Replica Management and Optimisation for Data Grids

David Gordon Cameron

Department of Physics and Astronomy
University of Glasgow

Thesis submitted for the degree of
Doctor of Philosophy

January 2005

Abstract

Grid computing is foreseen as the solution to the large data-handling requirements of the next generation of high energy physics experiments. This emerging paradigm enables distributed computational and data storage resources to be aggregated into one unified utility. This thesis describes and evaluates services developed by the European DataGrid project for managing distributed data, showing that they are sufficiently scaleable and stable to be used as production services for these experiments. A strategy to optimise data access based on an economic model for file trading is introduced and simulation of the Grid environment is used to evaluate its effectiveness. Results show that this model can be used to reduce data access latencies and usage of network resources by ensuring replicated data is distributed effectively around the Grid.

To My Parents

Author's Declaration

This thesis represents work carried out in the last three years as part of the Experimental Particle Physics group at the University of Glasgow. The data management services described in Chapter 3 and the OptorSim Grid simulation tool described in Chapter 6 were co-developed by myself and others working for the European DataGrid project. I developed the performance testing framework for the data management services and obtained the results using this framework which are presented in Chapter 4. I also contributed a significant fraction of effort throughout the last three years to all stages of OptorSim development - design, coding, testing, debugging and documentation - and all the results shown in Chapter 7 are my own work.

Acknowledgements

I would like to thank Prof. David Saxon for giving me the opportunity to work in the Experimental Particle Physics group at the University of Glasgow and PPARC for their funding over the three years.

I especially wish to thank my supervisor Prof. Tony Doyle for all his help and support especially while writing up, always willing to give his time to me despite an extremely busy schedule of seemingly endless meetings and the same goes for my second supervisor Rick St Denis. I owe a lot to the optimisation team I worked with for the three years and much of the work presented here is the result of collective effort from the whole team, so thanks to Kurt for introducing me to Austrian humour and punctuality, Floriano and Ruben for their more laid-back approach, and to those from Glasgow: Paul, Will and Caitriana especially for her calming influence on the office and constant help without which this thesis would have taken much longer to complete.

Everyone in the PPE group at Glasgow deserves thanks, but I especially want to thank the secretary for my first two years Catherine MacIntyre for being so friendly and always willing to help me out of the maze of expenses forms. The lunchtime gang also provided light relief each day: Alan, Dave T, Greig, Ian, Jack, Pete, Iain H and Joe.

Thanks to everyone at CERN for making my stay so pleasant: Peter, Sophie, Heinz, Kurt again, and especially my office mates James, for his extremely educational ranting, and Gavin, for his equally "educational" musical tastes. Also thanks to the l'Athénée/Servette fellow students: Alex, Christian, James S, James C, Janice, Laura, Leo and Rob.

And of course a big thanks to Chris C-T, Caitriana again and my flatmate Craig for keeping me relatively sane during the last fun-filled months of writing up, always reminding me that I will get there in the end eventually. A final thanks to my parents for always supporting me in everything I do, in good times and bad.

Outline

This thesis covers the issues concerned with data management in a Data Grid environment with a special focus on the problem of replica optimisation. Chapter 1 discusses the history of distributed computing and current efforts in this area along with the data requirements and computing models for high energy physics experiments at the Large Hadron Collider at CERN. The idea of Data Grids and their constituent components are explained in Chapter 2, leading to the definition of a Data Grid used throughout the rest of the thesis. The European DataGrid project is the focus of Chapter 3. The services developed by the project are discussed with particular emphasis on data management services. These data management services are tested in Chapter 4, to evaluate their scalability and stability in a production environment. The issue of optimisation is discussed in Chapter 5 and a solution to replica optimisation based on an economic model for file trading is explained in detail. Chapter 6 describes the architecture of the Grid simulator OptorSim, designed to evaluate the performance of Data Grid optimisation strategies and Chapter 7 presents results obtained using OptorSim to test these strategies in various Grid scenarios. Finally, conclusions are presented in Chapter 8.

A description of work carried out on the ScotGrid website is given in Appendix A, Appendix B contains a glossary of acronyms used in this thesis, and a brief explanation of UML diagrams is provided in Appendix C.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1964 Martin Greenberger wrote an article for The Atlantic entitled "The Computers of Tomorrow" [65], in which he proposed that computing development would follow the same lines as the development of electricity, evolving into a utility that could be used for any kind of computational problem. The electrical power Grid provides a reliable, constant source of power that any device designed with the appropriate wiring can plug into and use, without caring about which power plant generated the power. Similarly, the computational Grid was foreseen to be a utility of processing power providing standard interfaces to which any computer can link to harness its power, with the source of the power completely transparent to the user.

40 years later, this idea is becoming a reality. All over the world computational Grids are being constructed so that distributed resources can be aggregated and shared among different users and organisations. This development has been driven by the scale of many current and future scientific projects in which the amount of data produced and the computational resources required to analyse it are far greater than any previously seen. With such large requirements, it is not possible to use traditional cluster computing or super-computing at a single location to solve these problems, and so scientists are looking to link geographically distributed resources together to form Grids. The power of a Grid can be accessed from anywhere in the world, providing the user has the correct access rights, but software employed to manage the resources and data hide from the user the details of the particular

resources that are used. To the user the Grid looks like one single large computer to which they can simply "plug-in" and the Grid does whatever they want it to do, automatically finding the required resources.

Building a Grid is not easy, however, both in terms of the cost of hardware (CPUs, data storage devices and networks to link resources together) and developing the software to manage the resources (resource scheduling, information services, security, data management etc). The distributed nature of a Grid also brings other problems of a social and political nature, when people from many countries and backgrounds have to agree on standard definitions and protocols for the Grid.

Despite these problems many Grids are approaching the stage of being used as production facilities. In the next few years it will become apparent whether they will remain as specialised infrastructures for scientific experiments or will be taken up by commercial interests and the general public, to become the "next generation Internet".

1.1 A Brief History of Networked Computing

Networked computing has been around for almost 40 years [134] but only really in the last 10 years with the explosion of the World Wide Web has the concept become commonplace in the mind of both the general public and business.

In 1965, the year after Greenberger's article, the first wide-area network was created between two computers (one at MIT's Lincoln Lab and the other at System Development Corporation in California) linked together over a 1200 bps phone line. This experiment was performed by the Advanced Research Projects Agency (ARPA), set up by the US Department of Defence in response to the USSR's launch of the first man-made object into space, Sputnik 1. Over the next 4 years the ARPANET was conceived, designed to form a network of computers all over the USA to deal with any national emergencies. The initial ARPANET was formed in 1969 between 4 computers, the original sketch of which is shown in Figure 1.1.

Throughout the 1970s the ARPANET gradually grew in size, but two important inventions in this decade laid the foundations for today's networks: Ethernet and

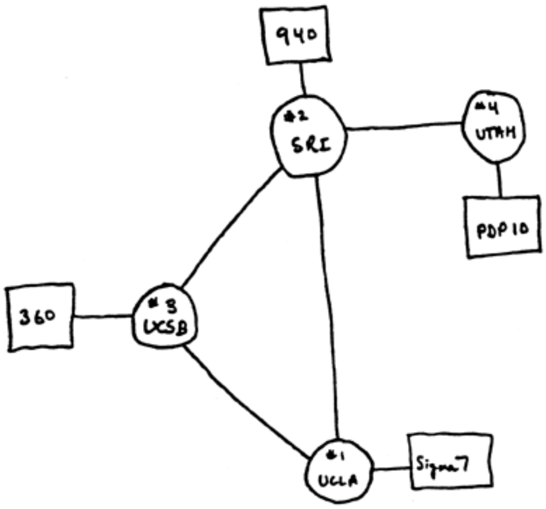


Figure 1.1: The 4 nodes on the original ARPANET: (1) UCLA, (2) SRI, (3) UCSB, (4) UTAH. From [79].

TCP/IP. Ethernet [92] enables multiple computers in a building or campus to be linked together in a Local Area Network (LAN), and thus the ARPANET slowly evolved into a connected set of LANs. TCP/IP [33] (originally just called TCP, but later split into TCP/IP) forms the combined Transmission Control Protocol and Internet Protocol. IP controls the forwarding and addressing of packets of data between hosts and TCP provides a reliable delivery service, controlling packet flow and dealing with lost packets. These two protocols combined gave a reliable way to transmit data between any two points on a network. The move in the early 1980s to using TCP/IP as the standard networking protocols led to the first definition of "an internet" as a connected set of computers communicating using TCP/IP and "the Internet" as connected TCP/IP internets.

The conversion from human-friendly host-names for network nodes (host.gla.ac.uk) to numeric IP addresses (194.25.1.45) was originally performed using one file containing a table of the two forms, passed around all the hosts on the Internet periodically. However as the number of Internet hosts grew into thousands, a more scalable solution was required. The Domain Name System (DNS) [93] was invented in 1984 and is a distributed hierarchical Internet directory service enabling fast resolution of any host-name to IP address. A search for a host-name starts at the nearest name server, then progressively moves higher up the hierarchy until the host is found. This distributed database can almost be thought of as the first metadata service for distributed computing.

Around the early 1990s the first computer clusters were created. The falling prices of PCs and improvements in Ethernet technology meant many computers could be linked together to form a single parallel computer. This was much cheaper than buying a single large mainframe computer with the equivalent processing power.

Until the mid-1990s the Internet was still only used by academic and military organisations for e-mail, newsgroups and file transfer. In 1991 Tim Berners-Lee released the World Wide Web [15], designed as an easy way to navigate through the massive amount of documents produced by CERN, using hypertext. This concept, first conceived in the 1940s [22], although the name "hypertext" was not coined until 20 years later, meant any part of a document, for example a reference to another document, could link straight to that other document, creating a tangled "web" of documents rather than the traditional hierarchical structure used previously. A few years later, with the introduction of sophisticated web browsers and home Internet access, the web really took off.

1.2 Current trends

Several new innovations in networked computing have brought it into the eyes of the public and business as well as academics. Various levels of hype have surrounded all these ideas at some point but each of them is still relevant and they form the cornerstones of Grid computing.

1.2.1 Web services

The World Wide Web Consortium (W3C) [130] was formed in 1994 by Tim Berners-Lee to develop new standards and protocols for information exchange on the web such as the eXtensible Markup Language (XML). XML provides a general infrastructure on which communities can base language to describe their applications, such as Web Services [131], which enable applications running on different platforms or frameworks to communicate with each other using defined means. Web services are now commonplace among industry since the defined standards and relatively small resources required make them easy to build.

1.2.2 Distributed computing

Distributed computing has been given much publicity by the SETI@Home project [5, 115], which harnesses idle computing resources to analyse radio signals from space in the hope of finding messages from intelligent extraterrestrial life. Almost 5 million people have contributed to this project so far. The client software, which acts as a screen-saver when the computer is not being used, fetches new data over the network, analyses it and sends the results back to the central server. Several other projects have been developed using a similar idea, such as Folding@Home [55], which runs protein folding simulations in the hope of understanding diseases like Alzheimer's, and Climate*Prediction*.net [39], which explores the sensitivity of current climate models by running simulations thousands of times with slight changes to the input parameters. This idea has also been used by particle physicists in the Distributed Particle Accelerator Design project [45], which simulates the pion-to-muon decay channel when protons hit the target rod of the RAL Neutrino Factory, in order to optimise the efficiency of this stage of the accelerator. Although particle physicists are looking to Grid computing to solve their data analysis problems, the LHC@home project [88] was started in October 2004 to run detailed simulations of the main LHC beam in order to study the stability of particle orbits around the ring, using the same technology as SETI@Home. These projects rely on the interest of the public and their willingness to give up their spare computing cycles, but for larger-scale projects something more reliable and consistent is required.

1.2.3 Peer-to-peer computing

The Peer-to-peer (P2P) networking model [102] forms a completely decentralised system, where every host, or peer, has equal status to any other peer. The early Internet was a P2P system with no centralised control, but with the growth of the World Wide Web it evolved toward a client/server model, with web browsers on users' home computers acting as clients, connecting to a web server hosting the required web page to download information. P2P systems, however, enable direct communication between two peers and are symmetric, meaning any peer can act as a client or server. The most common types of P2P applications are file sharing and distributed computing and storage applications. Projects such as JXTA [109] are developing P2P protocols to enable P2P applications to operate between anything from PCs to mobile phones.

In 1999 Napster [95] was released, which provided an easy way for users to share music files in MP3 format with other users around the world. When a Napster client started running, it registered itself and the music files the user wanted to share to a central index server. When the user wanted to download a song, they sent a query to this server, which returned a list of other users sharing the file. The user then connected directly to another Napster client to download the song.

The dramatic rise in popularity and media attention given to Napster-type file sharing programs (there are over 100 available to date) has been mainly due to the fact that copyrighted material such and music and films are freely available for anyone with an internet connection. Napster is not P2P in the strictest sense, since it uses a centralised indexing server, but many of the new generation of file sharing applications do not use a central server, rather they communicate directly with other clients. This makes them true P2P applications and also makes it very hard to prosecute copyright violations by users, since no one has overall control of the system. Even though some companies now run legal music downloading services (for example the re-launched Napster and Apple's iTunes [6]), other file sharing programs have found new ways around the law enforcers, such as FreeNet [38]. It uses a completely decentralised network and encrypted communication between nodes, giving anonymity to all users.

1.2.4 Grid computing

The term Grid computing first appeared in the late 1990s [57], the idea being to link together distributed heterogeneous computational resources together to form a single resource. The name Grid comes from analogy to the electrical power Grid, which provides reliable on-demand power to all consumers; likewise the computational Grid provides an on-demand computing utility [37]. This idea had existed before under the name metacomputing [117], where a metacomputer was a network of heterogeneous computing resources transparently available to a user, and the Information Power Grid [86]. However, only now has the level of technology and interest reached a point where Grid computing can be seriously looked at as a solution to computational and data intensive problems. Several groups are working on the construction of Grids, mainly for scientific projects, but industry is becoming increasingly involved in what is seen as the next important phase in the evolution of computing.

The hype around Grid computing has displaced the P2P hype but in reality Grid computing is not about replacing current technologies but harnessing them all together (cluster computing, web services, P2P etc.) to work as one unified utility.

1.3 Physics Experiments at the LHC

At CERN (the European Organization for Nuclear Research) [34] in Geneva, Switzerland, construction is currently underway of the Large Hadron Collider (LHC). The LHC is a giant particle accelerator, consisting of a circular tunnel with a circumference of 27 km, around which beams of protons (and heavy ions such as lead nuclei) will be accelerated in opposite directions. At four points on the ring, the beams of particles cross and collide with each other at extremely high energies to produce many other kinds of particles. Large detectors built around the collision points can measure what happens in each collision and the data can be analysed to discover new physics and confirm (or disprove) existing physics theories.

The experiments building each detector are ALICE [4], ATLAS [7], CMS [40] and LHCb [87], and each is designed to study a different area of particle physics. ATLAS and CMS are "general purpose" detectors looking at into fields ranging from Higgs physics to supersymmetry, ALICE will examine heavy ion collisions to study the strong interaction at extreme energy densities, and LHCb will measure CP violation to very high precision in the b-quark sector.

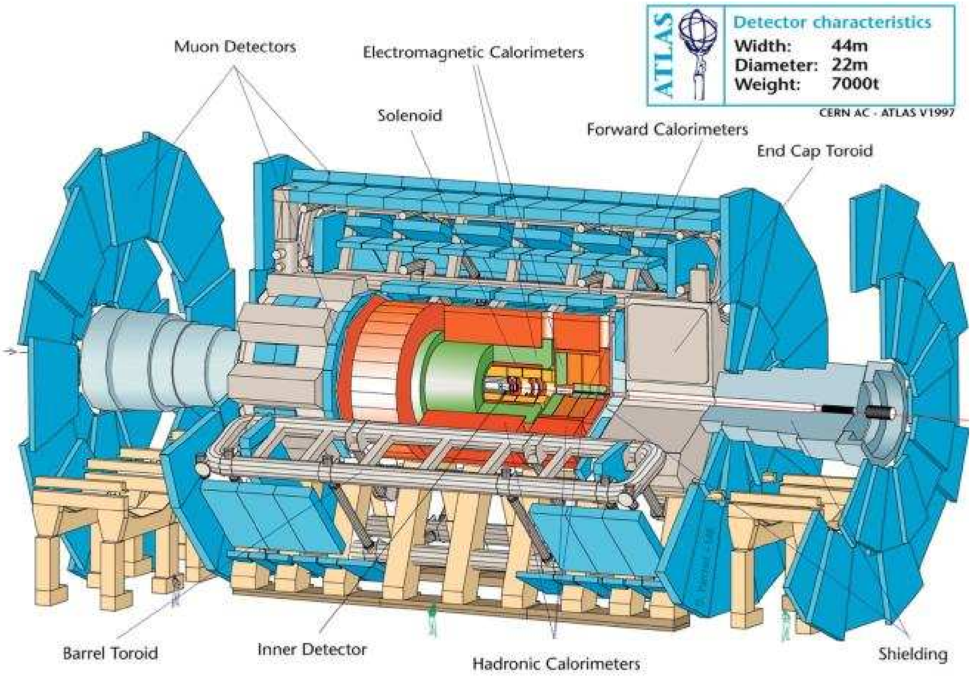


Figure 1.2: Cut-away view of the ATLAS detector.

A diagram of the ATLAS detector showing its component parts is shown in Figure 1.2. The detector has a length of 42 m, a diameter of 22 m and a total weight of 7000 tons. It has many different parts, designed to detect different types of particles and all together they add up to 200 million individual detector channels.

1.3.1 Data Output

In the LHC accelerator bunches of particles cross through each other at a rate of 40 million times per second. Depending on the detector, between 1 and 20 collisions, or events, will occur each time the bunches cross in one of the detectors. These very high event rates are used because the most important phenomena that the physicists want to observe occur with only a very low probability in any single event. Of the millions of events per second, perhaps only 100 may be of special interest and these are selected, using a fast real-time filtering (triggering) system, for storage and later analysis. A first level of filtering uses a subset of the detector elements and looks for basic physics properties of each event, disregarding most of the events before any detailed data processing is performed. Second and third level triggering is performed by massive computational farms and these select which events are useful based on other calculated physics properties interesting to the particular experiment.

The measurements of each event from the detector elements are called raw data and the size of this raw data is between 125 KB and 25 MB per event, depending on the experiment [16]. However, most physics analyses do not look at the raw data directly but use a number of reconstructed data products which summarise the data in some way. Physical quantities such as tracks, energy clusters and particle ID are reconstructed from the raw data and this constitutes Event Summary Data (ESD). The location of vertices, invariant masses etc are determined from the ESD to produce Analysis Object Data (AOD), and key parameters of each unique event are stored as TAG data. Physics analyses tend to use the smaller data products, referring back to the corresponding larger files when necessary. The rough sizes of these sets of summary data for the CMS detector are given in Table 1.1 (other experiments follow roughly the same scale).

Reconstructed Data Product	**Size (MB)**
Raw Data Event	1
Simulated Event	2
Data Event Summary (DST/ESD)	0.5
Analysis Object Data	0.1
TAG Data	0.05

Table 1.1: Sizes of reconstructed data products for CMS. From [119].

Event simulation also plays an important role in the physics analysis process, and this involves simulating the behaviour and output of detector elements as events occur, in order to test the effectiveness of event reconstruction algorithms. The amount of data produced by this process can be as much as or greater than the amount of real data from the detector itself. Furthermore, this data is produced at many different sites around the world and must somehow be available to all users.

Assuming an experiment running time of 100 days per year and 100 events recorded per second, roughly 10^9 raw events will be recorded per year. If each of these have a size of 1 MB, this means a total data size of 1 PB (10^{15} Bytes) per year, just for raw events. Factoring in the reconstructed data products and the simulated data, the four LHC experiments combined are expected to generate on the order of 10 PB of data per year. This scale of data is unprecedented in the high energy physics community, or in any field of science. As a comparison, Run II of the CDF experiment at FNAL which has been going since 2001 is expected to produce around 1 PB of data in its first few years of operation [89]. A comparison of event rates and sizes for several past, current and future high energy physics experiments is shown in Figure 1.3. The event rate corresponds to the number of events output from the detector after the first level trigger.

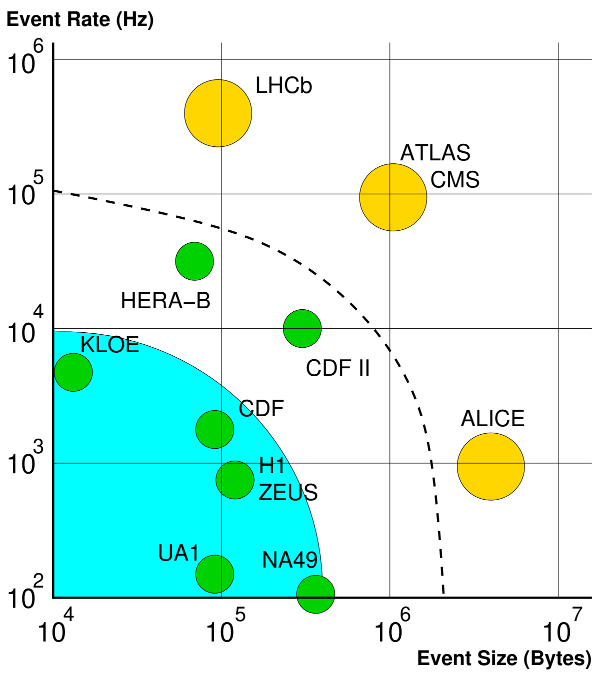


Figure 1.3: Event rates and sizes for high energy physics experiments after the first level trigger. From [46].

The blue shaded quadrant in the bottom left corner covers the scale of data for which traditional computing models can be used, and these models were used for the experiments contained within this region. On this scale all the data can be stored on the site it was produced and all the computational analysis of the data can also be done on-site. The area within the dotted line encompasses experiments which are moving toward distributing data and/or analysis. Outside this area, the four LHC experiments either have large event sizes (ALICE), large event rates (LHCb), or both (ATLAS and CMS) and are opting for a purely Grid solution.

1.3.2 MONARC Model

The MONARC (Models of Networked Analysis at Regional Centres for LHC Experiments) project [94] was started in the late 1990s in order to develop a computing model to handle the needs of these experiments. The final model produced by the project [2] was based on distributed computing following a hierarchical architecture, centred on CERN. A multiple tier system of hardware resources was proposed, composed as follows:

- A central Tier-0 site, hosted by CERN, with around 5,000 users.
- Tier-1 Regional Centres, serving around 1000 users within a large geographic region or a country.
- Tier-2 centres serve part of a geographic region, typically about 200 active users.
- The computing facilities available at different institutes constitute the Tier-3 centres, conceived as relatively small structures connected to a reference Tier-2 Centre.
- Users' personal desktops define Tier-4 centres.

This architecture is depicted in Figure 1.4. For each level down the hierarchy the sizes of resources available and the number of users decreases by roughly a factor of 5. This hierarchical architecture allows the data produced at CERN to flow down the tiers, with different types of data going to different levels, for example ESD to

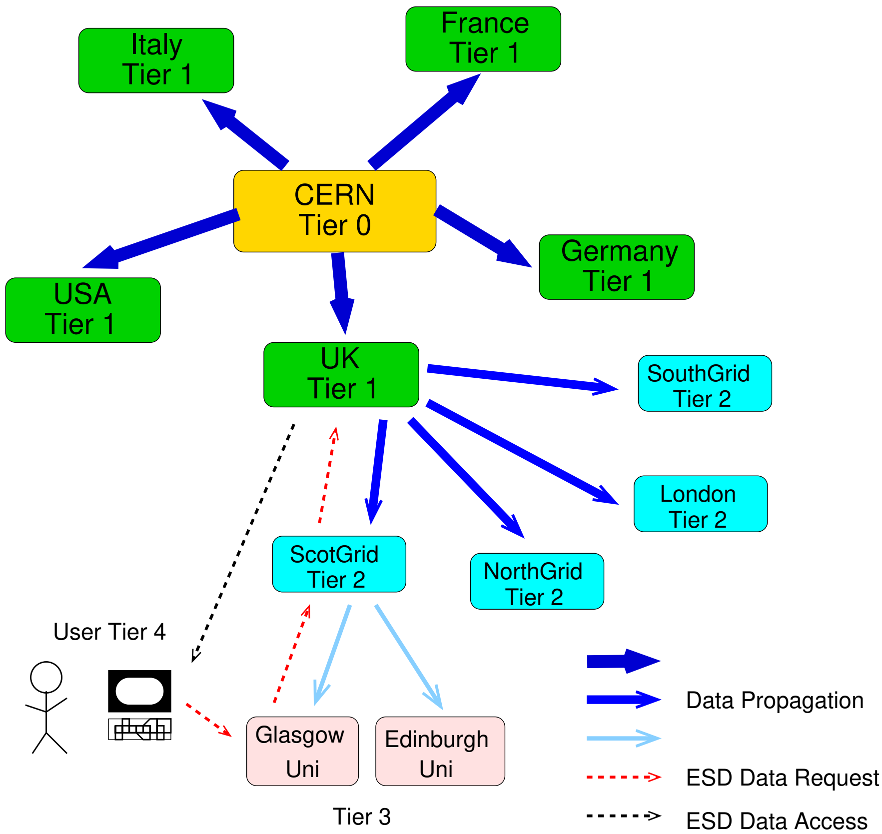


Figure 1.4: Architecture of the MONARC computing Grid model.

Tier-1 and AOD to Tier-2 sites. Requests for data are propagated up the tiers until the data is found. In Figure 1.4 a user request for Event Summary Data ends at the UK Tier-1 site, from which the user can access the data directly.

Physicists are able to analyse the data effectively no matter where they are located and regional centres provide a way to utilise the expertise and resources residing in computing centres throughout the world. It is difficult to concentrate resources (not only hardware but more importantly, personnel and support resources) in a single location, and so a regional centre architecture provides greater total computing resources for the experiments by allowing flexibility in how these resources are configured and located. Also, physicists from all around the world can participate in analysis efforts without physically being at CERN.

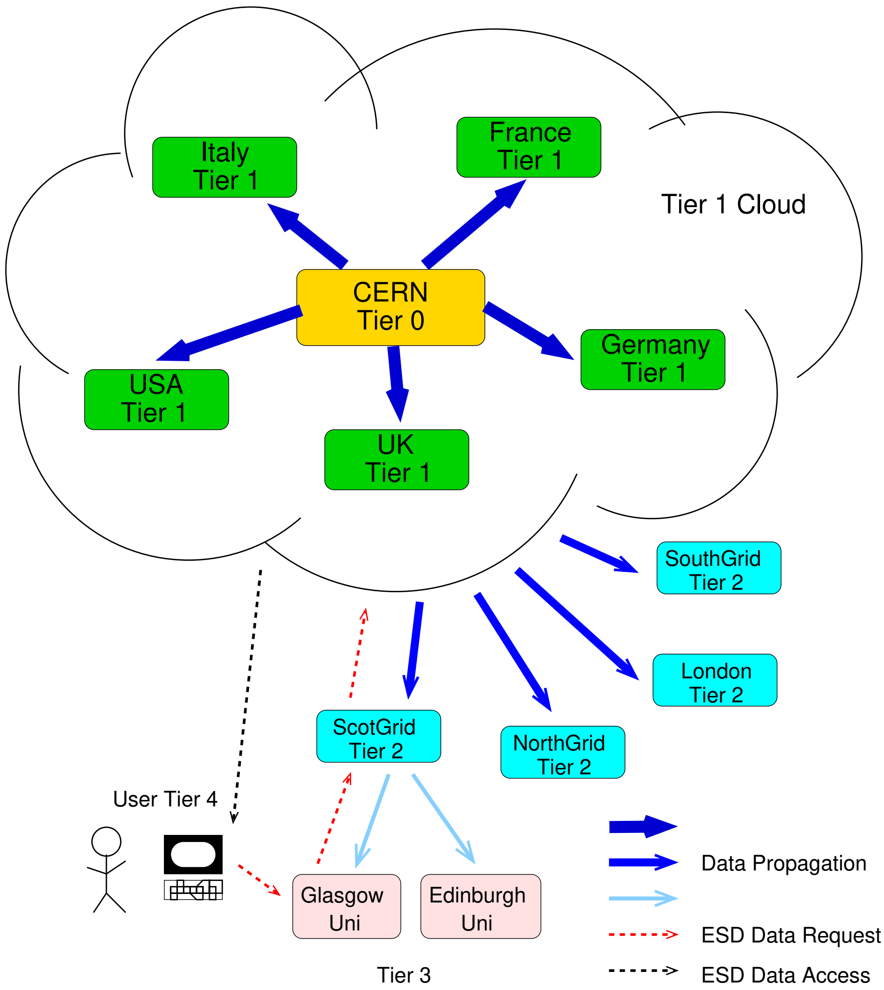


Figure 1.5: Architecture of the MONARC cloud model showing the Tier-1 cloud accessible to all Tier-2 centres.

The original, strictly hierarchical view has now changed slightly so that the Tier-1 centres from a “cloud” from which all Tier-2 centres are linked, as shown in Figure 1.5. This means that Tier-2 centres are not associated with a specific Tier-1 centre and can access the data shared among all of them. Also, most of the simulated data produced at Tier-2 sites can be replicated to a Tier-1 centre and thus be made available to all users.

1.3.3 LHC Computing Grid

To deploy, coordinate and manage the Grid resources required for LHC physics analysis at different centres, the LHC Computing Grid (LCG) project was created. It has the task of preparing the computing infrastructure for the experiments and organising the libraries, tools, frameworks and middleware required to support the experiment applications. It is grouped into 4 areas. The *applications area* develops and manages the applications and services common to all experiments. The *computing fabrics area* maintains the hardware resources. The *grid deployment area* is responsible for packaging and deploying software, monitoring and user support. The *grid technology area* integrates existing Grid technologies into LCG.

Hardware

The hardware resources on each LCG Tier-1 and Tier-2 site today (2nd quarter 2004) and required by 2007 are shown in Tables 1.2 and 1.3[1]. The 2004 figures are based on average values for all the sites listed in [82]. Many Tier-2 sites do not intend to supply tape storage, hence the average tape storage size is very low in Table 1.3.

Hardware	**2004**	**2007**
CPU	300 kSI2k	2000 - 2500 kSI2k
Disk	40 TB	1.4 - 1.8 PB
Tape	180 TB	1 - 1.5 PB/year

Table 1.2: Hardware resources at Tier-1 centres. From [82] and [119].

Hardware	**2004**	**2007**
CPU	90 kSI2k	300 - 500 KSI2k
Disk	10 TB	300 - 500 TB
Tape	0.5 TB	100 - 150 TB/year

Table 1.3: Hardware resources at Tier-2 centres. From [82] and [119].

[1]CPU power units: A standard PC with a 1GHz processor supplies around 0.5 kSI2k.

The target for 2007 is that Tier-2 sites should have roughly one-fifth of the resources of a Tier-1 centre. The current resources almost match this ratio but CPU resources are at roughly 20% of the 2007 target, while disk is only at 2% of the required space in 2007.

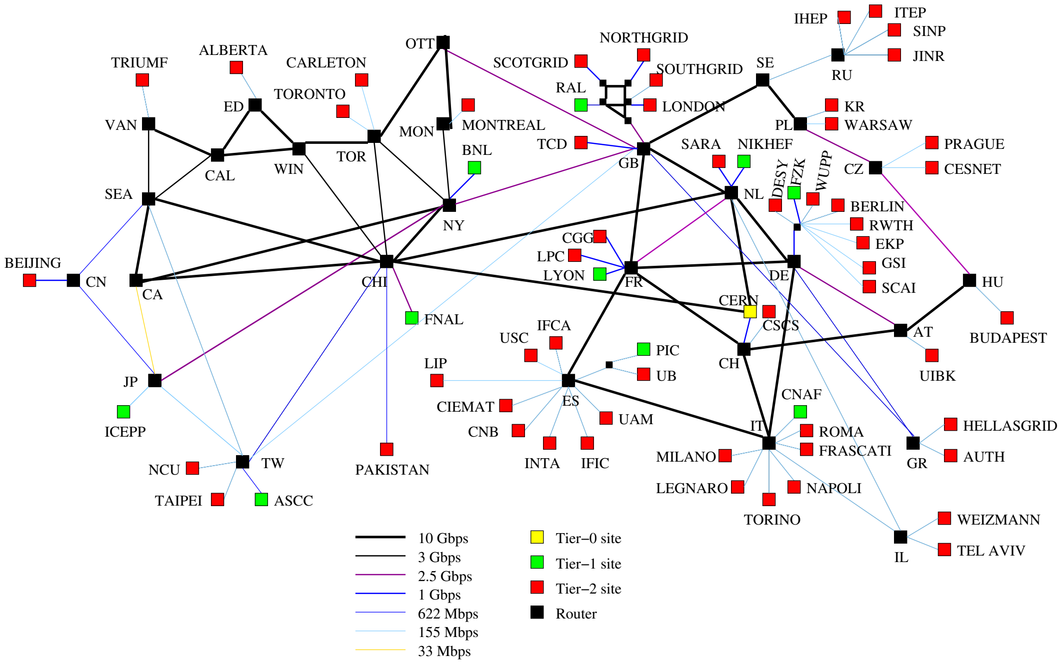


Figure 1.6: LCG testbed status at September 2004. (Figure by C. Nicholson, from [99]).

The status of the LCG testbed in September 2004 is shown in Figure 1.6. There are 10 Tier-1 and 54 Tier-2 sites with combined resources of 5509 kSI2k (from 14020 worker nodes), 681 TB of disk space and 2040 TB of tape space. These figures represent a snapshot in time of the rapidly evolving Grid.

Software

The Applications area of LCG develops and maintains physics applications software and infrastructure that is common among the LHC experiments. This software includes a persistency framework for creating a pool of persistent objects, core libraries and services and simulation software. In terms of Grid software/middleware, LCG does not do any development but brings together existing products. LCG Grid software is based on the Globus Toolkit and Condor/Condor-G (packaged within the Virtual Data Toolkit), which are described in the next chapter, and middleware components from the European DataGrid, which is the focus of Chapter 3.

1.4 Summary

In this chapter the concept of and motivations for Grid computing have been outlined. The evolution of computing over the last 50 years has led to many exciting new technologies such as web services and peer-to-peer computing. Grid computing aims to harness these to provide a computation utility for science and for industry. Similarly, particle physics has evolved substantially over the 50 years since CERN was created and the particular need for Grid computing for the new LHC experiments comes from the unprecedented scale of data they will produce. Only by distributing the computational load globally can physicists cope with the requirements demanded by these experiments.

Chapter 2

What is a Grid?

In this chapter the concept of a computational Grid is explained in detail and various views on the definition of a Grid are discussed, leading to a working definition that will be used for the rest of this thesis. The architecture of a Grid can be split into several layers within which components interact according to certain standards. The evolution of these standards is on-going but is converging toward the common agreements necessary so that different Grid projects can work together.

Many Grid projects are currently under development and the majority are based on the Globus Toolkit, which provides the low-level services necessary to bind resources together into a Grid. The scientific community and in particular those involved in high energy physics have invested a lot of effort into Grid research to tackle their unique computational and data needs.

2.1 An Overview of the Idea of Grids

Computational Grids started out being compared to electrical power Grids, but the range of applications and uses for a computational Grid make it a far more complex system than that simple analogy suggests. Consequently, there is much heated debate over what exactly defines a Grid and to what specifications one should be built.

2.1.1 Analogy with Electrical Power Grid

The electrical power Grid [37] provides a constant reliable source of power for all electrical applications. Forty years ago, in [65], three key differences were made between the electrical power Grid and the computational power Grid which still hold true today. Firstly, to use the resources of the electrical power Grid, all the user has to do is plug in the electrical item (TV, iron, kettle etc.) and throw a switch. Computers on the other hand are very complex - even using a computer which is not connected to the (computational) Grid is not trivial - and thus a standard, easy to use way of accessing the computational Grid resources is required.

All electrical devices connect to the electrical Grid in the same way using standard plugs, and transformers are used to control the voltage required for each application. Similarly, for the computational Grid standard interfaces to use the resources are required so that many different types of applications can make use of its power using standard means. Most electrical applications around today were invented after the means for supplying electricity through a Grid, so the computational power Grid could drive the invention of applications specifically designed to use it, rather than trying to adapt existing applications to fit into the Grid's way of working.

Lastly, using electricity is a one-way process - the user takes what is provided by the power company and there is no feedback in the other direction. But the actions of the computational Grid are driven by the applications - the user tells the Grid what to do - so there has to be two-way communication between the user and provider of the resource.

The main point of these three differences is that a computational Grid is more complex than the electrical power Grid, and the vision of simply plugging in to access worldwide computational resources will take a long time to realise. The problems that Grids must address can be summarised in four key issues:

Heterogeneity. Many computing technologies exist in terms of architecture, operating system etc. and potentially many different applications could make use of the same computational resources. The software running the Grid must be sufficiently adaptable that any system can be easily integrated into the Grid, and any application can easily plug-in to access the resources.

Scalability. It is expected that Grids will grow over time, both in terms of the number and distribution of individual sites and the resource capacity at each site. The resource management components must be designed to cope with a large and varying number of resources and ensure each resource is used to its full potential.

Dynamism. The Grid is a far from static environment. With so many resources outside central control, the reliability of any resource cannot be guaranteed and it is to be expected that resources will come and go periodically. Even if resources fail at critical points in the running of an application, the Grid must be able to seamlessly recover from the failure.

Multiple Administrative Domains. The distributed resources on a Grid are very likely to be owned by different institutes or companies and therefore administered differently according to local policies. The Grid must ensure its resource usage policies fit in with any local policies set by local administrators.

To satisfy all these requirements is a large challenge and many current projects concentrate on fulfilling one of these needs and for example impose limitations on the types of operating systems that can be used on the Grid or do not provide any mechanisms for fault recovery. While these are mainly experimental prototype Grids, in order to be useful and provide a true utility-like Grid all four of these requirements must be satisfied.

2.1.2 Defining Grids

The definition of a Grid has now moved on from the simple analogy with the electrical power Grid. In [60] the idea of *virtual organisations* (VOs) was introduced. These are "dynamic collections of individuals, institutions, and resources" for which the Grid provides an infrastructure to solve their problems. A VO is defined by a set of rules and conditions by which they share the Grid resources and examples include an international team collaborating on the design of a new aircraft, a group using satellite imaging to study climate modelling, or members of an experiment in high energy physics. With such a wide variety of VOs it is important to try to find sets of standards that are acceptable to all and that enable interoperability of the Grid between each of the VOs sharing its resources.

For this to work, it is suggested that standard protocols and services are defined which connect applications to the Grid resources. Protocols define how elements in a system interact with one and other and just as HTTP is the standard protocol for the Web, a similar standard is required for Grid resources. Such a protocol should provide mechanisms to discover, share and give secure access to these resources. A service provides a specific capability and is defined in terms of the protocol it uses to interact with users and other services, and its expected behaviour. Services could for example schedule computational resources to users' jobs or replicate sets of data. In other words, a standard Grid can be thought of as applications provided with standard services which interact via standard protocols. The implementation of each service or protocol and how they are used is dependent on the individual application but the way in which each service is accessed is the same for all.

In [78] the requirements for a production Grid are discussed which leads to the definition of a minimum set of services needed to support such a Grid. These include resource discovery and scheduling, access to computational and data resources, information monitoring, security and system management. Providing these services will make the Grid a common infrastructure for all higher-level, or application-specific services.

A different approach to defining a Grid is given in [97]; rather than focus on how a Grid is constructed in terms of protocols, services etc., a rigorous definition is given of what a Grid should provide, i.e. the functionality of the Grid instead of the components that form it. The Abstract State Machine, which formally models distributed systems mathematically, is used to examine an application executing in a system, and determine whether the system fulfils the criteria of a Grid.

A simple three-point checklist that characterises a Grid is given in [56]. To be truly called a Grid, a system must:

1. Coordinate resources that are not subject to centralised control. The resources on a Grid are owned and managed by different institutes or companies but the Grid must manage access to them and deal with issues of security, payment,

agreements with local policies etc.

2. Use standard, open, general purpose protocols and interfaces. The protocols and interfaces on a Grid must be standardised and accepted by all, extensible, portable and interoperable.

3. Deliver non-trivial qualities of service. The Grid must ensure the required level of resource availability, security, throughput and response time for the users so that the utility of the combined system is significantly greater than that of the sum of its parts.

While these three characteristics can certainly define some Grids, especially scientific Grids, they can be a little too restrictive or too vague for those in industry wishing to exploit Grid technologies [62]. For example, some Grids may need some centralised control to deliver the desired quality of service and requiring open standards is a business model feature rather than a technological aspect of Grid architecture.

In summary, the definition of a Grid is still not entirely agreed upon. With such a new technology and wide variety of uses, it is understandable that there is no overriding definition that fits all cases yet. Here, a working definition is used which reflects the idea of data-intensive computing in the definition of a Data Grid.

A Data Grid provides transparent secure access to data stored across geographically distributed heterogeneous storage resources whilst fulfilling the three criteria described above: de-centralised control of resources, using standard open protocols and delivering non-trivial quality of service.

