, as the number of jobs increases. Likewise, there is little significant change in the hit rate. These metrics are harder to extrapolate beyond the measured range than the job time, but are likely to follow similar trends to the prototype in the last chapter.

8.4 Varying D

Section 8.2 explained how the small ratio of SE size to the overall dataset size, D, led to lack of effectiveness of the replication algorithms for this configuration. The last section showed that this held true even for high numbers of jobs running, and so this section examines the dependency of the algorithms on D. Using the Queue Access Cost scheduler, and submitting 1000 jobs to the grid, with the other parameters as before, the overall size of the dataset was successively halved, thus increasing the fraction which could be stored by a Tier-2 SE. The dataset size was reduced rather than the SE size increased because of the memory limitations which were mentioned in Section 8.1.2. D was varied from 1.2×10^{-3} to 7.5×10^{-2}, bringing it closer to the more realistic level of $\mathcal{O}(10^{-1})$. The results of this test are shown in Figure 8.5.

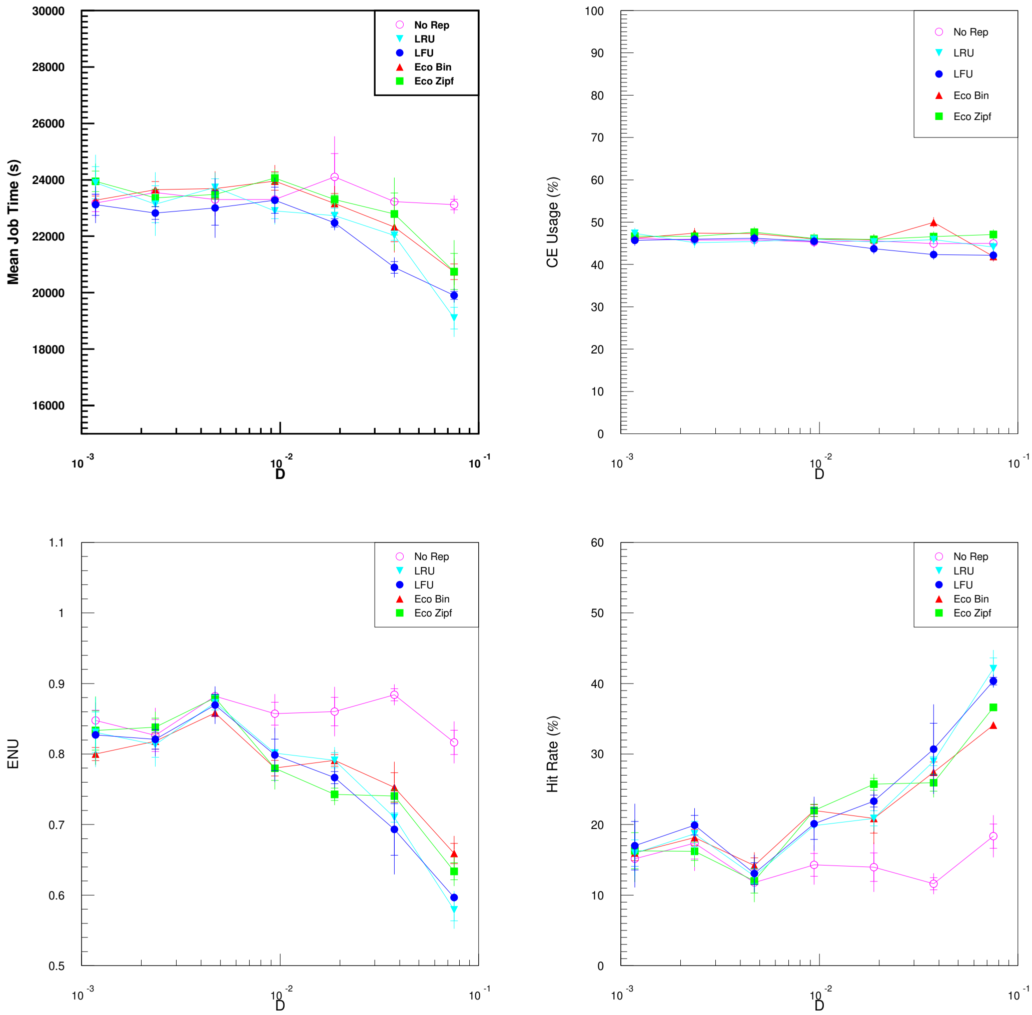

Figure 8.5: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, Queue Access Cost scheduler, varying the value of D.

Firstly, as should be expected, the mean job time without replication is independent of D. For $D \lesssim 10^{-2}$, replication gives no advantage; for $D > 10^{-2}$, the mean job time drops rapidly for all the replication strategies. For the highest value of D

tested, the LRU and LFU are slightly faster than the economic models, which agrees with the results from the simple topologies. This gives a pattern of results similar to that seen for the prototype LCG topology, which would similarly be exaggerated for higher numbers of jobs. Replication, therefore, continues to be an important way to reduce job times and the LRU and LFU strategies are the most effective.

There is little variation in the CE usage. The ENU falls as D increases, for all the replication algorithms. Without replication it naturally remains constant. This shows the increasing effectiveness of the replication strategies as their access histories contain a more representative sample of the whole dataset, and is also reflected in the increasing hit rate. For both ENU and hit rate, the simple strategies perform better than the economic models.

8.5 Varying P

While the most important resource in a data grid may be the data storage resources, it is also important to examine the effect of the computing resources. OptorSim's model of compute resources, as Chapter 5 explained, is quite simple and so a detailed investigation of the effects of compute power is not possible. It is possible, though, to vary the value of P, the average compute power at a site, and determine to what extent the jobs are bound by processing time compared to data transfer time.

With the Queue Access Cost scheduler, and all other parameters as before, P was varied and the behaviour of the different replication strategies tested. The results for the mean job time are shown in Figure 8.6. There is a drop of about 6% in the mean job time as P is increased by over a factor of 10 from just under 200 kSI2000 to 3267 kSI2000, with no significant differences in the response of the different replication algorithms. This is a relatively small decrease in the time taken for such a large increase in the computing power available, showing that the limiting factor in this grid is data transfer rather than computing time.

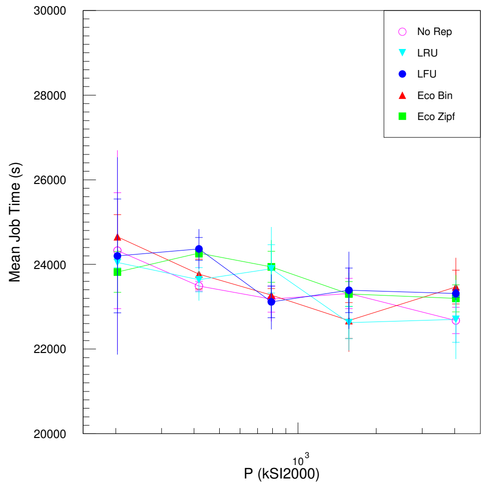

Figure 8.6: Mean job time for different replication algorithms, Queue Access Cost scheduler, varying the value of P.

8.6 Varying C

As the data transfer time is the limiting factor, it is interesting to examine the dependence of the results on the site connectivity C. In this section, C is varied for the Tier-2 sites and the performance of the replication algorithms measured, using the Queue Access Cost scheduler. Varying C just for the Tier-1 sites has no effect, because the bottlenecks for this configuration are at the Tier-2s rather than the Tier-1s.

The links to all the Tier-2 sites were therefore varied from 155 Mbps to 10 Gbps, and the results are shown in Figure 8.7. This shows a sharp decrease in the job time as the Tier-2 connectivity increases from 155 to 622 Mbps. As it is increased further, the job time levels off beyond 1 Gbps. This is due to the fact that the Tier-1 sites in the simulation model have an average connectivity of 882 Mbps, and so when the Tier-2 connectivity rises above this, the bottleneck is at the Tier-1s rather than the Tier-2s. It is therefore clear that the overall performance of these analysis-type jobs is controlled by the lowest bandwidth in the network. For improved grid performance in data-intensive analysis jobs, Tier-2 sites should therefore have as high a bandwidth

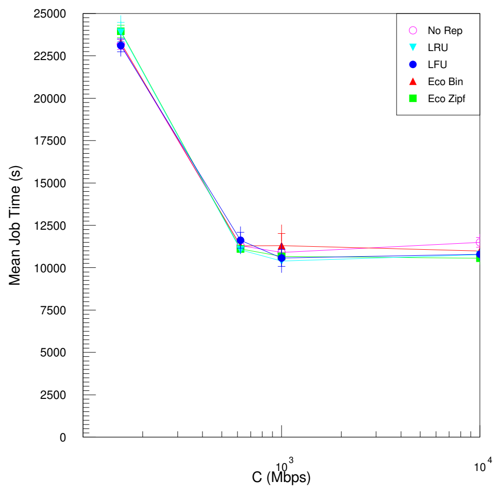

Figure 8.7: Mean job time for different replication algorithms, Queue Access Cost scheduler, varying the value of C for the Tier-2 sites.

as possible.

It is also instructive to look at the effect of connectivity on grid performance across all the topologies considered thus far. Figure 8.8 shows the mean job time as a function of connectivity for the topologies presented in this and the previous chapter. This shows that when the other simulation parameters, such as processing time for files, are similar, there is a general trend for mean job time to decrease as the connectivity increases.

The exceptions to this trend in Figure 8.8 are the LCG 2008 topology and Hybrid 3. In these topologies, although the general connectivity was low, the connectivity of the master site or sites was high; Hybrid 3 had connectivity to the master site of 10 Gbps, while the LCG topology had average connectivity to the Tier-1 sites of 2.7 Gbps and connectivity to CERN of 22 Gbps. Using these values would place these data points in the same trend as the other topologies, although it must also be remembered that the values of processing power, sizes of datasets and so on are different for the LCG configuration. It can also be seen that for the case with no replication, there is little difference in mean job time for the four topologies (Tree, Ring 2, Hybrid 1 and Hybrid 2) which have the same degree of connectivity to the

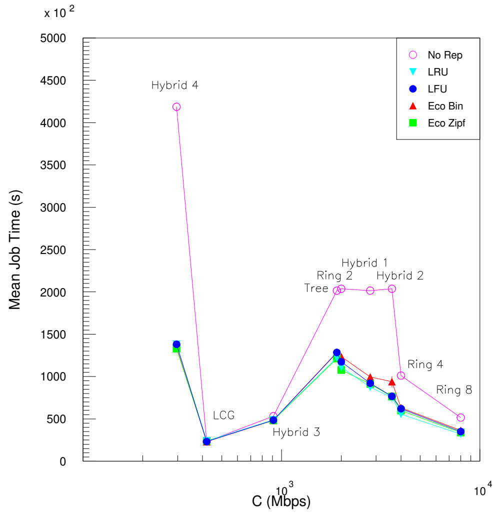

Figure 8.8: Mean job time for different replication algorithms, Queue Access Cost scheduler, considering C for all topologies.

master site. Connectivity to the data source or sources is therefore more important than the underlying topology.

8.7 Effects of Background Network Traffic

The previous section showed the effect of varying the average site connectivity for Tier-2 sites. All these results have been obtained with background network traffic applied, however; it is instructive now to consider the effects of that traffic itself. The performance of the different strategies is therefore compared with the background network traffic switched off and on; the results for mean job time and CE usage are shown in Figure 8.9. The ENU and hit rate were unaffected and are not shown.

This shows an increase in the mean job time of about 33% when background traffic is present, for all the replication algorithms. The CE usage also increases, by about 50%. The increase in job time is clearly due to the longer time taken in transferring files when the bandwidth is lower, as the average bandwidth available with the background traffic profile which was used is about 35% lower than the maximum available without background traffic. The increase in CE usage, however,

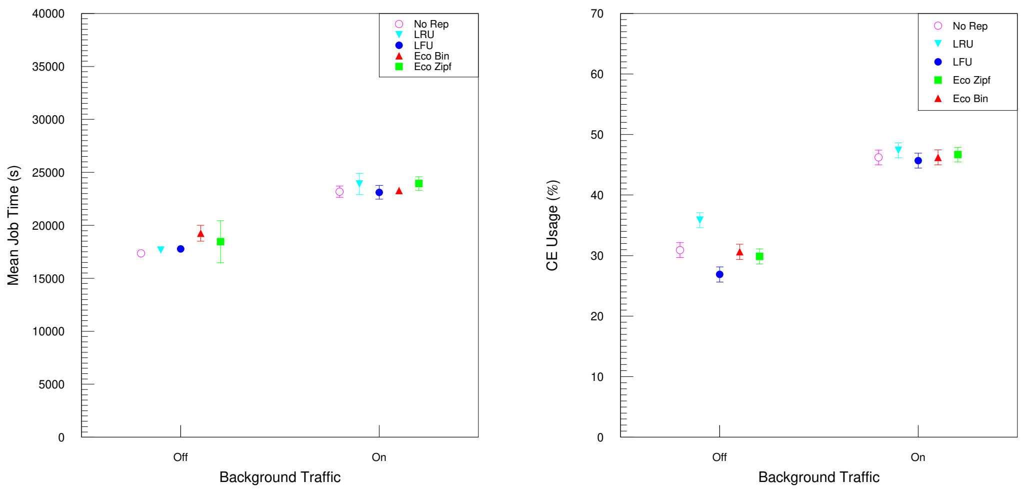

Figure 8.9: (a) Mean job time and (b) CE usage for different replication algorithms, with Queue Access Cost scheduler, background network traffic on and off.

is due to the higher bandwidth without background allowing concentration of jobs at fewer sites. When the network is more congested, jobs are distributed more evenly and thus the overall CE usage is higher. These results show that the state of the general-purpose research networks is important in planning grid resources, especially for Tier-2 sites which will not have a dedicated connection to a data source.

8.8 Effects of Site Policies

In the previous experiments, site policies - the types of jobs which would be accepted by each site - were set according to their planned usage. Here, the effect of site policies on the overall running of the grid is investigated. This was done by defining two extremes of policy. In the first, designated *All Job Types*, all sites accepted all job types. In the second, designated *One Job Type*, each site would accept only one job type, with an even distribution of sites for each job type. The CAF, being a special case, still accepted all job types. The default set of site policies is therefore in between these two extremes, and is designated in the results below as *Mixed*. The results are shown in Figure 8.10.