2.1.3 Grid Components

Whilst the architecture of individual Grid projects may vary, a general collection of components necessary for any Grid can be defined, from which each project implements its own version of each component. The Grid architecture can be split into distinct layers, each with its own functionality and Figure 2.1 shows these different layers and the components necessary within each layer.

The *Grid Fabric* is made up from the heterogeneous hardware resources and local management systems for each resource. Computational processing power could be

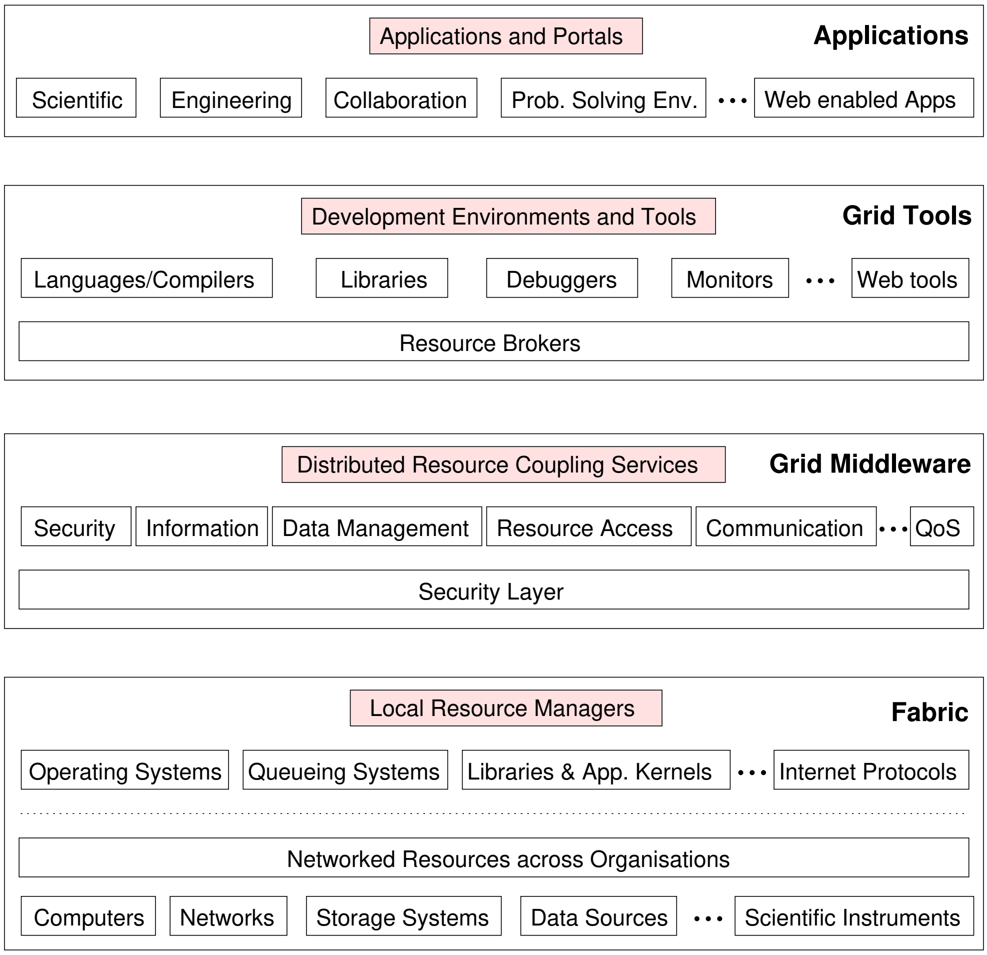


Figure 2.1: The components required to construct a Grid, split into layers containing Applications, Grid Tools, Grid Middleware and Fabric. Adapted from [8].

provided by individual PCs or workstations running a variety of operating systems, or by a cluster running its own internal resource management system such as Condor or PBS. The Grid fabric includes storage facilities, the method of storing data and the data itself. Data can be stored in a database or file system physically located on a disk or tape drive. Instruments used to collect the data are also part of the Grid fabric.

Grid Middleware is what binds the Grid resources together into one unit. Using the information provided by each part of the fabric, it can enable secure efficient access to the data and computational resources. Services in this layer include security services to control access rights, services to provide information on the available resources and data management services to provide access to the distributed data.

Grid Tools provide ways for applications to interface to the Grid. These high level services provide standard languages and libraries that allow applications to be developed which can run on the Grid. Resource brokers control the scheduling needs of these applications, using the information services provided by the middleware layer, and monitoring services feed back information on the use of Grid resources.

Grid Applications are the programs that will run on the Grid to solve the large scale problems of scientists and engineers. They are written specifically for the Grid using the components in the Grid Tools layer and should give the user transparent access to the Grid's resources.

Clearly there are a large number of components necessary to build a Grid and there are many complex interactions between them. The only way that Grids can develop coherently is if they all follow agreed sets of standards - this way interoperability and acceptance by the wider community is made easier.

2.2 Defining Grid Standards

Standards groups exist to provide frameworks and rules around which people develop their own implementations. The more people that agree to follow a standard, the more implementations of it will be produced, hence more people use the implementations, so more people agree to follow the standard and so on. The problem is to gain the initial momentum and critical mass necessary before the standard becomes widely enough accepted that it truly is a standard.

2.2.1 Internet Standards Bodies

Internet standards bodies have been around for years and have helped to standardise the way that applications integrate into and use the Internet. Groups such as the Organization for the Advancement of Structured Information Standards (OASIS) [103], which tries to drive the adoption of e-business standards, and the Distributed Management Task Force (DMTF) [44], an industry organisation leading the development of management standards, and the Internet Engineering Task Force (IETF) [73] have all provided defined means by which people can exploit the Internet.

The World Wide Web Consortium (W3C) [130] creates standards for the Web, by developing specifications, guidelines, software and tools for getting the most out of the Web. One of their most popular standards is the eXtensible Markup Language (XML), a flexible text-based description language. It was originally designed for describing documents but grew to describe any kind of data and is now commonly used for message exchange over the Web by applications such as Web Services. It is this kind of open, versatile and easy-to-use standard that it is hoped Grid standards bodies will be able to develop.

2.2.2 The Global Grid Forum

Because so many diverse Grids are evolving, it was important to start early to define Grid standards. The Grid Forum was created in 1999, became the Global Grid Forum (GGF) [32, 63] in 2000, and bases its structure on the IETF. It is a community of researchers and practitioners from academia and industry and its aim is to promote the development of Grid technologies via the creation of technical specifications and guidelines. It is organised into working groups and research groups which investigate a range of topics related to distributed Grid systems and produce "best practises" and recommendations for the implementation of Grid software. Working groups are short lived (up to 2 years) and focus on a particular problem for which they generally propose technical specifications or recommendations documents. Research groups last indefinitely and address longer-term problems which cannot be solved in a short period of time. They produce mainly informational documents but may "spawn" a working group to look at some specific part of the problem.

The large range of areas covered by the GGF include architecture, scheduling and resource management, security, applications and programming environments, information systems and data management. The 28 working and 27 research groups (as at September 2004) in these areas cover many aspects of Grids and have produced many standards which are used in Grid projects around the world.

2.2.3 Open Grid Services

The most well-known standards currently are the Open Grid Services Architecture/Infrastructure (OGSA/OGSI) standards [58]. The OGSA working group defined the motivation and requirements for the standard [59] and the OGSI working group focused on its technical implementations [125]. The basic idea of OGSA is that everything on the Grid (computational and storage resources, applications, programs, databases etc.) is represented by a service - an entity that provides some capability through the exchange of messages. Taking a service-oriented view is one solution to the requirement for interoperability in a Grid environment, since it allows standard interface definitions for services behind which can lie a range of different implementations of the services. This enables consistent transparent resource access across multiple distributed platforms on the Grid.

These *Grid Services* are defined as Web services that provide a set of well-defined interfaces that follow specific conventions. The Web services framework provides efficient mechanisms for registration and discovery of services by defining interface definitions separately from their particular implementation. The Grid service implements one or more interfaces which define some set of operations which can be invoked through messages. The OGSA framework also provides Grid services to handle other Grid services, such as factory services for creation and services to manage the lifetime of transient services. The service definitions provided by OGSA are rather low level and are intended to be used as building blocks for higher-level services like data management or monitoring services. But the flexibility of the framework means that these services can be implemented in any number of different ways, while still exposing the standard interfaces which enable interoperability between them all.

2.2.4 Web Services Resource Framework

Whilst addressing the complete range of aspects, the scope of OGSA was thought to be too broad, and developments in Web Services technology, particularly WS-Addressing, which provides a transport-neutral way to address Web Services, have superseded a lot of the OGSA specification. As a response to this, in early 2004 the

Globus Alliance and IBM released the specifications for the Web Services Resource Framework (WS-RF) [41], which attempts to bridge the gap between Web and Grid services. The WS-RF approach is concerned with stateful resources in a Web service context, that is Web services that have the ability to access and manipulate state (data values that persist across and change due to Web service interactions).

WS-RF defines conventions that enable the discovery of and interaction with such stateful resources as might be found on a Grid. The technical specifications within WS-RF address resource lifetime, resource properties, stale resource information, heterogeneous Web service collections and resource fault mechanisms. WS-RF has been termed a "refactoring" of the OGSI concepts to exploit the recent Web service developments which have made some parts of OGSI redundant. The greater reliance on accepted Web services standards rather than emerging Grid standards also provides a simpler way for developers to integrate their applications into the Grid.

2.2.5 The Enterprise Grid Alliance

The Enterprise Grid Alliance [50] is an industry-led consortium aimed at developing Grids for commercial enterprises. Like the GGF, they encourage the adoption of technical standards and technologies for enterprise Grids. By increasing awareness of Grid computing to business, they also hope to accelerate the move of industry into Grid computing, and defining clear open standards is part of this.

2.3 Past and Current Grid Projects

There are a large number of Grid projects worldwide working on different aspects of Grid construction. Some are creating applications, some the Grid middleware to run the Grid, some are building the physical resources necessary and some are combining all three in one project.

2.3.1 Grid Middleware and Tools

The main focus of this section will be on the tools that the European DataGrid (the subject of Chapter 3) is built on, namely Globus and Condor/Condor-G, but other historical and current developments will also be mentioned.

Globus Toolkit

The Globus Alliance [64] is conducting research into and developing Grid technologies. They assist in building large-scale Grid testbeds and develop Grid-enabled applications, but their main product is the open-source Globus Toolkit. It includes software and libraries for resource monitoring, discovery and management, security and file management and it is packaged as a set of components which Grid developers can use to build applications. Through working with the GGF, the Globus Alliance aims to provide the standard set of services and protocols which allows seamless access to Grid resources.

The European DataGrid is based on version 2 of the toolkit, released in 2002, which provided an implementation of all the basic services required to build the core elements of a Grid. Version 3 of the toolkit was the first full-scale implementation of OGSA and version 4 will follow the Web Services Resource Framework which has overtaken OGSA as the standard for Grid services interactions. The architecture of the Globus Toolkit can be split into several service layers, illustrated in Figure 2.2.

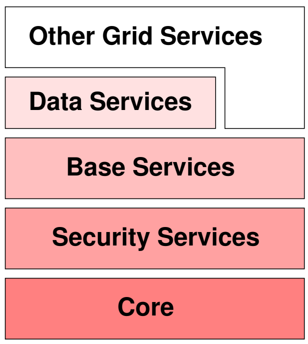


Figure 2.2: The Grid services layers in the Globus Toolkit.

Starting from the bottom, the Core services are the basic infrastructure needed to build any other Grid services. The Core contains system-level services to interact with the services hosting environment (a web service engine such as Tomcat) and management services to activate and deactivate service instances. Security services which use the Grid Security Infrastructure control access rights to Grid services for individual users and collective groups of users. The base services layer contains many services for managing Grid jobs. It provides the Globus Resource Allocation Manager (GRAM) service to monitor and control jobs, an index service to provide information on available resources and the GridFTP file transfer service to move data around. Data services help keep track of data on the Grid using a Replica Location Service (RLS)[1] and allow different sets of data to be collected into subsets for a job to work on. Other Grid services include higher level job handling and reporting services interface between these layers of Grid services and the user applications.

Condor-G

The Condor project [90] develops mechanisms to support high throughput computing, where high performance computing capabilities are required for long periods of time. Condor provides a full workload management system including a job queueing mechanism, scheduling policy and job priority schemes, resource monitoring and resource management for clusters of networked computers. It can be used with any groups of computers, even those linked together differently from traditional clusters, to harness CPU power from unused resources. For example, a cluster for students in a university may be busy during the day but idle at night, hence Condor can harness these unused CPU cycles for computationally-intensive jobs.

The Condor-G system [61] combines features of Condor and Globus to provide a full Grid resource management system. The local cluster management capabilities of Condor are used in conjunction with the ability of Globus to deal with security and resource discovery and access in a multi-domain environment. This means that, to the user, the resources spread across multiple domains appear as if they are all contained within a single domain and Condor-G handles all aspects of job management, resource selection, security and fault tolerance.

[1] Both Globus and the European DataGrid project have their own implementations of the RLS framework which they co-developed. The EDG RLS is described in detail in the next chapter.

Others

Legion [69] was one of the first projects to look at metacomputing (an early term for Grid computing) and aimed to enable a metasystem of heterogeneous, distributed, high-performance computers to interact with each other to form a world-wide virtual computer. Legion is organised into classes and metaclasses (classes of classes). All software and hardware components are objects and they respond to method invocations from other Legion objects. APIs are defined for interaction between objects, but not the language in which they are implemented or a communication protocol. Each object has its own class object which can create new instances and provide state information about the object. Since the system is object-oriented users can override the functionality of a class with their own and this design means functionality and components can be added or removed as required.

The Grid Economy project [23] is conducting research into developing economy-based resource management and scheduling systems for Grids. The architecture is based on the Grid Architecture for Computational Economy (GRACE) which contains a Resource Broker, protocols and a manager for resource trading and an information system provided by Globus MDS [54]. The Nimrod/G Resource Broker [24] was developed as part of this project. It uses an economic model for resource scheduling whereby the user or application specifies a desired quality of service with certain deadline or budget constraints, then the resource broker finds the appropriate resource to match. Lower level services from for example Globus provide resource discovery mechanisms and the GRACE services are used to trade between the resource owners and resource broker.

The NetSolve system [31] enables users to access computational resources distributed around a network (local or wide area) to run jobs remotely. It employs a load-balancing policy to make sure the available resources are used as efficiently as possible and is based on a client-server approach. A request to use a remote resource is sent from a NetSolve client to a NetSolve agent, which chooses the best resource to solve the problem. It was mainly intended for solving computationally-intense science problems; therefore using the C, Fortran or MATLAB clients the user can ask

NetSolve to solve a particular equation. NetSolve can use any remote host with a scientific linear algebra package installed to solve the equation and returns the result to the user.

PUNCH [81] is a platform for Internet computing accessible via any Web browser. It dynamically assembles virtual systems at run-time, given the requirements of the job, to give the user a virtual desktop on which to run their application. After verifying the user is allowed to run the application, PUNCH searches for an appropriate machine for the job. It uses a virtual file system service to mount the disks with the required data and application code on to the machine. It then starts the machine running the application and routes the display back to the user's terminal.

2.3.2 Grids for Science

The first Grids were conceived by computing science, but physical sciences provided the need for them. The scale of scientific experiments has grown so fast that traditional methods of computing used to solve problems are inadequate. In high energy physics in particular, the size in terms of the data produced by the experiments and the number of institutes and researchers involved in each experiment brought the need for distributed computing, and now drives Grid development. Hence the first prototype Grids are being developed by scientific projects and the success of these projects will be regarded as a benchmark of the potential of Grid computing for other areas of life.

The Information Power Grid

NASA is developing the Information Power Grid (IPG) [96] to increase their ability to solve their large data and computationally intensive problems. They are creating an infrastructure to integrate the resources at geographically distributed NASA centres to provide a common resource, while still allowing local control over these resources. The Globus Toolkit is being used as the basis for the software running the IPG and on top of this NASA will build the facilities for constructing collaborative problem-solving environments.

NEESgrid

The Network for Earthquake Engineering Simulation Grid (NEESgrid) [98] is being set up to link together earthquake researchers and their distributed instruments and computational resources. NEESgrid provides a shared infrastructure to help the scientists build more complex and accurate models and simulations of the results of earthquakes on people and property.

Earth System Grid

The latest high-resolution long-duration simulations performed by climate researchers produce tens of petabytes of data. The Earth System Grid (ESG) [47] will make this data available to teams of distributed researchers so that the data sets become community resources easily available to all the ESG collaborators. The ESG integrates and extends a range of Grid technologies including the Globus Toolkit for resource discovery, access and authentication.

2.3.3 Grids for Particle Physics

High energy particle physics has a unique need for Grid computing. The amount of data produced by the next generation of experiments is far greater than can be handled by traditional computing models. Several loosely-connected projects are aiming to solve the computational requirements of the high energy physics community.

GridPP

GridPP [66] is the UK Grid for Particle Physics. It aims to build a Grid from resources at UK institutes for analysis of physics data from experiments in the LHC at CERN (see Section 1.3) and others. GridPP is helping to build applications for physics analysis using the Grid, contributing to middleware development, and providing the hardware infrastructure at the different institutes.

GriPhyN

The Grid Physics Network (GriPhyN) [67] is mainly concerned with providing access to *Virtual Data* for US physicists working on two of the LHC experiments (CMS and ATLAS), the gravitational wave experiment LIGO and the Sloan Digital Sky Survey. Virtual Data means that the community of researchers has access to a very large virtual space of data created from the real experimental data which is geographically distributed. GriPhyN is creating a Virtual Data Toolkit to enable this seamless access to distributed data.

PPDG

The Particle Physics Data Grid (PPDG) [106] is a collaboration of several American laboratories and universities creating production Grid systems for particle physics analysis from several experiments. It brings together several existing Grid tools such as Condor and the Globus Toolkit and uses them to build applications for the US particle physics community.

Grid3

Grid3 [68] worked with GriPhyN and PPDG to deploy and manage a Grid spread over 25 sites in the US and South Korea. Various applications from physics, astrophysics, biology and astronomy are using this facility for analysis of large-scale scientific problems.

Open Science Grid

The Open Science Grid (OSG) [101] aims to link together the infrastructure and the applications provided by the above three projects to provide a framework for all scientific Grid efforts in the US. Its main task is to provide coordination and a forum for reaching common decisions among the many individual organisations and institutes involved.

SAM-GRID

The Sequential Access to Metadata (SAM) project [9] has developed a distributed data management system for the data produced by physics experiments at FNAL. SAM aims to transparently deliver and manage data caches on heterogeneous systems or "stations" where SAM is deployed, bringing together the necessary processing and data resources for the physics analysis job. SAM-GRID [10] builds on SAM to provide Grid job submission, management and information services. SAM takes care of data management while Condor/Condor-G and GRAM handle the resource brokering and job submission.

LCG

The LHC Computing Grid (LCG) was discussed previously in Section 1.3.3. Its main task is the deployment of existing Grid middleware in a consistent package, creating a Grid for the analysis of data from the LHC experiments. It uses the Globus Toolkit and Condor-G along with services developed by the European DataGrid.

EDG

The European DataGrid (EDG) project [52] was set up to enable access to geographically distributed computing resources for three data-intensive applications: high energy physics, biological and medical image processing and Earth observation science. It has developed middleware services based on the Globus Toolkit but enhanced to reflect the data-led approach required by the applications and the hierarchical MONARC model for high energy physics experiments described in Section 1.3. EDG is the focus of the next chapter and will be described in more detail there.

EGEE

The Enabling Grids for E-science in Europe (EGEE) project [49] is a continuation of many of the concepts formed in EDG and aims to create a 24 hour production Grid for scientists around Europe. It mainly develops and supports middleware which is deployed on Grids by projects such as the LCG.

2.3.4 Commercial Grids

Many companies are now offering Grid solutions to the computation and data handling needs of industry. Platform Computing offer a commercial distribution of the Globus Toolkit which in addition contains their own Community Scheduler Framework for job scheduling. IBM's Grid Toolbox is also based on the Globus Toolkit, providing a set of OGSA-compliant Grid services integrated into their WebSphere Application Server that businesses can use to Grid-enable their applications.

Avaki, a commercial spin-off from the Legion project, focuses on simplifying access to data from multiple heterogeneous sources. A data service layer lies between any number of data sources and any number of data applications and for the applications it appears as if there is one virtual data source. Entropia's DCGrid is similar to Condor, harnessing unused cycles of PCs on a company's network to run computationally intensive programs. It provides mechanisms for job prioritising, monitoring and rescheduling if the resource becomes unavailable or starts being used again.

Whilst there are several companies offering Grid software, the scale of investment from industry into the hardware infrastructure is small compared to that of academic research or government funded Grids. The main reasons are that the technology to run Grids has yet to prove itself reliable or cheap enough to develop to justify this scale of investment from industry. The academic community will first prove whether Grid technology is useful, then commercial enterprises will show whether it is profitable.

2.4 Summary

Finding an answer to the question "What is a Grid?" on which everyone agrees has been shown to be very difficult. Saying that it is just like the electrical power Grid is too simple an answer to a complex problem. However, with the help of agreed upon standards discussed among different interests in forums such as the Global Grid Forum consensus can emerge on the way certain things should happen within a Grid. Architecture standards such as OGSA give some basis for everyone to work from, but are unrestrictive enough in their implementation that any "Grid" project can easily fit around them.

Due to the scale of many Grid projects, these standards take a long time to adopt, and for many businesses, the idea of using Grid computing still seems a long way in the future. So whether the current standards such as OGSA or WS-RF will be the standards of tomorrow remains to be seen. It can only be assumed that if and when Grid computing takes off, it will be obvious what the best (and most profitable) way is to do it and then using it will eventually be as simple and transparent as using the electrical Grid.

Chapter 3

The European DataGrid

The European DataGrid (EDG) [52] project was funded by the European Union to provide to scientific researchers the means to access geographically distributed computational and storage facilities. It was driven by the huge data handling needs of three fields of science: high energy physics, biological and medical image processing, and Earth observation science, and ran from January 2001 to March 2004.

The scale of the data produced by the latest experiments in these areas means that it is not feasible to store it or perform analysis on it at a single site. EDG has enabled resources distributed over multiple sites to be linked together and accessed in a uniform, transparent way by developing Grid middleware for the purposes of secure resource access and discovery, data management and information monitoring among multiple heterogeneous resources.

EDG was split into Work Packages (WPs) to cover the different areas of development. A WP existed for each of workload management, data management, information services, fabric management, mass storage management, network services and testbed management. There were three WPs to represent each of the applications and a project management WP. The Architecture Task Force and Quality Assurance Group, responsible for overall architecture and design and quality control respectively, both contained a member from each technical WP.

EDG contributed heavily to defining standards in the Global Grid Forum and there were many collaborations between members of EDG and other Grid projects.

There were also many publications on middleware and other Grid topics written by EDG members. This chapter describes the components developed by each of the Work Packages and in particular the data management services from Work Package 2.

3.1 Middleware Development

The architecture of EDG [48] and the remit of each middleware WP can be illustrated by the Grid Middleware section of Figure 2.1, along with some pieces of the surrounding Grid Tools and Fabric sections. The relevant architecture area is shown again in Figure 3.1.

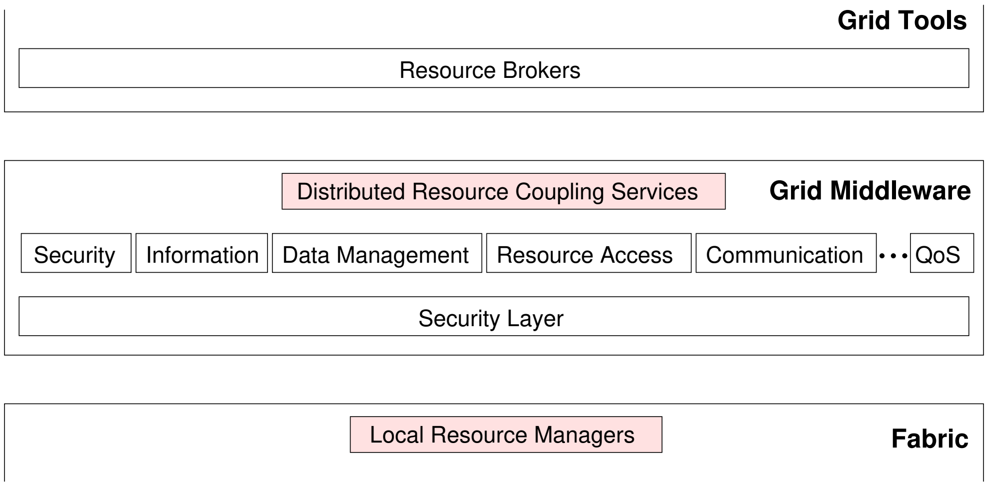


Figure 3.1: The area of Grid architecture covered by EDG.

The components developed by EDG to implement this architecture and their interactions are shown in Figure 3.2. The next sections describe each of these components in detail.

3.1.1 Computing Element

A Computing Element (CE) represents computing resources. It has the task of linking computing nodes in the fabric into the Grid in order that the middleware can gain knowledge of these resources and use them for Grid jobs. The CE consists of a Gatekeeper through which all communication between Grid and non-Grid components

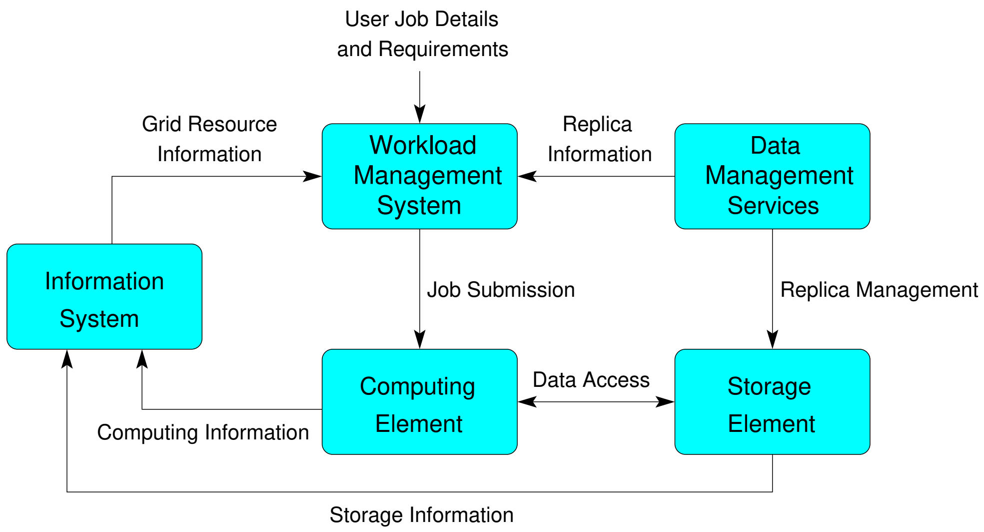


Figure 3.2: Grid middleware components developed by EDG and their interactions.

must pass and a Job Manager, which interfaces to the local resource management system (RMS) on the site. This is illustrated in Figure 3.3.

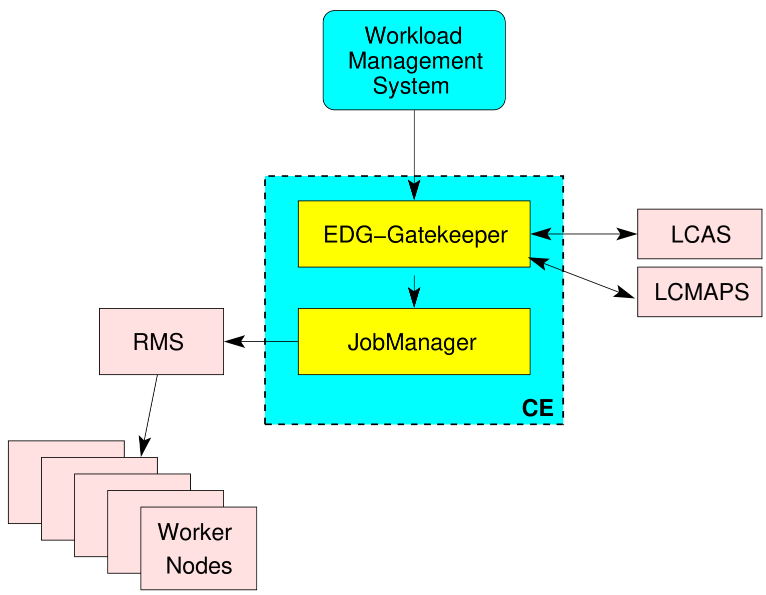


Figure 3.3: The Computing Element and its relationships with Grid and non-Grid components.

Grid jobs which must run on a computing resource are submitted by the Workload

Management System to the Gatekeeper, which first checks whether the owner of the job is authorised to use the resource according to the locally defined policy stored in the Local Centre Authorization Service (LCAS). This policy may specify a fair-share policy so that certain Virtual Organisations (VOs) are alloted certain percentages of resource usage or keep a list of users who are banned for abuse of Grid resources. If the job is allowed according to the policy, the job owner is mapped to a local account using the Local Credential Mapping Service (LCMAPS) so the job can run on the local system and a Job Manager is started to handle the local execution.

The physical computational resources may be clusters managed by a local batch system such as PBS or Condor, or stand-alone PCs. The Job Manager enables the Grid job to be executed on any of these systems so that the configuration of a site's resources is completely transparent to the Grid middleware and the user. It handles the submission of the job to the local Resource Management System (RMS), provides monitoring information during the lifetime of the job and information on any output files produced.

3.1.2 Workload Management System

The application jobs running on the Grid have differing requirements on the resources they use, for example there may be a minimum memory or CPU clock speed requirement for the job. The Workload Management System (WMS) has the responsibility of matching resources to requirements in such a way that the resources are used efficiently but usage limits are not exceeded. It also submits and monitors jobs via interactions with the CE and maintains a persistent database of job information to enable recovery in the case of job failure.

The WMS interacts with several other Grid middleware components as shown in Figure 3.2: the Information System to obtain information on the Grid resources available and their characteristics, the Data Management Services to learn information on the location of data required for the job, the CE to submit and monitor jobs and security services for user authentication and authorisation. To a user, however this is all transparent. They supply the description of the job and its requirements in Job Description Language (JDL) to the user interface component of the WMS and the interactions between other services are dealt with by the WMS itself.

The Resource Broker (RB) is the "intelligent" part of the WMS. It is responsible for optimising the use of Grid resources through efficient match-making between jobs and resources. It gathers information on all the possible resources that satisfy the JDL requirements and ranks them according to for example the time to access the required data or any number of scheduling strategies that are easily interchangeable within the RB. Once match-making has been completed, job management is done using Condor-G [61] which forwards incoming requests to the Gatekeeper of the chosen CE.

3.1.3 Storage Element

The Storage Element (SE) provides transparent access to data stored in several types of mass storage systems using a uniform interface based on the Storage Resource Manager (SRM) model [116]. It can be thought of as a layer between the Grid middleware services and storage, but does not store files itself. In the SRM model the control interface for file creation, access and copying is separate from the data transfer which is carried out via separate protocols. Given a request for a data file, the SE caches the file and returns a Transport URL (TURL) which can be used to access the file, according to the specified data transfer protocol, such as GridFTP.

The SE is not an exact implementation of the SRM interface, but it performs most of the core tasks and can provide extra functionality that is easy to implement since it is not reliant on the SRM standard. The first SE interface used a simple XML messaging system but as Web Services became widely used by other data management components, a Web Service implementation of the SE interface was created. The final SE release enabled file access from and adding files into mass storage and third-party file transfer between two SEs.

3.1.4 Information System

The information system developed by EDG is the Relational Grid Monitoring Architecture (R-GMA) which acts as both an information and monitoring service. Grid middleware services can supply information about themselves and gain information about other services through R-GMA. R-GMA is based on the Grid Monitoring Architecture (GMA) defined by the GGF, shown in Figure 3.4. In this model, Producers of information make themselves known to a Registry. Consumers of information send a query to the Registry, which finds suitable Producers to answer the query. The relational version implemented in R-GMA means global queries can be made over all published information.

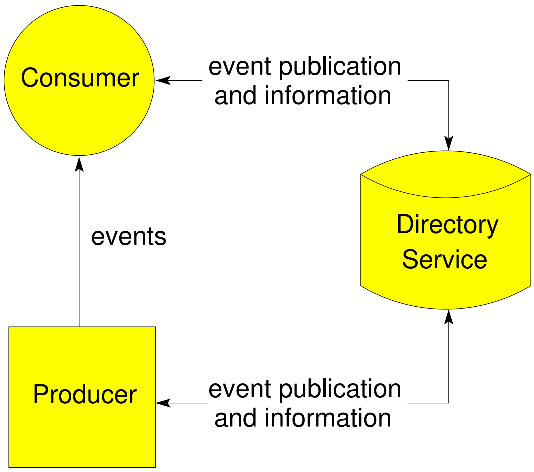


Figure 3.4: The Grid Monitoring Architecture implemented in R-GMA.

There are two means of information update between Consumers and Producers: in the pull model the Consumer regularly queries the Producer for the latest information which may be the same or different from the previous query. In the push model the Consumer asks that it be updated whenever the information that it is interested in from the Producer changes.

3.1.5 Network Monitoring

Network monitoring is used to provide information on how network resources are being used to Grid users, other middleware services and network administrators. The network monitoring architecture can be divided into measurement, collection and visualisation/processing layers, as shown in Figure 3.5.

The measurement layer consists of a set of tools developed in and external to

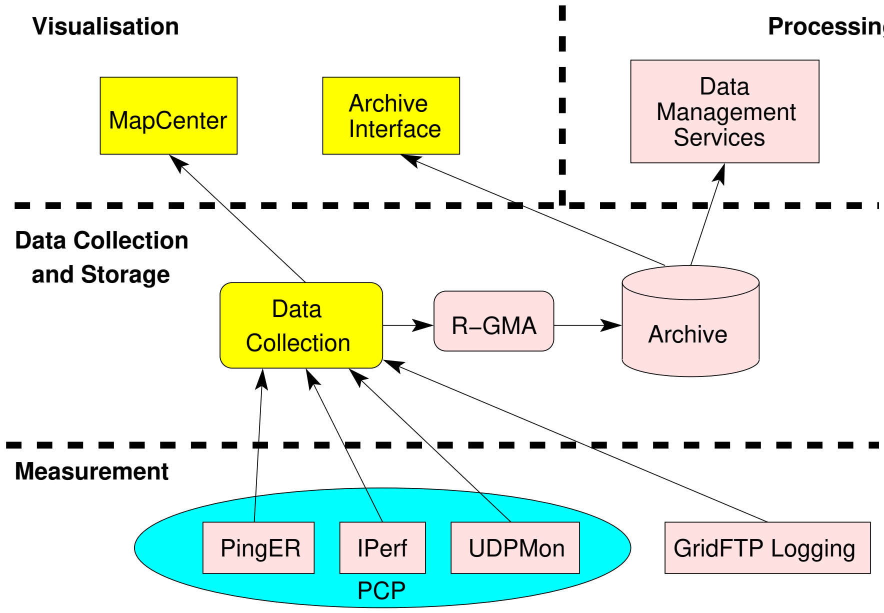


Figure 3.5: Architecture of the EDG network monitoring services.

EDG for measuring various aspects of network usage. Intrusive tools such as Iperf, UDPMon and PingER measure available bandwidth throughput and round trip time between sites for TCP and UDP network protocols. A Probes Coordination Protocol (PCP) is used to coordinate the scheduled use of the different tools so that their results do not interfere with each other. Passive performance statistics are also collected from GridFTP transfer logs.

The raw data collected in the measurement layer is stored and made available to other Grid services in the processing layer using R-GMA. Within R-GMA this information is mapped to CEs and SEs so that network information can be used by data management services to predict data transfer times between CEs and SEs.

The data is presented to users and administrators in the visualisation layer. A web browser can be used to view the status of the testbed using the MapCenter tool and the archive of measurements using a graphical interface. This interface can display a range of information including a matrix of site to site throughput or total Grid traffic per day. Such information could be used, for example, to tune parameters of GridFTP such as the number of parallel streams used.

An important service provided by network monitoring is the estimate of "network costs" for file transfers between two sites, i.e. the time to copy a file between two locations based on the throughput observations between the sites. This is used by the optimisation component of the data management services (described in detail below) which uses these cost estimates to choose the best file to access from a set of replicas.

3.1.6 Security

On a Grid, security mechanisms are required in order to control which resources users are allowed to use. There are three parts to the security systems developed by EDG: the authentication and delegation framework, global authorisation management and local authorisation management.

Authentication is how users and services prove their identity and is based on Globus' Grid Security Infrastructure (GSI), which uses public key cryptography. It provides encrypted secure communication between Grid services, and enables a "single sign-on" for users so that their credentials need only be verified once to use a large pool of resources, instead of every time they use a different resource or service. Users are identified via certificates which are issued and signed by a Certificate Authority (CA), similar to a passport system. If a service trusts the CA it can trust the user whose certificate is signed by the CA and give them the appropriate access rights. Single sign-on is achieved by the user creating a limited lifetime passphraseless proxy certificate to act on their behalf. This means the user can be authenticated to multiple resources without typing in a password every time. In addition a client application may delegate its credentials to a remote process so that it can run on the client's behalf.

Authorisation defines the rules by which users are able to access each resource. Coarse-grained authorisation means the system can make decisions such as: 'does the user have access rights to this storage resource?'. Fine-grained authorisation, which can make decisions such as: 'what kind of access does the user have on this file?', can only be handled inside the service because the actual file to access is only known during the execution of the service. In EDG only coarse-grained authorisation has been addressed.

In EDG every user is part of a Virtual Organisation (VO) whose members are for example all involved in one of the high energy physics experiments at CERN. Global authorisation decisions are made at the VO level so if a university is only involved in the ATLAS experiment, it may only allow users in the ATLAS VO to use its resources. The Virtual Organisation Membership Service (VOMS) was developed for this purpose and essentially consists of a database of users and their associated VOs. A user contacts the VOMS service to request authorisation with a particular VO for some purpose. If the user is a member of the VO they are granted a VOMS proxy certificate in a similar style to the GSI proxy certificates described above. The VOMS proxy certificate is then used to authorise the user to use each local system. The resource provider chooses the appropriate level of access for the VO and the VO itself grants access controls to each individual user. This means the resource provider only has to worry about access controls for a few VOs which will not change rapidly over time and not hundreds of users who may change frequently.

The goal of local authorisation is to map the grid user to a local user account to run their jobs on a local system. This is done using the LCAS and LCMAPS components shown in Figure 3.3. LCAS adds access controls to the EDG Gatekeeper and could contain a list of banned users or specify limits on job parameters such as a maximum wall-clock time to control which jobs are run on the CE. LCMAPS grants the local credentials to the user, for example a Unix account name and group. Authorisation for Java Web Services is done as part of the edg-java-security package, which is described as part of the next section.

3.2 Data Management

In any Data Grid, data is the most important resource, hence it must be managed effectively and efficiently. A key concept in Data Grids is replication of data, whereby multiple copies of data are stored at different geographical locations, making access to data faster and more reliable for all users. In EDG a set of integrated data management services was developed [19] to enable movement and replication of data at high speed, management of distributed replicated data and associated metadata, and optimised access to data.

The first prototype data management system was implemented in C++ and comprised the edg-replica-manager [120], based on the Globus toolkit and the Grid Data Mirroring Package (GDMP) [121]. GDMP was a service for mirroring file sets between Storage Elements and together with the edg-replica-manager it provided basic replication functionality. After experience gained from deployment of these prototypes, it was decided to adopt the web services paradigm and implement the data management services in Java. The second generation data management system includes the following services: the Replica Location Service, the Replica Metadata Catalog, and the Replica Optimization Service. The primary interface between users and these services is the Replica Manager client.

The design of the data management system is modular, with several independent services interacting via the Replica Manager, a logical single point of entry to the system for users and other external services. The Replica Manager coordinates the interactions between all components of the systems and uses the underlying file transport services for replica creation and deletion. Query functionality and cataloging for distributed replicated data are provided by the Replica Metadata Catalog (RMC) and Replica Location Service (RLS). Optimised access to replicas is provided by the Replica Optimization Service (ROS), which aims to minimise file access times by directing file requests to appropriate replicas.

The Replica Manager is implemented as a client side tool whilst the Replica Metadata Catalog, Replica Location Service and the Replica Optimization Service are all stand-alone services. One advantage of such a design is that if any service is unavailable, the Replica Manager can still provide the functionality that does not make use of that particular service. Also, critical service components may have more than one instance to provide a higher level of availability and to avoid service bottlenecks. However, a lot of the coordinating logic happens at the client side, so asynchronous interaction is not possible. Also, in case of failures on the client side, there is no way to automatically re-try the operations.

3.2.1 Web Service Design

Web service technologies [131] provide an easy and standardised way to logically connect distributed services via XML (eXtensible Markup Language) messaging. They provide a platform and language independent way of accessing the information held by the service and, as such, are highly suited to a multi-language, multi-domain environment such as a Data Grid.

A web service is hosted within an application server and exposes an interface in Web Service Description Language (WSDL) format [132] which describes the operations the service performs. For the RLS, RMC and ROS these are operations such as querying for the locations of replicas of a particular dataset or finding the best replica to access. Clients for the services can be automatically generated in any language from the WSDL interface. The client and server communicate using the language-independent Simple Object Access Protocol (SOAP) with XML messages passed around using HTTP, or HTTPS for encrypted messages.