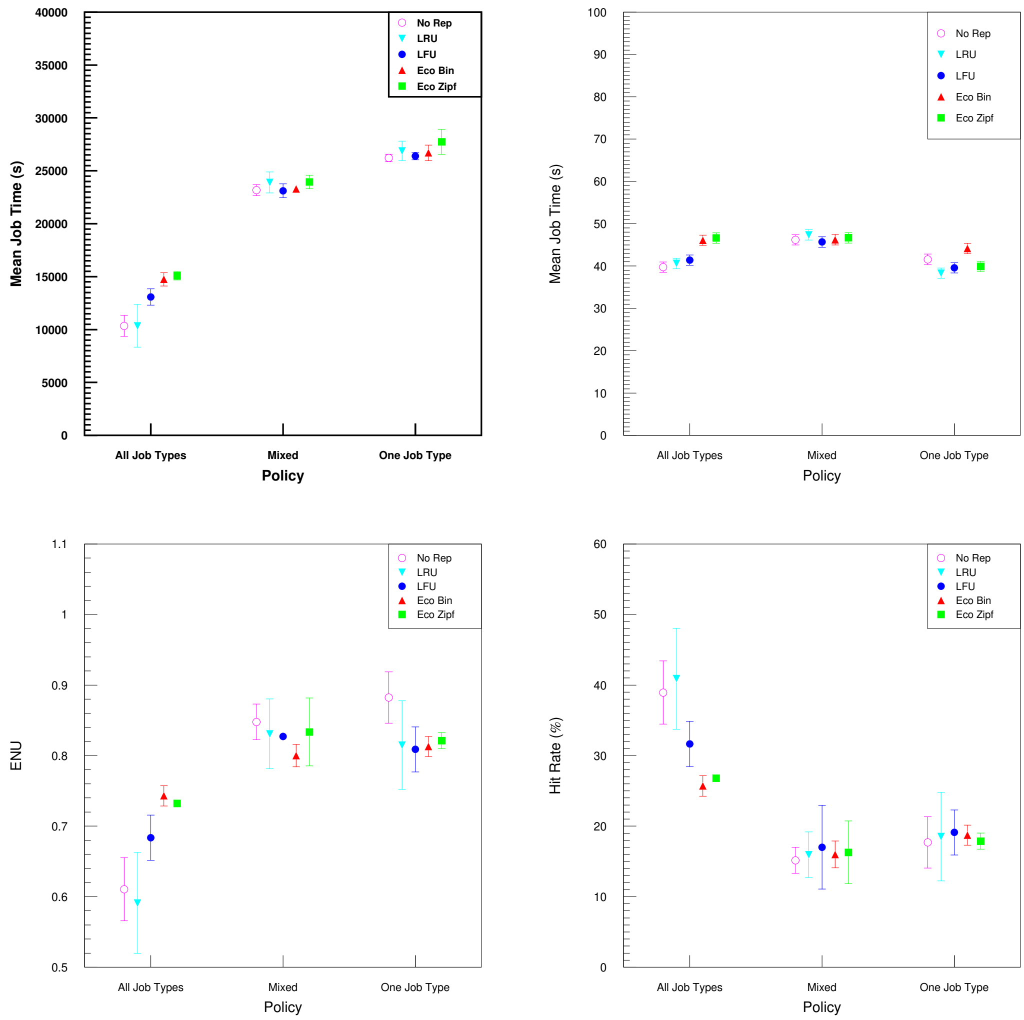

Figure 8.10: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, with different site policies.

These results show that the overall pattern of site policies on the grid have a powerful effect on performance. The mean job time with the *All Job Types* policy is about 60% lower than with the *One Job Type* policy. This is true across all the replication strategies, although the effect is strongest with no replication and with the LRU. While there is no great difference in CE usage with the three policy types, *All Job Types* gives a lower ENU (about 25% lower than the others) and a higher hit rate (60 - 70% higher than the others). It seems clear that an egalitarian approach, in which resources are shared as much as possible, yields benefits to all grid users. It would therefore be recommended that experiment collaborations make every effort to share their resources widely.

8.9 Effects of Zipf-like Access Pattern

In all the experiments presented so far, jobs have accessed their files using a sequential access pattern. Other access patterns are also possible, however, some of which were described in Chapter 5. Perhaps the most likely of these, especially in a chaotic analysis situation, is the Zipf-like access pattern, which models the situation when a few files are very popular. An example could be files containing data from a set of possible Higgs events, which would naturally attract a great deal of attention from LHC physicists.

It is still unclear, however, what the dominant access pattern in LCG will be. In [128], examination of access patterns for the D0 experiment at FNAL showed that although the least popular files followed a Zipf-like pattern, there were a large number of popular files which were all accessed with the same frequency, which corresponds to the use of the sequential access pattern in OptorSim. This observation may be specific to the D0 sample studied, or may be applicable to HEP experiments in general, but gives strong motivation to examine the relative effects of sequential and Zipf access patterns with OptorSim.

The scheduling test from Section 8.2 was therefore repeated with the Zipf-like access pattern rather than the sequential access pattern. The Zipf parameter α was set to 0.85, which is in the middle of the observed range of 0.7 - 1 for web page accesses. The results with the Queue Access Cost scheduler, compared to the results with sequential access, are shown in Figure 8.11. This shows quite a different pattern

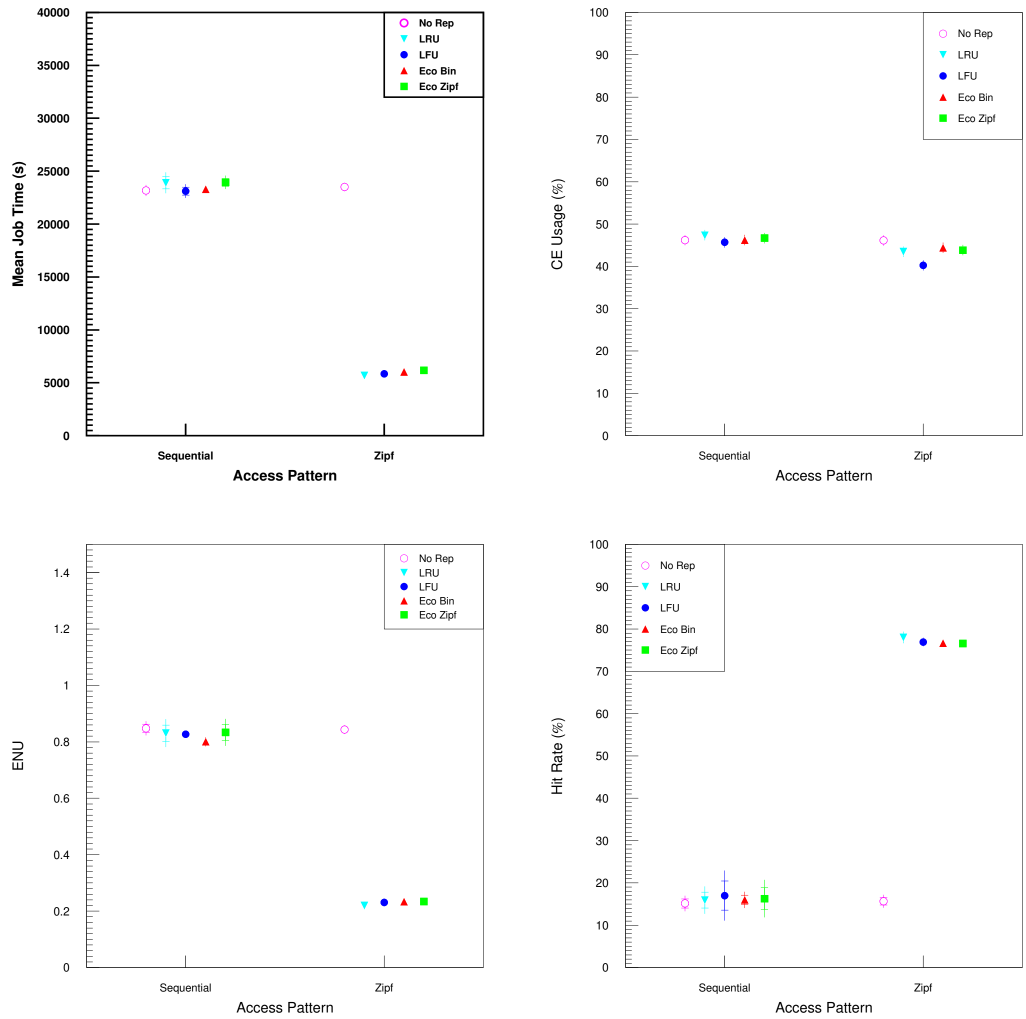

Figure 8.11: (a) Mean job time, (b) CE usage, (c) ENU and (d) hit rate for different replication algorithms, with the Queue Access Cost scheduler and Zipf-like access pattern.

to the previous results. Although the four replication algorithms still have very similar performances, they are now about 75% faster than without replication. ENU is correspondingly lower and hit rate higher, while CE usage is unchanged. This is due to the way in which a few files from each job's fileset are accessed many times during the jobs, while others are accessed infrequently. This allows the access histories to predict file values more accurately than with the sequential pattern, where they may see a file only once. As the number of jobs and the proportion of the whole dataset seen by an individual SE increases, however, the results with sequential access should tend towards a similar pattern as for the Zipf access. This is borne out by the results from varying D with sequential access. The presence of any Zipf-like element, even if combined with a sequential pattern as in the D0 sample, would make dynamic replication highly desirable.

8.10 Summary

In this chapter, detailed simulations of dynamic file replication and deletion on LCG resources for 2008 were performed. These showed, first of all, that the Queue Access Cost scheduling policy both reduces job running times significantly and gives the best use of grid resources. While there was not a clear difference between the scheduling algorithms, in terms of their performance, for the simple topologies in the last chapter, this shows that in a large and complex grid scenario it is best to use a scheduler which accounts for both CE status and data location when scheduling.

With sequential file access patterns, replication at run-time (recalling that there were already replicas present at a number of locations through the grid at the start of the simulation) was initially less effective here than in the simple grids. However, this was shown to be due to the small value of the storage metric D. At higher values of SE size compared to dataset size, where the replication algorithms had broader knowledge of the dataset, replication did give significantly faster job times, with the LRU and LFU slightly faster than the economic models. Recalling from the previous chapter that the number of initial replicas had a significant impact on the performance, fewer copies of the experiment AOD at Tier-1 sites would also give greater need for dynamic replication.

Simulation of large numbers of jobs was found to be restricted for this configuration, due to the large memory requirements; future work could involve refactoring of the code to make better use of memory, so that yet larger simulations could be run without requiring high machine specifications. Making use of results from the prototype LCG model in the previous chapter showed, however, that the mean job time increases linearly with time and that the economic models should be approximately 15-20% faster than without replication while the LRU and LFU should be 30-40% faster, for sequential access.

In addition, it was shown that if the file access pattern involved some files being more popular than others, such as a Zipf-like pattern, the advantages of dynamic replication were greatly increased. The actual file access patterns for LHC analysis will therefore have a significant impact on the effectiveness of replication. Again, the LFU and LRU performed slightly better than the economic models. There was little difference between the two varieties of the economic model, although the Zipf-based version gave slightly better network usage and hit rate in some circumstances. This indicates that strengths of the economic model may lie in the fact that not every file has to be replicated, rather than in the actual valuation of the files. Further refinement of the algorithms may lead to improved performance, but at present their higher complexity suggests that it would be better to use a simpler algorithm such as the LFU.

The importance of sharing grid resources was also shown, with large gains in job times, network usage and hit rate when sites shared resources freely. While complete freedom of resource access may not be possible, for a number of reasons, this suggests that site policies should be made as broad as possible within the LHC context.

Chapter 9

Conclusions

The computing and data handling needs of the next generation of high energy physics experiments, such as the LHC at CERN, have driven the development of grid technology. Grid computing allows resources to be shared between geographically scattered users, thus allowing LHC physicists to access data and computing resources without knowing about the underlying infrastructure of even the location of the resources. It removes the burden of a single central resource with its inherent risk of failure.

In such a grid environment, the data management middleware is responsible both for file movement and replication. The file transfer and catalogues used in the LHC Computing Grid were discussed and the performance of the LCG File Catalogue investigated and compared to the Replica Location Service used in the European DataGrid project. The LFC was shown to be stable and scalable up to $\mathcal{O}(10^7)$ catalogue entries and $\mathcal{O}(100)$ concurrent operations with no indication of performance degradation. With 1 million entries in the catalogue, it was found that most operations took about 20 ms. It was also shown that in a secure, consistent environment, the LFC's performance was improved with respect to the RLS implementation. Using security, while the rate for single operations was reduced by a factor of 20 with respect to the insecure rates, the rate for multiple operations within a transaction was reduced by only 20%. The LFC and gLite's FiReMan catalogue are now being tested by the LHC experiments for their particular requirements, and the LFC has been installed as a production service for the ZEUS experiment at DESY, SEE-GRID in Greece and TW-Grid in Taipei. Further development and feedback from users as LHC data-taking approaches will undoubtedly continue to improve the performance of the LFC and of the LCG data management middleware in general.

The grid simulator OptorSim was presented and shown to be a useful tool for the investigation of grid scheduling and replica optimisation strategies. Developed for the purpose of testing replica optimisation strategies, it is based on the architecture of the European DataGrid. Any grid topology can be simulated, with a large choice of parameters which can be varied. Following validation of the simulation code and algorithms, OptorSim was used to simulate some fairly simple topologies and examine the performance of the different algorithms. Following on from this, a model was constructed of LCG as it is expected to be in 2008, the first full year of data-taking for the LHC. This consists of CERN as a central Tier-0 site, 11 Tier-1 sites in different regions of the world, and over 100 Tier-2 sites.

OptorSim was used to evaluate a number of scheduling and replica optimisation algorithms in both the simple and LCG topologies, aiming primarily to reduce the job time seen by users but also to give the best use of the grid's storage, computational and network resources. It was shown that while for the simple grids the best scheduler differed, for more complex grids and especially for the LCG configuration, the Queue Access Cost scheduler performed best. This policy, which made use of information about how busy a site was as well as the location of required data, reduced job times by up to a factor of 5 compared with a random scheduling policy. It also gave greatly improved use of grid resources.

The value of file replication was found to be dependent on several factors. A distinction was made between "static" file replication, in which replicas were made from CERN to the Tier-1s in a planned fashion, and kept at these sites, and "dynamic" replication which was performed by the Replica Optimisation Service while jobs are running. Dynamic replication was found to be more useful the fewer static replicas were present. It was also found that dynamic replication became more important as the number of jobs running on the grid increased, and as sites' access histories developed, giving better knowledge of the overall dataset. Dynamic replica optimisation would therefore lead to improved performance, with jobs running up to about 40%

faster, in a grid such as LCG.

The performances of the different replication strategies were found, on the whole, to be similar. The simpler strategies of LRU and LFU were found to perform up to 20% and 30% better, respectively, than the economic models which were proposed, and so the economic models would not be recommended for a present-day grid. It is also important that all grid resource providers and users trust any automatic file replication and deletion tools, thus giving the LRU and LFU an added advantage. In future, however, if grid resources were sold or hired as commodities, an economic model would provide a natural mechanism with which to implement replica optimisation. This could be a fruitful avenue for future work in OptorSim, perhaps implementing budgets and a grid currency for the auction protocol. Other improvements could include more detailed modelling of compute resources at sites, the simulation of output files from jobs, and temporary unavailability of resources. All of these would bring OptorSim a step closer to the conditions on a real grid. Ideally, it would be good to take data from the real, running LCG to feed into OptorSim and, in turn, use OptorSim to inform decision-making for the real grid.

There are many other possibilities for future research using OptorSim. So far, for example, replication has been at the level of files containing many events. It would be interesting to examine the effect of different granularities of replication, particularly at the event level. The focus here has also been on analysis-type jobs, so it would also be useful to simulate reconstruction and Monte Carlo simulation jobs on the same grid. Another interesting possibility would be to compare the effects of a central or distributed file catalogue, both of which are deployment models which may be used for the LFC, and which is a very pertinent question for file management in HEP experiments.