The translation from each client language to the SOAP messages understood by the server is performed by a SOAP implementation such as Axis (for Java), gSOAP (C++) or SOAPpy (Python). Clients bind to the service which is defined by a URL representing its location, and they can then invoke methods of the service as if it was a local object. The details of the communications between client and server and language translations are hidden by the SOAP implementation. Persistent data associated with a service such as the location of replicas is stored in a relational database, usually on the same machine as the service.

All the EDG data management services have been designed and deployed as web services and the RLS and RMC have been used with open source and, in the LCG project, commercial application servers and databases. The servers are written in Java, with Java and C++ clients provided, although it is possible to generate clients in any other language from the WSDL.

3.2.2 Replica Manager

For the user, the main entry point to the data management services is through the Replica Manager client interface that is provided via C++ and Java APIs and a Command Line Interface. The actual choice of the service component to be used can be specified through configuration files, and Java dynamic class loading features are exploited to make each particular component available at execution time. The interactions between the Replica Manager and these components and other EDG components is shown in Figure 3.6.

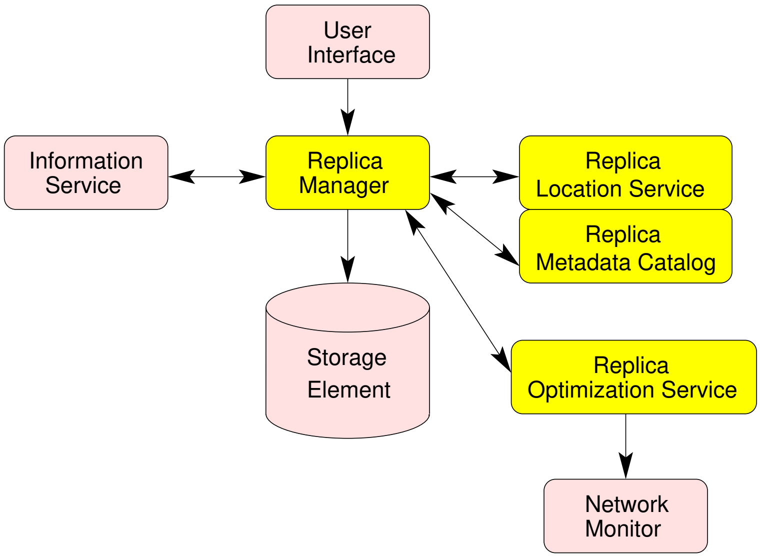


Figure 3.6: Interactions between the Replica Manager and other components.

The Replica Manager uses the other data management services to obtain information on data location but it also uses many external services. An Information Service such as MDS (Monitoring and Discovery Service) or R-GMA (Relational Grid Monitoring Architecture) needs to be present, as well as storage resources with a well-defined interface, in our case SRM (Storage Resource Manager) or the EDG-SE (EDG Storage Element). The Replica Manager also makes use of file transport mechanisms such as GridFTP.

3.2.3 Replica Location Service

In a highly geographically distributed environment, providing global access to data can be facilitated via replication, which can also reduce access latencies and improve system robustness and scalability. However, the existence of multiple replicas of files in a system introduces additional issues. The replicas must be kept consistent, they must be locateable and their lifetime must be managed. The Replica Location Service (RLS) is a system that maintains and provides access to information about the physical locations of copies of files [35].

The RLS architecture defines two types of component: the Local Replica Catalog (LRC) and the Replica Location Index (RLI). The LRC maintains information about replicas at a single site or on a single storage resource, thus maintaining reliable, up to date information about the independent local state. The RLI is a (distributed) index that maintains soft state collective information obtained from any number of LRCs.

Globally Unique IDentifiers (GUIDs) are guaranteed unique identifiers for data on the Grid and are based on Universally Unique IDentifiers (UUIDs). In the LRC each GUID is mapped to one or more physical file names identified by Storage URLs (SURLs), which represent the physical location of each replica of the data. The RLI stores mappings between GUIDs and the LRCs that hold a mapping for that GUID. A query for a replica is a two stage process. The client first queries a RLI in order to determine which LRCs contain mappings for a given GUID. One or more of the identified LRCs is then queried to find the associated SURLs. As almost all files are assumed to be read-only in EDG applications, all the replicas of a given file should be consistent with each other.

An LRC is configured at deployment time to subscribe to one or more RLIs. The LRCs periodically publish the list of GUIDs they maintain to the set of RLIs that index them using a soft state protocol, meaning that the information in the RLI will time out and must be refreshed periodically. The soft state information is sent to the RLIs in a compressed format using bloom filter objects [17]. A possible deployment architecture is shown in Figure 3.7. Here the middle RLI contains complete global information since all LRCs publish state to it, however the other two hold information on only two LRCs, one of which is common between them.

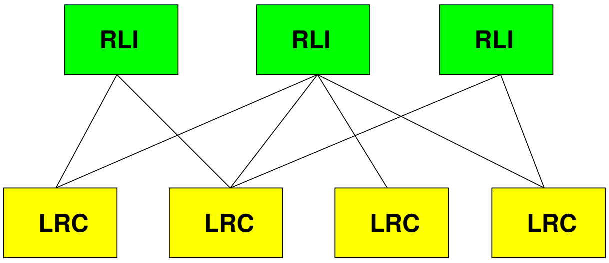


Figure 3.7: A possible RLS architecture. From [35].

An LRC is typically deployed on a per site basis, or on a per storage resource basis, depending on the site's resources, needs and configuration. A site will typically deploy one or more RLIs depending on usage patterns and need. The LRC can also be deployed to work in stand-alone mode instead of fully distributed mode, providing the functionality of a replica catalog operating in a fully centralised manner. In stand-alone mode, one central LRC holds the GUID to SURL mappings for all the distributed Grid files.

3.2.4 Replica Metadata Catalog Service

The GUIDs stored in the RLS are neither intuitive nor user friendly. The Replica Metadata Catalog (RMC) allows the user to define and store Logical File Name (LFN) aliases to GUIDs. Many LFNs may exist for one GUID but each LFN must be unique within the RMC. Below are a sample GUID and LFN:

guid:63047220-2dd5-11d9-9669-0800200c9a66

lfn:data/cal_test/2004-09-01-001

The relationship between LFNs, GUIDs and SURLs and how they are stored in the catalogs is summarised in Figure 3.8. In addition, the RMC can store GUID metadata such as file size, owner and creation date. The RMC is not intended to manage all generic experimental metadata however it is possible to use the RMC to maintain

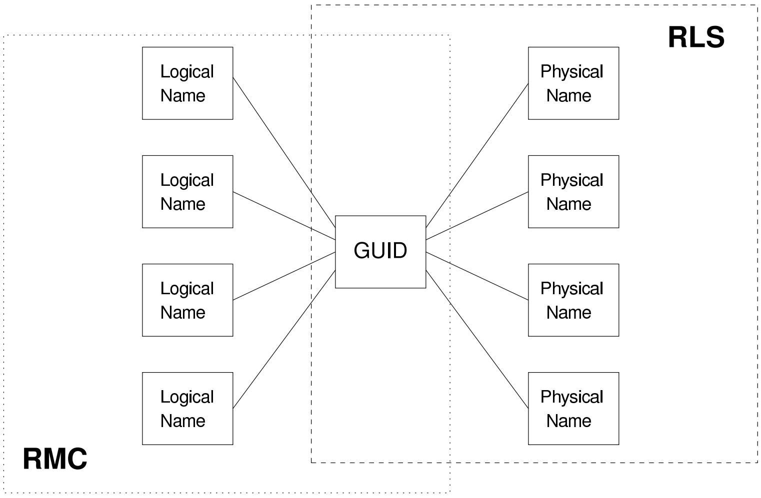


Figure 3.8: The Logical File Name to GUID mapping is maintained in the Replica Metadata Catalog, the GUID to physical file name (SURL) mapping in the RLS.

$\mathcal{O}(10)$ items of user definable metadata. This metadata provides a means for a user to query the file catalog based upon application-defined attributes.

The reason for providing a separate RMC service to the RLS for the LFN mapping is the different expected usage patterns of the LFN and replica lookups. The LFN to GUID mapping and the corresponding metadata are used by the users for pre-selection of the data to be processed. The replica lookup happens at job scheduling time when the locations of the replicas need to be known and at application runtime when the file is to be processed.

3.2.5 Replica Optimization Service

Optimisation of the use of computing, storage and network resources is essential for application jobs to be executed efficiently. The Replica Optimization Service (ROS) [12] focuses on the selection of the best replica of a data file for a given job, taking into account the location of the computing resources and network and storage access latencies.

The network monitoring services (Section 3.1.5) provide the information for the ROS on network latencies between the various Grid resources. This information is used to calculate the expected transfer time of a given file with a specific size. The ROS can also be used by the Resource Broker to schedule user jobs to the site from which the data files required can be accessed in the shortest time. The ROS is implemented as a lightweight web service that gathers information from the European DataGrid network monitoring service and performs file access optimisation calculations based on this information.

3.2.6 Service Interactions

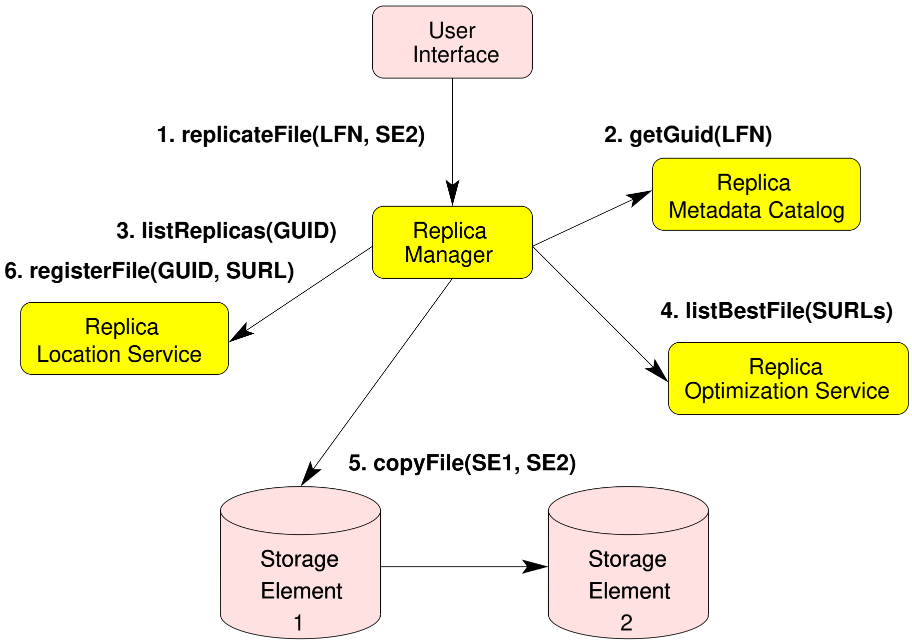


Figure 3.9: Typical usage of the EDG data management services.

The interaction between the various data management services can be explained through a simple case of a user wishing to make a copy of a file currently available on the Grid to another Grid site (Figure 3.9). The user supplies the LFN of the file and the destination storage location to the Replica Manager (1). The Replica Manager contacts the RMC to obtain the GUID of the file (2), then uses this to query the RLS for the locations of all currently existing replicas (3). The ROS calculates the best site from which the file should be copied based on network monitoring information (4). The Replica Manager then copies the file (5) and registers the new replica information in the RLS (6).

3.2.7 Security

Secure access to data management services via the use of GSI certificates is enabled by the edg-java-security package. It consists of two main components: one dedicated to authentication of any service requests (Trust Manager) and one dedicated to authorisation of any service requests (Authorization Manager).

The authentication mechanism developed by EDG is an extension of the standard Java implementation of the SSL/TLS (Secure Socket Layer/Transport Layer Security) protocols [43]. These protocols provide communications privacy over the Internet using cryptography, allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. Mutual authentication in SSL/TLS is based on public-key certificates that are signed by trusted Certification Authorities (CA). The user and the server verify each others' identities by proving in cryptographic terms that they have access to the private key that matches with the certificate.

The Globus Toolkit introduces the use of time-limited GSI proxy certificates, usually with a lifetime of 12 hours. A proxy certificate consists of a new certificate signed by the end entity (user or server) that created it, rather than by a CA. This proxy certificate causes problems since it is signed by an entity that is not a CA. This causes the mutual authentication to fail in any implementation that fully conforms to the SSL standard. For the GSI proxy to work, the SSL implementation in the Trust Manager was changed to a non-standard form that accepts proxy certificates.

The EDG Java security package implements a coarse-grained authorisation model. The authorisation mechanism is positioned in front of the service and the authorisation decision is made before the actual call to the service. The Authorization Manager within the EDG Java security package implements a role-based access control policy. The Authorization Manager decides whether a particular user, extracted from the certificate presented by a client, can be associated with a given role, or attribute. The Authorization Manager can optionally perform a translation phase: after establishing that a subject indeed is authorised, it translates the attribute into a local-ID value understandable by a specific application. This service is similar to the role played by the LCAS/LCMAPS system in the Computing Element. Security inevitably adds a layer of complexity to any application, however it is absolutely necessary for the users and the service providers and therefore there always has to be some trade-off between the security requirements and the level of simplicity required by the application.

3.3 Summary

In this chapter an overview of the European DataGrid project has been presented. Building on top of the Globus Toolkit and Condor-G, the middleware components developed by EDG together provide a complete Grid system for use by high energy physics, biology and Earth observation sciences applications. Several work packages worked in close collaboration to produce components and services for accessing distributed computational and storage resources.

A large number of computing resources can be used by the Grid via the Computing Element middleware and jobs can be submitted to them via the Workload Management System. The Storage Element means mass storage resources can also be transparently accessed by users and applications. R-GMA and Network Monitoring provide Grid information for all other Grid components and security mechanisms ensure the resources are used correctly.

The vast amount of data the applications generate is handled by a set of integrated data management services, designed around the Web Services model. These services enable secure movement and replication of data at high speed from one geographical site to another, management of distributed replicated data, optimisation of access to data, and the provision of a metadata management tool.

Chapter 4

Performance Evaluation of Data Management Services

In this chapter the properties and performance of the European DataGrid data management services are investigated. Middleware components must be designed to withstand heavy and unpredictable use and be able to scale well with the demands of the Grid. Results presented here show the data management services can handle the loads as expected and scale well, however the use of security mechanisms adds a large overhead.

Clients to use these services are written in three forms: C++ API (Application Program Interface), Java API, and CLI (Command Line Interface). It was envisaged that the CLI, typing a command by hand on the command line of a terminal, would be mainly used for testing an installation or individual command. The APIs on the other hand would be used directly by applications' code and would avoid the need for the user to interact directly with the middleware. In this chapter all three clients for each component are tested first without any security, and then with security. The extra overhead security introduces is significant, but it is a necessary factor if secure access is required.

4.1 Testbed Setup

In order to test the performance of each service, a testbed consisting of 11 machines in 5 sites was used. All the machines had similar specifications and operating systems and ran identical versions of the data management services: LRC (v2.1.2), RMC (v2.1.2) and ROS (v2.1.4-cg). The application server used to deploy the services was Apache Tomcat 4 and for storing data on the server side, MySQL 4 was used. The list of machines is given in Table 4.1. For most of the performance tests small test applications were developed; these are packaged with the software and can therefore be re-run to check the results obtained.

Machine	Location	Function
lxshare0313	CERN, Switzerland	Client, LRC, ROS
lxshare0344	CERN, Switzerland	Client
lxshare0343	CERN, Switzerland	LRC, RMC
lxshare0346	CERN, Switzerland	LRC, RMC
pccms144	CERN, Switzerland	LRC
grid04	Glasgow, Scotland	Client, LRC
grid05	Glasgow, Scotland	Client, LRC
grid06	Glasgow, Scotland	Client, LRC
is01vidgrid	Vienna, Austria	LRC
grid	Innsbruck, Austria	LRC
eio01	Tel Aviv, Israel	LRC

Table 4.1: Testbed used for evaluation of data management services, showing the services installed on each node.

4.2 Replica Location Service

The Replica Location Service (RLS) is described in detail in Section 3.2.3. It is used to find the locations of physical copies of files in the Grid, given the Globally Unique Identifier (GUID) of the data. The complete service comprises a number of Local Replica Catalogs (LRCs) which hold information on the files at one particular site, and a number of Replica Location Indexes (RLIs) which are sent regular updates from LRCs and queried by the user or client API. However, within EDG and LCG the RLS has only been used with a single LRC per VO, therefore all the RLS results presented here show the performance of a single LRC.

Three main functions of the catalog were tested: insert (add a GUID to physical file name (SURL) mapping), query (get the SURLs corresponding to the given GUID) and delete (remove a GUID to SURL mapping). For all tests the testbed shown in Table 4.1 was used, with pairs of clients and servers on the same local area network to avoid differences in results caused by the time to communicate between the two.

4.2.1 C++ API

The performance of the C++ client was tested by timing how long it took to insert a certain number of mappings, how long it took to delete them and how long one query took, for varying number of entries in the catalog. This tests how the number of entries in the catalog affects the time to complete each operation. Figure 4.1(a) shows the total time to insert and delete a certain number of mappings, and Figure 4.1(b) shows how the time to query one entry varies with the number of entries in the LRC.

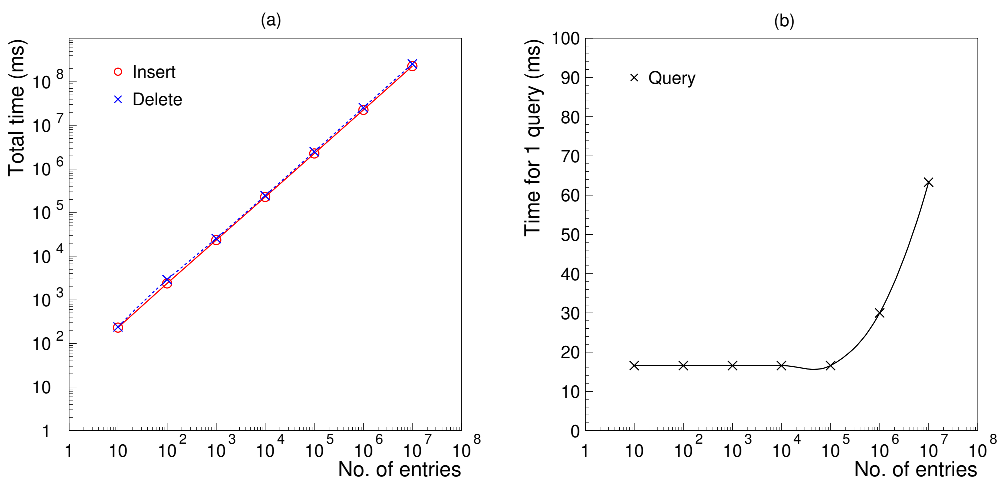


Figure 4.1: (a) Total time to add and delete mappings and (b) query the LRC using the C++ API.

The results show that insert and delete operations have very stable behaviour, in that the total time to insert or delete mappings scales linearly with the number of mappings inserted or deleted. A single transaction with a single client thread takes about 25 - 29 ms with the tendency that delete operations are slightly slower than inserts. The query time of around 16 ms is independent of the number of entries in the catalog up to around 1 million entries, when it tends to increase. This is due to the underlying database, which then takes longer to query the more entries it contains.

4.2.2 Java API

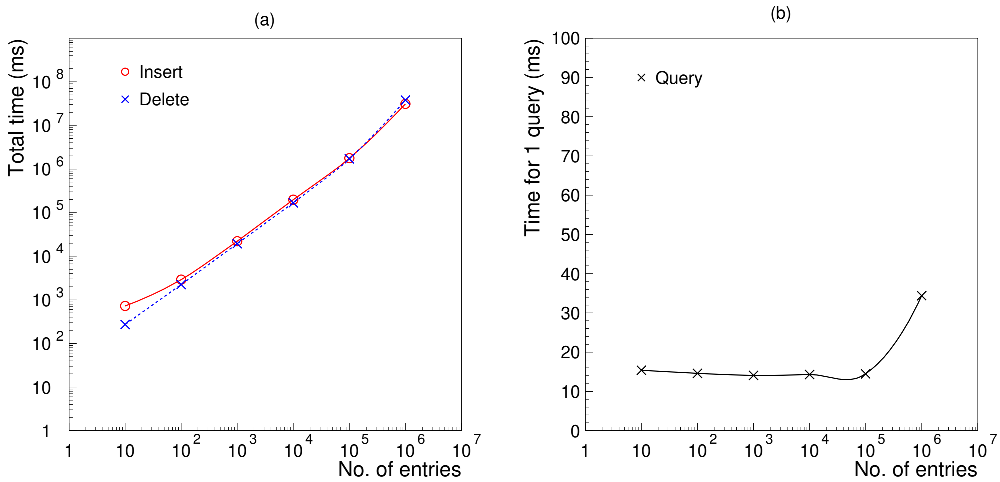


Figure 4.2: Total time to (a) add and delete mappings and (b) query the LRC using the Java API.

Similar tests were performed using the Java API and these results are shown in Figure 4.2. The results are very similar to the C++ results, however Java performs dynamic class loading which means the first time a class is used, some time is spent reading it into memory. Therefore the first insert takes much longer than the following inserts, as shown by Figure 4.2(a), where 10 inserts take around 3 times as long using Java than they do using C++. However, with a larger number of operations this effect becomes negligible. Query performance is also similar to C++, with the time remaining around 16 ms up to 1 million entries, when it increases to 35 ms.

Taking advantage of the multiple threading capabilities of Java, it was possible to simulate many concurrent users of the catalog to analyse its performance under heavy loads. Firstly, the time to insert a GUID to SURL mapping into the LRC as a function of number of entries in the LRC and number of concurrent threads was measured. Figure 4.3 shows, for different numbers of concurrent threads, how the average time to insert a mapping (averaged over every 1000 inserts) varies over the course of inserting 500,000 mappings into an empty LRC.

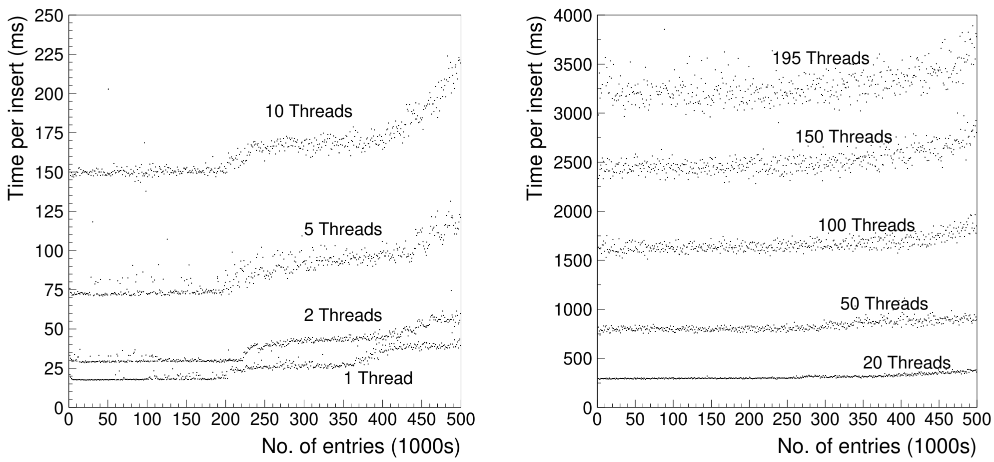


Figure 4.3: Time to insert one mapping into the LRC as the number of entries and number of concurrent threads varies.

The shapes of the plots are very similar for different numbers of concurrent threads, in that the insert time is constant for around the first 200,000 entries, but after this it gradually increases as the number of entries increases. The effect is more pronounced with a lower number of threads, for example with one thread the increase is almost 100% between 0 and 500,000 entries, whereas it is only around 20% when 195 threads are used. However, the data becomes more noisy when more threads are used which tends to obscure the trend.

To measure the effective throughput of the LRC, i.e. the time to complete a certain number of operations, the total time to insert 500,000 mappings was measured for different numbers of concurrent threads. Figure 4.4 shows that the time falls rapidly with increasing numbers of threads, bottoming out after 10 or 20 threads. For 20 threads the total time taken is about 40% less than using one thread. Although the

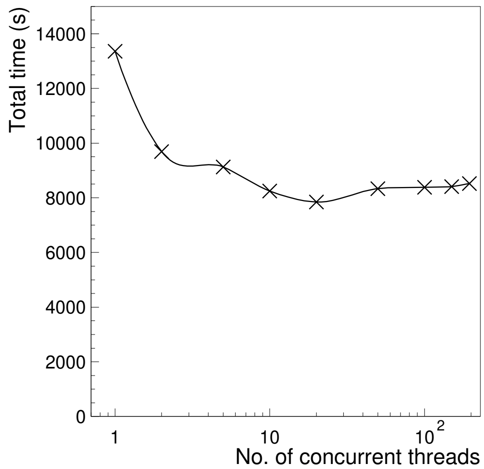


Figure 4.4: Total time to add 500,000 mappings to the LRC using concurrent threads.

time for an individual operation is slower the more concurrent operations are taking place, the overall throughput actually increases, showing the ability of the LRC to handle multiple processes concurrently.

One client running 195 client processes at the same time is an unrealistic situation - it is much more likely that concurrent transactions would arise from different client machines. It was thought that the large amount of memory and processing time consumed on one client machine may have caused the noisy data for large numbers of threads rather than the stress on the server. Therefore a test was run whereby the client load was split between two separate machines, with each one running 25 threads. A comparison of the performance seen by one client running 50 threads alone, two clients both running 25 threads and one client running 25 threads alone is shown in Figure 4.5.

It is clear that the data for the single client running 50 threads is as noisy as it is for the two separate clients running 25 threads each. Also, both are far noisier than the single client running 25 threads, hence we can conclude that the spread of data points is, if not wholly, then at least largely happening on the server-side rather than the client-side. It is interesting to note that the average time for inserts using

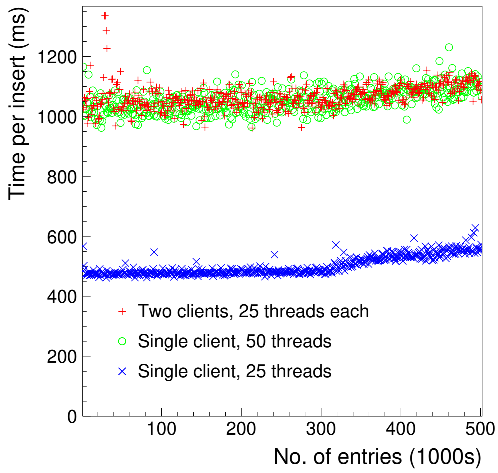


Figure 4.5: Comparison of LRC insert time between one and two clients each running 25 threads and one client running 50 threads.

only 50 threads is just over twice that of the inserts using 25 threads, confirming the previous result that the throughput decreases with the number of threads after around 20 threads.

A comparison between insert and query performance was done with a client running 10 concurrent threads, where at any given moment 5 threads would be inserting a mapping and 5 threads would be querying a mapping. Figure 4.6 shows the insert time rising from 140 ms to 200 ms as the number of entries increases from 0 to 500,000 as it did in Figure 4.3. However the query time stays at a constant 100 ms and does not vary with the number of entries. Since, after an initial period of filling up the catalog, the main operations on the catalog will be query operations, this result shows excellent scalability for long-term usage.

To investigate performance in realistic situations, it was decided to simulate users using the LRC in a production environment. The two typical situations mentioned above which might arise in such an environment were tested:

- *Start-up:* An initial period, adding data to a catalog: many inserts, a few queries and no deletes, starting from an empty catalog.

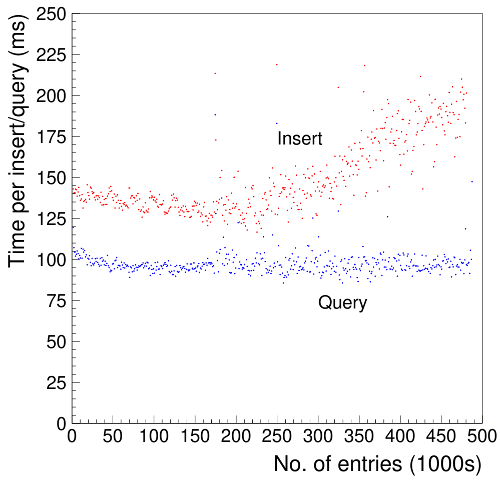


Figure 4.6: Time to insert mappings and query one GUID for different numbers of entries in the LRC, using 5 concurrent inserting threads and 5 concurrent querying threads.

- *Normal:* The expected pattern of operations, for example by users doing physics analysis: a few inserts, many queries and a few deletes starting from a reasonably full catalog.

Results simulating the first situation are shown in Figure 4.7(a). Here 500,000 mappings were inserted and 50,000 queries were performed using different numbers of concurrent threads and the total time measured. The results are very similar to Figure 4.4, showing that the extra 50,000 queries have little effect on the total time. Again, the total time bottoms out at around 20 threads.

Figure 4.7(b) shows results from simulating the second situation and for this plot 100,000 inserts, 500,000 queries and 100,000 deletes were performed on a catalog that initially held 500,000 entries. The minimum total time is not so well defined and even though there were 150,000 more operations than in Figure 4.7(a) on a large catalog compared to an empty one, the total time is less for all numbers of threads (except 1 thread) than in Figure 4.7(a). This confirms the result shown in Figure 4.6, that query operations are much less time consuming than insert operations.

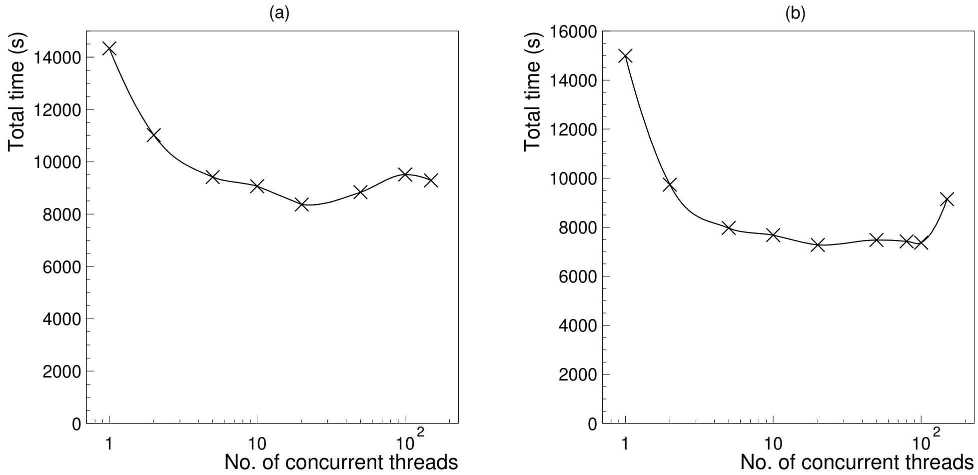


Figure 4.7: Total time for (a) Start-up and (b) Normal operations on the LRC using concurrent threads.

4.2.3 C++ API vs Java API

APIs for the data management services are provided in two languages due to the many possible applications that could use each service. Popular opinion holds that C++ in general out-performs Java, although several studies, mainly using numerically intensive code [18, 21], have shown the difference to be "negligible" (around 20%). More relevant to this study are a set of benchmarks presented in [11], which shows the performance of Java server applications to be on a par with other language implementations. Java also offers the advantages of portability, ease of use and automatic memory management. In this section, the performance of the two different client implementations is compared (for all services the servers are implemented in Java).

A direct comparison of how the insert time for one thread varies with the number of entries in the catalog for C++ and Java is given in Figure 4.8. The performance is very similar up to about 150,000 entries, where the time for each operation using Java increases, while for C++ it remains roughly constant. The reason for this could be the amount of memory used with the Java client slowing it down, since the GUID for each mapping is stored after the insert in a list, which becomes rather large when it

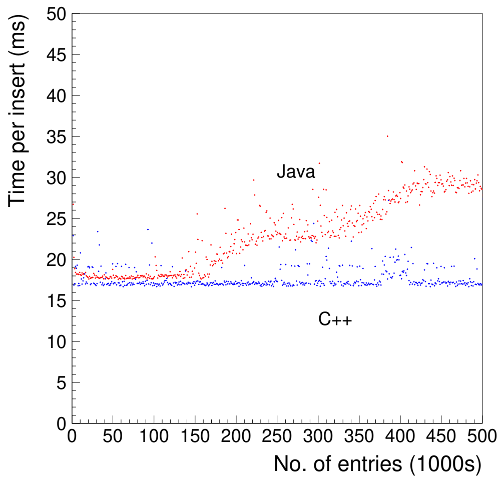


Figure 4.8: Average time to insert one mapping into the LRC using C++ and Java API as the number of entries varies.

contains hundreds of thousands of elements. The total integrated time for all 500,000 operations was just over 9,000 s with C++ and almost 13,500 s for Java, a difference of almost 50%.

4.2.4 Server and Client Comparison

In this section the details of each catalog operation are studied by comparing the times spent on the client-side and server-side of the service. The operation of web services was described in detail earlier in Section 3.2.1 but to summarise, they offer a standard way for different software applications running on different platforms and implemented in different languages to interact using XML-based SOAP messaging. All the data management services expose a standard interface in WSDL from which client stubs can be generated in any of the common programming languages. A client user application can then bind to the service and invoke methods as if it was a local object.

The performance of the server should be independent of the client used (Java in this case). The fraction of time spent, for each catalog operation, on server-side processing is shown in Figure 4.9. The server-side time was defined as the time spent completing the necessary operations on the database to create the GUID to SURL mapping.

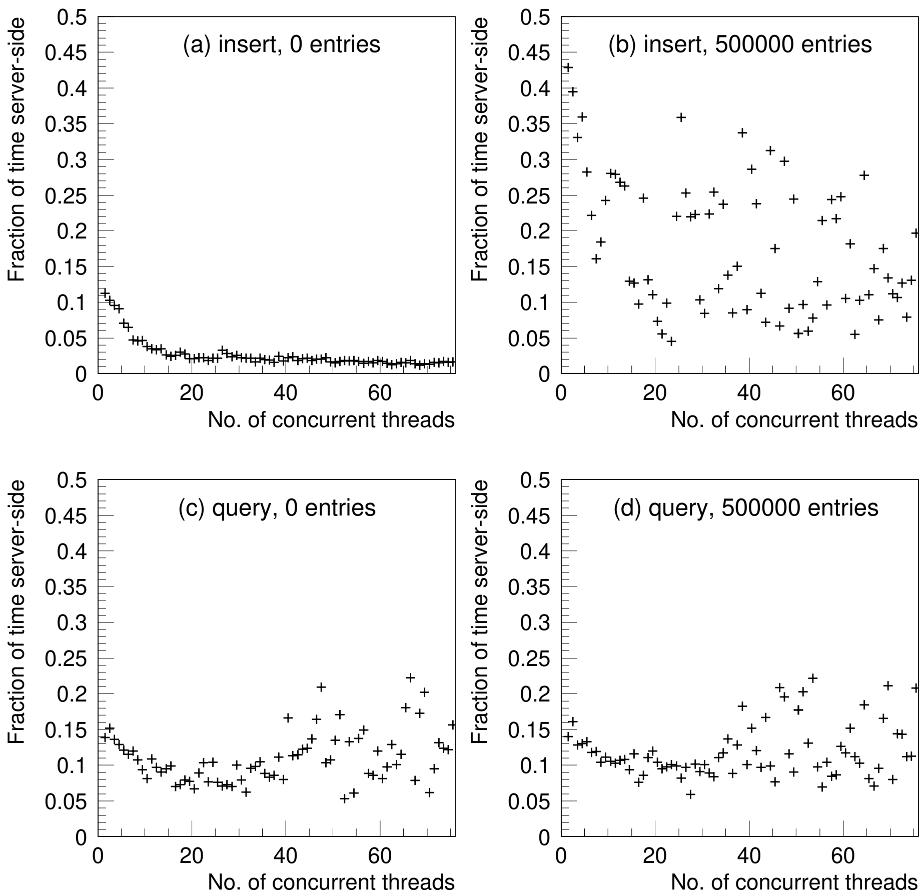


Figure 4.9: Fraction of time spent in server-side processing of the LRC for different numbers of concurrent threads.

Four different situations were tested: insert and query operations on an empty catalog and the same operations on a catalog containing 500,000 entries. The results are quite surprising when compared with those in Figure 4.3. On an empty catalog, the fraction of time spent server-side performing insert operations drops dramatically the more concurrent threads are used. However since Figure 4.3 shows that as the number of threads increases the insert time also increases, it can be deduced that the thread-handling capabilities on the server-side (i.e. Tomcat 4) are much better than on the client-side (Java concurrent threading library).

This does not seem to be the case though when the catalog is reasonably full (Figure 4.9(b)). Here the data is extremely noisy but the fraction of time in server-side processing is much higher than with an empty catalog, which is to be expected since the database operations take longer the more entries there are. The length of time spent in these operations is also less predictable and depends on internal database optimisations occurring during each transaction, contributing to the noisy data.

The results for queries (Figures 4.9(c) and (d)) are almost identical for the different sized catalogs, which agrees with the previous findings that query time is independent of the number of entries in the catalog (see Figure 4.6). Up to 30 threads the general trend is for less time to be spent server-side the more threads there are but above this the data becomes very noisy.

A comparison of the server-side performance using an empty catalog and a reasonably full one can be seen in Figure 4.10. The two histograms show the times for 1000 inserts using one thread and show clearly the times are far more spread out when the catalog is full than when it is empty. The average times are 8.2 ms for the empty catalog and 31.0 ms for the catalog with 500,000 entries, almost a 4-fold increase. The few times with very high values in Figure 4.10(b) (up to 850 ms) are rare but could potentially cause problems with other services using the RLS timing out while waiting for a response. The secondary peak seen in Figure 4.10(b) is probably due to the same reasons as given above, i.e. an operation triggering internal database optimisation operations occurring when there are a large number of entries.

4.2.5 Command Line Interface

The command line interface (CLI) is implemented in Java using the Java API and allows any operation on the catalog to be performed from a terminal command line. This feature was provided at the request of users as it gives a convenient way to test a service installation or for new users to learn the functionality of the service. It was

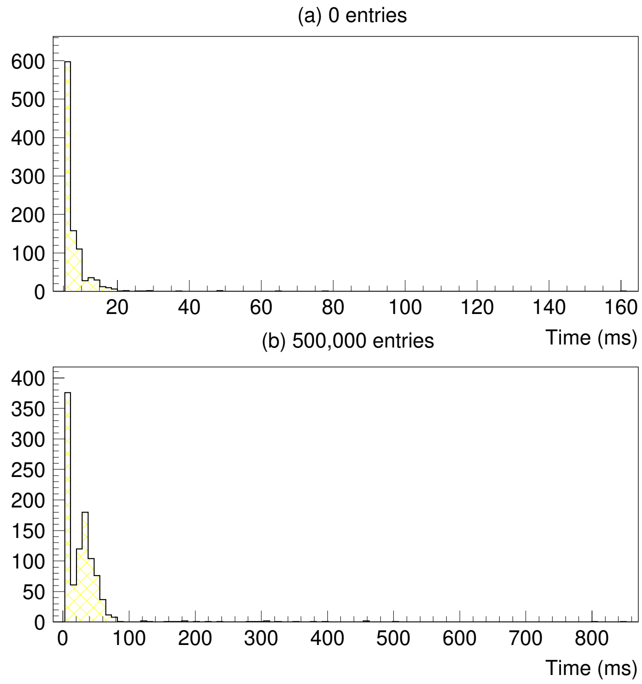


Figure 4.10: LRC insert time on server-side for inserting 1000 entries into (a) an empty catalog and (b) a catalog with 500,000 entries.

not intended to be used for large-scale operations and therefore was not optimised for performance. The reason Java was chosen and not C++ was that there was a delay in providing the C++ API due to the complexity of the security interaction through an auto-generated SOAP client. Had the CLI been implemented in C++ this would have given a performance improvement, however this would not have been a significant or important improvement. Table 4.2 contains some timing statistics showing the time to execute different parts of the command `addMapping` (in insecure mode) used to insert a GUID to SURL mapping into the LRC.

The total time to execute the command was 3.2 seconds and this time is broken down into the following areas: The start-up script sets various options such as logging parameters and the class-path for the Java executable and this, along with the time to start the Java Virtual Machine (JVM), took 0.8 s. After parsing the command

Time (s)	Operation
0 - 0.8	Start-up script and JVM start-up time
0.8 - 1.0	Parse command and options
1.0 - 2.2	Get LRC service locator
2.2 - 2.3	Get LRC object
2.3 - 3.2	Call to the lrc.addMapping() method
3.2	End

Table 4.2: Timing statistics for adding a mapping in the LRC using the CLI.

line it took 1.3 s to get the LRC service locator (i.e. bind to the web service) - during this time many external classes had to be loaded in.