Appendix A

Artificial Neural Networks for Higgs Discovery at the LHC

This chapter contains a brief summary of work carried out in the first half of 2003, investigating the use of artificial neural networks in particle discovery at the LHC. One of the LHC's priorities is the discovery of the Higgs boson. Its existence is an important part of the Standard Model of physics; its mass, however, is unknown. It has implications for the Standard Model itself and for a number of extensions to the Standard Model which have been proposed. There are many ways in which a Higgs could be produced, some more likely than others [129] [130]. It is possible that adaptive learning techniques, such as those used in artificial neural networks or support vector machines, could improve the possibility of detection in the less likely channels, and so this chapter evaluates their use in the $HZ \rightarrow llb\bar{b}$ channel.

A.1 Artificial Neural Networks

An artificial neural network (ANN) is an information processing structure inspired by the way biological nervous systems work. Their advantages include adaptive learning, self-organisation, real-time operation and fault tolerance; they are especially well suited to pattern recognition tasks. In a high-energy physics context, this includes particle identification and the ability to separate signal from background, with much evidence (such as [131, 132]) to suggest that they can perform better than traditional cuts-based analyses.

The most common type of ANN is the multi-layer perceptron (MLP), which consists of an input layer of nodes or neurons, an output layer and zero or more (usually one or two) hidden layers. A simple example is shown in Figure A.1. The notation used to denote MLP structure will be $I - L_1 - L_2 - O$, where I is the number of neurons in the input layer, O is the number in the output layer (usually 1) and L_x is the number in the x^{th} hidden layer.

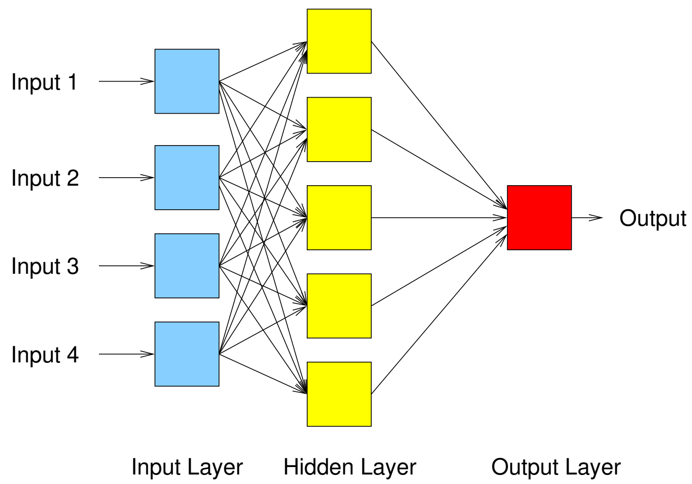

Figure A.1: A simple multi-layer perceptron, with a 4-5-1 structure.

An MLP works by allocating each neuron an activation function, which gives the neuron a rule for when to activate and process the input data. The input layer does no processing, simply passing the data into the first hidden layer. If a particular input to a neuron in the hidden layer obeys the activation rule for that neuron, it performs some transformation on the data and passes it on to the output layer. Neural networks must be trained to be useful; at the start of training, the weights on the neurons are set to small random values, and these are adjusted as training proceeds until the network is able to do useful work. If an MLP was being trained for particle identification, for example, it would be trained on a subset of real data until it was able to identify the particle in question.

A.2 Analysis of the $HZ \rightarrow llb\bar{b}$ Channel

The Feynman diagram for the $HZ \rightarrow llb\bar{b}$ production channel is shown in Figure A.2. It is highly unlikely to be used for Higgs discovery at the LHC, with [133] quoting

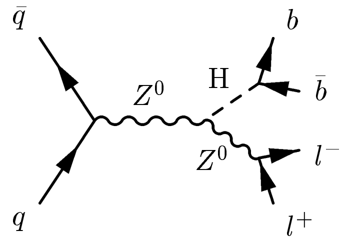

Figure A.2: Feynman diagram for the $HZ \rightarrow llb\bar{b}$ channel.

an $S/\sqrt{B}$ of 1.1. $S/\sqrt{B}$ is the standard measure of significance, where $S(B)$ is the number of signal (background) events. An $S/\sqrt{B}$ of 5 would correspond to a discovery-level signal. The aim of this study was to see whether the sensitivity could be improved by using ANNs rather than cuts-based analysis, as the technique could then be applied to more important channels such as $ttH \rightarrow b\bar{b}$. A Higgs mass M_H of 100 GeV was used, to allow comparison with other studies at that mass.

The signal and possible backgrounds for the $llb\bar{b}$ final state are shown, with their production cross-sections, in Table A.1. This is for 14 TeV proton-proton collisions with an integrated luminosity of 30 fb^{-1} (i.e. 3 years' low-luminosity running of the LHC).

Process	Cross-Section (fb)
ZH	6.8×10^{14}
ZZ	5.7×10^{15}
$t\bar{t}$	4.9×10^{18}
Z + jets	3.5×10^{19}

Table A.1: Production cross-sections for the ZH signal and for backgrounds for 14 TeV pp collisions at the LHC, with $M_H = 100$ GeV.

Monte Carlo datasets based on these values had previously been produced for a cuts-based analysis [134] and some standard pre-selection cuts applied. Two different ANN approaches were considered, using the PAW MLP package [135]. Firstly, the two ANNs were trained against the $t\bar{t}$ and $Z + jets$ backgrounds respectively. (The ZZ background is too similar to the signal for ANN training). Several different network structures were tried for each, with various input variables. Those with the lowest error on an independent test sample were chosen. For the network trained against $t\bar{t}$, this was a 4-16-16-1 structure with the inputs being jet separation, separation between H and Z bosons, Z mass and sum of the transverse momenta. For the $Z+jets$ network, the structure was 5-10-20-1 with jet separation, separation between H and Z bosons, sum of jet energies, sum of transverse momenta and sum of the two largest lepton-jet angles as the inputs. After training, the two ANNs were used to analyse another independent sample of signal and background events. Their outputs were plotted against each other as shown in Figure A.3(a) and two-dimensional cuts made to calculate expected numbers of events. The axis marked *Anti-ttbar* refers to the output of the ANN trained against the $t\bar{t}$ background, while the axis marked *Anti-(Z+jets)* refers to the output of the ANN trained against the $Z + jets$ background.

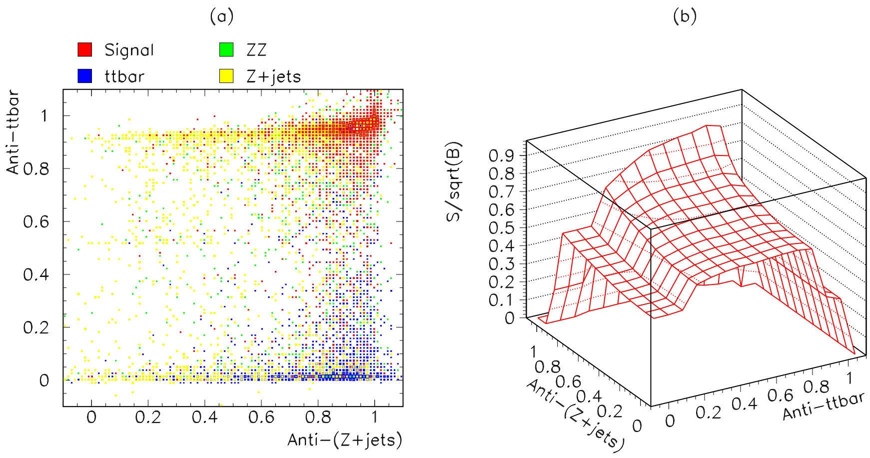

Figure A.3: (a) Neural network outputs and (b) $S/\sqrt{B}$ for the 2-ANN case.