The call to the *addMapping()* method took around 0.9 s, due to the effect of dynamic class loading in Java the first time a class is used within this method. Compared to the average of around 20 ms for this operation observed above in the Java API tests, this is very large, and because every time the CLI is used a new JVM is started up, the time to execute the command is the same every time. In short, the time taken to insert a GUID to SURL mapping using the command line interface is about 2 orders of magnitude longer than the average time taken using the Java or C++ API. Therefore, the command line tool is only recommended for simple testing and not for large scale operations on the catalog.

4.2.6 Comparison with Globus RLS

Similar performance tests on the C-based Globus implementation of the Replica Location Service framework are presented in [36]. This version is not implemented as a web service but uses Remote Procedure Calls (RPC) to communicate between client and server. As with the EDG RLS peak performance was seen when around 10 concurrent client threads were using the service. A comparison between the different rates achieved when adding mappings to the two catalogs for 1 and 10 concurrent clients is shown in Table 4.3.

The Globus RLS tests were performed with the back-end MySQL database in

RLS type	Inserts per s (1 thread)	Inserts per s (10 threads)
EDG	37	62
Globus (with DB flush)	84	84
Globus (no DB flush)	460	700

Table 4.3: Inserts per second rates achieved with the EDG and Globus RLS, with 1 and 10 concurrent client threads. The Globus RLS figures are taken from [36].

two modes of operation: with and without immediate flushing of data to disk. If the flush is on (as it is in the EDG RLS), whenever a transaction such as adding or deleting a mapping is performed, the data is written immediately to disk. With the flush switched off, this data is only written periodically, and so performance is vastly improved (up to 8 times faster) but at the cost of possibly corrupting the database with inconsistent data. When the database flush is on, the difference in insert rates between the EDG and Globus services is around 60% for 1 thread and around 25% for 10 threads. The cause of the lower rate achieved with the EDG RLS at this level is mainly due to the larger communications overhead introduced by SOAP web services as compared to RPC. However, the benefits of using a web services design (ease of use and deployment, portability, conformity with emerging Grid standards) outweigh the lack of performance as compared to RPC for the majority of Grid applications.

4.3 Replica Metadata Catalog Service

The Replica Metadata Catalog (RMC) can be regarded as an add-on to the RLS system and is used by the Replica Manager to provide a complete view on LFN:GUID :SURL mapping (Figure 3.8). Users can define their own LFN aliases to GUIDs to make their data-naming more manageable and user friendly. The RMC service is implemented in exactly the same way as the LRC, i.e. a web service architecture with a database back-end to hold the LFN to GUID mappings. In fact the way the RMC and LRC are used is exactly the same, only the data stored is different and thus one would expect similar performance from both components. This section describes the results of the some of the tests that were run on the LRC, performed on the RMC.

4.3.1 C++ API

First the performance of insert, query and delete operations with varying numbers of entries in the catalog was tested. The results are shown in Figure 4.11 and show very similar insert/delete and query rates as for a single LRC. On average, a single insert operation takes about 22 ms as compared to 24 ms for a delete operation using a single client thread. Query operations take around 20 ms up to 1 million entries, when it increases to 30 ms.

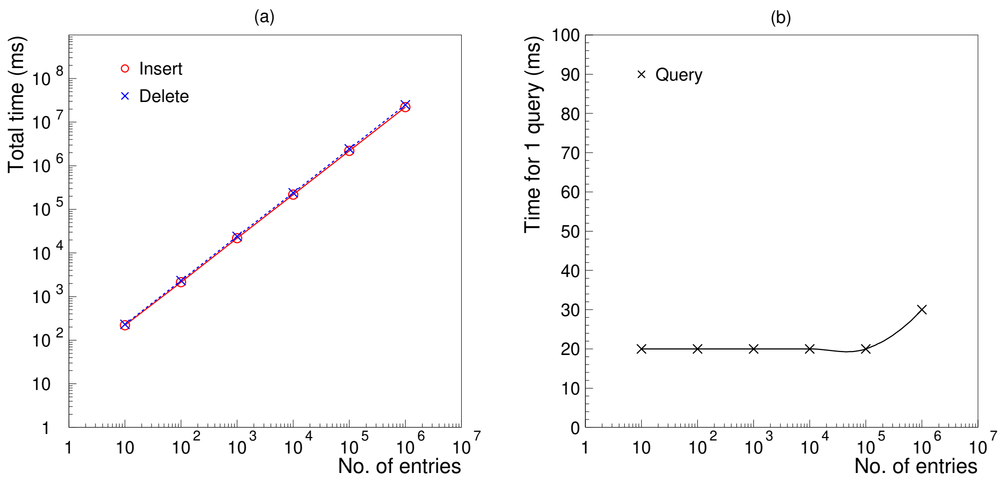


Figure 4.11: (a) Total time to add and delete mappings and (b) query the RMC using the C++ API.

In the EDG model, many users may be interested in the same data sets and thus there can be many user defined aliases (LFNs) to a single unique file identifier (GUID). Therefore the behaviour of the RMC was analysed in the case where multiple LFNs were mapped to a single GUID. Figure 4.12 shows the time to insert and delete 10 GUIDs with different numbers of LFNs mapped to each GUID and the time to query for 1 LFN with varying numbers of LFNs per GUID.

The insert/delete times increase linearly as one might expect, since each new LFN mapping to the GUID is treated similarly to inserting a new mapping, thus the effect

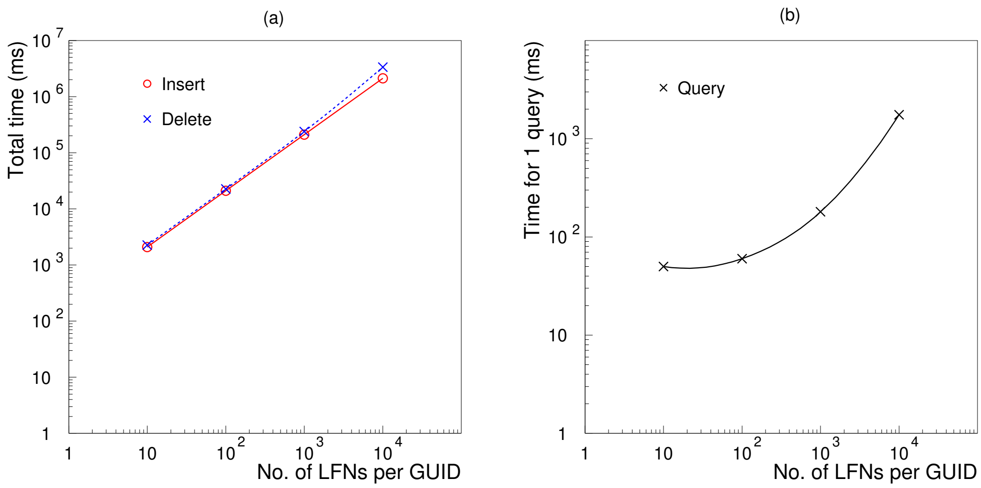


Figure 4.12: Total time to (a) insert and delete 10 GUIDs with varying number of LFNs, and (b) query for one LFN in the RMC

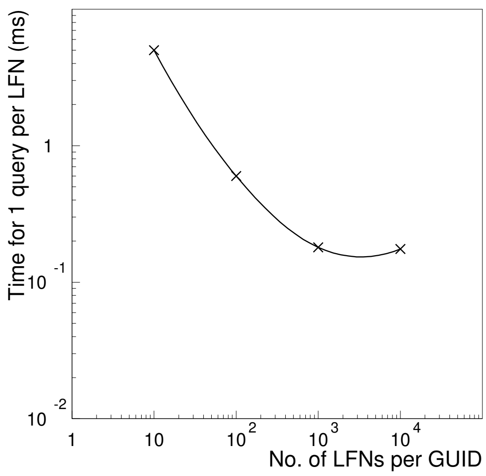


Figure 4.13: RMC query time per mapping for different numbers of LFNs mapped to one GUID.

is to give similar results to Figure 4.11 in terms of number of operations performed. Query operations take longer the more LFNs exist for a single GUID, however the query time per LFN mapped to the GUID (Figure 4.13) actually decreases the more mappings there are, showing that the RMC performance scales well with the number of mappings up to 1000 mappings, which is far more mappings than would be expected for any application.

4.3.2 Java API

The performance of the RMC Java API was measured in a similar way to the performance of the LRC Java API. A Java client class was written which used multiple threads to simulate many users concurrently accessing the RMC. It performs the required number of operations (create, query and delete) as quickly as possible using all the threads available and for each type of operation the average time to complete it was measured over every 1000 operations.

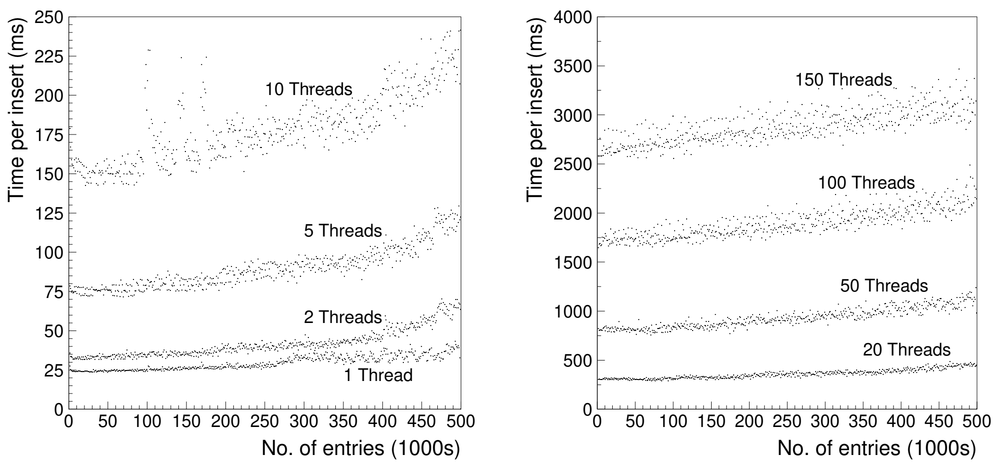


Figure 4.14: Time to insert one mapping into the RMC as the number of entries and number of concurrent threads varies.

Firstly, the time to insert a GUID to LFN mapping into the RMC was measured as the number of entries in the catalog and the number of concurrent threads changed. Figure 4.14 shows how the average time to insert a mapping varies from using an empty RMC to using a RMC with 500,000 entries for 1 to 150 concurrent threads. As with the LRC, the shapes of the plots are very similar for the different numbers of concurrent threads, in that the insert time gradually increases with the number of entries in the catalog. As the catalog fills up, the insert times become more noisy.

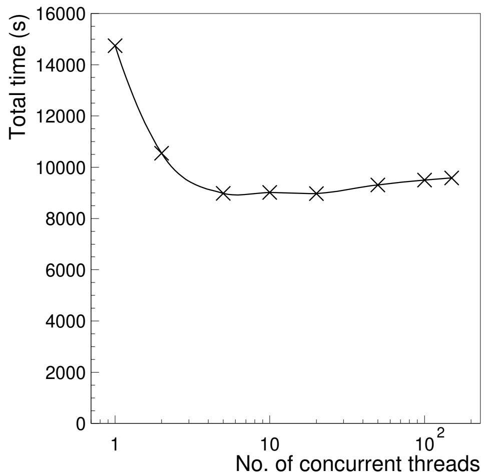


Figure 4.15: Total time to add 500,000 mappings to the RMC using concurrent threads.

Next, the total time to insert 500,000 mappings was measured for different numbers of concurrent threads. This effectively gives the throughput performance of the RMC. Figure 4.15 shows that the throughput reaches a maximum at around 5-20 threads, a similar result as was seen with the LRC. The total time using 20 threads improved by around 40% compared to using a single thread. All the results for the RMC, as expected, are very similar to the single LRC results, since both services are almost identical in their implementation.

Metadata is "data about data" and is used to organise and manage data so that sets of data can be categorised according to certain attributes. Each application will have its own specific metadata schema which defines the attributes associated with their data, for example in the LHC high energy physics experiments described in Section 1.3, TAG data is metadata for each event, and is used in analyses to select events with certain physics attributes. The method of storing this metadata is application dependent, but for all applications the RMC enables low-level file metadata to be stored, such as date created or file size. Application metadata is likely to consist of hundreds of attributes, whereas the RMC was designed to store O(10) attributes, although the underlying technology (database behind a web service) is essentially the same.

Inserting metadata attributes into the RMC takes time, so the effects on the time to include them were measured and the results are shown in Figure 4.16. In this figure, the variation of the time to insert a mapping including attributes with the number of entries in the catalog is plotted, when 1, 5, and 10 attributes were added to each mapping. In all cases half the attributes were string attributes and half were integer attributes and one thread was used to perform the inserts.

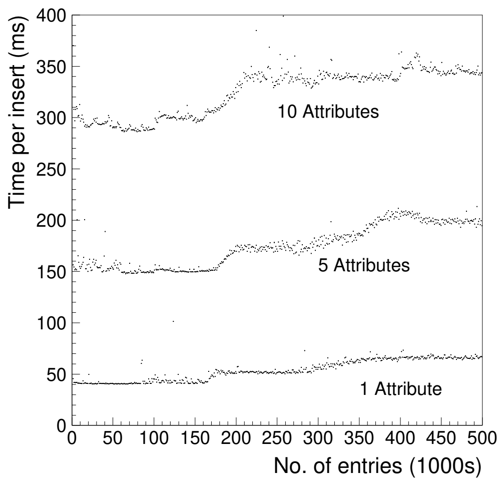


Figure 4.16: Variation of RMC insert time with different numbers of entries in the catalog and different numbers of attributes.

It can be seen that adding an attribute to an alias takes an equivalent amount of time to adding the mapping itself, i.e. 25-30 ms. The number of attributes added and hence the increasing amount of information held by the database does not affect the insert timings, as the three sets of results in Figure 4.16 have essentially the same shape as the results for 1 thread in Figure 4.14, in that there is a very gradual increase in insert time as the number of entries in the catalog increases.

4.3.3 Command Line Interface

The command line interface is implemented in Java using the Java API. Table 4.4 shows some timing statistics giving the time to execute different parts of the command

Time (s)	Operation
0 - 1.0	Start-up script and JVM start-up time
1.0 - 1.1	Parse command and options
1.1 - 2.1	Get RMC service locator
2.1 - 2.3	Get RMC object
2.3 - 3.0	Call to the rmc.addAlias() method
3.0	End

Table 4.4: Timing statistics for adding a GUID to LFN mapping in the RMC using the CLI.

`addAlias` (in insecure mode) used to insert a GUID to LFN mapping into the RMC. The results are very similar to the LRC command line interface results shown in Table 4.2 and the same conclusions can be drawn, that the command line interface is about 2 orders of magnitude slower than the average time taken using the Java or C++ API, and so is not recommended for large scale operations.

4.4 Replica Optimization Service

Given a set of replicas of a particular data file that are spread over different locations on the Grid, the Replica Optimization Service (ROS) finds the one which can be accessed in the fastest time, according to network bandwidth information. Within the EDG testbed, replica selection is based on the network cost functions provided by the network services described in Section 3.1.5. Network monitoring was performed on the five major testbed sites, namely CERN, CNAF (Bologna), IN2P3 (Lyon), NIKHEF (Amsterdam) and RAL (Oxford). The estimation of network transfer time is used by the Replica Optimization Service to locate the best replica. Experiments showed that the estimated transfer times for files with sizes greater than around 100 MB corresponded well to the real transfer times. On average, estimation errors were lower than 15%, and for links between IN2P3 and NIKHEF less than 3%.

4.4.1 Java API

The API method *getBestNetworkCost()*, given a set of SURLs corresponding to replicas of a particular file, returns best SURL to access and the network cost (i.e. time) to access the SURL. The ROS calls the network monitoring service, supplying it the file size and locations of all replicas, which then returns the estimated transfer time for each replica. From this the ROS calculates the best replica to use.

A test was performed to measure the response time of calling *getBestNetwork-Cost()* and, in order to measure the overhead of the ROS, these results were compared to calling the network costs directly, i.e. without the ROS. The method was called a certain number of times and the average response time for each call is shown in Figure 4.17.

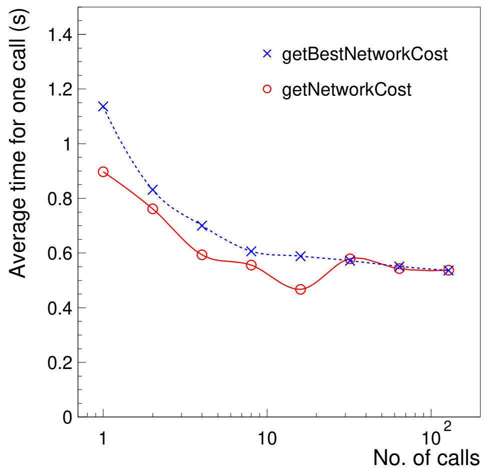


Figure 4.17: Average time to call *getBestNetworkCost()* and *getNetworkCost()* of the ROS with different numbers of calls to the methods.

It is important to note that the response time of the first call to *getBestNetwork-Cost()* is larger than for the successive calls. This is due to the time spent dynamically loading the required Java classes. Both sets of results show a gradual decrease in the average response time for each method. The difference between calling the ROS and the network monitor directly is very small, especially when more calls are made and thus it can be seen that the overhead of the ROS is minimal. This is expected since the ROS simply gathers information from other sources and performs a very simple calculation; most of the time is spent at the client-server communication level.

In the next set of tests the scalability of the network costs provider was measured by calling the ROS with multiple client threads. *getNetworkCost()* was called 32 times for various numbers of concurrent threads. The workload was evenly distributed among the threads running on the client node. For instance, in the case of one thread, the ROS client issued 32 calls; in the case of two threads, each ROS client issued 16 calls; finally, in the case of 32 threads, all 32 ROS clients issued 1 call each. The time was measured from the beginning of the first call until all client threads received the results back from the ROS.

The previous results showed that the bottleneck of retrieving the network costs of the best replica is not the ROS but the central network monitoring archive. Thus, these tests essentially evaluate the scalability of the network monitoring service. The results in Figure 4.18 show the response time for the 32 calls is reduced by around 10% going from 1 to 32 threads.

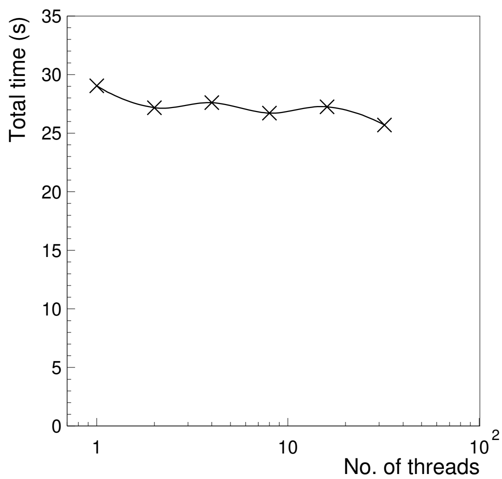


Figure 4.18: Total time to call *getNetworkCost()* 32 times with different numbers of concurrent threads.

This is not as large a reduction as was seen with the RLS and RMC services and could be due to the fact that while the ROS can handle multiple threads accessing the service, the underlying network monitoring service cannot.

4.4.2 Command Line Interface

As with the other data management services, the ROS command line interface is implemented in Java using the Java API. Table 4.5 shows some timing statistics indicating the time to execute different parts of the command `getNetworkCost` used to obtain the cost of transferring a file between two sites.

Time (s)	Operation
0 - 1.2	Start-up script and JVM start-up time
1.2 - 1.4	Parse command and options
1.4 - 2.7	Get ROS service locator
2.7 - 4.4	call getNetworkCost() method
4.4	End

Table 4.5: Timing statistics for calling `getNetworkCost` in the ROS using the CLI.

The total time to execute the command was about 4.4 s. The start-up script and the time to start the Java Virtual Machine took almost 1.2 s. After parsing the command line it took 1.3 s to get the ROS service locator. The time for calling the method *getNetworkCost()* is some 1.7 s which includes calling the remote network monitor service running on a different site. The same conclusions can be reached as for the other services - the CLI is very much slower than the API due to the overheads involved in the Java implementation, which is worse for the ROS than for the other services due to the ROS's use of an external service whose classes have to be dynamically loaded every time the command is used.

4.5 Replica Manager

The Replica Manager acts as an entry point to all the data management services. It uses the RLS and the RMC for cataloging data and the ROS for optimising file access.

In the original design of the data management components, it was envisaged that all operations would be performed via the Replica Manager, which would deal with the other services internally. In other words the services could not be accessed directly but only through the Replica Manager acting as a client. However, at the request of users, CLIs and public APIs were made available for all the services, even though almost all of the required functionality can be provided by the Replica Manager with little overhead.

The Replica Manager is implemented as a client side tool only. This has several advantages like easy installation and configuration but some disadvantages such as no server side processing to perform for example file check-summing, queued file transfers, check-pointing/restarting and rollback mechanisms. The Replica Manager provides four types of commands:

- **Management commands**: These commands combine data transfer and registration in the catalog services.
- **Catalog commands**: Interfaces to the catalogs. The basic functionality has been covered in the RLS section above.
- **Optimization commands**: Replica optimisation commands for optimised file locations and transfers.
- **File transfer commands**: Basic (third party) file transfer mechanism without updating the replica catalog.

Since the Replica Manager is a client side tool that effectively combines all the clients of the services described above (RLS, RMC, ROS), its performance depends mainly on the performance and scalability of these services. The only part of the Replica Manager that has not been tested so far is file transfer operations, so this is examined briefly in this section.

The Replica Manager uses primarily the GridFTP protocol [108] and thus has similar performance characteristics to conventional GridFTP file transfers. However, since the Replica Manager does additional existence checks and the Java implementation creates some overhead, the file transfer is slower by a constant factor as compared with a pure GridFTP file transfer using for example the Globus command `globus-url-copy`.

For the end-user it is of major importance to know how many files can be transferred without errors using the Replica Manager. For this purpose a large scale test was run which involved transferring 1,800 1 GB files between NIKHEF (Netherlands) and CNAF (Italy) and registering them in the RLS using the Replica Manager command line interface with the command `copyAndRegisterFile`.

The 1.8 TB of data was successfully transferred in around 62 hours. Each file transfer plus registration took between 1m45s and 1m49s, depending on the network throughput. The files were transferred using parallel GridFTP streams. As compared to a single file transfer using `globus-url-copy` (1m35s), the Replica Manager has an overhead of around 12 seconds due to the cataloging (RLS) and the file existence checks (Storage Element). This overhead can partially be overcome using the command `bulkCopyAndRegisterFile` where the additional startup time of the Java Virtual Machine and some class loading are negligible as they are only needed for the first file transfer.

4.6 Security

When dealing with large amounts of potentially sensitive data, it is of great importance to restrict read and write access only to trusted users. The edg-java-security package, described earlier in Section 3.2.7 interacts with all the other data management services to enable secure access to each service. However, adding a layer of security inevitably adds delays to each operation, as the security package has to make decisions on whether to permit access to the services. In this section the quantitative effect this has on the performance of the services is examined.

4.6.1 Java API

Using the security modules creates an overhead due to the creation of a secure communication channel between client and server, the authorisation of client requests according to a certain access control policy, etc. Table 4.6 shows the time to insert a number of GUID to SURL mappings into the LRC using a Java client with a secure and insecure service.

Mappings	Secure service (s)	Insecure service (s)
1	0.77	0.07
10	7.07	0.54
100	55.44	3.38
1000	527.12	28.61

Table 4.6: LRC Java client performance with security on and off.

There is a roughly constant overhead to each operation when using security due to the reasons discussed above. Since the security mechanisms are used every time the method is called the effect is cumulative with the number of times it is called and means the operations take 10 to 20 times longer than they do without security. A comparison between insert and query times, using a similar set-up to that which gave the results in Figure 4.6, is shown in Figure 4.19.

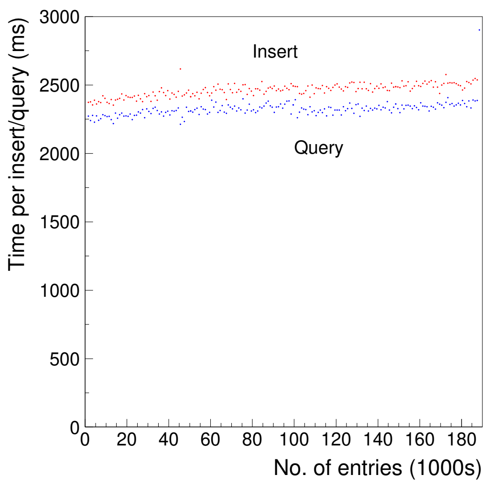


Figure 4.19: Time to insert mappings and query one GUID for different numbers of entries in the LRC, using 5 concurrent inserting clients and 5 concurrent querying clients, with security enabled.

It can be seen that the query times are faster than insert, as before, but the times for both hardly vary with the number of entries in the catalog. This is due to the large effect of security, which does not depend on the size of the catalog, making the time for the database operations and SOAP messaging negligible by comparison.

Some timing statistics for the server-side processing involved in calling the *addMapping()* method with security are shown in Table 4.7. Note that this was not the first time the method was called, hence the Java class loading which is done the first time a class is used was done in a previous method call and does not affect the times given here.

Time (s)	Operation
0 - 0.061	Initialise security classes
0.061 - 0.115	Examine certificate and get keys
0.115 - 0.250	Find CA certificate to validate client certificate
0.250 - 0.292	Accept client certificate
0.292 - 0.299	Database operations
0.299 - 0.304	Return from method
0.304	End

Table 4.7: Server-side timing statistics for calling *addMapping()* using the Java API with a secure LRC.

Firstly, around 60 ms was spent constructing and initialising the security classes required, then it took 55 ms to check the client certificate was a valid Grid certificate. In order to find out if a client can be entrusted to perform the operations on the catalog, its certificate is validated against the Certificate Authority (CA) certificate of the CA that issued it. It took 135 ms to find the correct CA certificate and a further 40 ms to check the client certificate against it. The database operations to modify the catalog took 7 ms and there were a further 5 ms until the eventual return from the method.

In total the method call took 304 ms, which compared to the average of 25 ms it takes when not using security, is a long time, but the security overhead caused by the validation of certificates is unfortunately unavoidable. However, if there was a method for performing bulk operations on the catalog, this would mean the security check would only take place once and so the effect would be much smaller.

4.6.2 Command Line Interface

In previous results given which did not use security the time to perform an operation on a catalog service using the command line interface was at least 3 seconds. Therefore the relative effects of adding security, which mainly affects server-side operations, should be less than with the API. The timings observed for using the command `addMapping` on the LRC using the CLI are shown in Table 4.8.

Time (s)	Operation
0 - 0.7	Start-up script and JVM start-up time
0.7 - 0.8	Parse command and options
0.8 - 1.5	**Look for VOMS credentials in client certificate**
1.5 - 2.3	Get LRC service locator
2.3 - 2.4	Get LRC object
2.4 - 3.7	Call to the lrc.addMapping() method
2.4 - 3.0	Class loading
3.0 - 3.6	**Security check**
3.0 - 3.03	**Examine certificate and get keys**
3.03 - 3.36	**Class loading**
3.36 - 3.60	**Find CA certificate to validate client certificate**
3.6 - 3.7	Database operations and end command
3.7	End

Table 4.8: Timing statistics for calling addMapping using the LRC command line interface (security parts in bold).

The times for the parts not involving security (starting JVM, parsing command line options etc.) are similar to those observed in Table 4.2. Security adds delays in two places: checking for VOMS credentials in the client certificate and authentication on the server-side. The process of examining the client certificate for VOMS credentials takes 700 ms and the security authentication takes around 600 ms, twice as long as with the API, due to the class loading that takes place.

In summary, security increases the time to insert a mapping into the LRC by around 20% when using the command line interface and therefore is not a major hindrance to that performance. However, for normal use when performance is important the API should be used in preference to the CLI, and with the API security does severely affect the performance, slowing it down by a factor of at least 10. But in certain applications, such as medical imaging, security is of paramount importance and therefore the penalties of using security are unavoidable.

4.7 Summary

In this chapter the performance of several components of the European DataGrid data management software has been examined. Comparison of timing statistics from C++ and Java clients has shown that their performance is similar but C++ has a slight advantage. For each call to a catalog service (RLS or RMC), the database operations on the server take under 10 ms (for an empty catalog) and the rest of the time is spent in messaging between the client and server.

The time to query a catalog (20 ms) is independent of the number of entries within it, up to around 1 million entries, and the performance penalty at this point is due to the underlying database. The Java API, when used to add entries, gives poorer performance with a higher number of mappings, whereas the C++ API gives roughly constant timings. However, the catalogs can easily handle up to 200 concurrent operations and in fact throughput reaches a maximum when 10 - 20 concurrent accesses take place.

Table 4.9 shows, for the typical use case described in Figure 3.9 of a user copying a 1 GB file currently on the Grid to another Grid site, the approximate time for each operation using the Java API with secure and insecure services. The figures are given assuming this was not the first time the services had been used, therefore the effects of class loading are not included.

Operation	Service used	Time (secure) (s)	Time (insecure) (s)
getGuid	RMC	0.3	0.02
listReplicas	RLS	0.3	0.02
listBestFile	ROS	0.9	0.6
copyFile	RM/SE	95.0	95.0
registerFile	RLS	0.3	0.02
Total		96.8	95.66

Table 4.9: Timings for each part of the use case described in Figure 3.9 when using secure and insecure services.

Clearly, when operations involve the transfer of large files (1 GB), the overhead of data management services, whether secure or insecure, is negligible ($< 2\%$). However, if the file sizes are MB in size, they could be transferred in less than one second and in this case the services overhead would be of the same order of magnitude as the transfer time. For those operations which do not involve data transfer, such as a simple query for a set of data, the reaction time of the service is the limiting factor and here there would be a large difference in performance when secure services are used, however further work on bulk operations could possibly limit these effects. Overall, the services cope well under stressful situations, dealing with many concurrent users placing high demands on them and efficiently handling large amounts of data. The RLS has already been used in a production context in the LHC Computing Grid for a CMS data challenge from March to May 2004, when it was used to store information on over two million replicated files.

Chapter 5

Grid Optimisation

So far we have described how Grids work and shown that Grid middleware services have been developed to the extent to which they can be used in production environments. Once the concept has been shown to work, it is important to make it work as efficiently as possible. In this chapter the process of optimising the use of the Grid will be examined. The two main areas of relevance, especially for a Data Grid, are optimising job scheduling and replication of data. The aim in both of these areas is to minimise the time spent accessing data whilst ensuring resources are used to their full potential. The scheduling policy proposed here is based on minimising the estimated access time to data for a running job. It aims to schedule the job as close to the data as possible whilst not overloading resources. For replica optimisation we present an economic model for file trading, with each Grid site buying and selling files to increase its own profit. As a result of this local optimisation it is hoped that replication and data access will be optimised globally.

This chapter is structured as follows: firstly the issues surrounding distributed scheduling and how it differs from local scheduling are discussed. Some answers to the Grid scheduling problem are then presented for comparison with our own solution based on the estimated data access time. Replica optimisation and its various stages are described, then the architecture and reasoning of the economic model we propose is discussed in detail.

5.1 Scheduling Optimisation

Scheduling is the process of deciding when and where a task should be performed in a system of one or more elements capable of performing the task. In a localised cluster computing environment a central scheduler distributes jobs to homogeneous worker nodes and, since the scheduler has full knowledge and control of the resources available, informed scheduling can be done based on well-known algorithms. In a distributed Grid environment there are more factors to take into consideration and the scheduling decision can become complex.

5.1.1 Local Scheduling

Schedulers for clusters or farms of PCs have generally dealt with environments where the resources are dedicated and the scheduler has complete control over their use. Typically all the nodes are homogeneous, or at least, a typical user job can run on any of the nodes available. When a job is sent to the scheduler the user specifies a limit on the time they expect it to run for, the number of processors required (if sub-jobs can run in parallel) and the executable. The job is held in an ordered queue, which may be first come-first served, or users may have different priorities assigned to them to decide their position in the queue. The scheduling policy is completely under the control of the administrator of the system as is the policy of which nodes can be used when and by which user.

Among the many schedulers that currently exist are the Portable Batch System (PBS), Load Sharing Facility (LSF), Network Queueing Environment (NQE) and IBM's LoadLeveler. PBS is used for local job scheduling in the ScotGrid cluster (see Appendix A). All these schedulers operate in a similar way, following the client-server model. A server runs the scheduling service, accepting job submissions from clients, queueing the requests, submitting the jobs to processing nodes and returning the results to the client when the job finishes. The scheduling process is transparent to the users, and much the same should be true for a Grid scheduling process.

5.1.2 Distributed Scheduling

A distributed environment presents many challenges which make scheduling a more complicated task. The generalised architecture of a Grid is shown in Figure 5.1. Users submit jobs via a User Interface to the Workload Management System (WMS). The Resource Broker uses information on the Grid resources to find candidate sites for each job and then schedules the job for submission to the most suitable site. The WMS distributes jobs to the Computing Elements, which each do their own internal local scheduling, as described in the previous section.

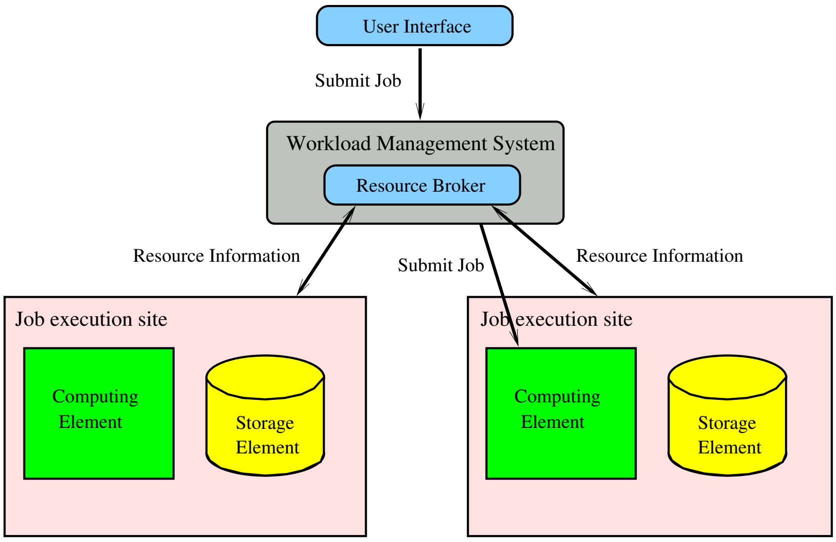


Figure 5.1: Generalised architecture of a Grid, showing the components related to scheduling.

The Resource Broker decisions are complicated by several factors: firstly, the resources are heterogeneous - anything from a single PC to a cluster to a supercomputer could be linked into the Grid, each with a variety of architectures and operating systems, and so the scheduler cannot assume all resources are suitable for all jobs. These resources may not be exclusively for Grid use and may be further restricted to only allow certain Grid users to run jobs, and so the scheduler has to fit in to the local policies defined at each site. This also brings in the issue of security, in that the local administrators have to trust the Grid scheduler not to abuse their resources, and so the scheduler must have security mechanisms so that it can trust its users. Linked to the policy issue is the idea of accounting for use by different groups or Virtual Organisations. In an academic environment paying for resources used is not such a big issue but for commercial Grids there must be a way for users to tell the scheduler what they are willing to pay for, and the scheduler must deliver the required quality of service.

On a more technical level, resources on a Grid may go up or down without warning, due for example to interrupted network connections or the system being deliberately taken off-line for hardware updates. The scheduler must therefore know what the state of the Grid is at scheduling time and any advance scheduling would clearly be risky. There must be some way to match up the resources required by a job and resources provided by the Grid, for example limits on running time, operating system and memory available. Mechanisms for this are discussed in the next section.

Many distributed systems are flexible in the jobs that each site can run because the scheduler can ship the application software to the job execution site. This means there is no pre-requisite for the application to be installed on the execution node, it can simply be installed and uninstalled on the fly before and after the job runs. However, in the case of high energy physics the size of the applications (on the order of Gigabytes) makes this approach infeasible, and so the scheduler must take into account requirements on the (pre-installed) software available at each site.

5.1.3 Grid Scheduling Policies

Much research has been carried out in the field of distributed scheduling, however most of it does not take data requirements into account. In this section, a solution to the resource matching problem is outlined, then an overview is given of some policies which do not look at data requirements, some which do, some economic approaches and finally our scheduling policy based on the access time to data.

Resource Matching in a Grid Environment

In an environment in which jobs with differing requirements are to be executed on a Grid with various types of resources, there must be some matchmaking service that ensures a suitable resource is found for each job. The ClassAd language [110] was written to describe jobs and resources in Condor [90] (the name is derived from classified advertisements in newspapers). Using this language, jobs can advertise their needs and resources can specify what they are willing to supply. A matchmaking service uses the ClassAds to find resources that match the jobs.

The language uses key-value pairs to express required values for certain attributes of the job and attributes of each resource. The job ClassAd can then define a ranking mechanism for the resources that match its requirements, such as processor speed or number of free CPUs. In a Grid many separate entities are used in the execution of a job and so an extension to the matchmaking framework called gangmatching [111] was introduced to overcome the limitations of bilateral matchmaking, where one job is matched to one resource. Using gangmatching, a job can specify requirements on multiple resources such as CPUs, storage and even requirements such as software licenses which may vary between sites.

ClassAds are used as the Job Description Language for the Workload Management System of EDG (Section 3.1.2) and users can specify input and output data sets as well as the usual hardware and software requirements of the job.

Non-data-centric Scheduling

There are several groups which have proposed solutions to distributed scheduling, but which do not take data requirements into account. In [42] an iterative approach is used to schedule multiple (possibly linked) jobs simultaneously, finding an optimal group of resources with just enough machines to run the jobs. It was found in [74] that continual adaptive scheduling, where job times are monitored and used to predict future job times, worked best when the jobs all ran for similar lengths of times but a first-come first-served policy was better in other situations. A novel approach where a job is submitted to many queues is presented in [122]. When the job is ready to run on one resource it notifies all the other queues to remove the job. In [70], it was found that for parallel jobs, using the maximum available resources possible for each job gave the best results. This kind of research has been on-going for many years but the main theme behind all these ideas is that scheduling must be adaptive to the changes in the environment.

Data-centric Scheduling

As the area of data-intensive processing in a distributed environment is relatively new, much less research has been carried out considering data location in scheduling decisions. In [124], I/O communities are defined which bind together job execution and storage sites. The communities locally share data, but they are free to define exactly what "locally" means themselves (for example the data could really be on site or the site could be willing to cache from a remote site). ClassAds are used for matchmaking and they can specify data and rank sites on the locality of this data.

Data location is considered in the scheduling algorithms discussed in [3]. Directed Acyclic Graphs are used to define related tasks and sub-tasks, and the scheduling policy is based on heuristics like Min-Finish or Max-Finish (run first the task with the shortest completion time or longest completion time respectively). However the scheduling for all the tasks is done at the start and the scheduler assumes static Grid conditions - an assumption which would not be true in a real Grid environment.

Scheduling jobs to sites that already contain the required data is shown to give the highest job throughput in [112]. The assumption is made that the data required for a job is all in one file, therefore all the data is either available on site or not. In our case jobs may require multiple files which may be distributed over several sites. Thus scheduling the job to the data is not as simple as finding where the data is and sending the job to where it can access all the data locally.

Economic Scheduling

Economic approaches to Grid scheduling [24, 25, 51] involve resources competing to provide a service (time on a processor) for a price (real money or "Grid credits").

This model solves many of the problems of scheduling in a distributed heterogeneous environment. If the local resource policy states that certain users cannot use the resource, it can price its resource out the range of these users or simply refuse to sell it to them. By making the resources compete for the users money, much of the work is taken away from the scheduler, since the resources compete to gain tasks from the scheduler, rather than the scheduler having to look for resources itself. This "pull" model is in contrast to the traditional "push" model used by most scheduling systems. Users place constraints on the deadline and/or budget for their jobs and the scheduler must try to finish all the jobs before the deadline without spending more than the allocated budget. The use of economic models in a Grid environment will be discussed in more detail in Section 5.4.

Scheduling using Access Costs

If a job is data-intensive then the scheduler should make sure the job is scheduled to where it can most easily access the data required. The scheduling algorithm we propose [27] for these types of jobs is based on the "access cost" of obtaining the data for the job. Access cost here is not monetary cost but the estimated time to copy all the files needed for the job into a local cache or storage so the running job can access them directly. From a set of sites that are suitable for the job (according to local policy and the job requirements), the job is scheduled to the site with the minimum access cost.

The data requirements for each job are supplied to the scheduler in the form of unique identifiers for the files containing the data (LFNs in the LFN:GUID:SURL model used in EDG). The files may have multiple replicas distributed around the Grid, so network monitoring (Section 3.1.5) and replica location services (Section 3.2.3) or other mechanisms described in the next section are used to find the copy that can be accessed in the fastest time from the site on which the job is running. This time is the access cost of one file and the access cost of the job is the total access cost of all the files required. From a set of potential sites that satisfy all the other requirements for the job (operating system, minimum CPU speed etc.), the scheduler submits the job to the site with the lowest access cost for the job. This ensures the data is retrieved as quickly as possible and hence minimises the main time-consuming factor in data-intensive jobs.

Here we make several assumptions about the nature of data-intensive jobs and use high energy physics analysis processes as a model. Our definition of a job is a task that must access a list of files and perform some computational processing on the data in the files. This processing could possibly lead to output data, but the size of this data is very small compared to the input data in the type of job in which we are interested. We assume that the process time of a job is related to the amount of data required, i.e. the process time per file is approximately constant (for files of approximately equal size). Therefore the limiting factor in the time to run a job will be the time taken to access the data, and scheduling policies which use the access cost metric should be the most effective at maximising job throughput.

Using the access cost of the current job alone does not take into account the workload of the Grid sites, therefore we introduce an extension to the access cost policy, queue access cost. The queue access cost is the sum of the access costs of all the jobs waiting in the queue at a site and the job is submitted to the site with the minimal queue access cost value. The queue access cost gives a rough estimate of how long a job would have to wait in a queue before it starts running. Queue access cost therefore takes into account data location and workload in one metric. Results presented later in Chapter 7 will show that for most situations queue access cost scheduling out-performs other policies, giving lower run times for users' jobs and making the best use of available resources.

5.2 Replica Optimisation

In the Data Grid environment the availability of data and efficient access to it is of prime importance. Replicating data to different locations increases the likelihood of data being available and decreases the time to access the data. However, in a Grid with finite storage resources and network bandwidth, replication must be controlled in such a way as to maximise the job throughput with the resources available. The solution we propose to handle data replication uses an economic model combined with an auction protocol for file trading over a peer-to-peer (P2P) network between sites. Agents located on each Grid site use the file trading mechanisms to optimise data access for jobs running on their local site, and as a result data access across the whole Grid is optimised.

Our model is primarily designed to address the issue of latency, i.e. providing the most efficient access to data for jobs, rather than general availability of data. Many replication solutions are designed for P2P networks with a high level of unreliability for each node and which deal with relatively small files. P2P nodes may be users' home computers for example which are switched off every night. The Data Grids being constructed for large scientific experiments such as those at the LHC (Section 1.3) consist of sites which will be generally reliable, only going offline in the case of scheduled downtime or serious failure, and so data on these sites should generally be constantly available. Even though the number of files involved is similar in both situations (millions), the files in high energy physics are Gigabytes in size compared to Megabytes in P2P networks, hence the efficient access to this size of data becomes much more important than availability.

Replication studies were first applied to databases [1, 133] for cases where high availability was required. Databases or parts of databases were replicated among different sites using algorithms which depended on the previous read/write access patterns on the data. Since the data in the database could change, maintaining consistency across all replicas was a big issue. However, in Data Grids most data is read-only and hence there are few concerns over consistency between replicated data.

Replication strategies may be used to control the number and location of replicas of a given data file, or there may be an implicit strategy which brings about an optimised network for the users. The Gnutella network [80] for example performs no replication itself but popular files automatically become more widely available as more copies are downloaded by users who then make them available from their own computers.

Freenet [38] uses a replication strategy in which replication is triggered when the number of requests for a file reach a certain level. In return for the use of the network, users must provide a portion of their own storage space which can be used by the network to store any replicated data. This data is encrypted so users cannot be held responsible for any illicit data FreeNet places on their computer. Storage is managed using a Least Recently Used algorithm, where the file least recently accessed is deleted to make room for a new replica. File popularity also triggers replication in [112]. Here, the file is either replicated to a random site or to the site with the smallest queue of jobs waiting to run, depending on the strategy used. This creates a global distribution of replicas related to their popularity but does not give optimum local availability for replicas at a particular site. A similar approach is followed in [114], whereby at set time intervals the availability of a file on the network is calculated. If it is too low, additional replicas are created and if it is too high, replicas are deleted. This strategy is also used in [113], where a set level of availability is maintained for each file by creating new replicas and deleting excess replicas. When a decision has been made to create a new replica, the destination site is chosen to be the site which is predicted to have the highest number of future requests for the file.

The difference between these strategies and our economic strategy is that we have agents on every site which only aim to optimise file access for the local site. This means the agents do not need to directly know about the global state of the Grid. Global optimisation is the emergent result of local optimisation.

In the EDG Replica Optimization Service (ROS) (Section 3.2.5), automatic replication takes place whenever a file is requested that is not available on local storage (storage on the same site as the Computing Element running the job). When a job requires a file, it contacts the Replica Location Service to obtain the locations of all the replicas then uses the ROS to find the best one to access. If a replica already exists on local storage, this one is used but if not, the replica which can be accessed in the fastest time according to the current network traffic is copied to the local storage then accessed from there. Like Gnutella, the most popular files become more widespread automatically as a result of the replication strategy.

In our model, replica optimisation is performed by agents, or *Optimisers*, one for each Grid site. The architecture and interactions of the Optimisers with other Grid components is shown in Figure 5.2. The Computing Elements gain access to files required for the jobs they are running via the Optimiser inside the Replica Manager.

The Optimiser controls the files stored on its local Storage Element and interacts with Replica Managers on other sites to copy files around the Grid.

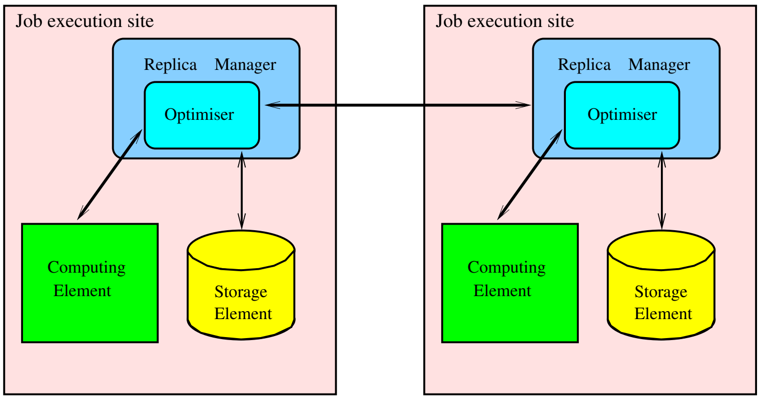


Figure 5.2: Optimiser interactions with other Grid components.

Every Optimiser has two goals: (1) to minimise a single job's execution time and (2) to maximise the usefulness of locally stored files. Users want their jobs to be executed in the quickest time; therefore an Optimiser aims to minimise the execution time of every job that is executed in its Grid site. The Optimiser therefore aims to keep locally those files which are most useful for the jobs that are being executed currently and for potential future jobs.

Whenever an Optimiser is considering a file request, it executes the following three tasks:

- *Replication Decision.* If a requested file is not present on a site's local storage, this process decides whether local replication of this file should take place. If the Optimiser decides not to replicate a file then the job must remotely access that file, copying it to a temporary local cache.

- *Replica Selection.* This process selects the best replica to replicate locally or to copy to the local cache from those currently available throughout the Grid. In general, the selection criterion depends on the chosen evaluation measure.

- *File Replacement.* When a file has been selected for replication to the site's local storage, there might not be sufficient spare capacity. In this case, one or more local files must be deleted. The selection criterion for deciding which locally stored files to delete depends on some estimate of future value in keeping each file.

A specific combination of the algorithms for each stage defines a replica optimisation strategy. Some strategies that we have implemented and tested in the Grid simulator OptorSim (see Chapter 6, [13, 14]) are shown in Table 5.1.

Strategy	Replication Decision	Replica Selection	File Replacement
No Replication	Never Replicate	RLS + Net Mon	None
Least Recently Used	Always Replicate	RLS + Net Mon	Least Recently Used
Least Frequently Used	Always Replicate	RLS + Net Mon	Least Frequently Used
Economic Model	Prediction Function	Auction Protocol	Prediction Function

Table 5.1: Examples of replica optimisation strategies.

5.2.1 Replication Decision

In a Data Grid, when a file is required by a job and is not available on local storage, it may either be replicated to local storage and read from there, or read remotely to a temporary cache from the storage on some other site. Therefore a replica optimisation strategy must make decisions on whether or not to copy each file into local storage. If the file is replicated locally, the next time it is requested the job can read it quickly and the time to complete the job will be reduced[1]. But if replicating the file required the deletion of other files, future jobs requiring those files which were deleted will take longer. A balance must therefore be found whereby only the most popular files are stored locally. The simplest strategies are to never replicate or to always replicate to local storage. Never replicating would result in very bad performance and create huge bottlenecks in the network, whereas always replicating (as in the EDG ROS) tends to create more replicas of the more popular files, as mentioned previously.

The best solution is to evaluate on a file by file basis whether a file is worth having compared to all the other files already there. This requires an algorithm to predict

[1]It is assumed that reading a file from storage on the same site is much faster than reading from a remote site over a wide area network.

the future "value" of each file, based on some information such as previous file access history. In [30] an economy-based prediction function was presented which used patterns of previous file access history to predict future file accesses. The optimisers made a decision to "invest" in a file (i.e. replicate it to local storage) if the prediction function showed profit could be made from selling the file in future. This prediction function is explained in more detail in Section 5.4.

5.2.2 Replica Selection

Whether a file is to be replicated or copied to a local cache, a remote source file must be selected from among the existing replicas currently on the Grid (assuming there is at least one copy available). There are several ways in which this can be done: looking up the locations of other replicas and using the network status to pick the best one, via a P2P auction process, or using ClassAds. The first case has been described already for the EDG ROS and selection based on ClassAds is described in [128]. The P2P auction process we have developed as part of the economic model is described later in Section 5.4.

5.2.3 File Replacement

When the decision has been made to replicate, a suitable source file has been found, and the local storage does not have enough space to accommodate the new replica, one or more files currently on the local storage must be deleted. The situation of file replacement on Grid storage resources is analogous to the well-researched problem of cache management in a single CPU's memory and disk space. Reading data from memory takes a fraction of the time that it does from disk, so it is desirable to keep the most commonly used data in a memory cache. Unlike a Grid, where files may be read remotely from other sites, data must be read into memory before it is read by the CPU, hence a careful choice has to be made on which data to delete. On the Grid, the algorithm must also take into account that some data should not be deleted, such as files that are currently being used and files that are the last remaining replica of a particular set of data. This latter problem can be solved by using "master" replicas which cannot be deleted, and these could be stored at the location where the data is first produced, or it may be more efficient to sometimes change which replica is the master replica. Several cache management algorithms as well as the file value prediction function used in the economic model are described in the next sections.

5.2.4 Other Problems

There are of course other issues of a social rather than technical nature to solve in the Grid environment before any algorithms could be used to manage storage space. Optimisation algorithms must be given complete control of what is stored at each site on the Grid, and the owners and local administrators of the resources on these sites may not be happy allowing the Grid this level of control. Although the use of a Grid is supposed to be transparent, many users like to know exactly where their files are and would rather not trust the Grid to move their data around continuously. These issues are outside the scope of this work; here we concentrate on finding the algorithm which gives the best performance technically, not which is the most acceptable to users.

5.3 Non-economic Replica Optimisation Strategies

The non-economic replica optimisation strategies we investigate (Least Recently Used and Least Frequently Used in Table 5.1) always replicate to the local storage and use a replica location service along with network measurements to make the replica selection. The file replacement algorithms are taken from recent studies into cache management.

Two traditional cache replacement algorithms, on which almost all others are based are Least Recently Used (LRU) and Least Frequently Used (LFU). In the LRU algorithm the file or block of data which has the oldest last access time (i.e. was accessed least recently) is selected for deletion. Using LFU, the file or block of data which has been accessed the least number of times in the past is deleted. They both have their own advantages and disadvantages. LRU is very simple, requires little information to be stored and has a low complexity. However just because a block has been accessed recently does not necessarily mean it is a popular block, so little used blocks can stay in the cache for a long time. LFU, on the other hand uses more information and so can make well informed decisions on the potential usefulness of each block, but all this information takes up space and is time-consuming to process. Most current systems for memory caching favour simplicity over accuracy and use the LRU algorithm for cache replacement. Much recent research focuses on combining the simplicity of LRU with the performance benefits of LFU.

LRU-K [100] uses the time to the K^{th} most recent access as an indicator of which block to delete, making it as accurate as LFU but as simple as LRU. The 2Q algorithm [77] divides the memory cache in two, using one section for a main queue in which proven frequently accessed blocks lie and the other for new blocks yet to prove if they are worth promoting to the main section. Least Recently Frequently Used (LRFU) [83] is similar to LFU but weights each reference to a block according to how long ago the reference was, so that blocks which were accessed relatively often in the past are more likely to be deleted than blocks accessed recently. The Low Inter-reference Recency Set (LIRS) replacement policy [75] uses the number of blocks referenced between two consecutive references to a block to evaluate the least useful blocks.

The use of cache management algorithms has also been investigated for web proxy caching, where to reduce network traffic, the most visited pages are held in a local cache [20]. The Least Cost Beneficial based on the K backward references (LCB-K) algorithm [104] was developed for Storage Resource Managers for use in a Grid environment. In general, all these algorithms give similar to or slightly better performance than LFU, but most require various parameters to be tuned to be most effective, or only work best for certain cache sizes or access patterns. Therefore only LRU and LFU are investigated in detail and compared with the economic model in Chapter 7.

5.4 Economy-based Replica Optimisation Strategies

The use of economic models has been discussed briefly already in the context of optimised job scheduling. Here, we apply an economic model to the process of replica optimisation. In our model the Optimisers on each Grid site are independent agents responsible for providing the fastest access to data for the jobs running on the site. Optimised access to data should be the outcome of the Optimiser's main aim, which is to maximise its own profit. The Optimiser trades via an auction protocol with other sites, buying files it hopes it can sell to Computing Elements on its own site and to other Optimisers. A prediction function based on previous file access history and some assumptions of file similarity (explained later) are used to determine the relative worth of files. Through this process of locally optimising access at each site, global optimisation should result.

5.4.1 Reasons for using an Economic Model

There are many reasons for employing an economic model, some of which are related to those advantages stated earlier for economic scheduling policies. Using this model we are able to make optimised replication decisions over a very large parameters set in a distributed manner. Centralised systems do not scale well and the Grid optimisation service must be able to deal with up to thousands of nodes and millions of files totalling Petabytes of data. By restricting the scope of each Optimiser to local optimisation we make the reasoning problem far more manageable, and by allowing economic interactions to lead the system toward global optimisation through emergent market-place behaviour, we also make the system optimal.

The optimisation service needs to be both reliable (it should re-optimise its configuration to account for failures elsewhere on the Grid) and robust (no single point of failure should be capable of bringing down the system). Optimisation based on an economic model is fully distributed in nature and is therefore reliable and robust. The Grid is a dynamic environment in which the state of resources can change without warning. Thus it is extremely important to be able to perform dynamic optimisation while jobs are executing. By using an economic model we can exploit the dynamism of the market to take informed decisions at job execution time. We do not need to know a priori at job scheduling time which file replicas will be accessed for a particular job - all the more important since some jobs will not know the data they need to access until they are actually executing.

An economic model provides a natural framework for possible accounting and payment mechanisms for Grid resources. In general a Grid is set up by various Virtual Organisations (VOs), which contribute differently to its establishment. These VOs are made up of people from different institutes or businesses which invest different amounts into the resources which make up the Grid. It is likely that the VOs and the real organisations need a way to regulate the use of the Grid according to the effort that each of them made to establish it. An economic model provides a suitable way to mediate this interaction between organisations. Even if no real money is involved in assigning resources, as in most scientific research Grids, users may be given "virtual" money according to their status. The way this money is spent can be used to account for the Grid use by users and VOs.

5.4.2 Agent Architecture

The economic model we have developed consists of the agents which perform the reasoning and an auction protocol the agents use to communicate and trade with each other. In each optimisation agent there are three distinct elements which may be present on each site, as shown in Figure 5.3.

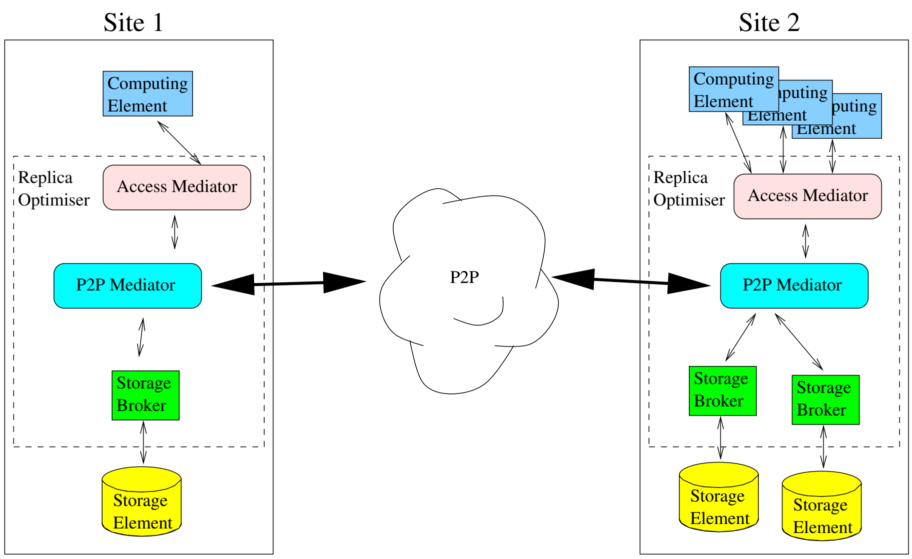


Figure 5.3: Components of the optimisation agents. From [14].

The network of agents has a P2P structure and each node may contain the following components:

Access Mediator (AM). This component is the entry point to the optimisation agents for jobs running on Computing Elements (CEs) which require access to files. For each requested file, the AM starts an auction to identify the cheapest replica of the file. It gathers bids for the file from local and remote *Storage Brokers*, selects the winner of the auction and performs file payments.

P2P Mediator (P2PM). These components are responsible for establishing and maintaining a peer to peer communications infrastructure between Grid sites. They propagate auction messages between AMs and SBs.

Storage Broker (SB). This component is responsible for maximising the profit gained from selling files in its corresponding Storage Element (SE) and decides which files to replicate to or delete from the SE. If the SB receives a request for a file and already has the file stored in its SE, it responds immediately with a bid reflecting the price for which it is willing to sell the file. If the file is not stored in the SE, the SB may start a *nested auction* in order to obtain a local replica of the file and be able to reply to the parent auction in the knowledge that it will have the file at some point in the immediate future.

5.4.3 Auction Protocol

In order that the optimisation agents can buy and sell files from each other, a fair, efficient trading mechanism is required. This mechanism must give every agent equal status and reduce the risk of collusion or any kind of price-fixing among the sellers. It must be efficient and simple so that it itself does not over-use Grid resources in its attempts to optimise them. It must also be reliable and stable, in terms of both the physical instability of the Grid resources and the potential instability of the market, avoiding for example hyper-inflation of file prices. Many market trading mechanisms exist in real life financial markets, such as bargaining, tenders, auctions, monopolies and oligopolies.

Reverse Vickrey Auctions

In our model we have chosen to use an auction process - specifically a Vickrey auction process [129]. Vickrey auctions are *second-price sealed-bid* auctions and have been shown to be fairer and more efficient than other auctions such as English or Dutch auctions, in both computer science [71, 127] and other areas of commerce [91]. They involve a single negotiation round, in which each bidder submits a bid to the auctioneer. Other bidders cannot see the bid. The winner of the auction is the bidder that made the highest bid, but this bidder only pays the price of the second-highest bid. An advantage of this type of auction over others is that the dominant strategy for each bidder is to bid his true valuation. In fact, bidding more than the true valuation is clearly a risky strategy: if he won the auction he could be asked to pay a higher price than he would have otherwise. On the other hand, bidding less than the true valuation would reduce the chances of winning. Bidding the true valuation is then the best strategy for an agent. This maximises his chances of winning the auction and, if he won, he would pay a price less than his valuation (the second best price).

The second-price auction also the reduces the problem of the "winner's curse". If it is assumed that the average of all prices bid is the true value of the item, then if the highest bidder pays the price he bid, he will almost certainly be paying more than the true value. Therefore paying the second-highest price means the value paid is closer to the true value, although with a very high number of bidders this effect is reduced.

In our case the role of the auctioneer is played by the AM, while the SBs play the role of bidders. However, the AM does not start an auction for selling an item (a file in our case) but for buying it. The SBs bid the price they are willing to sell the file for and the winning SB is paid the second-lowest bidding price by the AM. In other words, our economic model uses a *reverse Vickrey auction*. Using these type of auctions maintains a fair marketplace and, since there is only one round of bidding, reduces the messaging overhead involved in the process.

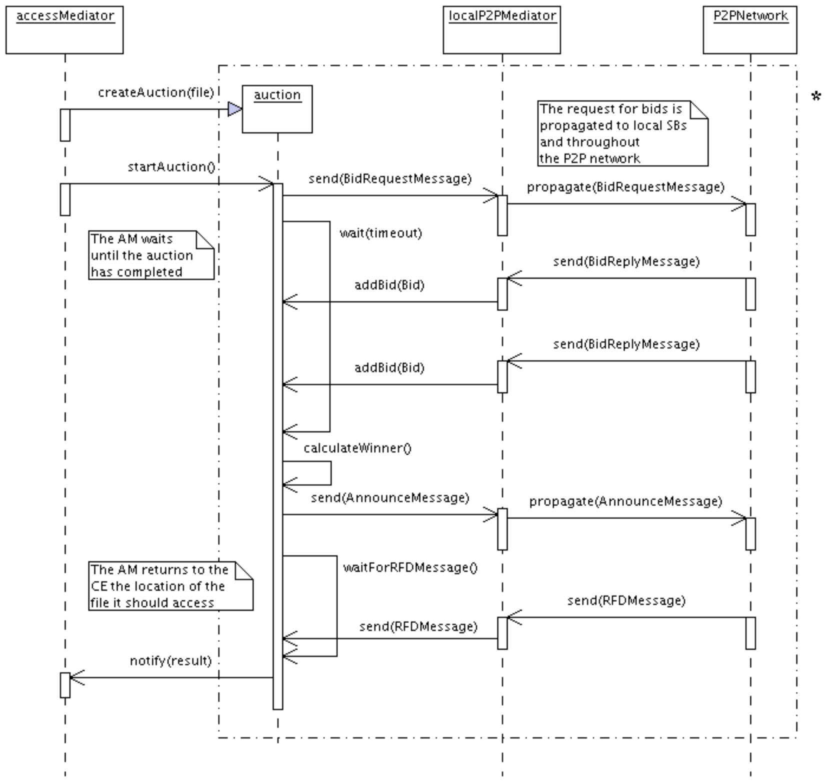


Figure 5.4: Peer-to-peer auction process represented by a UML sequence diagram.

First-level Auctions

The Unified Modelling Language (UML) [126] is a family of graphical notations used to describe and design object-oriented software systems. It provides a useful way of explaining diagrammatically at many different levels how a system works and the notations used in each type of diagram are explained in Appendix C. UML sequence diagrams are used to describe a sequence of interactions between different objects in a system and the file auction process is shown as a sequence diagram in Figure 5.4.

When a file is required by a CE, the AM starts a reverse Vickrey auction, and waits until the end of the auction before returning the location of the winning replica to the CE. An auction first issues a *BidRequest* message for the required file, which is propagated by the local P2P Mediator to the local SBs and other SBs via the P2P network. Depending on the topology of the network and the maximum distance for auction propagation (a parameter of the auction), the message will reach a set of Grid sites. Once it has issued a request for bids, the auction waits so potential file sellers can reply with *BidReply* messages containing offers to sell the file. After a fixed timeout, the auction calculates the winner (the SB that submitted the lowest bid from the bids received) and the price to pay the winner (the value of the second lowest bid). An *Announce* message is propagated over the P2P network informing the SBs of the auction result. If the auctioneer received no bids whilst waiting, the auction will have no winner.

When the winner has been notified, the auction waits until the winning replica is ready on the site. This is because a SB can bid for a file even if the related SE does not store a replica of the requested file (see below for details). After any replication process on the winning site has completed, the auction is notified via a *ReadyForDownload* (RFD) message and the physical location of the winning replica is returned to the CE by the AM.

Nested Auctions

Once a SB receives a request for a bid, it first checks if its SE stores the required file. If the file is present, it calculates a bid and replies with a *BidReply* message. This is shown in the top part of the UML sequence diagram in Figure 5.5. The local P2PM gathers bids from all local SBs and forwards them to the P2PM on the site that started the auction. This P2PM will forward the messages to the corresponding AM which started the auction.

If the file is not stored locally, the SB may start a "nested auction". The nested auction process constitutes most of Figure 5.5. If the SB decides that owning the file would be economically beneficial according to some price evaluation method a nested auction is used to obtain a copy. The auction is termed "nested" because the time for which the auction waits for bids is shorter than the original "parent" auction. Thus when the nested auction finishes there is sufficient time to send a bid back to the parent auction before the parent auction's timeout expires. Even though the SB does not yet have a copy of the file, if the nested auction was successful the SB can return a bid and win the parent auction in the knowledge that it will have the file in

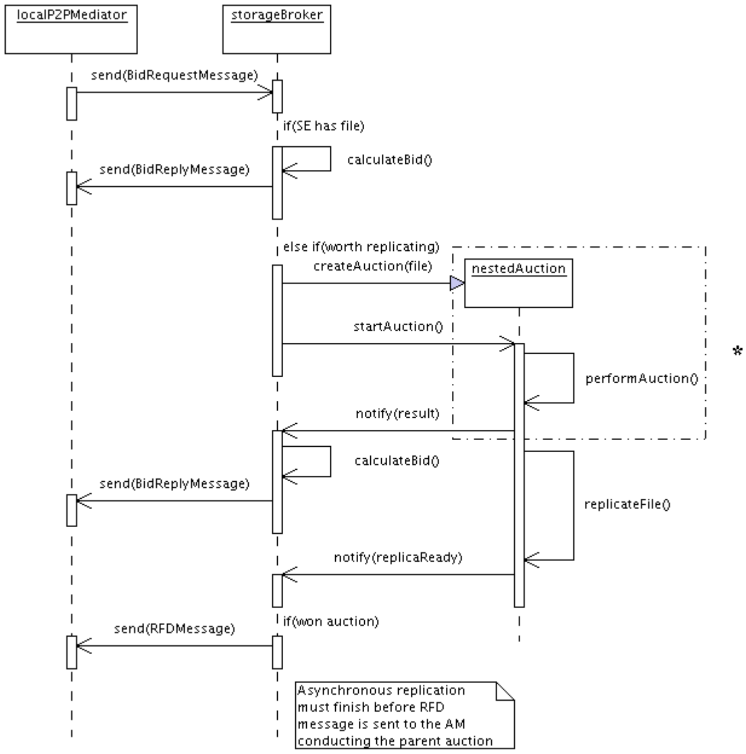


Figure 5.5: Nested auction process represented by a UML sequence diagram. The part within the box marked * is the same as the corresponding process in Figure 5.4.

the immediate future.

The nested auction is conducted in the same way as in Figure 5.4, with the SB as auctioneer instead of the AM. The box marked * in Figure 5.5 is the same process as the similarly marked box in Figure 5.4. Once a nested auction has successfully terminated, the SB calculates a bid for the file and sends it back to the parent auction. It is worth emphasising that we separate bidding in the parent auction from any replication activity. Asynchronous replication is carried out regardless of whether the SB won the parent auction. If the SB did win the parent auction however, it waits until replication has completed before sending the ReadyForDownload message back to the AM.

There is an important difference between parent and nested auctions. The goal of the former is to locate the cheapest replica required by some job executing on a CE. The best replica might be located either on the same site as the CE or on any remote site. Nested auctions always aim to replicate a file to a SE, as this will increase the SB's expected future income. In other words, the mechanism underlying the auction protocol performs long term optimisation allowing automatic replication toward "data hot-spots". Also, nested auctions allow replication to third party sites: sites where the file was not initially needed. Third party replication provides a good mechanism for distributing required files amongst a neighbourhood of nearby SEs. It means that collections of similar files group together in areas in which they are most requested, and reduces the bottlenecks caused by having all such files on one single site.

The auctioning process does introduce overheads such as extra messaging and time spent waiting for bids to arrive across the network. Our use of Vickrey auctions with one bidding round means these overheads are minimised, and the amount of network bandwidth used for auction messages is tiny compared to the sizes of files that are copied around the Grid. Using a distributed P2P network rather than a centralised replica location service means the system is more robust to failures and the use of third party replication should bring the files on the Grid to a stable optimal distribution.

5.4.4 File Value Prediction Functions

In the economic model there needs to be some mechanism to give value to the goods, i.e. files, being traded. The value is defined as the expected money that can be made from selling the files in the future, hence we need a "prediction function" to estimate the potential profit to be made [29]. In general the future pattern of file requests on a Grid is unknown and the best we can do is base the prediction function on observations of the past history of file requests. We assume that file requests are not entirely random and so the patterns seen in the past history give a reasonable guide to what might be expected for future patterns.

We can associate the file space $\{F\}$ which represents the set of all potentially requested files with the file-ID space $\{f\}$ which is a set of integer positive numbers.

One number is mapped to each file and the smaller the difference in two numbers, the more likely the two files will be requested together in any given sequence of requests.

Then we can represent the history of file requests as a random walk in the space of file identifiers $\{f\}$. A random walk consists of a sequence of identifiers, starting from an initial identifier f_0 and adding a sequence of random walk steps from f_0 to f_i. Each generic step s is an independent random variable which may assume values within the interval $[-S, +S]$, where S is the maximum difference between the identifiers of two successively requested files.

An important issue is then how to assign identifiers to each file. One solution is at the time when the files are created or put "into the Grid", the files are assigned identifiers of similar value to the files produced at the same time or by the same process. This assumes that files produced at the same time will have similar probability of being requested which may be likely for many applications (e.g. analysis of physics simulation data) but perhaps not for others (analysis of real data produced by a detector). A better way would be to dynamically (re)assign identifiers based on those already assigned to files requested around the same time. At first this system might not be reliable, but as statistics built up of the file request patterns the system should become more aware of the similarity of different groups of files and assign the identifiers more accurately. However, the development of a dynamic assignment system is left to future work.

Binomial-based Prediction Function

We define the value of a file $V(f)$ to be the number of times the file will be requested over some future period of time and use the random walk assumptions above to calculate an analytical form of $V(f)$. Each step s of the random walk can be modelled as a discrete random variable with binomial distribution. The binomial distribution returns the probability p to obtain k successes performing n independent trials of a certain test when the probability of success of each single trial is q. It is given by:

$$p(k) = \binom{n}{k} q^k (1-q)^{n-k} \qquad 0 \leq k \leq n$$

We can regard the random walk step size s as the sum of $2S$ values chosen independently with probability $\frac{1}{2}$ from the set $\{-\frac{1}{2}, \frac{1}{2}\}$. The sum of an even number of these values always gives an integer between $-S$ and S. Therefore setting $q = \frac{1}{2}$, $n = 2S$ and $k = s$, we have

$$p(s) = \frac{1}{2^{2S}} \binom{2S}{s+S}$$

The file requested at the i^{th} step in the random walk can be interpreted as the sum of $2iS$ values chosen independently with probability $\frac{1}{2}$ from the set $\{-\frac{1}{2}, \frac{1}{2}\}$. Therefore, setting is to be the distance in file space travelled in i steps $f_i - f_0$, the probability of receiving a request for file f at step i is given by

$$p_i(f) = \frac{1}{2^{2iS}} \binom{2iS}{f - \bar{f} + iS}, \qquad |f - \bar{f}| \leq iS$$

approximating f_0 to $\bar{f}$. If we have R sequences of file requests each containing n requests and starting from the same position $\bar{f}$, then the ratio

$$\frac{r_i(f)}{R} \tag{5.1}$$

where $r_i(f)$ is the number of times file f was requested at step i throughout the R sequences, will approximate to $p_i(f)$ for increasing R. Summing up all terms $r_i(f)$ for i going from 1 to n, i.e. $\sum_{i=1}^{n} r_i(f)$, we obtain the total number of times that file f has been requested during the R sequences. Dividing this sum by R, we get the average number of requests for f in a single sequence and this is equal to the sum of all terms like (5.1) (for i going from 1 to n). Assuming $p_i(f)$ is the best estimation of (5.1) we can give, then the most probable number of times file f will be requested during the next n requests, which is exactly what we are looking for, is given by

$$V(f) = \sum_{i=1}^{n} p_i(f) = \sum_{i=1}^{n} \frac{1}{2^{2iS}} \binom{2iS}{f - \bar{f} + iS} \tag{5.2}$$

In order to calculate this value we have to define n, S and $\bar{f}$. Firstly we have to fix a time interval T' in the past that serves as the basis for our estimation. Then T is the future interval for which we intend to do the prediction. So, r is the number of requests received in the last period of length T'. By assuming that the arrival rate of requests will stay the same, n is obtained by:

$$n = r\frac{T}{T'}. \tag{5.3}$$

We give an estimation of the maximum step size S based on past steps in the random walk. The file requested at step i is the sum of all the steps since f_0 was requested and is given by

$$f_i = f_0 + \sum_{j=1}^{i} s_i.$$

Since each s_j is an independent random variable with symmetric binomial distribution, each f_i is also a random variable with the same distribution, and its variance is

$$\sigma_{f_i}^2 = \frac{iS}{2}.$$

So going back r requests into the past, each f_{-j} should have variance $jS/2$. The best estimation we have for this variance is $(f_0 - f_{-j})^2$. Thus, weighting in favour of recently requested files, we obtain an average for S

$$S = \frac{2}{r}\sum_{j=1}^{r}\frac{(f_0 - f_{-j})^2}{j}. \tag{5.4}$$

The simplest way to fix a central value $\bar{f}$ for our sequence of distributions on the basis of previous requests is to calculate a weighted average among last r files requested, where the weights decrease going toward the past. That is, the more recently a file has been requested, the more important it is in calculating $\bar{f}$.

$$\bar{f} = \sum_{j=1}^{r}\frac{f_{-j}}{j} \tag{5.5}$$

Therefore, using (5.3), (5.4) and (5.5) we can evaluate $p_i(f)$ and hence the future value of the file $V(f)$ (5.2).

Zipf-based Prediction Function

Another possible file request distribution pattern is a Zipf distribution [135]. Zipf's law was originally based on observations of the relative frequency of words used in the English language. It was found that the spread of frequency vs rank roughly follows an inverse power law, i.e. a few words occur very frequently while many words occur rarely. The distribution can be described as $P_i \sim i^{-\alpha}$, where P_i is the frequency of occurrence of the i^{th} ranked item and α is around 1. Any distribution following this pattern can be termed a Zipf distribution. As an example, the number of occurrences of each word in Charles Dickens' *David Copperfield*, ranked by the number of occurrences, is shown in Figure 5.6, overlaid with the shape of $i^{-\alpha}$ when $\alpha = 1$.

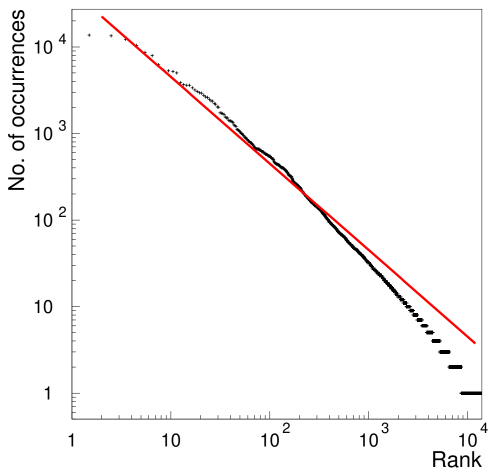


Figure 5.6: Relative frequency of words in David Copperfield and the shape of i^{-1}.

Several other collections approximately follow Zipf's law, for example the sizes of cities, income distribution in a population and the strength of earthquakes. All of these distributions have a few occurrences of exceptionally high values and many occurrences of low values. Recent interest around Zipf's law has centred around its use when modelling Internet traffic and the popularity of web pages. It is used in web proxy caching, where to reduce network traffic it is desirable to cache the most popular web pages. In [20] it was found that web proxy traces roughly followed a Zipf-like distribution with the value of α between 0.64 and 0.83 depending on the trace.

Due to the conceptual similarity between web workloads and Grid file requests, we propose an economic prediction function based on the assumption of a Zipf-like distribution of these requests. The recent history of file requests is used to construct a Zipf-like distribution based on the frequency of occurrence of each file in the history. The files are ranked according to how many times they have been requested in the past history, with the more popular files having higher rank. If the file requests follow a Zipf-like distribution this should give a pattern like that shown in Figure 5.6. Then any file with rank i will have a projected number of requests and hence a value

$$V(f) = \frac{T}{T'} i^{-\alpha} \tag{5.6}$$

where T and T' have the same meanings as in (5.3) and α is calculated by fitting Zipf's law to the observed distribution.

Other prediction functions based on different analyses of file request patterns are feasible but in this work we only consider the binomial and Zipf-based prediction functions just described, and compare their effects on the performance of the Grid in Chapter 7.

5.5 Summary

This chapter has described some of the mechanisms used to optimise the use of the resources on the Grid, for both providers and consumers of the resources. It has been shown that job scheduling in a Grid environment is more complex than localised job scheduling. The problems related to heterogeneity, scalability, unreliability and security on a Grid mean that resource matching and giving users the required quality of service become difficult. The solution we propose for Data Grids, where efficient access to data is a key factor, is to use a metric based on the estimated access cost of the data required for a given job. Using this metric should ensure that when access to data is required, minimal network resources are used and hence minimal time is spent moving data around the Grid.

Replication of data also plays an important part in optimising the resources available for data-intensive Grid jobs. Three distinct stages of replica optimisation (replication decision, replica selection and file replacement) must be addressed by any replication strategy. We have proposed an economic approach to replica optimisation which provides a solution to these three stages. Using a P2P auction protocol, files are traded between independent optimisation agents and a future file value prediction function is used to calculate when it is beneficial to replicate and which files are expendable. The use of a Vickrey auction means the marketplace is fair and efficient and the P2P nature of the communications makes the system robust to failures. The two versions of economic prediction functions presented here (binomial and Zipf-based) use a history of file requests to estimate potential future file requests and hence how much profit can be gained from purchasing any particular file. The economic model offers many advantages over traditional cache management systems, giving more reliability and scalability and providing the basis for any accounting or payment mechanisms that may be used by Grids in the future. The implementation of this model into a Grid simulator to evaluate its properties is the subject of the next chapter.

Chapter 6

OptorSim

At the present time (the second half of 2004) there are no Grids in existence that are at the stage of being used to their full extent or potential as part of scientists' day-to-day work. Over the summer of 2004 the LHC Computing Grid for particle physics experiments at CERN (see Section 1.3.3) expanded to 65 sites (Figure 1.6) and was shown to be sufficiently stable and large enough to be used in a production environment. However it is still not at the stage where it is in common use by the average physicist. It is not possible to test different optimisation strategies using different Grid scenarios, topologies and user patterns in a controlled and repeatable way with the resources available today. Without the means to fully test our strategies on live Grids, and in order to evaluate their effectiveness on future Grids, we use computer simulation of the conditions of the Grid environment.

As part of the European DataGrid project (see Chapter 3) the Grid simulation package OptorSim was developed to allow investigations into possible optimisation strategies that could be used in the Replica Optimization Service. The scope of OptorSim has since been expanded to simulate a variety of generic Grid scenarios and optimisation strategies. Results from OptorSim simulations have been presented in [13, 14, 27] and further results will be shown in Chapter 7. This chapter describes the design of the simulation and the implementation of the ideas described in the previous chapter. The Unified Modelling Language (UML) is again used here to illustrate the implementation of concepts described in the last chapter and other features in OptorSim.

6.1 Motivations for Creating OptorSim

Our primary aims in developing a simulation were to emulate a Grid environment as realistically as possible and to provide some method of evaluating various strategies that might be used to optimise a Grid. Before work started to build a new simulation, other simulations that had already been developed to model distributed computing environments were evaluated to find out if any could be used as a basis for this work.

The most likely candidate was the simulation used for the MONARC project (see Section 1.3.2, [94]) to test potential computing models for LHC experiments. The simulation modelled to a high degree of detail the estimated output data and subsequent reconstructed data products produced at CERN and the physics analysis processes performed by physicists at other distributed sites. The main point of the simulation was to show that a multi-tier system such as the one shown in Figure 1.4 was feasible and to calculate the size of the resources required at each site. We decided not to use this simulation mainly due to the threading model used, which modelled each job as a separate Java thread. It was found that the amount of memory required made it impossible to simulate more than around 1000 physics analysis jobs, whereas we were interested in simulating upward of 100,000 jobs. We also wanted the focus of the simulation to be the effects of optimisation strategies, but with the MONARC simulation, there were too many other parameters that would affect any given simulation run and swamp any changes to the Grid performance caused by the optimisation strategy. It was also very specific to LHC high energy physics and the multi-tier model whereas we required more flexibility in the architecture of the Grid and the types of Grid jobs we could simulate.

GridSim [26] is another large-scale simulation, designed to study economic resource brokering in a distributed environment. However, at the time of the evaluation, GridSim concentrated heavily on complex scheduling decisions and did not deal with data movement around the simulated Grid. Other simulations we considered included the Bricks Grid simulator [123], which also focused on scheduling policies rather than data movement and MicroGrid [118], which modelled in detail the Globus Toolkit and applications that use it. The Ptolemy project [84] provided a framework for simulation of concurrent systems, however, at the time, we could not have easily adapted any of these tools to our requirements. Therefore we decided to create our own simulation, OptorSim.