A similar procedure was then followed for a single ANN, trained against both types of background. The best structure found for this was 6-10-20-1 with the input variables being all of those mentioned above. This ANN was then used to analyse independent signal and background samples; its output is shown in Figure A.4(a). (Note that "Number of events" in the figure refers to output from the ANN, not to physical event numbers). The expected numbers of physical events were then calculated for different cuts on its output.

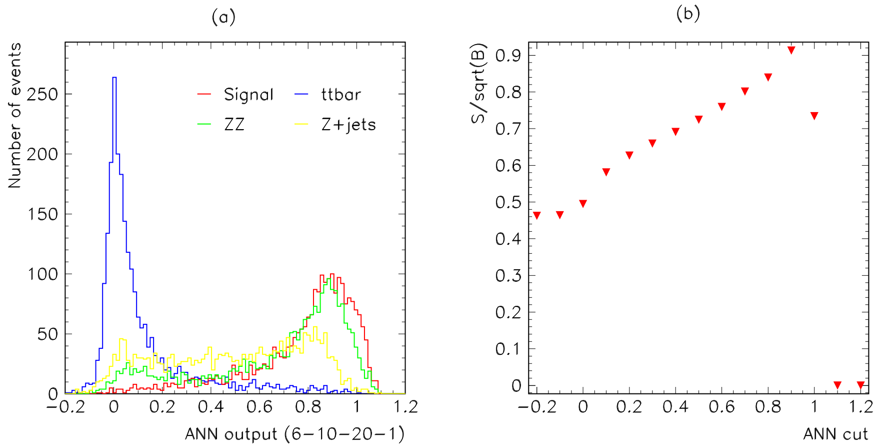

Figure A.4: (a) Neural network outputs and (b) $S/\sqrt{B}$ for the 1-ANN case.

For both the two-ANN and single-ANN cases, the ratio of signal to background (S/B) and the significance ($S/\sqrt{B}$) were then found (Figures A.3(b), A.4(b)) and compared to the cuts-based results given by by [134]. The results are shown in Table A.2, where ERW refers to the results of [134] using the same cuts as [133] but with updated simulation code, and JK refers to the results of [134] using different cuts.

This shows a clear improvement over the cuts-based analysis. In comparisons of cuts with similar background figures, the two-ANN case gave an improvement in signal of $\sim$13%, and $S/\sqrt{B}$ is improved in both cases: with the single ANN, by $\sim$7% and with the two ANNs, by $\sim$15%. The single ANN also had a more complex structure, having six inputs rather than four or five. The overall significance, however, is still low compared with other channels [129] and it is therefore unlikely that this mode will contribute to the discovery of a low-mass Higgs.

	ERW	JK	1 ANN	2 ANNs
N_{HZ} (m_H=100 GeV)	96	77	64	97
N_{ZZ}	243	151	127	199
N_{Zjj}	10031	7090	4381	8200
$N_{t\bar{t}}$	1196	833	346	1236
$N_{Background}$ (total)	11469	8074	4854	9667
S/B	0.84%	0.95%	1.31%	1.00%
S/$\sqrt{B}$	0.90	0.85	0.91	0.98

Table A.2: Comparison of ANN and cuts-based results. N_x is the expected number of events of type x.

A.3 Summary

The study of the $HZ \to llb\bar{b}$ channel for $M_H = 100GeV$ using ANNs has shown that improvement of the sensitivity can be made, although it is still not enough to make it a viable discovery channel. This kind of analysis can, however, be applied to more significant channels such as $ttH \to b\bar{b}$, (which has an expected $S/\sqrt{B}$ of 5), complementing current cuts-based analysis work which may eventually help in the discovery of the Higgs boson.

References

[1] A. T. Doyle. Data Centric Issues: Particle Physics and Grid Data Management. In *UK e-Science All Hands Conference*, Sheffield, UK, September 2002.

[2] David G. Cameron. Replica Management and Optimisation for Data Grids, January 2005. Ph.D. Thesis, University of Glasgow.

[3] A.L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. In *13th IEEE Symposium on High Performance Distributed Computing (HPDC-13)*, Honolulu, Hawaii, USA, June 2004.

[4] J.P. Baud and J. Casey. Evolution of LCG-2 Data Management. Presentation at *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September, 2004.

[5] L. O. Hertzberger. Does HEP Still Hold Challenges for Computer Science? *Computer Physics Communications*, 57(1–3):15–22, December 1989.

[6] P. Zanella. 30 Years of Computing at CERN - Part 1. *CERN Computer News Letter*, 36(2), 2001. CERN-CNL-2001-002.

[7] C. Jones. Computing at CERN: the mainframe era. *CERN Courier*, 44(7), 2004.

[8] J. C. R. Licklider. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11, March 1960.

[9] J. C. R. Licklider and R. W. Taylor. The computer as a communication device. *Science and Technology*, April 1968.

[10] W3C: World Wide Web Consortium. `http://www.w3.org/`.

[11] The Globus Alliance. The Globus Toolkit. `http://www.globus.org/toolkit/`.

[12] Gnutella. `http://www.gnutella.com/`.

[13] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, California, July 2000.

[14] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. SETI@home-massively distributed computing for SETI. *Computing in Science and Engineering*, 3(1):78–83, Jan - Feb 2001.

[15] LHC@home. `http://athome.web.cern.ch`.

[16] F. J. Corbató and V. A. Vyssotsky. Introduction and Overview of the Multics System. In *AFIPS Fall Joint Computer Conference*, pages 185–196, November 1965.

[17] The ALICE Collaboration. ALICE Computing Model. Technical Report CERN-LHCC-2004-038/G-086, CERN, January 2005.

[18] The ATLAS Collaboration. The ATLAS Computing Model. Technical Report CERN-LHCC-2004-037/G-085, CERN, January 2005.

[19] The CMS Collaboration. The CMS Computing Model. Technical Report CERN-LHCC-2004-035/G-083, CERN, January 2005.

[20] The LHCb Collaboration. LHCb Computing Model. Technical Report CERN-LHCC-2004-036/G-084, CERN, January 2005.

[21] The LCG Project. LHC Computing Grid Technical Design Report. Technical Report CERN-LHCC-2005-024, CERN, June 2005.

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

[23] Memorandum of Understanding for Collaboration in the Deployment and Exploitation of the Worldwide LHC Computing Grid. Technical Report CERN-C-RRB-2005-01, CERN, May 2005.

[24] A. Szalay and J. Gray. The World-Wide Telescope. *Science*, 293(5537):2037–2040, September 2001.

[25] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, and D. Williams. High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. In *Proceedings of SuperComputing 2001*, Denver, Colorado, November 2001.

[26] S. R. Amendolia, F. Estrella, W. Hassan, T. Hauer, D. Manset, R. McClatchey, D. Rogulin, and T. Solomonides. MammoGrid: A Service Oriented Architecture based Medical Grid Application. In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing*, Wuhan, China, 2004.

[27] F. Gagliardi and M. Bégin. EGEE: providing a production quality Grid for e-Science. In *International Symposium on Emergence of Globally Distributed Data*, Sardinia, Italy, June 2005.

[28] The CMS Collaboration. CMS: The Computing Project Technical Design Report. Technical Report CERN-LHCC-2005-023, CERN, June 2005.

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

[30] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[31] I. Foster. What is the Grid? A Three Point Checklist. *GRIDtoday*, 1(6), July 2002.

[32] PBS. Portable Batch System. `http://www.openpbs.org`.

[33] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[34] Global Grid Forum. GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20), March 2003.