6.2 Simulation Requirements

The main idea of OptorSim was, given a Grid topology and resources, a set of jobs and an optimisation strategy, simulate data movement around the Grid as these jobs run and supply information on various factors that could be used to evaluate the performance of the optimisation strategy. It had to allow a variety of different Grids and user jobs to be simulated and enable any results to be repeatable, while also taking into account the many random factors in a Grid environment.

6.2.1 Simulated Grid Architecture

One of the aims of OptorSim was to evaluate optimisation strategies that would eventually be used in the Replica Optimization Service, so the architecture of the simulated Grid had to mimic as closely as possible the structure of the European DataGrid. However, not all components are important or affected by the optimisation strategies we are testing, so we model in detail only the relevant ones, such as those associated with data management.

Figure 6.1 shows the main components of the architecture. The Grid has several Grid sites, each of which have a Computing Element (CE), a Replica Manager (RM) and a Storage Element (SE). The CEs run Grid jobs which are submitted to them by Users via a Resource Broker (RB) according to a specific scheduling policy. The CE interacts with a RM to obtain the data required for each job. Instead of a centralised Replica Optimization Service, each RM contains an individual Optimiser which uses a replica optimisation strategy to make replication decisions about data files. These files are stored in SEs and may be replicated between SEs on different sites. The RM is used to move the data files between sites and the Replica Location Service consists

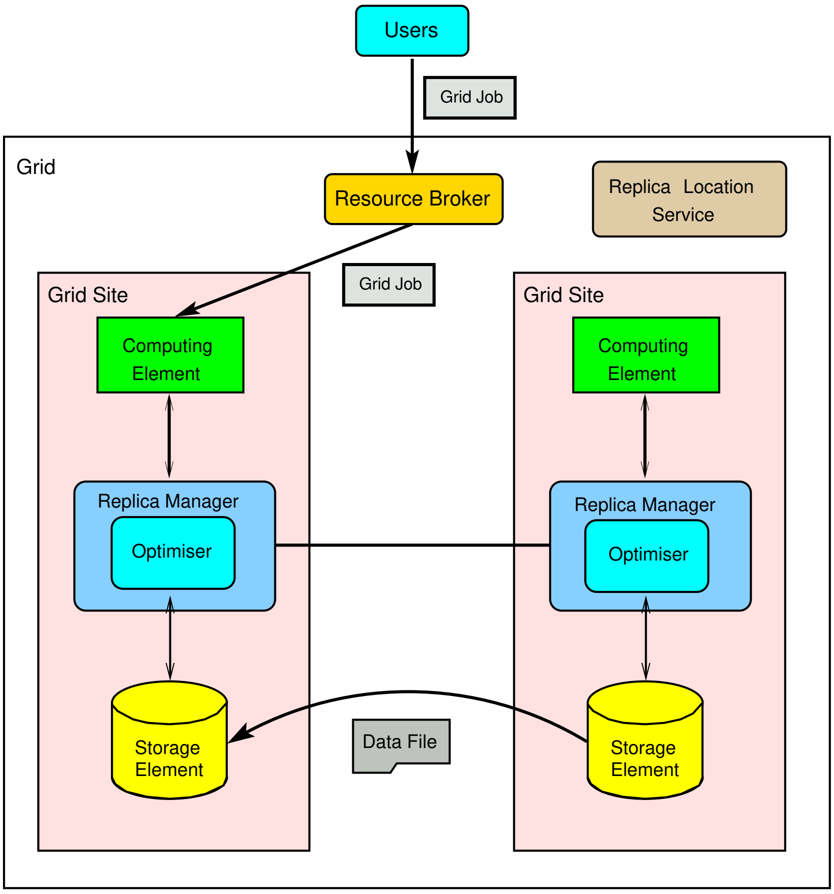


Figure 6.1: Generalised simulation Grid architecture, showing the components of the EDG architecture important to replica optimisation.

of a single Local Replica Catalog holding logical to physical file name mappings for the whole Grid.

6.2.2 Inputs to the Simulation

There are many parameters that can be varied to simulate different Grid scenarios and which might affect any results from the simulation. The Grid network topology has a large effect on the structure of the overall Grid and in OptorSim, any Grid layout, from the MONARC-style hierarchical model to a flat peer-to-peer structure can be used. The jobs which run on the simulated CEs are defined by a set of files they must access and each job has a certain probability of being selected to run on the Grid. The policy of each CE can also be defined, i.e. the set of jobs it is willing to run. A specific scheduling policy and replica optimisation strategy is fed into the simulation along with some initial conditions such as the location of master files and the rate that users submit jobs. A summary of the main parameters that can be specified before running OptorSim are shown in Table 6.1.

Parameter	Description
Grid configuration	A text file describing the Grid topology and resources
Job configuration	A text file describing the jobs to run on the simulated Grid
Number of jobs	The number of jobs to simulate
Users	Determines the rate at which jobs are submitted to the Grid
Scheduler	The scheduling policy
Optimiser	The replica optimisation strategy
dt	The past period of time over which to keep file access history
Access Pattern	The pattern of file access within each job
File Distribution	The distribution of files at the start of the simulation
Process Time	The time taken by the CE to process each file

Table 6.1: Adjustable parameters in OptorSim.

6.2.3 Running of the Simulation

The interactions between each component in the lifetime of a Grid job is shown in the UML sequence diagram in Figure 6.2. When the simulation starts running, users submit jobs at a specified rate to the RB, which schedules them to CEs according to the scheduling policy defined. The file access pattern determines the order in which files are requested and the CE contacts the Optimiser to find the best replica to access for each file required. Triggered by these requests, the Optimiser makes decisions on replication of data depending on the optimisation strategy. Once the Optimiser has supplied the location of the replica the CE is to access the CE either reads it directly (if the file is on a SE on the same site) or reads it remotely, copying to a local cache (if the file is on another site). For simplicity, bandwidth is assumed to be infinite within a Grid site; inter-site network connections are supplied in the Grid configuration file (see Section 6.4.2 for more information on how we model the network). The data analysis process is not modelled in detail, so when the CE has read the file, it waits

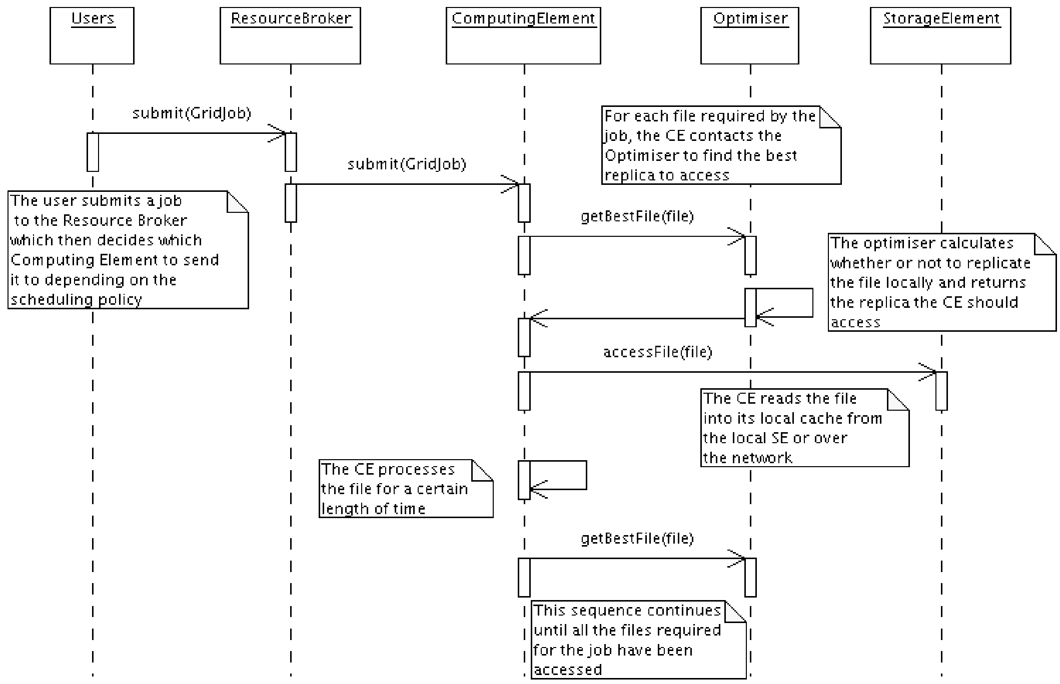


Figure 6.2: UML sequence diagram of a typical Grid job.

for a certain length of time to simulate the delay caused by processing the file. The CE then requests the next file it needs for the job and continues until it has processed all the files in the job. The simulation runs until the users stop submitting jobs and the CEs have finished processing all the jobs.

6.2.4 Outputs from the Simulation

In order to evaluate how the optimisation strategy affects the performance of the Grid, there are several ways in which information is output to the simulation user. At the end of each simulation run, detailed statistics are provided for each component on each site, each site as a whole, and the entire Grid itself. This includes information such as the files accessed for each job by each CE, the time taken to run each job, and the number of files replicated to each SE. The use of these metrics in evaluating the optimisation strategies is discussed in Section 7.1.2. If OptorSim is run from a command line, this information is output to the terminal window. Besides textual output, OptorSim also has a Graphical User Interface (GUI) to monitor the state of the Grid as the simulation runs and examine it at the end of a run. A screen shot of the GUI midway through a simulation is shown in Figure 6.3.

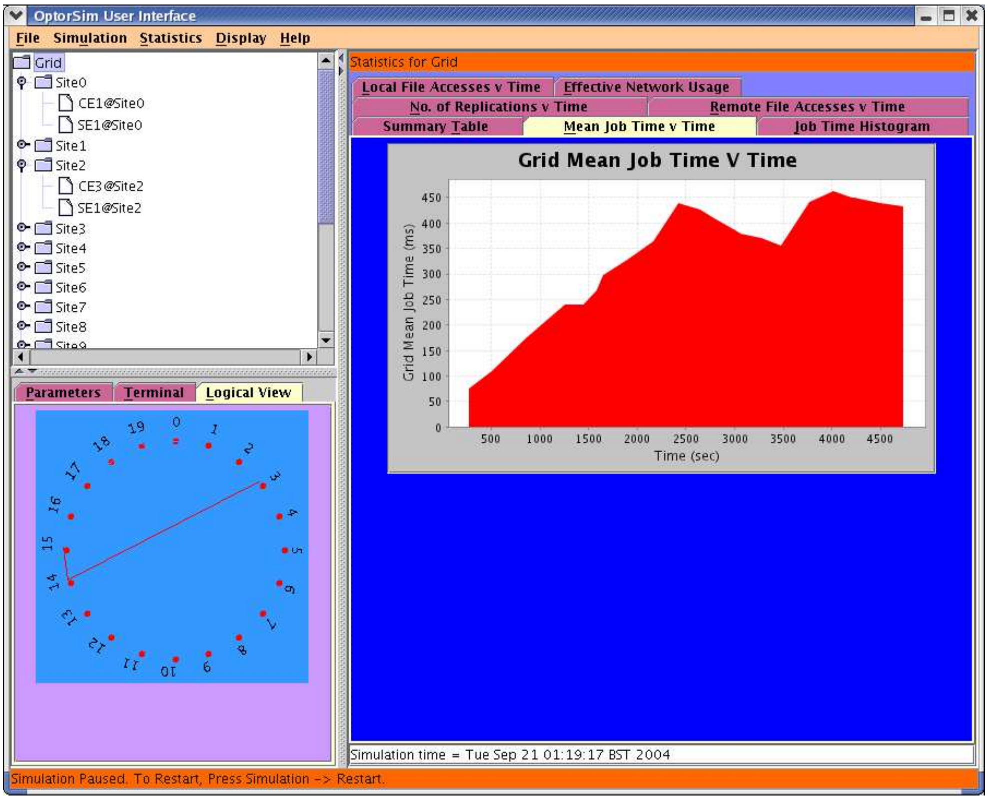


Figure 6.3: Screen shot of the OptorSim GUI.

The top left pane of the window (the resource view) shows the Grid sites, which can each be expanded to show the CEs and SEs it contains. The bottom left pane (the information view) shows a logical view of the network, with the lines between each numbered Grid site representing a file transfer between those sites. Other tabs on this pane can be selected to show the parameters used in this simulation run and the terminal output. There are several graphs which can be displayed on the right-hand pane (the graph view), which show information on the element selected in the resource view, in this case graphs showing the status of the entire Grid. The graph selected is the variation over time of the time taken to run each job on the Grid. A set of drop-down menus at the top of the main window provide various features such as exporting graphs and choosing which views to display. A full description of the GUI can be found in [53].

6.3 Implementation

OptorSim is implemented in Java^{TM}, and there were several reasons for choosing this language. The OGSA/WS-RF Grid architectures discussed in Chapter 2 describe distinct services which interact via standard interfaces with the implementation encapsulated within the service, much like an object-oriented system. Hence, simulation of this environment lends itself well to an object-oriented implementation, as the functionality of each component shown in Figure 6.1 can easily be encapsulated within that component. This kind of system also makes interchanging different implementations of various components, such as the replica optimisation strategy, very simple. We want to simulate independent optimisation agents, and Java's concurrent threading facilities make this easy, so that many processes can run simultaneously. Many of the other Grid simulators mentioned in Section 6.1 are written in Java, showing that it can be used successfully for modelling distributed systems. Although Java performance can be relatively slow compared to C++ in numerically intensive applications [18, 21], the initial design of our simulation meant it was not numerically intensive and the advantages of Java (threading, ease of use, portability etc.) outweighed any performance deficits it might have.

In OptorSim each CE runs as a thread, as does the RB and the Users (one thread represents all users). This means that none of these components are directly dependent on any other to perform their tasks, for example the RB can submit a job to a CE's queue at the same time as the CE is executing a job. Threads are also used to model the P2P Mediator and Storage Broker components of the economic model (Figure 5.3) to handle message passing, and in auctions to allow asynchronous replication.

6.3.1 Development History

The development of OptorSim started early in 2002 and very soon a prototype version was released with all the basic features of the Grid architecture, but only very simple replica optimisation implemented. There followed a series of releases in quick succession with increasing functionality. Version 0.4 was the first major release to have a reasonably stable code base and to incorporate the economic model prediction functions (without the auction protocol). The auction protocol was added in version 0.5 to give a complete implementation of the economic model along with a prototype GUI. Version 0.6 was an internal release only, which was stable enough to provide results on the performance of the optimisation algorithms but the code was not of the required quality for an external release.

Date	Version	Features
March 2002	0.1	A first implementation demonstrating some logic of the simulation
April 2002	0.2	Modularisation of access pattern generators and (simple) replica optimisation strategies.
May 2002	0.3	Internal release used for testing.
June 2002	0.4	Integration of economic model prediction function (without auction protocol)
March 2003	0.5	Large number of new features including the auction protocol and a GUI.
June 2003	0.6	Internal release used mainly for testing and obtaining the results in [27].
February 2004	1.0	Large refactoring to make code more modular and expandable, and includes background network traffic.
November 2004	2.0	New time model, brand new GUI and many bugs fixed.

Table 6.2: Development history of OptorSim.

Version 1.0 was heavily re-factored from version 0.6 to make the code more usable by external users. Many classes and interfaces were modularised so that users could add their own code without requiring significant changes to the core OptorSim code. It was also possible to simulate the effects of background network traffic on the Grid. Two major changes were introduced in version 2.0, a new time model and a new

GUI design [53]. The new time model moved OptorSim from a time-driven simulation to an event-driven simulation, the reasons and significance of this is described later in Section 6.4.1. A summary of all the major releases of OptorSim is given in Table 6.2. Apart from the developers, OptorSim has been used and extended by several researchers from Europe to the Far East, for example in [105] OptorSim is used by the authors to evaluate an optimisation strategy which considers data locality within a region of well connected sites instead of on a single site. They found that when the network bandwidth within a region of sites was higher than the inter-region bandwidth, their strategy outperformed the traditional LRU strategy by up to 30%.

6.4 Components within OptorSim

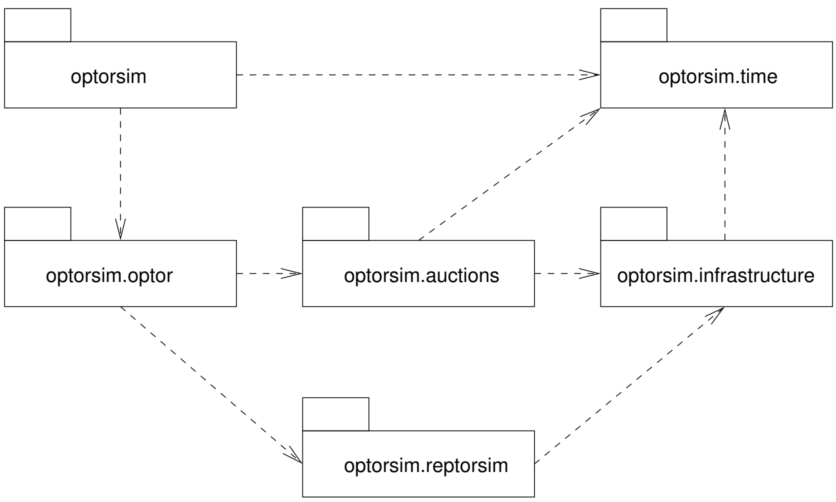


Figure 6.4: UML package diagram of OptorSim, showing the dependencies between each of the packages.

OptorSim simulates many areas of a Grid and these areas can be divided into packages, each of which contains a collection of related classes. The package diagram shown in Figure 6.4 describes those within OptorSim and their dependency relations. Starting at the lowest level, the *optorsim.time* package (7 classes) deals with how time is measured within the simulation. This is covered in more detail in Section 6.4.1. *optorsim.infrastructure* contains 20 classes that simulate the underlying Grid infrastructure including the network, the Grid sites and the basic components of a site: Computing Elements and Storage Elements. The network infrastructure and how background network traffic is simulated is explained further in Section 6.4.2. The P2P network and messaging system along with the auctioning process described in Section 5.4 is contained in the *optorsim.auctions* package (16 classes). The functionality of the replica management components is implemented in the *optorsim.reptorsim* package (4 classes), including the Replica Manager and Replica Location Service. The replica optimisation strategies are in the *optorsim.optor* package (18 classes), and their implementation is described in Section 6.4.6. *optorsim* is the highest level package, and it has 30 classes to simulate the Resource Broker, users and control the GUI. Overall OptorSim consists of 95 classes totalling around 15,000 lines of code.

The next few sections cover in more detail the following components of the simulation:

- Timing: how time is measured in OptorSim.
- Network: the model of the underlying network along with how we simulate background non-Grid traffic.
- Job Selection and Submission Rate: modelling different types of Grid users.
- File Access Patterns: the order in which files are accessed within a Grid job.
- Scheduling Policies: the implementation of the policies to decide where to submit jobs.
- Replica Optimisation Strategies: the various strategies implemented in OptorSim.

6.4.1 Timing

Until version 2.0, OptorSim used a time-driven simulation model. This meant the time taken to simulate a job running on the Grid was the same time as it would take in real life. If it took 10 seconds to copy a file between two sites on the Grid, in the simulation the CE would wait 10 seconds until the transfer had finished before processing the file. The time to complete a job in the simulation time was measured simply by measuring the real time taken using the host computer's internal clock. The idea behind this model was that calculation times for the optimisation strategies would be taken into account when evaluating how effective they were. In other words if a strategy reduces the time to run a simulated job by distributing files better around the Grid, but requires a significant amount of time to do its calculations, it may not be the best choice of strategy. This model has the large disadvantage of taking a long time to run any simulated jobs which would take a long time in real life, for example jobs running on sites with very low bandwidth network links. However, it was possible to scale down certain parameters such as the file sizes so that the simulation would run faster.

After much experience using this timing model, it became clear that in the simulation, and hence in a real Grid, for the sizes of files we used (of the order of 1 GB) the calculations for the optimisation strategies we had included did not take a significant length of time compared to file transfer between sites and file processing time. It was observed that all the threads spent most of the time waiting, for example waiting for a file transfer to complete, waiting for jobs to be submitted etc. Therefore, a decision was taken to move OptorSim toward a event-driven timing model. In this model, simulation time is separated completely from real time so that simulation time is managed by the simulation itself. The hope was that the time to run a simulation would be greatly reduced if, instead of all threads waiting for something to finish, time was simply advanced to that point and the threads woken up.

This new timing model is implemented in OptorSim using a separate timing thread, which continually looks to see if all the other threads are waiting. Only this thread has the power to advance simulation time, so if any other threads are performing calculations, time stands still. Once all the other threads are waiting, the timing thread finds out when the next event is due to happen, advances the simulation time to the time that event will occur, and notifies the waiting thread. The difference between the two timing models is illustrated in Figure 6.5.

The two parts of this figure show how real time and simulation time progress using (a) a time-driven and (b) an event-driven model. For the time-driven model,

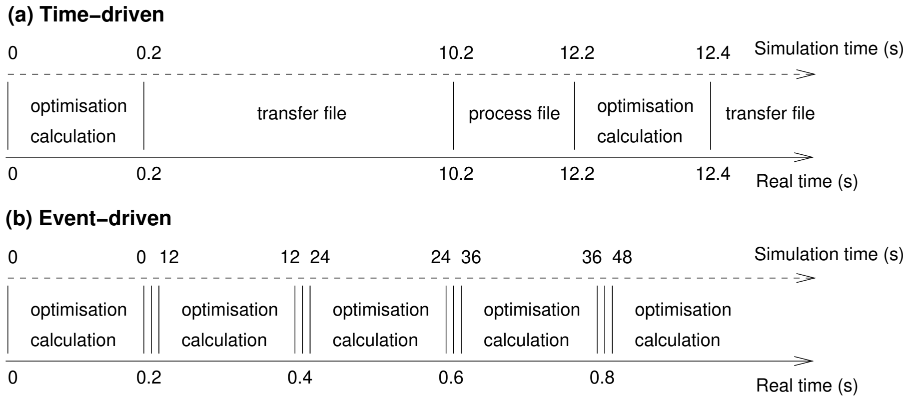


Figure 6.5: (a) Time-driven and (b) event-driven timing models. This figure shows the relation between simulation time and real time for the two timing models.

the simulation time is the same as the real time; the optimisation calculation takes 0.2 seconds, while the file transfer and processing combined take 12 seconds. With the event-driven model, the optimisation calculation still takes 0.2 seconds of real time, but simulation time remains at zero. During the file transfer the thread is waiting, so the timing thread can advance simulation time 10 seconds to the next event, which is the end of the file transfer. Similarly for the processing time the thread is just waiting, so simulation time can again be advanced by 2 seconds, to the next event which is another optimisation calculation following a request for the next file. Simulation time is thus advanced 12 seconds in a negligible amount of real time, so the real time to access and process each file for the job is only 0.2 seconds, compared to 12.2 seconds using the time-driven model.

The disadvantage of using the event-driven model is that the calculation time for various processes including the replica optimisation strategy is not included in the measured simulation time. However, as was stated earlier, for the file sizes and strategies we have used so far in OptorSim, this time is not significant compared to the other time-consuming processes such as file replication. The advantages of the event-driven model far outweigh this small inaccuracy - simulations now run from 10 to 100 times faster than they did with the old time model and results obtained using the event-driven model were checked and are consistent with the results obtained for the time-driven model. The option of using the time-driven model is still available in OptorSim as it is useful for demonstration purposes when using the GUI or if file sizes are very small and hence calculation time is significant.

6.4.2 Network

In OptorSim the network topology is specified in the Grid configuration file, which is read at the start of the simulation. An example of a simple Grid topology is shown in Figure 6.6. Grid sites are defined as sites with CEs and/or SEs present, whereas routers have no CEs or SEs. The numbers on each link give the bandwidth capacity in Mbit/s (M) or Gbit/s (G). In the situation depicted there are 8 Grid sites with fast connections to a relatively slow backbone.

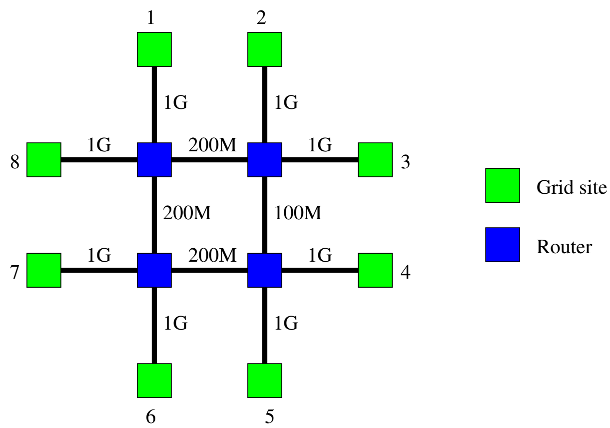


Figure 6.6: A simple Grid topology with 8 sites connected with high speed links to a slow backbone.

When a file is copied between two sites, the rate at which data is transferred is determined by the network link with the least available bandwidth, i.e. the bottleneck. For example, if a file is replicated from site 2 to site 3, the rate of data transfer is 1 Gbit/s, but if it is replicated from site 2 to site 1 the rate is 200 Mbit/s. If there are multiple routes between a pair of sites, the fastest one is chosen, so for a file replication between site 2 and site 5, the data will be routed around the left side of the backbone and the transfer rate will be 200 Mbit/s.

We take a very high-level view of the network, so that a file transfer is just a constant stream of data using a certain amount of bandwidth. We do not go as far as simulating individual packets of data or any particular data transfer protocol. If several data transfers are sharing the same link, the available bandwidth is divided equally among each one, whether each transfer actually uses its full share or not. This provides a reasonably accurate way of modelling real networks, but it can lead to data transfers being slower than they should be in certain situations. For example, if a file is being replicated from site 3 to site 8, the data transfer rate is 200 Mbit/s. If site 2 starts to replicate another file from site 3, in theory the bandwidth available should be 800 Mbit/s. However, because the bandwidth on the link from site 3 to the backbone is divided evenly between the two data transfers, a maximum of 500 Mbit/s is available for each one.

When a file replication across the network starts in OptorSim, the time it will finish is calculated based on the current available bandwidth, and the thread initiating the transfer waits until this time. If another file transfer starts or finishes during this time, all the other currently executing file transfers are notified so they can re-calculate the time that they will finish based on any changes in the bandwidth available.

Background Traffic

On the real Grid, any file transfers will in general use network infrastructure that is shared by other non-Grid traffic. This background traffic could use a significant amount of the bandwidth capacity and could vary throughout the day. To characterise this traffic, network monitoring data was collected from various sources, including traffic on links between sites on the Grid testbeds we simulate in Chapter 7. The sizes of the data samples varied from a period of a few days to around two months, and the mean value was taken over each half hour of the day.

Example results from monitoring two links (Lyon to CERN and FNAL to Imperial) are shown in Figure 6.7. The capacity of each of these links was 100 Mbit/s. For the first link the amount of bandwidth available varies by up to 20% during the day, with the heaviest traffic being in the afternoon and early evening. There is also some variation in the second link, but it is less than 2% and this link is much more heavily

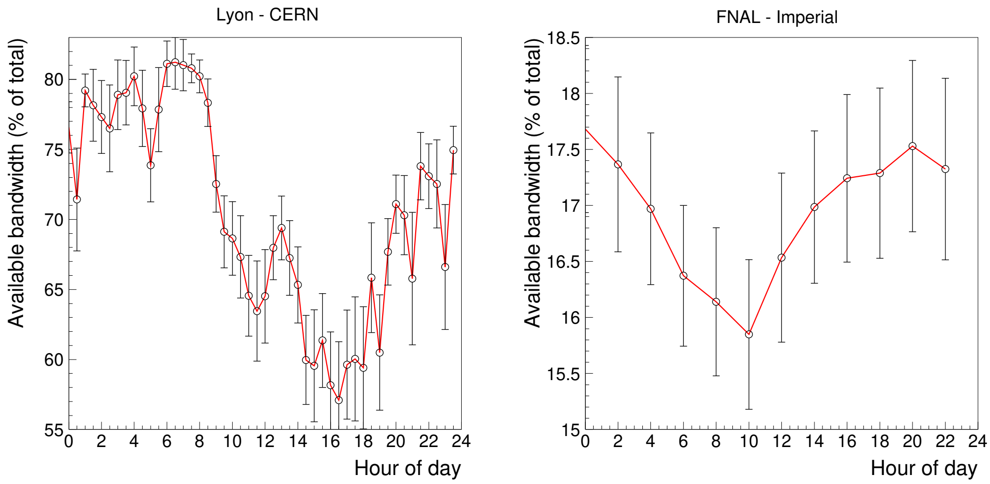


Figure 6.7: Variation of non-Grid network traffic over the course of a day for links Lyon to CERN and FNAL to Imperial. From [99].

used, with only 16-17% available. This difference is expected as the FNAL-Imperial link is used for transatlantic traffic, whereas Lyon and CERN are relatively close to each other. Information gathered from these network links and others is used in OptorSim to place limits on the fraction of a particular network link available for Grid traffic at a given time of day.

6.4.3 Job Selection and Submission Rate

The jobs selected to be submitted to the Grid and the rate at which they are submitted is defined by the type of users specified in the parameters configuration file. In OptorSim there are three types of users currently implemented, as shown in the UML class diagram in Figure 6.8.

A *Users* interface defines two methods that all types of users must implement. It also extends the *Runnable* interface so that the users can run as a separate thread. The next job selected to be submitted to the Grid and the time for which the users wait between submitting jobs are defined by the methods *getNextJob()* and *waitForNextJob()* respectively. Currently all types of users use *getNextJob()* defined in *SimpleUsers*, which chooses jobs according to the probabilities given in the job con-

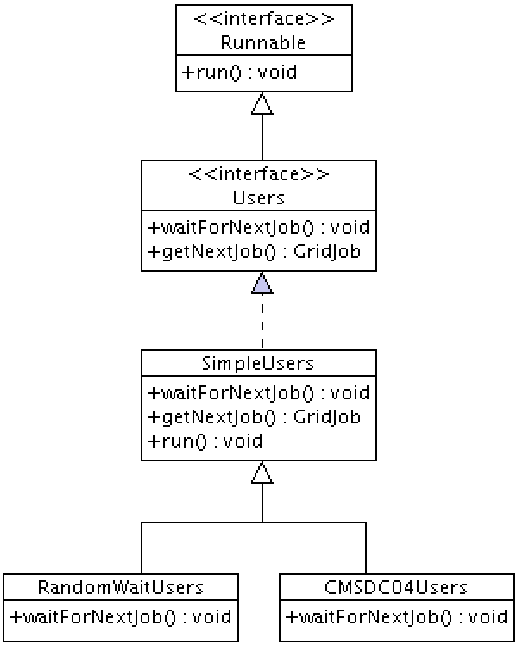


Figure 6.8: UML class diagram showing the different types of users implemented in OptorSim.

figuration file. In this class' implementation of *waitForNextJob()* it simply waits for a constant set time (specified in the parameters configuration file) between jobs. The delay between jobs defined in *RandomWaitUsers* is uniformly randomly distributed between zero and twice the time given in the parameters configuration file, so that on average the same number of jobs are submitted as the simple users.

The final class of users, *CMSDC04Users* represents a more realistic job submission rate. Between March and May 2004 the CMS experiment [40] held a "data challenge" using simulated data to examine how their computing framework performed. Part of this involved physicists analysing simulated data using the LHC Computing Grid and the CMSDC04Users class models their pattern of job submission using information from [76]. It was found that the rate of job submission varied for each hour of the day according roughly to a normal distribution, which peaked at 3pm and had a width (FWHM) of around 6 hours. The simulated submission rate follows this pattern and aims to represent a typical day of users in Europe submitting physics analysis jobs.

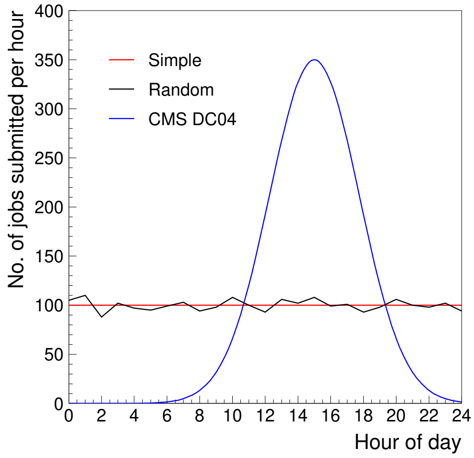


Figure 6.9: Number of jobs submitted per hour for simple, random and CMS DC04 users.

The number of jobs submitted every hour over the course of a day for each of the user types is shown in Figure 6.9. The simple users submit 100 jobs per hour and the random users between 90 and 110. The CMS DC04 users do not start submitting jobs until around 7am, peak at 3pm when 350 are submitted in an hour then the rate decreases into the evening. The number of jobs submitted over the whole day is 2400 for all types of users.

6.4.4 File Access Patterns

The file access pattern determines the order in which files are requested during the course of a job. A list of files is defined for each job and depending on the access pattern, some of these files will be accessed during a job once, multiple times, or never. There are five different choices of access patterns that can be used in OptorSim:

- *Sequential:* Files are accessed in the order that they are supplied in the description of the job.

- *Random:* Files are selected randomly from the list with a flat probability distribution.

- *Unitary Random Walk:* Each successive file is either the previous or next file in the list from the current file, with both having an equal probability.

- *Gaussian Random Walk:* Similar to unitary random walk, but the step size of the walk has a Gaussian distribution.

- *Zipf:* The probability P of accessing each file i in the list follows a Zipf-like distribution: $P_i \sim i^{-\alpha}$ where α is around 1 (see Figure 5.6 in Section 5.4.4)

These file access patterns are illustrated in Figure 6.10 (Gaussian random walk is similar to unitary random walk but with a variable step size). In these illustrative models a job was defined as requiring 100 files (Files 1 - 100). With sequential access patterns (Figure 6.10(a)) the request order is the order of the file list in the job definition i.e. File 1, File 2, ..., File 100. Random access patterns (Figure 6.10(b)) mean that every file has the same probability of being the next file to be requested and the unitary random walk (Figure 6.10(c)) steps randomly up or down the list of files. Using a Zipf distribution (Figure 6.10(d)), here with $\alpha = 0.5$, a few files are requested many times and the majority of files have a very small number of requests.

The most common access patterns for high energy physics jobs are expected to be sequential in the reconstruction phase and Zipf in the analysis phase. In reconstruction, a long list of raw data files are processed in order to reconstruct what happened in each event and produce data products describing the physics properties of these events. Some of these data files will be more interesting to physicists than others and so when the data is analysed it is very likely that the user access pattern will follow a Zipf distribution, i.e. rare events will be analysed more often than common events. It was also mentioned previously in Section 5.4.4 that Zipf distributions have been used to model web traffic, so it is not unreasonable to think that Grid traffic may some day follow the same pattern. Other Grid applications however may use different access patterns if for example the next file to be requested depends on the outcome of processing the current file.

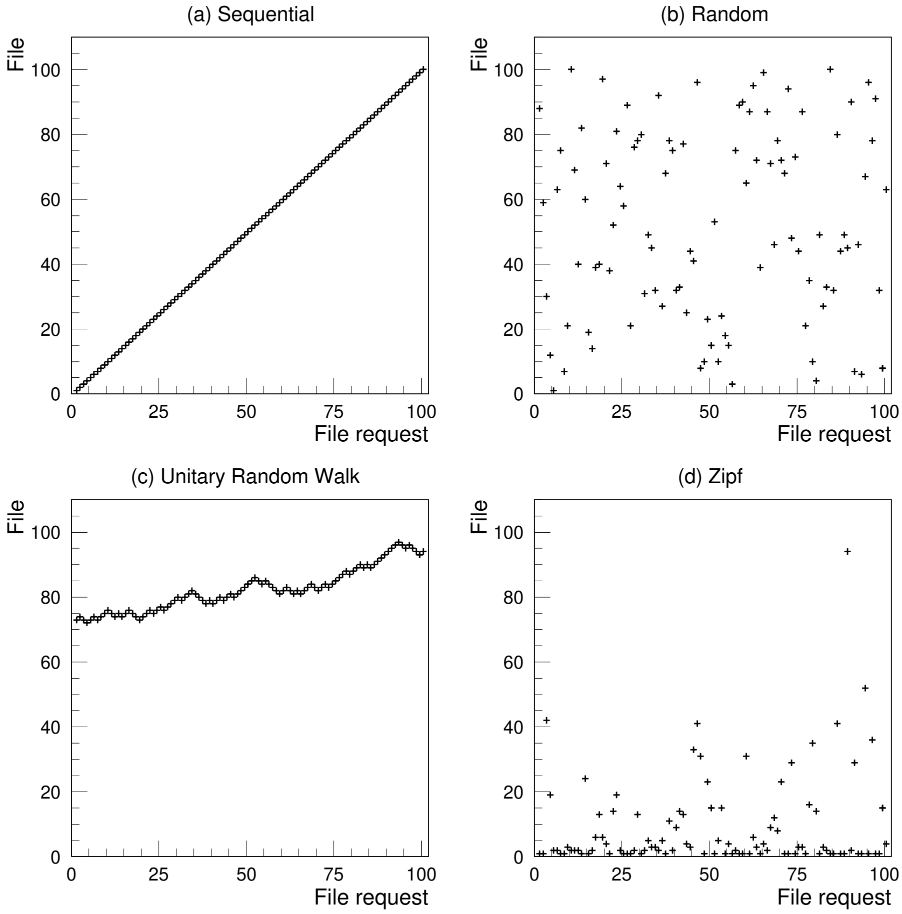


Figure 6.10: File access patterns: (a) Sequential, (b) Random, (c) Unitary Random Walk and (d) Zipf.

6.4.5 Scheduling Policies

The scheduling policy uses certain metrics to determine the CE that will process a Grid job. Using the access cost of the required data for the job as a metric was discussed previously in Section 5.1.3. In OptorSim we have implemented access cost scheduling as well as other more simple policies. There is a choice of four schedulers:

- *Random:* Jobs are scheduled to a random CE selected from the set of CEs that are willing to run the job.

- *Queue Length:* The site with the shortest queue of jobs waiting to run is chosen to run the job.

- *Job Access Cost:* Access cost is defined as the estimated time, based on current network bandwidth observations, for the given CE to obtain all the files required by the job. The job is submitted to the site with the minimum value of access cost.

- *Queue Access Cost:* Queue access cost (or Combined Cost) is the sum of the access costs for all the jobs in the queue at the given CE and the job is scheduled to the CE with the minimum value of queue access cost.

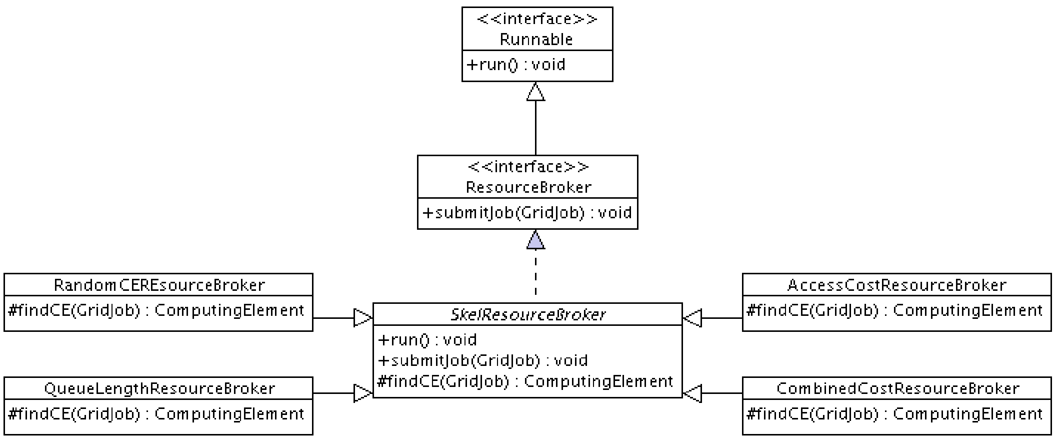


Figure 6.11: UML class diagram showing the different types of Resource Broker in OptorSim.

These scheduling policies are embedded in various implementations of the ResourceBroker interface, which are shown in the UML class diagram in Figure 6.11. ResourceBroker extends the Runnable interface so it runs as a thread and defines a *submitJob()* method to be called by the Users. An abstract skeleton implementation of this interface is provided by SkelResourceBroker which provides all the functionality of a Resource Broker such as the main *run()* method and *submitJob()*. It also defines an abstract method, *findCE()*, which is used to find the best CE to which to submit each job. Each class that extends this class supplies its own implementation of this method according to the metric it uses to rank the CEs.

6.4.6 Replica Optimisation Strategies

The most important functionality that a replica optimisation strategy provides is contained in the two methods *getAccessCost()* and *getBestFile()*. The former is used by some Resource Brokers to calculate the access cost of a job for a particular CE and the latter is used by CEs to obtain the location of the best replica to access. An interface, *Optimisable*, is defined which contains these two methods and all the different classes for each optimisation strategy must implement their own version of them. The implementing classes and subclasses are shown in the UML class diagram in Figure 6.12.

SkelOptor is an abstract simple implementation of *Optimisable* and defines the *getAccessCost()* which is used by all the Optimisable classes. It uses the Replica Location Service to look up all the replicas of each file required by the job, then using network bandwidth measurements finds the replica that can be accessed in the minimum time. This is done for each file passed to *getAccessCost()* and the method returns the sum of the access costs for all the files in the List. The *getBestFile()* implemented by this class does not perform any replication but simply uses the Replica Location Service and network information to locate the copy of the given file that can be accessed fastest. *SimpleOptimiser* provides a concrete version of *SkelOptor* with no extra functionality.

ReplicatingOptimiser uses *getBestFile()* from *SkelOptor* but then will always try to replicate the file to a SE on the same site as the CE calling the method if a replica does not already exist there. If replication fails because there is not sufficient space in the SE, *chooseFilesToDelete()* is called, which returns a List of files that should be deleted to create space for the new replica. This method is not defined in *Replicating Optimiser* but is provided by subclasses which use different strategies to calculate which files to delete. *LruOptimiser* and *LfuOptimiser* use Least Recently Used and

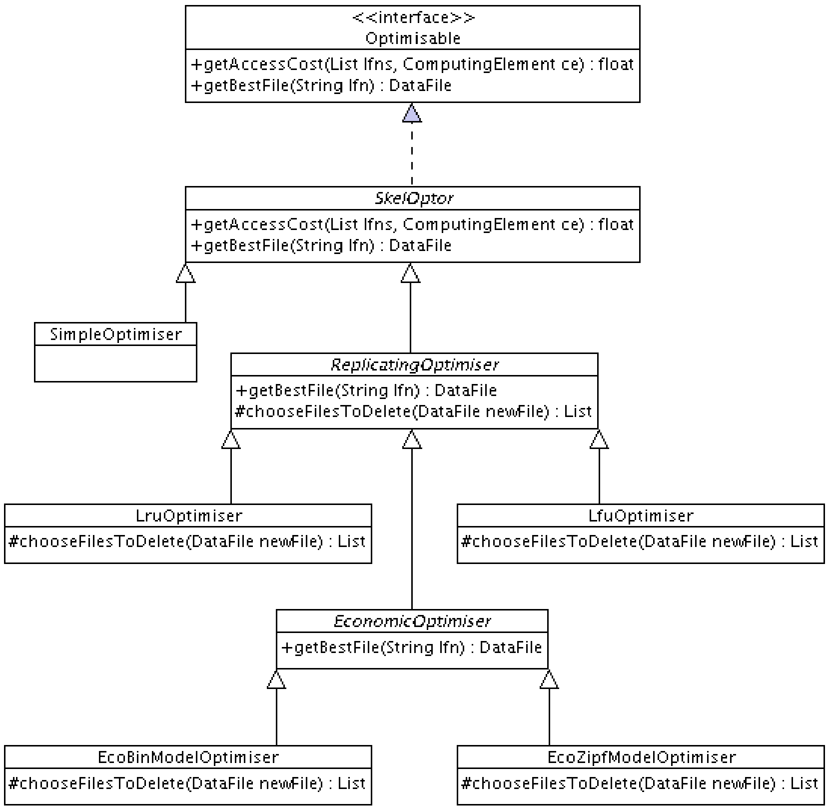


Figure 6.12: UML class diagram showing the Replica Optimiser classes in OptorSim.

Least Frequently Used algorithms respectively (see Section 5.3).

The *EconomicOptimiser* implementation of *getBestFile()* calls the CE's Access Mediator to tell it start an auction for the file. The Storage Brokers make the decision of whether to replicate the file or not, based on the estimated value of the files currently in their SEs and the value of the potential new replica. If the decision is made to purchase a replica, *chooseFilesToDelete()* returns the least valuable files based on the binomial or Zipf-based analysis of the file access history, depending on whether the particular class used is *EcoBinModelOptimiser* or *EcoZipfModelOptimiser*.

The aim of this class structure is so that any calls made to *getAccessCost()* or *getBestFile()* simply use the *Optimisable* interface, and the particular strategy to

Strategy	Replication Decision	Replica Selection	File Replacement
SimpleOptimiser	Never Replicate	RLS + Net Mon	None
LruOptimiser	Always Replicate	RLS + Net Mon	Least Recently Used
LfuOptimiser	Always Replicate	RLS + Net Mon	Least Frequently Used
EcoBinModelOptimiser	Binomial Pred. Func.	Auction Protocol	Binomial Pred. Func.
EcoZipfModelOptimiser	Zipf Pred. Func.	Auction Protocol	Zipf Pred. Func.

Table 6.3: Optimisation strategies used in implementations of the Optimisable interface.

be used is selected automatically at run-time based on the input parameters. The functionality of the five concrete implementations of *Optimisable* are summarised in Table 6.3, in terms of the three areas of replica optimisation discussed in Sections 5.2.1 - 5.2.3. This table shows how the classes in OptorSim correspond to the strategies shown in Table 5.1. The structure of these classes also make it easy to add classes with new strategies into the code.

6.5 Summary

This chapter has described the Grid simulator OptorSim and how it can be used to evaluate Grid optimisation strategies under realistic conditions. The simulation allows not only investigation of how to optimise the performance of current operational Grids, but also can be used to help design and build future Grids. OptorSim models Grid components related to data management and has a large parameter set that can be varied to simulate an array of conditions, so that any type of Grid, jobs and users can be examined. Days worth of Grid activity can be easily simulated in a few minutes and the simulation provides through text output and a GUI a range of information, graphs and statistics to show what is happening in the Grid.

The way the simulation is implemented in Java provides a portable, easy to use, extensible code base for future development. It is straightforward for any project looking into replica optimisation to add their own strategies without changing the core OptorSim code, and the license under which OptorSim is released allows unlimited modification and re-distribution. In the next chapter OptorSim is used to simulate different Grid situations and examine the effect of optimisation strategies on the

Grid's performance.

Chapter 7

Evaluation of Grid Optimisation Strategies

In this chapter OptorSim will be used to compare and evaluate the performance of optimisation strategies in a variety of simulated Grid scenarios. Metrics provided by OptorSim that are used to measure performance are first explained, then the replica optimisation strategies and scheduling policies discussed in the last two chapters are tested using very simple Grid topologies. This validates whether they perform as we expect they should, and that they have been implemented correctly in the simulation before they are used to test any complex situations. Then, using real examples of Grid testbeds and typical jobs from high energy physics users, the performance and sensitivity to certain parameters of the optimisation strategies is examined. Finally, for each configuration, the economic model is analysed in terms of the profits and losses made by the agents.

7.1 Simulation Set-up

In this section we describe briefly the input parameters that are varied to simulate different Grid situations and the outputs used to evaluate the optimisation strategies. We also explain how the results will be presented and the treatment of errors.

7.1.1 Input Parameters

The main input parameters into OptorSim were described in Section 6.2.2 and summarised in Table 6.1. Two important inputs are the Grid configuration and the set of jobs submitted to the Grid, and the combination of these two can be categorised using the following characteristics:

- D: The ratio of the average SE size to the total dataset size for all jobs, i.e.

$$D = \frac{\text{Average SE size}}{\text{Total dataset size}}$$

  A value of $D > 1$ indicates that each SE has more than enough space to hold a complete set of replicas, so little file deletion will take place and the choice of replica optimisation strategy should not greatly affect the performance of the Grid. A value of $D < 1$ means the SE sizes are relatively small and so choices have to be made of which replicas to keep. In this case the optimisation strategy will be more important.

- C: The average network connectivity of a site, i.e.

$$C = \frac{\sum_{\text{sites}} \text{Site connection bandwidth}}{\text{Number of sites}}$$

  This describes how well connected the sites are to the Grid, and hence how quickly data files can be copied around. If data is being heavily replicated, the value of C can have a large effect on the performance of the Grid. When sites have low bandwidth connections, the slow rate of data transfer could lead to large queues of jobs accumulating at the site, whereas a Grid with a high value of C can process jobs quickly and thus have little or no queues at its sites.

The characteristics of the Grid topology and jobs are summarised at the start of each section that uses a new configuration. Other important parameters that are varied to obtain the results presented here are:

- The number of jobs submitted to the Grid.

- The rate at which jobs are submitted to the Grid.

- The period of access history used in optimisation calculations.
- The pattern with which files are accessed during a job.

7.1.2 Evaluation Metrics

In order to evaluate and compare the effectiveness of the various optimisation strategies implemented in OptorSim, we use the following metrics:

Mean Job Time

The time to execute a job is measured from the time it is submitted to the Resource Broker to the time a CE finishes processing all the job's required files. The mean job time is defined as the sum of the times to execute every job, divided by the number of jobs completed. A typical Grid user would probably consider this to be the most important measure of how the optimisation strategy performs and it was used in [13, 14, 27] to evaluate various strategies.

Hit Rate

The aim of the replica optimisation strategies is to ensure that as many file requests as possible can be met with a file stored locally. The success of the strategy can then be measured by the hit rate, which is defined as the percentage of file requests satisfied with a local file, without any replication taking place. The term comes from memory caching algorithms (see Section 5.3) where the optimum algorithm maximises the number of hits (data requests satisfied by data in memory rather than on disk).

Network Usage

File replication is essential to a distributed Grid system but it takes time and uses network bandwidth. Thus, a good balance must be found where any replication is in the interest of reducing future network traffic. We define *effective network usage* as a measure of how well the optimisation strategy uses the network resources:

$$\text{effective network usage} = \frac{N_{\text{remote file accesses}} + N_{\text{file replications}}}{N_{\text{total file requests}}}$$

where $N_{\text{remote file accesses}}$ is the number of times a CE reads a file from a SE on a different site, $N_{\text{file replications}}$ is the total number of file replications, and $N_{\text{total file requests}}$ is the total number of file requests from all CEs. This gives the ratio of files transferred across the network to total files requested. For a given network topology, a lower effective network usage indicates the optimisation strategy is better at replicating files to the "correct" location.

CE Usage

Another resource which is of interest is the computational power usage, which we define as the percentage of time that a CE is transferring or processing data. A good job scheduling policy should be able to maximise the CE usage for the whole Grid by spreading the workload, avoiding the situation where some sites lie idle while others have long queues of jobs.

7.1.3 Presentation of Results

In the figures in this chapter, values of these evaluation metrics are used to show how the performance of different optimisation strategies varies when certain parameters are changed. The four strategies compared are Least Recently Used (LRU), Least Frequently Used (LFU), the binomial-based economic model (eco-bin) and the Zipf-based economic model (eco-zipf). As there are random factors in each simulation run, each result shown is the average taken from 10 or more simulation runs. The spread of values for mean job time running the simulation 1000 times with a fixed set of parameters with the Grid configuration used in Section 7.3 is shown in Figure 7.1. As each simulation run is independent of any other, we would expect that the distribution of mean job times approximates a normal distribution for a large number of simulation runs. This can be seen from the fitted normal distribution curve in Figure 7.1. Thus the mean $\bar{x}$, the standard deviation σ, and the standard error on the mean Δx of any N measurements made using repeated simulation runs have the standard forms:

$$\bar{x} = \sum_{i=1}^{N} x_i / N \qquad \sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N-1}} \qquad \Delta x = \frac{\sigma}{\sqrt{N}}$$

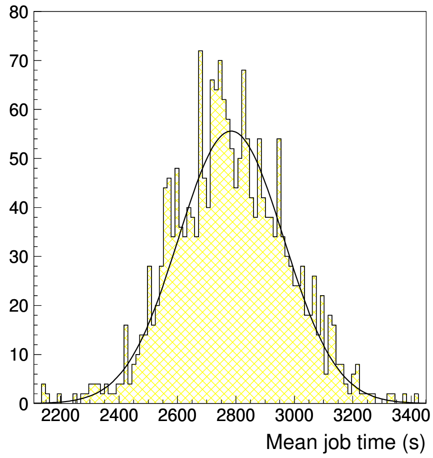


Figure 7.1: Values of mean job time for 1000 repeated simulation runs and fitted normal distribution curve.

Comparison of the standard deviations of results using the same parameters and different optimisation strategies gives a good indication of the relative stability of the strategies and the standard error in the mean allows a good estimation of the difference in performance of any two strategies. Therefore all results in this chapter show the mean as a point, the standard error in the mean as an error bar with ticks and standard deviation as an error bar with no ticks, calculated according to the above formulae.

7.1.4 Analysis of the Economic Model

At the end of Sections 7.3, 7.4 and 7.5, we analyse what happens inside the economic model for each testbed used in these sections by looking at the profits and losses made by the optimisation agents during the course of OptorSim simulations. The currency unit we use is Grid Credits (GC) and all agents start with zero GC. The architecture and function of the Access Mediator (AM) and Storage Broker (SB) agents was described in Section 5.4.2. AMs, which only buy files, can never make a profit, and their aim is to minimise the money they spend. SBs can make a profit or a loss as they buy and sell files from other SBs and sell to AMs.

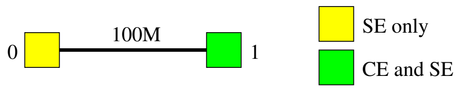
The relative values of particular files are calculated by SBs from the prediction functions implemented in the binomial-based or Zipf-based economic model, which were described in Section 5.4.4. The prediction function uses a past access history of file requests to predict the number of times a file will be accessed in the future, which is then used to assign a value to the file. The binomial-based model (eco-bin) finds the mean value and spread of file identifiers from the access history and constructs a binomial distribution centred around the mean value from which the value of any file identifier can be taken. Similarly, the Zipf-based model (eco-zipf) builds a Zipf-like distribution of the file identifiers in the access history, and the value of any file with a given identifier can be taken from this distribution.

In the current version of the economic model implemented in OptorSim, the values of files calculated by the SBs are separated from the pricing mechanism used in the auction process. If a SB is replying to an auction's request for bids, it returns a bid representing the estimated network cost of selling the file, i.e. the estimated time to transfer a copy of the file to the site conducting the auction. This pricing mechanism used in the auctioning process can be thought of as an agent paying for the use of the network resources of a site providing a file. When deciding whether or not to replicate a file, the file value prediction function assigns values which are unrelated to these values and essentially arbitrary, being used only to compare the relative worth of different files.

If the SB does not hold the file in its local SE, and if the estimated value according to the prediction function is more than that of the least valuable file it has, the SB starts a nested auction to obtain a replica of the more valuable file. The cost of buying a file is not taken into account when calculating whether it will be profitable to do so, i.e. the nested auction is performed after the decision has been made to replicate. In addition to the fact that the file values and auction prices are unrelated (and hence to avoid the problem of trying to relate them), this was done in order to simplify OptorSim and decrease the time taken to run the simulation. It avoids the situation where every SB starts a nested auction for every file request in order to evaluate whether or not to replicate, which would result in a huge amount of auction messaging traffic. In this case however more replication than is optimal could take place. Analysis and comparison of the model for each testbed should show whether the model is stable and scalable and ascertain which factors affect the profits and losses made by the agents.

7.2 Validation of Replica Optimisation Strategies

For the first results, we examine the behaviour of replica optimisation strategies in the simplest situation possible, depicted in Figure 7.2.



No. of sites	Total dataset size	D	C
2	1000 GB	0.55 - 1	100 Mbit/s

Figure 7.2: A simple Grid set up with two sites, of which one contains a CE and SE and the other only a SE.

This Grid consists of two sites connected by a 100 Mbit/s network link. One site contains a SE that stores the master replicas of all the data, and the other contains a CE and an empty SE. This situation is analogous to the disk/memory cache situation discussed earlier. One Grid job is defined which requires 1000 files, each 1 GB in size and 10 of these jobs are submitted to the Grid. The size of the SE on site 1 is varied from 100 GB to 1000 GB so that in all simulations (except where the size is 1000 GB), the replica optimisers must make decisions on replicating and deleting files because this SE cannot hold all the files required by the job. These SE sizes correspond to values of D between 0.55 and 1. Due to the simplicity of this set up, the random errors in the results are very small, and so the error bars are in most cases obscured by the point marking the mean value.

7.2.1 Storage Element Size

The mean job time and hit rate for several optimisation strategies and varying SE sizes simulating sequential file access patterns are shown in Figure 7.3. In these tests the delay between submitting jobs was large enough (100,000 seconds) that each job could run immediately without waiting for previous jobs to complete. This means the mean job time does not depend on any time spent in a queue, only on the number of replications, and this is why the mean job time and the hit rate appear to be the inverse of each other.

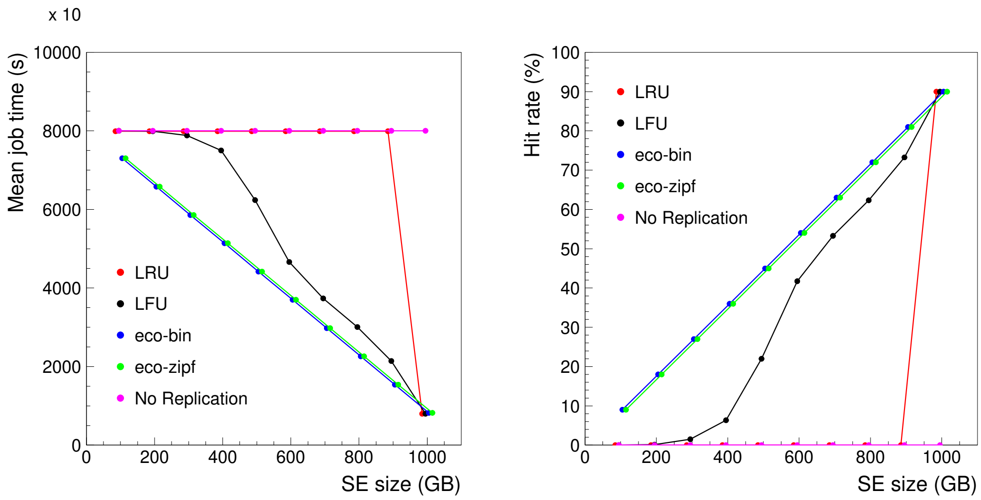


Figure 7.3: Mean job time and hit rate variation with site 0 SE size for a simple Grid using sequential access patterns.

The performance of LRU is the worst of the optimisation strategies and is as bad as having no replication at all. This is because with sequential access patterns a file is guaranteed to have been deleted before it is requested in the next job, if the SE cannot hold all the files required for the job. For example, if the SE capacity is 200 GB, files 1 to 200 can be replicated with no deletion. When file 201 comes to be replicated, file 1 is deleted, then for file 202, file 2 is deleted and so on until the end of the job when the SE has files 801 to 1000. At the start of the next job these files are deleted to replicate files 1 to 200 and this cycle continues for all the jobs, so that there is never any file available locally.

LFU performs better, with mean job time decreasing with increasing SE size. After each job, every file will have been accessed an equal number of times, and therefore when the next job starts files will be deleted randomly. Therefore there is a chance, which increases with SE size, that when a particular file is requested, it is still in the SE from the previous job. This means that with a larger SE size the performance of LFU approaches that of the economic models. Both eco-bin and eco-zipf give identical results, with an exact linear decrease in mean job time and increase in hit rate. The eco model strategies perform slightly better because even though all files are accessed an equal number of times, they tend to keep the same files in the SE throughout the simulation. This is because there is no point in deleting a file to replicate one which is equal in value. LFU on the other hand is less likely to have kept any given file for the duration of a whole job, and so it has to replicate this file again the next time it is requested.

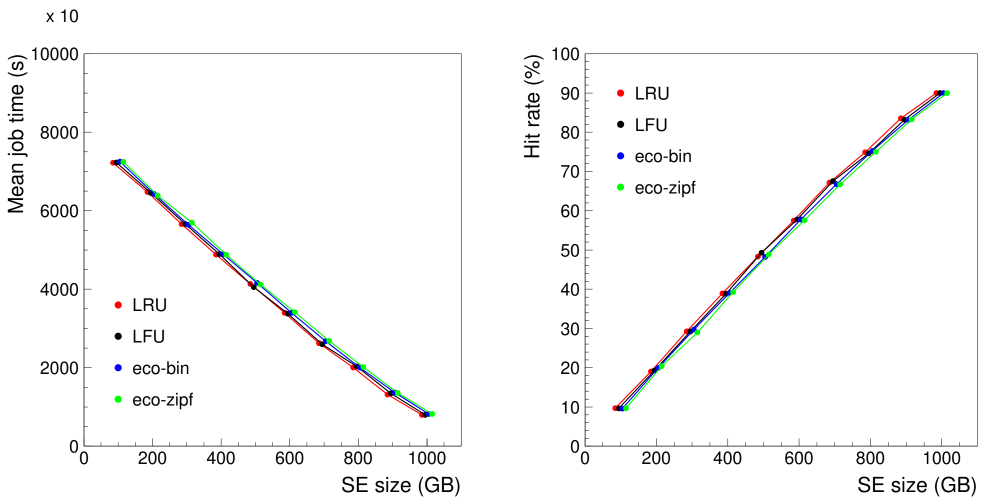


Figure 7.4: Mean job time and hit rate variation with site 0 SE size for a simple Grid using random access patterns.

Results for the same simulation set-up but with random access patterns are shown in Figure 7.4. All optimisation strategies have almost exactly the same performance, with the mean job time decreasing linearly and the hit rate increasing linearly with the size of the SE. Since the access pattern is entirely random, we would expect the hit rate to simply be the ratio of the SE size to the total size of files required for the job since this is the probability that the file is on the local SE (given the SE is full). This is shown to be true in the results, with a slight offset due to the SE filling up during the first job. The mean job time for the eco models is the same as in Figure 7.3 and here LRU and LFU match their performance, because the files that are deleted have no correlation with the files replacing them. The correlation of file requests assumed by the eco model (see Section 5.4.4) is no longer valid and all the strategies effectively act like they are deleting random files.

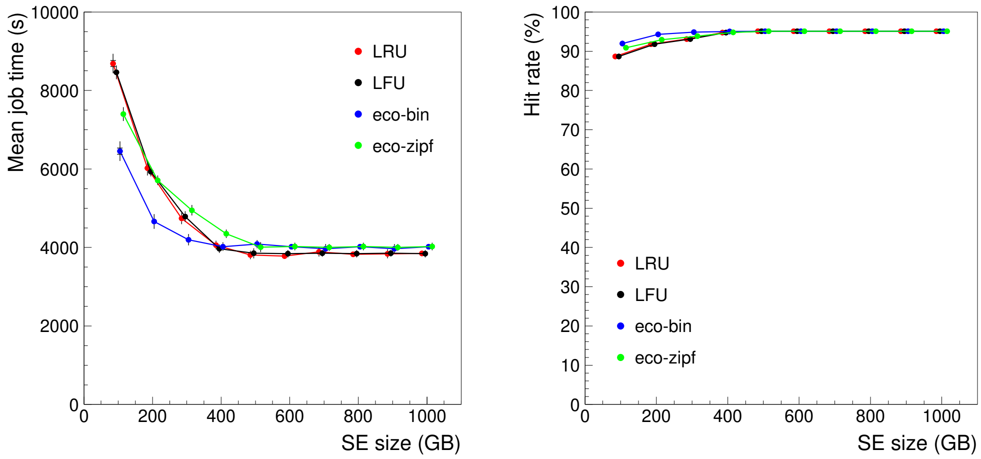


Figure 7.5: Mean job time and hit rate variation with site 0 SE size for a simple Grid using Zipf access patterns.

The performance of each optimisation strategy when Zipf access patterns are simulated is shown in Figure 7.5. Here $\alpha = 0.5$, so the access pattern is not as skewed as that shown in Figure 5.6. The mean job time quickly decreases for all strategies as the SE size increases but levels out above 400 GB. Similarly the hit rate increases steadily and levels out around 95% above 400 GB. The reason for this is that using the Zipf distribution, only a small fraction of the possible files needed for a job are actually requested (see Figure 6.10(d)). In this case there were 10 simulated jobs requiring 1000 files each, making a total of 10,000 file requests, but only 489 distinct files were actually requested. This meant that with an SE size 500 GB or higher all the required files could be accommodated, hence no optimisation decisions needed to be made.

The number of requests for each file, ranked by the number of requests, is shown

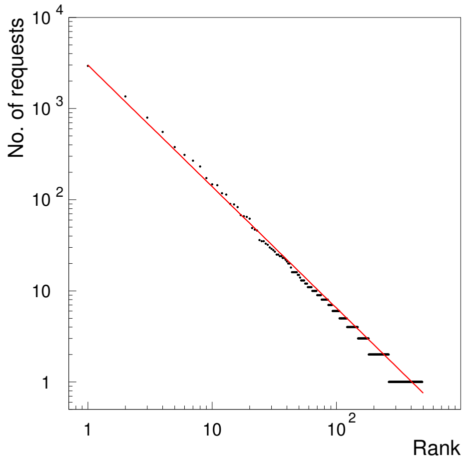


Figure 7.6: Number of requests for each file ranked by number of requests, running 10 jobs on a simple Grid using Zipf access patterns and the fitted line $i^{-0.5}$.

in Figure 7.6. It is very similar to the distribution shown in Figure 5.6, showing the characteristic power law shape common to Zipf distributions. With $\alpha = 0.5$, the 100 most popular files comprise almost 9300 of the total 10,000 requests, which explains why the mean job times using Zipf file access patterns are around an order of magnitude smaller than sequential or random access patterns, where all files are equally popular.

An important measure of the success of an optimisation strategy is how quickly the system tends to a stable optimum state. Figure 7.7 shows the job time for each of the 10 simulated jobs using an SE size of 500 GB and sequential access patterns. The time taken for the first job is the same for all optimisation strategies, but after this the job times for the eco models half in value and remain at a constant level, since the correct binomial and Zipf distributions can be constructed from the access patterns of a single job. For LFU, after the first job, the SE stores a random collection of 500 files. During the subsequent jobs, there will be some probablility that each file is available on the SE, but this probability is much less than it is for the eco models, which always keep the same 500 files stored locally, therefore the job time is around 50% higher for LFU.

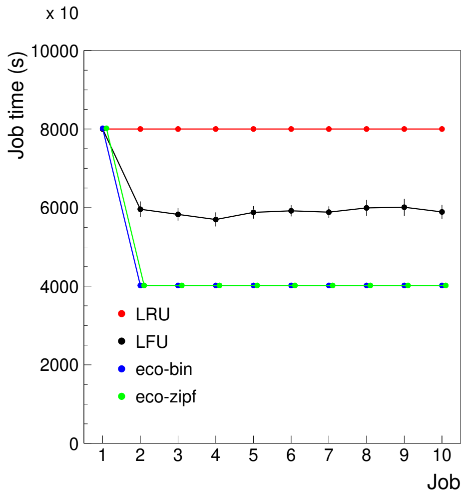


Figure 7.7: Job time for each of 10 jobs submitted to a simple Grid using sequential access patterns.

7.2.2 Access History Size

The history of file accesses is used by the optimisation strategies to predict future file request patterns and the length of the history used in the calculations must be carefully chosen to obtain optimum performance. If the history does not go back in time far enough, the statistics of file accesses will not be accurate, but if the history goes back too far, the optimisation strategies will be slow to react to any changes in the patterns. A large access history also uses more space and is computationally more intensive to process. The ideal length is short enough that it is not too time-consuming to process and changes in file request patterns are noticed quickly, but long enough to have a good view of the overall access patterns.

The mean job time and hit rate for different lengths of access history and sequential access patterns is shown in Figure 7.8. In this configuration the SE on site 1 had a capacity of 500 GB and the time between submitting each job was 100,000 seconds. This means that any access history length less than 100,000 seconds contains information on only one job and access history greater than 1 million seconds is a complete access history of the whole simulation (since 10 jobs were submitted). In general, with increasing access history, the performance of LRU decreases, LFU and

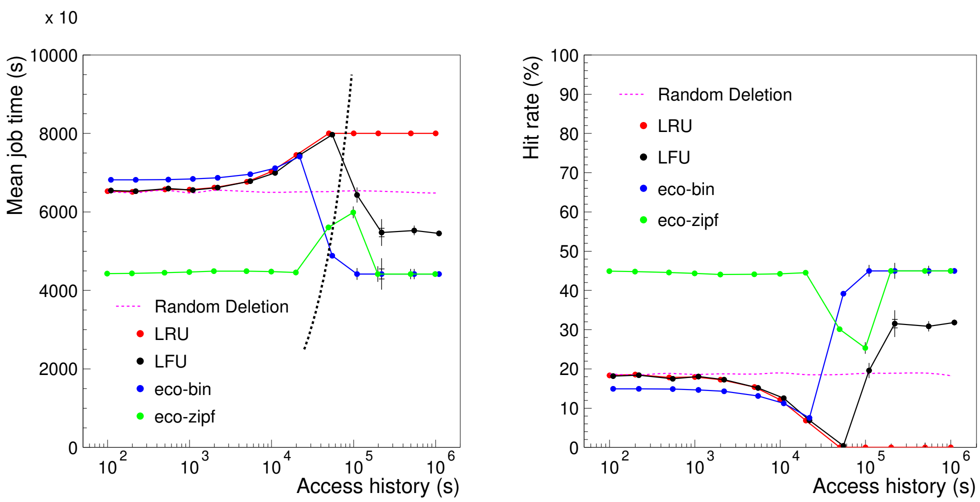


Figure 7.8: Mean job time and hit rate for varying access history with a simple Grid using sequential access patterns. The thick dotted black line corresponds to an access history length equal to the mean job time (i.e. the access history contains information on one full job).

eco-bin increase and eco-zipf remains roughly constant and the best performer. A strategy where files were deleted randomly is shown for comparison as this strategy does not depend on the access history length.

The poor performance of LRU and LFU with small access histories is because in these strategies, when a file is required to be deleted, the first choice is from files not on the access history. Files are selected randomly from these non-access history files until enough space is created. Therefore the performance approaches that of not using an access history but simply randomly deleting files. The performance of LFU and eco-bin gets worse until the access history contains information on at least one full job's worth of files (i.e. the access history exceeds the mean job time), when it rapidly improves. Using LFU, and an access history of say 400 files, when it comes to deleting a file, it will delete one of the 100 files not on the access history (the SE can hold 500 files). However if the access patterns are sequential and the access history is smaller than one complete job, the files not on the access history will be accessed before those on the access history. This means that randomly choosing a file to delete from all the files on the SE gives a higher chance that each file will be present when it is requested in the next job, and so random deletion gives a higher hit rate than LFU for particular lengths of access history.

Eco-bin performs badly with small access history because the file values change so rapidly that seemingly worthless files will be deleted when they are likely to be requested in the immediate future. This strategy requires a large access history of at least one full job to be able to assess well which files are worth keeping. Eco-zipf, like LRU and LFU, chooses files not on the access history to delete first, but these files can still be valued and are potentially worth more than the new replica, hence a reasonable economic evaluation of all files can still take place with a small access history. What tends to happen is that new files are valued equal to files already on the SE, so they are not replicated and the same files remain on the SE all the time. Similar values of access history cause a degradation of performance as did for LFU and eco-bin, but eco-zipf overall gives excellent performance for small and large access histories.

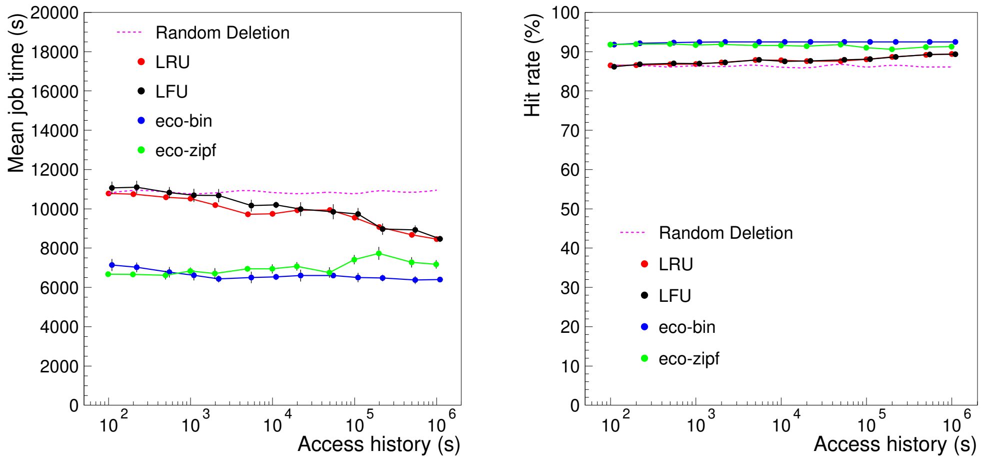


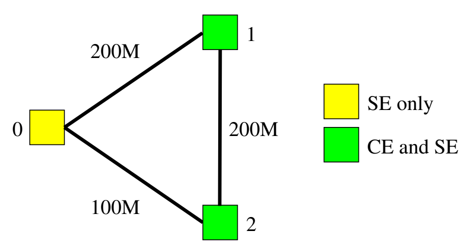
Figure 7.9: Mean job time and hit rate for varying access history with a simple Grid using Zipf access patterns.

Results for the same set up but with Zipf access patterns and an SE size of 100 GB are shown in Figure 7.9, showing very different results from sequential access patterns. LRU and LFU improve with larger access history, while both the economic models remain constant and approximately independent of the access history length. Because the access pattern is semi-random, there are none of the problems seen for sequential access patterns with LRU and LFU as the access history increases. Instead, the files not on the access history genuinely are the files least likely to be accessed next and the increasing access history merely makes the analysis of the distribution even more accurate.

Even with a small number of files in the access history, the binomial and Zipf distributions can easily be fitted to the pattern. In other words, the observed shape of the access patterns can be inferred from only a few data points, which is not the case for sequential access patterns. This explains why the economic models have a constant performance for all lengths of access history, with mean job times 20 - 40% faster than the other strategies.

In conclusion, studying the different replica optimisation strategies under these very simple conditions has shown that they behave as they might be expected to. Therefore in simulating more complex situations we should be able to belive the replica optimisation strategies will behave as they should, and any results presented should be an accurate representation of the performance of these strategies.

7.3 Validation of Scheduling Policies



No. of sites	Total dataset size	D	C
3	1000 GB	0.67	200 Mbit/s

Figure 7.10: A Grid set up with three sites, two of which contain CEs.

To examine the different scheduling policies implemented in OptorSim, we use another simple set-up, this time consisting of three sites. All the sites have SEs and two of them have CEs. The three sites are linked to each other by networks of varying bandwidth, as shown in Figure 7.10. As there is now a choice of locations to which to schedule jobs, the scheduling policy becomes important in the optimisation process. The four different scheduling algorithms explained in Section 6.4.5 are tested here - random, queue length, access cost and queue access cost.

In these tests there were 10 different jobs defined which could be submitted to the Grid, shown in Table 7.1. Each job required 100 files of size 1 GB which each took 10 seconds to process by the CE and all the files were unique to one job. The SE at site 0 held all the files initially and the SEs at sites 1 and 2 had a capacity of 500 GB (the CEs at both sites could run any of the 10 jobs). The probability of a particular job being submitted to the Grid also varied on a linear scale from job to job in order to make some jobs more popular than others. In each simulation, 100 jobs were submitted and since there were now several random parameters involved, such as selecting a job to submit, each result is the average of 10 simulation runs, and the standard deviation and error in the mean are shown as an error bar as described earlier.

Job	Files Required	Probability	Job	Files Required	Probability
1	File 1 - File 100	1.81%	6	File 501 - File 600	10.91%
2	File 101 - File 200	3.64%	7	File 601 - File 700	12.73%
3	File 201 - File 300	5.45%	8	File 701 - File 800	14.55%
4	File 301 - File 400	7.27%	9	File 801 - File 900	16.35%
5	File 401 - File 500	9.09%	10	File 901 - File 1000	18.18%

Table 7.1: The job set for the three site Grid, with the files required for each job and the probability each will be submitted by a user.

7.3.1 Scheduling Policy vs. Replica Optimisation Strategy

The first set of figures show the mean job time and CE usage for each combination of replica optimisation strategy and scheduling policy. The job submission rate could be important in determining the performance of the scheduling policy, since this determines the lengths of the queues of jobs waiting to run at the site, i.e. the workload.

If the jobs are submitted faster than the CE can process them, queues of jobs will build up and scheduling policies which take workload into account should perform better at balancing the load between the sites. On the other hand, if the CEs can process the jobs faster than they are submitted, the workload is not a deciding factor and the main cause of delay when the job is running will be how quickly the data can be transferred. In this case, scheduling policies which take into account the data access cost should perform well.

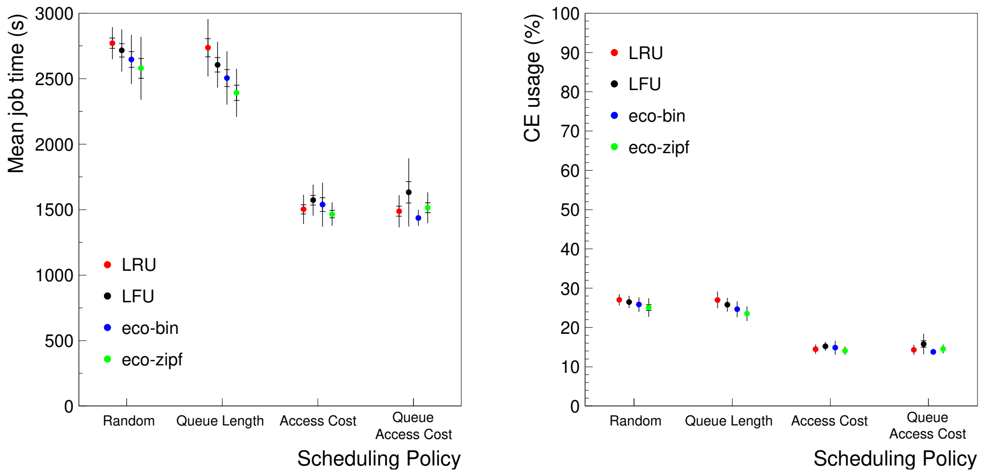


Figure 7.11: Mean job time and CE usage for a three site grid with a delay between job submission of 5000 seconds.

Figure 7.11 shows the mean job time and CE usage for the different combinations of scheduling policies and replica optimisation strategies when the jobs were submitted at a rate of one job every 5000 seconds. This meant that there were never any queues at the CEs, so all jobs could start running immediately after they were submitted. We would thus expect queue length scheduling to show the same characteristics as random scheduling and queue access cost to perform the same as access cost. This is shown to be true, with the access cost schedulers having mean job times just over half that of the other schedulers. There is little or no difference in the replica optimisation strategy used with access cost policies, because the scheduler is very effective at scheduling jobs to the data, thus there are fewer files replicated.

When there are queues of jobs at a site, it is very likely that the files contained in local storage will have changed by the time the job starts, so the replication strategy becomes more important, as can be seen in the next sets of results.

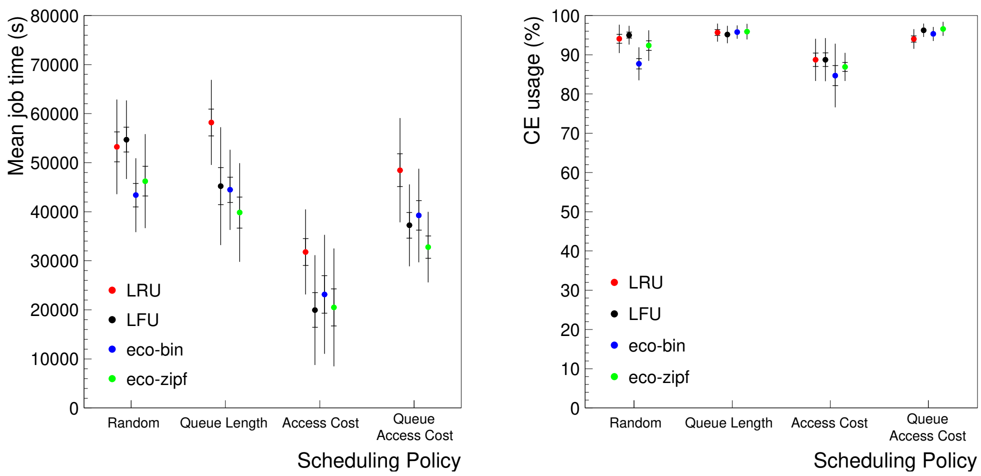


Figure 7.12: Mean job time and CE usage for a three site grid with a delay between job submission of 1000 seconds.

A job submission rate of one job every 1000 seconds was used to obtain the results shown in Figure 7.12. This rate meant that roughly half of the 100 jobs had been processed by the time they had all been submitted, and so at this time the queue at each site was around 20 - 30 jobs. The mean job time has increased by more than a factor of 10, because the time spent in the queue is counted as part of the job time, and the CE usage is near 100% for all scheduling policies because there is always a queue of jobs waiting to run.

Access cost scheduling reduces the mean job time but does not use all the CEs to their full potential. This is because the job is scheduled to the data, so if several consecutive jobs are submitted requiring similar data, they will all be scheduled to the CE that holds the greater number of the required files. Then the other CE could lie idle for some time before a job is submitted for which it can provide a lower access cost. There is not a large difference in how each optimisation strategy performs, taking into account the random factors associated with each simulation run, however

LRU is generally worse than the other three.