[35] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Services Architecture Working Group, Global Grid Forum, June 2002.

[36] GGF OGSI Working Group. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation, June 2003.

[37] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling Stateful Resources with Web Services, May 2004. Whitepaper.

[38] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 0:0–20, 2004.

[39] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Francisco, California, August 2001.

[40] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw. The Legion Resource Management System. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 162–178, 1999.

[41] Sybase, Inc. Sybase Avaki Enterprise Information Integration. `http://www.avaki.com`.

[42] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra. *NetSolve: Grid Enabling Scientific Computing Environments*. Elsevier, 2005. To appear.

[43] M. Beck, J. Dongarra, J. Huang, T. Moore, and J. S. Plank. Active Logistical State Management in GridSolve/L. In *4th International Symposium on Cluster Computing and the Grid (CCGrid2004)*, Chicago, Illinois, April 2004.

[44] Rajkumar Buyya and Srikumar Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, Seoul, South Korea, April 2004.

[45] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Cluster Computing*, 1:119–132, January 1998.

[46] F. Gagliardi, B. Jones, M. Reale, and S. Burke. European DataGrid Project: Experiences of deploying a large scale Testbed for e-Science applications. In *Performance Evaluation of Complex Systems: Techniques and Tools (Performance 2002)*, pages 480–499, January 2002. LNCS 2459.

[47] DataTAG. Research & technological development for a Data TransAtlantic Grid. `http://cern.ch/datatag`.

[48] GridPP. UK Computing for Particle Physics. `http://www.gridpp.ac.uk`.

[49] The GridPP Collaboration. GridPP: Development of the UK Computing Grid for Particle Physics. *J. Phys. G: Nucl. Part. Phys.*, 32:N1–N20, 2006.

[50] P. Avery and I. Foster. The GriPhyN Project: Towards Petascale Virtual-Data Grids, April 2000. NSF Award ITR-0086044.

[51] The iVDGL Project. An International Virtual-Data Grid Laboratory for Data Intensive Science, April 2001. NSF Award ITR-0122557.

[52] OSG. Open Science Grid. `http://www.opensciencegrid.org`.

[53] PPDG. The Particle Physics Data Grid. `http://www.ppdg.net`.

[54] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi. Access Grid: Immersive Group-to-Group Collaborative Visualization. In *Proceedings of the 4th International Immersive Projection Technology Workshop*, June 2000.

[55] J.S. Grethe, C. Baru, A. Gupta, M. James, B. Ludaescher, M.E. Martone, P.M. Papadopoulos, S.T. Peltier, A. Rajasekar, S. Santini, I.N. Zaslavsky, and M.H. Ellisman. Biomedical informatics research network: building a national collaboratory to hasten the derivation of new understanding and treatment of disease. *Studies in Health Technology and Informatics*, 112:100–109, 2005.

[56] M. Bubak, M. Malawaski, and K. Zając. The CrossGrid Architecture: Applications, Tools and Grid Services. In *Grid Computing: First European Across Grids Conference*, Santiago de Compostela, Spain, February 2004.

[57] E. Laure, F. Hemmer, A. Aimar, M. Barroso andP. Buncic, A. Di Meglio, L. Guy, P. Kunszt, S. Beco, F. Pacini, F. Prelz, M. Sgaravatto, A. Edlund, O. Mulmo, D. Groep, S.M. Fisher, and M. Livny. Middleware for the Next Generation Grid Infrastructure. In *Proceedings of Computing in High Energy Physics (CHEP 2004)*, Interlaken, Switzerland, September 2004.

[58] P. Buncic, J.F. Grosse-Oetringhaus, A.J. Peters, and P. Saiz. The Architecture of the AliEn System. In *Proceedings of Computing in High Energy Physics (CHEP 2004)*, Interlaken, Switzerland, September 2004.

[59] O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J.R. Hansen, A. Konstantinov, B. Kónya, J.L. Nielsen, F. Ould-Saada, and A. Wäänänen. The NorduGrid Architecture and Middleware for Scientific Applications. In *International Conference on Computational Science 2003 (ICCS2003). LNCS 2657*, June 2003.

[60] P. Buncic, F. Rademakers, R. Jones, R. Gardner, L.A.T. Bauerdick, L. Silvestris, P. Charpentier, A. Tsaregorodtsev, D. Foster, T. Wenaus, and F. Carmi-

nati. LHC Computing Grid Project: Architectural Roadmap towards Distributed Analysis. Technical Report CERN-LCG-2003-033, CERN, October 2003.

[61] M. Ernst, P. Fuhrmann, M. Gasthuber, T. Mkrtchyan, and C. Waldman. dCache, a Distributed Storage Data Caching System. In *Computing in High Energy Physics (CHEP)*, Beijing, China, September 2001.

[62] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for grid storage. In *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems*, April 2002.

[63] The EGEE Collaboration. EGEE Information and Monitoring Service (R-GMA): System Specification. Technical Report EGEE-JRA1-TEC-490223-R_GMA_SPECIFICATION-v2-0, CERN, July 2004.

[64] S. Andreozzi. GLUE Schema Implementation for the LDAP Data Model. Technical Report INFN/TC-04/16, INFN, September 2004.

[65] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A Framework for Constructing Scalable Replica Location Services. In *International IEEE Supercomputing Conference (SC 2002)*, Baltimore, USA, November 2002.

[66] J. Andreeva, A. Anjum, T. Barrass, D. Bonacorsi, J. Bunn, P. Capiluppi, M. Corvo, N. Darmenov, N. De Filippis, F. Donno, G. Donvito, G. Eulisse, A. Fanfani, F. Fanzago, A. Filine, C. Grandi, J. M. Hernnde, V. Innocente, A. Jan, S. Lacaprara, I. Legrand, S. Metson, H. Newman, D. Newbold, A. Pierro, L. Silvestris, C. Steenberg, H. Stockinger, L. Taylor, M. Thomas, L. Tuura, T. Wildish, and F. Van Lingen. Distributed Computing Grid Experiences in CMS DC04. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[67] M. Girone, M. Branco, R. Chytracek, D. Düllmann, M. Frank, L. Goossens, G. Govi, V. Innocente, J. T. Moscicki, I. Papadopoulos, H. Schmuecker, K. Karr, D. Malon, A. Vaniachine, A. Fanfani, C. Grandi, W. Tanenbaum, L. Tuura, Z. Xie, T. Barrass, and C. Cioffi. Experience with POOL in the LCG Data Challenges of Three LHC Experiments. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[68] J.P. Baud and J. Casey. Evolution of LCG-2 Data Management. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland, September 2004.

[69] The EGEE Collaboration. EGEE Users' Guide: gLite Data Management Catalog. Technical Report EGEE-TECH-570780-v1.0, CERN, April 2005.

[70] C. Munro and B. Koblitz. Performance Comparison of the LCG2 and gLite File Catalogues. In *Tenth International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2005)*, Zeuthen, Germany, May 2005.

[71] SPEC. Standard Performance Evaluation Corporation. `http://www.spec.org/cpu2000/results/cint2000.html`.

[72] Robert G. Sargent. Verification and Validation of Simulation Models. In *Winter Simulation Conference*, 1998.

[73] D. G. Cameron, R. Carvajal-Schiaffino, J. Ferguson, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini. OptorSim v2.0 Installation and User Guide, November 2004. `http://edg-wp2.web.cern.ch/edg-wp2/optimization/downloads/v2_0/edg-opto%rsim/doc/userguide-optorsim.ps`.

[74] The EDG WP2 Optimisation Team. OptorSim Release 2.0, November 2004. `http://edg-wp2.web.cern.ch/edg-wp2/optimization/downloads/v2_0/`.

[75] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[76] J. Moy. The OSPF Specification. Network Working Group. Request for Comments 1131, October 1989.

[77] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Dsitributions: Evidence and Implications. In *IEEE INFOCOM'99*, March 1999.

[78] Antonio Pierro. Fake Analysis at INFN with LCG-2 during DC04, 2004. `http://cmsdoc.cern.ch/cms/LCG/LCG-2/dc04/FakeAnalysis.html`.

[79] D. Cameron and R. Carvajal-Schiaffino and P. Millar and C. Nicholson and K. Stockinger and F. Zini. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. *Journal of Grid Computing*, 2(1):57–69, March 2004.

[80] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. Simulation of Dynamic Grid Replication Strategies in OptorSim. *Int. Journal of High Performance Computing Applications*, 17(4), 2003.

[81] SLAC Bandwidth Monitoring Team. SLAC WAN Bandwidth Measuring Tests. `http://www.slac.stanford.edu/comp/net/bandwidth-tests/antonia/html/slac%_wan_bw_tests.html`.

[82] SLAC Bandwidth Monitoring Team. SLAC WAN Bandwidth Measuring Tests at FNAL. `http://dmzmon0.deemz.net/~wanbanmon/html/slac_wan_bw_tests.html#summary%`.

[83] UK e-Science Grid Network Monitoring. `http://gridmon.ucs.ed.ac.uk/gridmon/`.

[84] GridNM. A Grid Network Monitoring Perl Client. `http://www.hep.ucl.ac.uk/~ytl/monitoring/gridnm/gridnm-client.html`.

[85] Iperf. A TCP/UDP Bandwidth Measurement Tool. `http://dast.nlanr.net/Projects/Iperf/`.

[86] Kihong Park and Walter Willinger. Self-Similar Network Traffic: An Overview. In Kihong Park and Walter Willinger, editors, *Self-Similar Network Traffic and Performance Evaluation*. Wiley Interscience, 1999.

[87] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[88] Kihong Park. Variation in throughput on a network, 2003. Private Communication.

[89] The MONARC Project. `http://monarc.web.cern.ch/`.

[90] Iosif Legrand. Multi-threaded, discrete event simulation of distributed computing system. In *Computing in High Energy and Nuclear Physics (CHEP)*, Padova, Italy, February 2000.

[91] Rajkumar Buyya and Manzur Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, 14:1175–1220, 2002.

[92] A. Takefusa, O. Tatebe, S. Matsuoka, , and Y. Morita. Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In *Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press*, June 2002.

[93] Grid Datafarm. `http://datafarm.apgrid.org`.

[94] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of Dynamic Data Replication Strategies in Data Grids. In *Proc. 12th Heterogeneous Computing Workshop (HCW2003)*, Nice, France, April 2003.

[95] The Network Simulator. `http://www.isi.edu/nsnam/ns/`.

[96] Kavitha Ranganathan and Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. In *Proceedings of the 11th International Symposium for High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[97] PARSEC. Parallel Simulation Environment for Complex Systems. `http://pcl.cs.ucla.edu/projects/parsec/`.

[98] EDGSim. Simulating the European Data Grid. `http://www.hep.ucl.ac.uk/~pac/EDGSim/`.

[99] GridMate. `http://www.caip.rutgers.edu/~xlli/gridmate.htm`.

[100] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, R. Taura, and A. Chien. The MicroGrid: a Scientific Tool for Modeling Computational Grids. In *Proceedings of SuperComputing 2000*, Dallas, Texas, 2000.

[101] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software - Practice and Experience*, 32:135–164, 2002.

[102] Community Scheduler Framework. `http://www.platform.com/products/Globus/`.

[103] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proceedings of Supercomputing '96*, 1996.

[104] S. S. Vadhiyar and J. J. Dongarra. A Metascheduler for the Grid. In *11th IEEE Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[105] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A Network based Information Library for a Global World-Wide Computing Infrastracture. In *High Performance Computing and Networking (HPCN '97)*, April 1997.

[106] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18:1061–1074, 2002.

[107] P. Chandra, Y. Chu, A. Fisher, J. Gao, C. Kosak, T. S. E. Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network*, 15(1):22–35., 2001.

[108] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In *Proceedings of the 2nd International Workshop on Middleware for Grid Computing*, Toronto, Canada, October 2004.

[109] EDG Work Package 1. WP1 - WMS Software Administrator and User Guide. Technical Report DataGrid-01-TEN-0118-1_2, CERN, November 2003.

[110] L. Dowdy and D. Foster. Comparative Models of the File Assignment Problem. *Computing Surveys*, 14(2):287–313, June 1982.

[111] S. Acharya and S. Zdonik. An Efficient Scheme for Dynamic Data Replication. Technical Report CS-93-43, Brown University, September 1993.

[112] O. Wolfson and S. Jajodia. Distributed algorithms for dynamic replication of data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, San Diego, California, June 1992.

[113] K. Ranganathan and I. Foster. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *Proceedings of the International Grid Computing Workshop*, pages 75–86, Denver, Colorado, November 2001.

[114] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. In *Proceedings of the 11th International Symposium for High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[115] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Simulation of Dynamic Data Replication Strategies in Data Grids. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, Nice, France, April 2003.

[116] R. Karedla, J.S. Love, and B.G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, March 1994.

[117] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.

[118] P. Cao and S. Irani. GreedyDual-Size: A Cost-Aware WWW Proxy Caching Algorithm. In *2nd Web Caching Workshop*, Boulder, Colorado, June 1997.

[119] L. Rizzo and L. Vicisano. Replacement Policies for a Proxy Cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, April 2000.

[120] C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, and W.S. Stornetta. Spawn: a Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[121] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the Internet - the POPCORN project. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, pages 592 – 601, Amsterdam, May 1998.

[122] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An Economic Paradigm for Query Processing and Data Migration in Mariposa. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.

[123] W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, 16(1):8–37, March 1961.

[124] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[125] The GEANT Project. `http://www.geant.net`.

[126] Abilene Backbone Network. `http://abilene.internet2.edu`.

[127] JANET. The UK's education and research network. `http://www.ja.net`.

[128] A. Iamnitchi and M. Ripeanu. Myth and reality: Usage behavior in a large data-intensive physics project. Technical Report TR2003-4, GriPhyN, 2003.

[129] The ATLAS Collaboration. ATLAS Detector and Physics Performance. Technical Report ATLAS TDR 15, CERN/LHCC 99-15, CERN, May 1999.

[130] The CMS Collaboration. The Compact Muon Solenoid: Technical Proposal. Technical Report CERN/LHCC 94-38, CERN, December 1994.

[131] P. Chiappetta, P. Colangelo, P. De Felice, G. Nardulli, and G. Pasquanello. Higgs Search by Neural Networks at LHC. *Phys. Lett. B*, 322:219–223, 1994.

[132] S. Dusini, F. Ferrari, I. Lazzizzera, A. Sartori, A. Sidoti, and G. Tecchioli. Searching the Higgs with the Neurochip TOTEM. *Nucl. Phys. Proc. Suppl.*, 65:320–323, 1998.

[133] Elzbieta Richter-Was. Revisiting the Observability of the WH and ZH, H $\rightarrow ll b\bar{b}$ Channel in 14 TeV pp and 2 TeV $p\bar{p}$ Collisions ($lb\bar{b}$ and $llb\bar{b}$ Final States. Technical Report ATL-COM-PHYS-2000-018, 2000.

[134] John Kennedy. $HZ \rightarrow llb\bar{b}$, 2003. University of Glasgow, Private Communication.

[135] J. Schwindling, B. Mansoulié, and O. Couet. Multi-Layer Perceptrons in PAW. `http://paw.web.cern.ch/paw/mlpfit/pawmlp.html`.