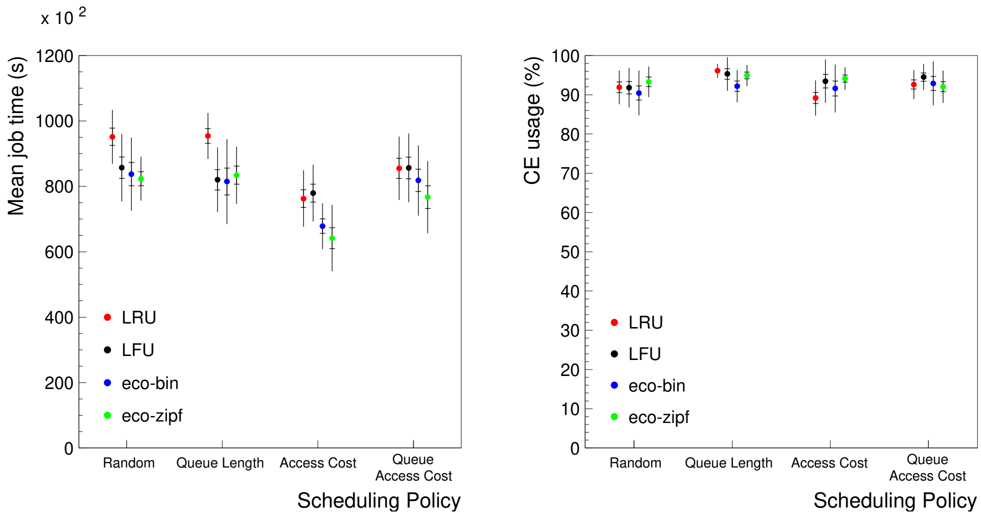


Figure 7.13: Mean job time and CE usage for a three site grid with a delay between job submission of 200 seconds.

A fast job submission rate of one job every 200 seconds was used in Figure 7.13. At this rate, only 4 jobs had been completed by the time all 100 had been submitted to the Grid. This meant that the scheduler had very limited information to make decisions on and this, along with the longer queues at each CE, explain the rise in mean job time. Access cost scheduling does not give as much of an improvement over random and queue length as before because by the time many of the jobs begin to run, the files stored locally have changed and so the CE does not necessarily have the best access cost for the job anymore. We can thus conclude that the access cost schedulers perform best when there is fresh information on the state of the Grid. The difference in performance of the scheduling policies becomes more pronounced when larger Grids with sites containing widely varying resources are used, as will be shown later in this chapter.

7.3.2 Network Variations

The next set of studies show how the relative bandwidth between different sites (and hence the value of C) affects the performance of the scheduling policy. The policy should be able to cope equally well with sets of sites connected by similarly sized network links and sets of sites connected by links with very different bandwidths. To test this the link between site 0 and site 2 was varied from 100 to 1000 Mbit/s while the other two links remained at 200 Mbit/s. This corresponded to varying the mean connectivity C from 200 to 733 Mbit/s.

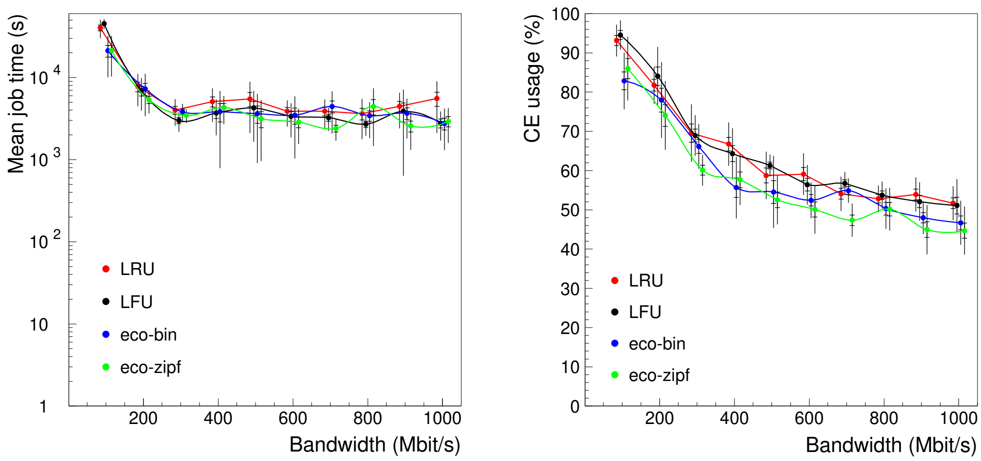
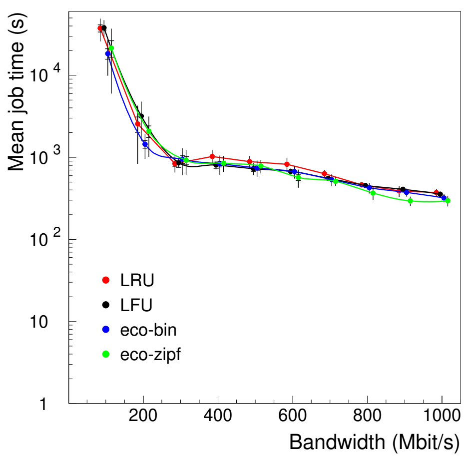


Figure 7.14: Mean job time and CE usage varying network bandwidth from site 0 to 2 for a three site Grid with random scheduling.

The mean job times and CE usage when using random scheduling and queue access cost scheduling are shown in Figures 7.14 and 7.15. The medium rate of job submission (1 job every 1000 seconds) was used for these simulation runs. With random scheduling, upward of around 300 Mbit/s there is no improvement in the mean job time with increased bandwidth although the CE usage does drop. With queue access cost scheduling however, the mean job times continue to fall as the bandwidth goes up, decreasing by a factor of 3 between 300 and 1000 Mbit/s. This shows that the queue access cost policy makes excellent use of the extra bandwidth available. The CE usage also decreases with the increase in bandwidth, due to the fact that jobs run faster with the greater bandwidth and so the CEs are idle more of the time. Using queue access cost scheduling, the CE usage is much lower with increased bandwidth than random scheduling but this is because the majority of the



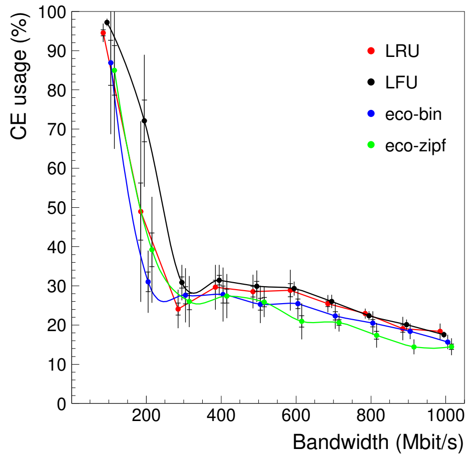


Figure 7.15: Mean job time and CE usage varying network bandwidth from site 0 to 2 for a three site Grid with queue access cost scheduling.

jobs are sent to the better connected CE. Not using all the resources to their full extent is the cost of reducing the times the jobs take to run.

The set of results in this section has shown that the scheduling policies behave as we expect they should, and that there are some situations where the choice of policy does not affect the result. However when the scheduling policy does make a difference, it can be significant and so the choice of policy is important. The differences are even more apparent in a more complex situation with a large set of resources, as we observe in later sections.

7.3.3 Economic Model Analysis

The amount of money spent and gained by each agent was monitored over the course of a simulation run, when the queue access cost scheduling policy was used with jobs submitted at a rate of one job every 1000 s (the medium job submission rate). The profit/loss for each agent is shown in Figure 7.16, for both eco-bin and eco-zipf.

The pattern of the results is very similar for both flavours of economic model although the overall amount of money in the Grid is slightly smaller for eco-zipf. In the early stages of the simulation only the SB at site 0 contains any files, and so it is

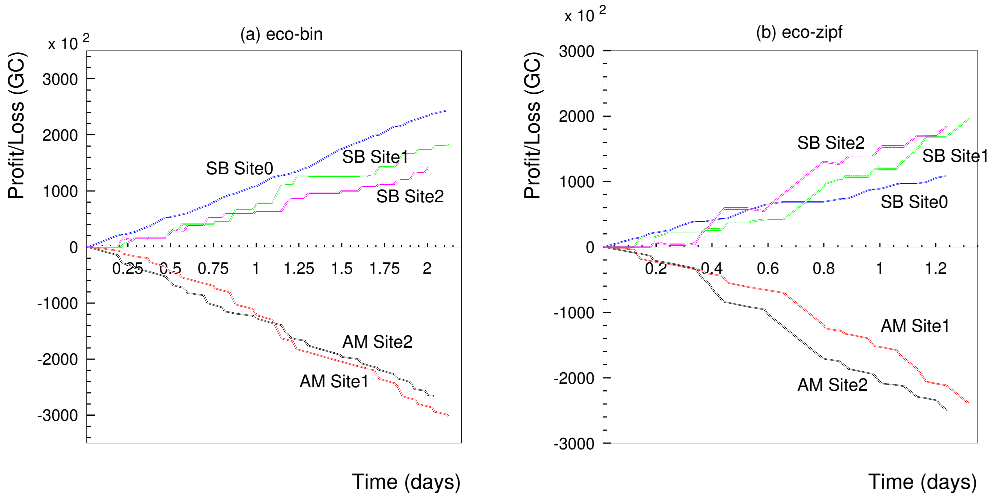


Figure 7.16: Profit/loss for agents on a three site Grid over time using (a) eco-bin and (b) eco-zipf.

the only seller in the market. If the SB at site 1 or 2 receives a request for bids from an AM on the same site and there is enough space for the file requested (and the SB does not already own the file) the SB will always start a nested auction to obtain a replica of the file. Once the nested auction has completed the SB will return a bid of zero GC to the AM, since the network cost of supplying the file to a local AM is zero. At the end of this auction the AM will have received a bid of zero GC from its local SB and a bid from the SB at site 0 of the estimated file transfer time. The result of this auction is then the local SB winning at the price bid by the SB at site 0, but since the local SB has also paid this price to the SB at site 0 to obtain the replica, the net profit is zero. This is the reason the SBs at site 1 and 2 do not make a profit until around 0.2 days have passed.

After this point these SBs have full SEs and can start selling their files at a profit to the AMs on their sites. The profits of all the SBs increase and the losses of the AMs increase roughly linearly, showing the market is stable, i.e. prices do not increase over time. The relative rates of increase of the different SBs tells us that with eco-bin the SB at site 0 wins around half the auctions whereas with eco-zipf the other two sites win the majority of auctions. This implies that less replication takes place when eco-zipf is used, because fewer files are purchased from site 0 by sites 1 and 2 than are bought by agents on the same site as each other. Site 2 is not as well connected to the other two sites as site 1 (see Figure 7.10) so we would expect that the SB at site 1 should make more profit due to this and due to the fact that the queue access cost scheduling policy means it should run more jobs anyway. Eco-bin behaves as expected, with the SB at site 1 making around 25% more than the SB at site 2 but with eco-zipf there is not as much difference because there is generally less replication taking place, so most files are purchased from local SBs. Throughout most of the simulation the SB at site 2 is more profitable although in the end the SB at site 1 finishes with a slightly higher profit.

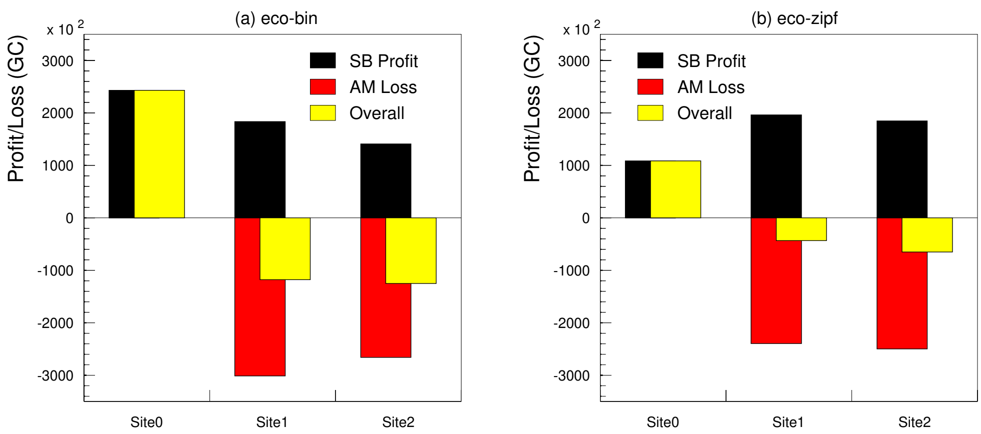
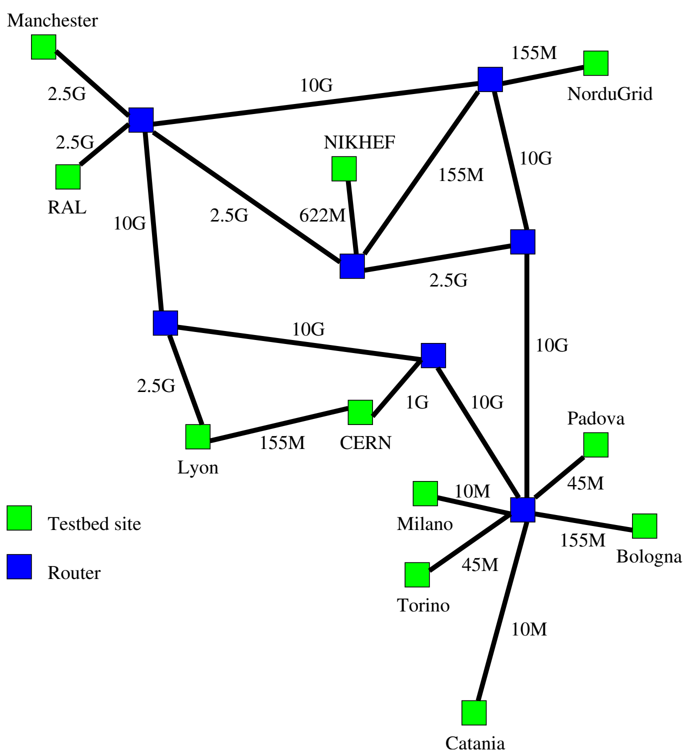


Figure 7.17: Total profit/loss for AMs and SBs at each site of the three site Grid using (a) eco-bin and (b) eco-zipf.

The total profits and losses for the AMs and SBs at each site at the end of each simulation are shown in the bar charts in Figure 7.17 for eco-bin and eco-zipf. The values correspond to the values for the same agents at the end of the simulations shown in Figure 7.16. We can see that for both eco-bin and eco-zipf site 0 makes a profit whereas sites 1 and 2 make a loss overall. Site 2's loss is greater than site 1's, as we expect, due to site 2 having the slower network connection. Eco-bin favours more the SB on the site with the master files (site 0), as this SB makes 2.5 times more profit than with eco-zipf, whereas the other two sites' SBs make around the same profit with both flavours. This, and the fact that overall sites 1 and 2 make smaller losses with eco-zipf suggests that for this situation eco-zipf is preferred over eco-bin. Looking at the job probability distribution for this set-up (Table 7.1) which resembles a Zipf-like power-law distribution, this is not entirely unexpected. This result is reinforced by the smaller mean job times for eco-zipf and queue access cost shown in Figure 7.12.

7.4 European DataGrid Testbed 1



No. of sites	Total dataset size	D	C
11	97 GB	0.37	867 Mbit/s

Figure 7.18: The European DataGrid testbed 1.

For the next set of tests we simulate the European DataGrid testbed 1 topology and resources, shown in Figure 7.18 and Table 7.2. This testbed ran the first major

Site	SE size	Site	SE size
Manchester	20 GB	Lyon	50 GB
NorduGrid	15 GB	Padova	20 GB
NIKHEF	33 GB	Bologna	30 GB
Catania	30 GB	Torino	10 GB
Milano	40 GB	RAL	50 GB
CERN	100 GB		

Table 7.2: SE sizes in EDG testbed 1.

release of European DataGrid middleware in 2002 and it consisted of 10 sites which contributed computing and storage resources and one site (CERN) which contained only storage resources. The storage resources at each site are given in Table 7.2 and are scaled down values from the actual resources that were available. The aim of using this testbed was to examine a realistic high energy physics situation where all raw data is initially produced and stored at CERN, but analysed at other Grid sites. The simulated jobs are based on a scaled down set of high energy physics analysis jobs from the CDF experiment use case [72]. Each file has a size of 1 GB and the total size of the whole set is 97 GB. The sizes of file sets required by each job are shown in Table 7.3.

Data Sample	Total Size (GB)
Central J/ψ	12
High p_t leptons	2
Inclusive electrons	5
Inclusive muons	14
High E_t photons	58
$Z^0 \to b\bar{b}$	6

Table 7.3: Estimated sizes of CDF secondary data sets (from [72]).

This testbed and set of jobs was used in [13] to evaluate replica optimisation strategies using an early version of OptorSim. Here we consider both scheduling and replica optimisation strategies and evaluate how effectively they distribute the workload and data. Contending network traffic based on real throughput measurements between the sites in Figure 7.18 (see Section 6.4.2) is also included in the simulation.

7.4.1 Scheduling Policy vs. Replica Optimisation Strategy

The first set of results compare combinations of scheduling policies and replica optimisation strategies, for two different kinds of users. The first kind of users (random users) submit jobs at random intervals, the delay between jobs having a uniform distribution between 0 and 100 seconds. The second kind (DC04 users) is modelled on the observed job submission patterns for physics analysis during the CMS Data Challenge 2004, where the majority of the jobs are submitted during working hours, reaching a maximum of 100 jobs per hour at 3pm. The job submission rates per hour for these types of users were shown in Figure 6.9. The total number of jobs submitted per simulation was 5000.

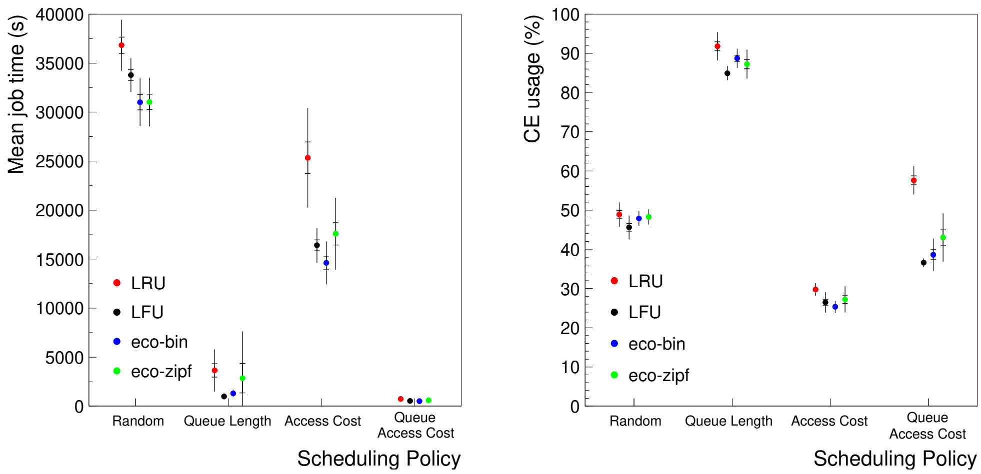


Figure 7.19: Mean job time and CE usage for EDG testbed 1 with varying scheduling algorithms and random job submission rate.

Figure 7.19 shows the mean job time and CE usage for each combination of scheduling policy and replica optimisation strategy simulating random users and Figure 7.20 shows the same results for DC04 users. The pattern of results for mean job time is very similar for both sets of users. The queue length and queue access cost scheduling policies give the lowest job times by a long way, with queue access cost 2-3 times faster than queue length. It seems that, in this situation, the workload of a site is the major factor to consider when scheduling, so the policies that take the

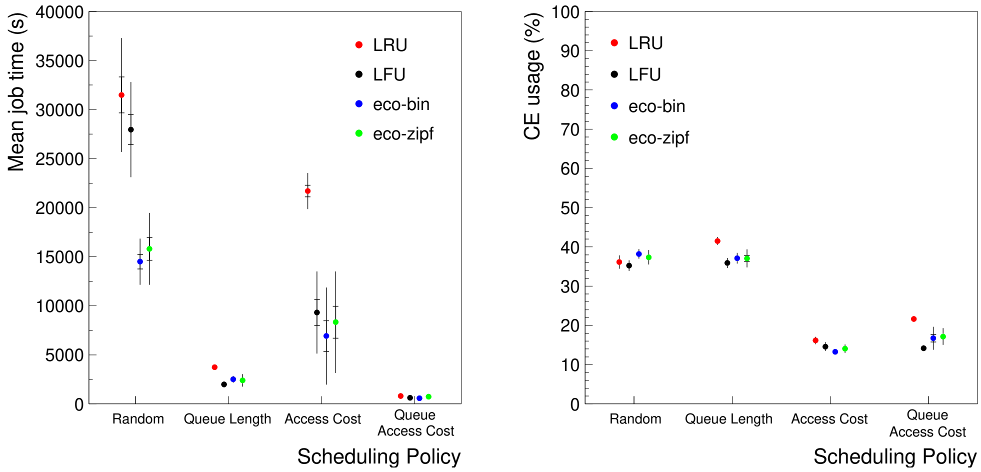


Figure 7.20: Mean job time and CE usage for EDG testbed 1 with varying scheduling algorithms with CMS DC04 job submission rate.

queue of jobs into account perform much better. Queue length has the highest CE usage with random users, because it always makes sure the workload is spread evenly around all the CEs. Taking the queue length into account does not make as much of a difference in the CE usage with DC04 users. This is because there are long periods during the night and early morning when all of the jobs from the previous day have been processed, and hence all of the CEs lie idle, whereas with random users there is a constant stream of jobs submitted.

7.4.2 Access Patterns

The mean job time and the effective network usage of the queue access cost policy for sequential and Zipf access patterns (here $\alpha = 0.5$ again) with DC04 users is shown in Figure 7.21. The mean job time is lower for Zipf access patterns because many files are accessed more than once in the same job, so if the first time a file is accessed it is copied to local storage, it is likely to be available there for subsequent accesses. This also explains the very low effective network usage for Zipf access patterns. There is not much difference between the replica optimisation strategies apart from LRU which gives poor performance with sequential access patterns and the eco models

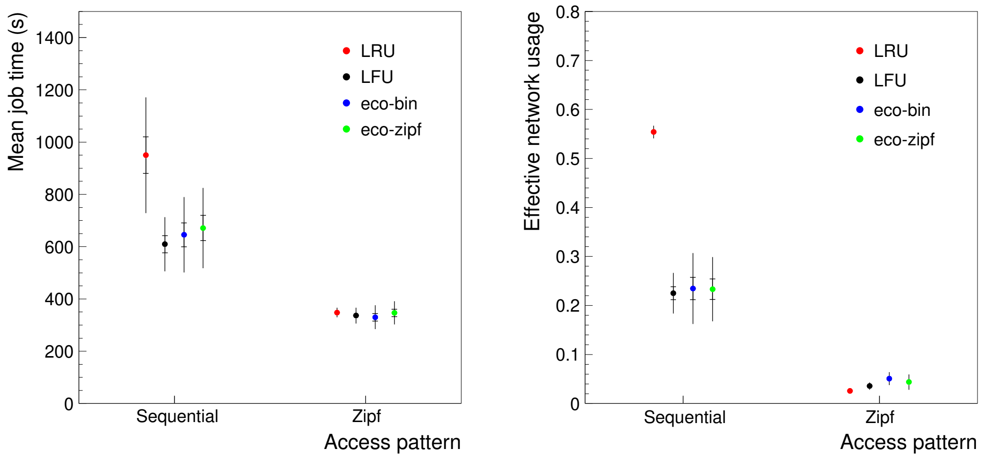


Figure 7.21: Mean job time and effective network usage for sequential and Zipf access patterns for EDG testbed 1 and CMS DC04 users.

seem to be a little more unstable than LRU and LFU.

7.4.3 Long-term Stability

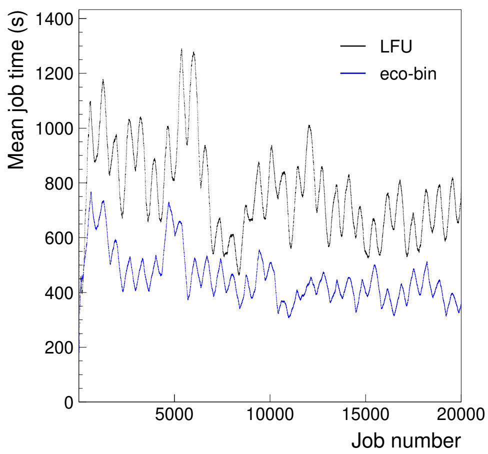


Figure 7.22: Variation of average job time over time using LFU and eco-bin for 20,000 jobs submitted by CMS DC04 users to EDG testbed 1.

To measure how quickly the replica optimisation strategy takes the system to a stable state, the average job time was measured over time and is shown for LFU and eco-bin in Figure 7.22. 20,000 jobs were submitted by DC04 users using the queue access cost scheduling policy and each point in the figure is the average job time for the previous 1000 jobs. The simulation ran for roughly 30 days, and each peak and trough in the figure corresponds to the jobs that had been queued at the end of the day (when queues were full) and jobs submitted at the start of the day (when the queues were empty) respectively. The economic model consistently gives lower mean job times and the variations each day are generally smaller than those of LFU.

7.4.4 Job Workload

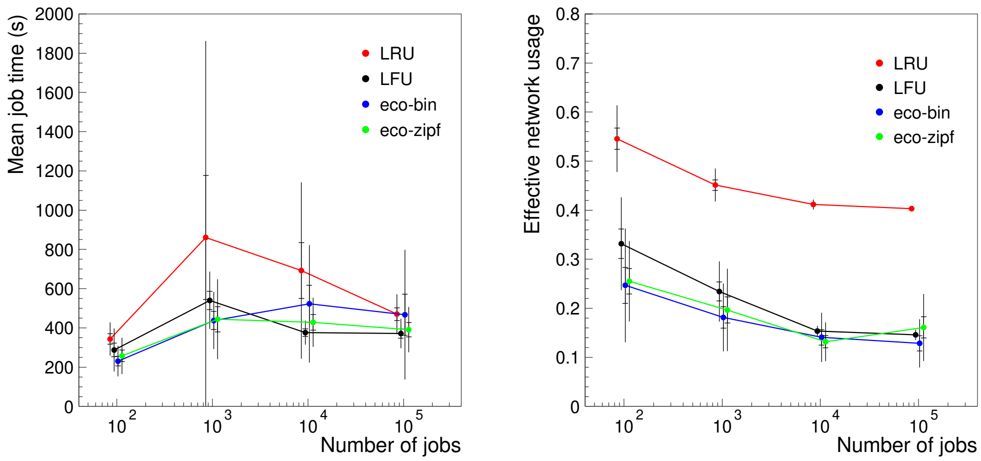


Figure 7.23: Mean job time and effective network usage for varying numbers of jobs submitted to EDG testbed 1.

Figure 7.23 shows results from running the simulation with varying numbers of jobs, from 100 to 100,000. The mean job time tends to start low, then rise to a peak at 1000 jobs, and then fall as the number of jobs increases, and this trend is closely related to the size of the queues at Grid sites. At the start of the simulation the queues are small, but they build up quickly while the files are copied around the Grid. Once the replication process has stabilised the jobs can run faster and hence the queues and the mean job time decrease. The effective network usage meanwhile decreases gradually with increasing numbers of jobs because the amount of replication decreases over time. The eco model strategies perform better with fewer jobs, while LFU is slightly better with more jobs, although the eco models consistently consume less network resources.

7.4.5 Storage Element Sizes

The sizes of the SEs and in turn the value of D may affect the performance of each replica optimisation strategy. We would expect that a lower value of D, i.e. less storage space available per site, would lead to longer job times and larger effective network usage because fewer replicas can be accommodated in the Grid. Figure 7.24 shows the performance of replica optimisation strategies with varying values of D, with $D = 0.37$ corresponding to the sizes listed in Table 7.2, and these values were then scaled by a factor of 2 each time to give each value of D in Figure 7.24. Queue access cost scheduling was used here with DC04 users submitting 5000 jobs.

For low values of D, the mean job time follows a power law relationship with D, showing that a small increase in the resources available can lead to a large fall in the observed job times. This is in contrast to the linear relationship seen for the simple two-site Grid configuration in Figure 7.3. LRU performs badly for all D, as expected, and the other three strategies vary in their effectiveness with different values of D. For very small SE sizes eco-zipf gives the best results, with LFU next best and eco-bin worst. This order is reversed when $D = 0.19$ and by the time $D = 0.37$ there is no discernible difference between the three. The mean job times with higher values of D are high and unstable, with standard deviations of up to 2500 seconds. This is because queue access cost scheduling does not take into account the processing time involved in the job. If a site has all the files available for all the jobs in its queue, it will have a queue access cost of zero and so is likely to have further jobs submitted to it, even though it will take a long time to clear the queue. In extreme cases, a site which is the first to replicate all the files for a particular job can receive all subsequent jobs of this type submitted to the Resource Broker which can result in very large job times.

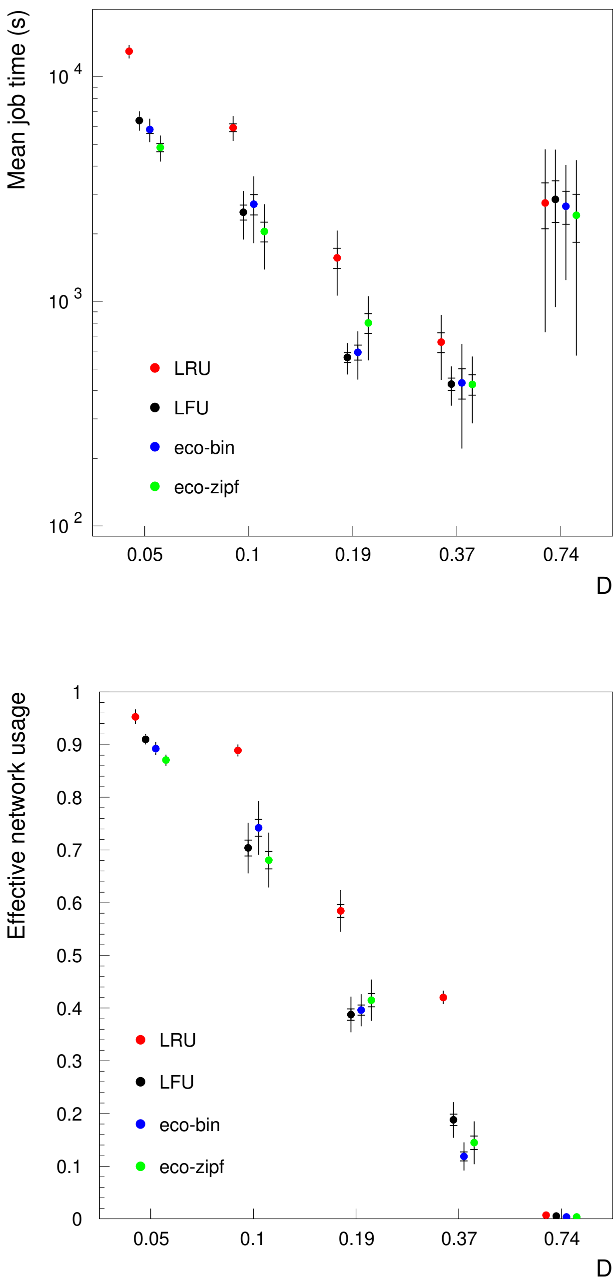


Figure 7.24: Mean job time and effective network usage with varying values of D for the EDG testbed 1.

The effective network usage decreases with increasing D, because much less file replication is required with the larger SE sizes. It drops to almost zero when $D = 0.74$ because in this situation some of the SEs can hold all the files and so no replica optimisation is required. The relative performance of the optimisation strategies follows a similar pattern to that seen for the mean job time, with LRU clearly the worst performer and no significant difference between the other three. Thus we can conclude that the sizes of SEs available to store replicas has a large effect on the performance of Grid jobs, but this effect does not depend significantly on the particular choice of replication strategy employed until space is severely limited, when eco-zipf gives the best results.

In conclusion, LRU performs badly compared to LFU and the eco models, which have roughly the same performance in terms of the time taken to run jobs, but the eco models make slightly better use of the network resources. Queue access cost scheduling gives a large improvement over other scheduling policies for situations where storage space is limited, which is the area of most interest to us here.

7.4.6 Economic Model Analysis

In this analysis of the economic model, 5000 jobs were submitted to the EDG testbed, following the pattern of CMS DC04 users' job submission. The profit made by the SB, the loss made by the AM and the overall balance of each site at the end of the simulation are shown in Figure 7.25 for eco-bin and eco-zipf. The sites are ordered by their network connectivity, smallest first.

Overall, the total profit or loss made by each site is similar for both eco-bin and eco-zipf, but they are more evenly distributed among the individual AMs and SBs with eco-zipf. Some of this difference however is due to the jobs that each site receives, which in turn is affected by the jobs the users submit to the Grid, which varies randomly with each simulation run. Catania and Milano have such slow network connections (10 Mbit/s) that they are never scheduled any jobs, hence they never buy or sell any files. Padova almost exactly breaks even for both eco-bin and eco-zipf, however the large scale of profit and loss for the AM and SB at this site is not due to the number of files traded but the fact that each file is so expensive due to

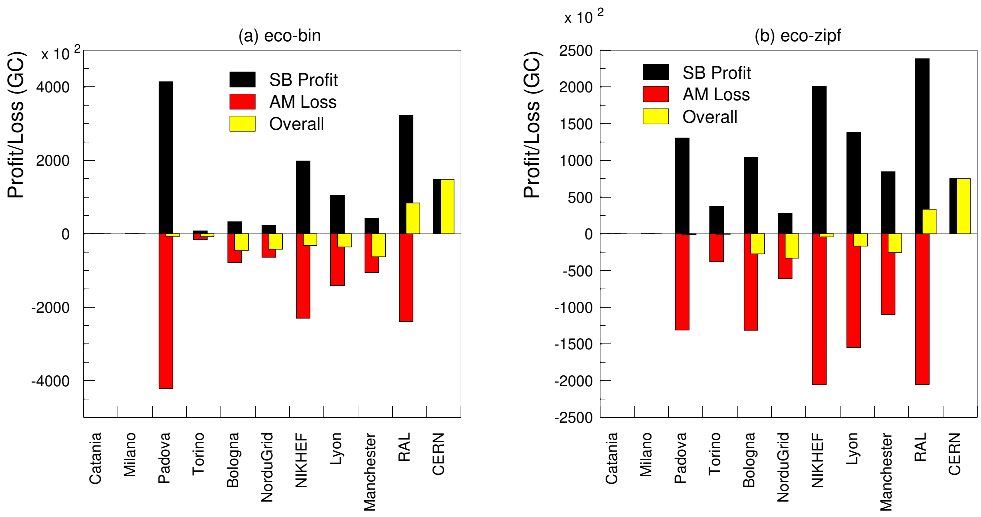


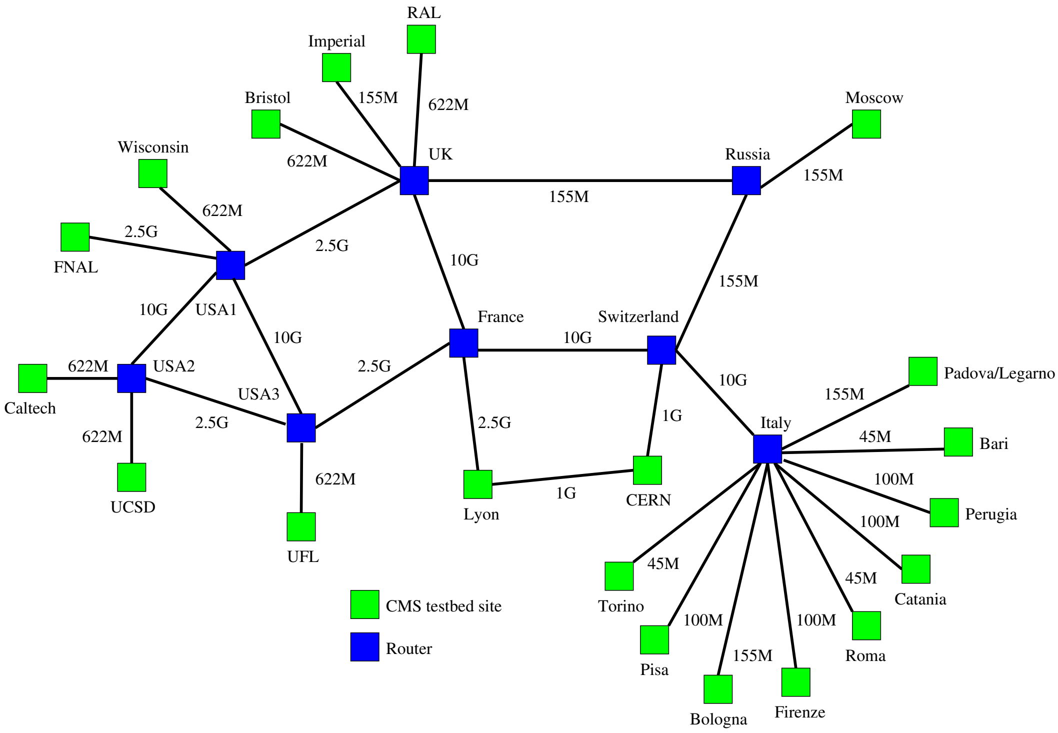
Figure 7.25: Total profit/loss for AMs and SBs at each EDG Testbed 1 site using (a) eco-bin and (b) eco-zipf.

the slow network connection. This trade-off between the number of jobs increasing and the price decreasing with site connectivity contributes to the lack of a general trend in the profits of each site.

Apart from CERN, which holds all the master files, the only site to make a profit with eco-bin or eco-zipf is RAL. It has the fastest network connection to the Grid, at 2.5 Gbit/s, but Manchester has the same connection speed yet makes a loss overall. The difference between these two sites is due to the SE sizes, with RAL having a capacity of 50 GB and Manchester 20 GB. The larger SE size means RAL has more opportunity to sell files to other sites, including its close neighbour Manchester which has little room to store the files it requires for its jobs. The combination of fast network connectivity and large SE size is therefore advantageous for making profits at the expense of other sites with lesser resources.

7.5 CMS Testbed 2002

The Grid we simulate in this section is based on a testbed used during a large scale production effort for CMS in 2002 [85] and we use the same set of jobs that was used



No. of sites	Total dataset size	D	C
20	97 GB	0.56	507 Mbit/s

Figure 7.26: The testbed used for CMS production in 2002.

in the previous section. The Grid topology, shown in Figure 7.26, comprises 20 sites in Europe and the USA. CERN and FNAL (where the data is originally produced) have a relatively small storage capacity of 100 GB each and a master copy of each file is stored at one of these sites. Every other site has a Computing Element and initially empty storage of capacity 50 GB. The storage values used are representative, being scaled down from the actual resources at these sites. We again include contending network traffic which is based on network monitoring data between some of the sites in Figure 7.26.

7.5.1 Scheduling Policy vs. Replica Optimisation Strategy

The same tests, comparing each combination of scheduling and replica optimisation algorithm as were done for the EDG testbed were repeated for the larger CMS testbed.

As before, two different groups of users were simulated: users submitting jobs at random times and users following the pattern of CMS Data Challenge users. The total number of jobs submitted was again 5000 but for the latter group the maximum number of jobs submitted per hour was increased to 300 due to the larger number of resources available.

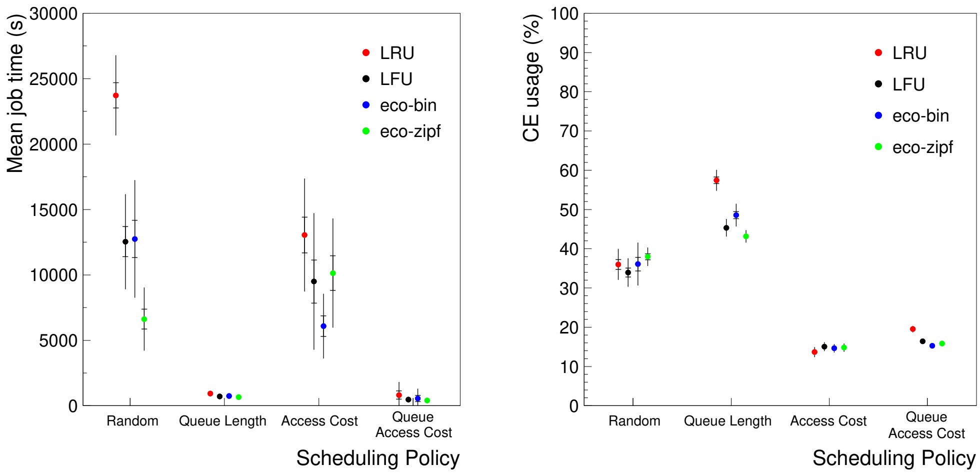


Figure 7.27: Mean job time and CE usage for CMS testbed 2002 with varying scheduling algorithms and random job submission rate.

Figures 7.27 and 7.28 show results similar to those seen for the EDG testbed (Figures 7.19 and 7.20). Queue access cost gives lower mean job times than any other scheduler, but does not use the available CEs as fully as some other policies. Queue length has the highest CE usage, again because it always schedules jobs to CEs that have no jobs in their queue if there are any available.

7.5.2 Job Workload

The mean job time and effective network usage for a varying number of jobs submitted to the Grid, using the queue access cost scheduling policy, are shown in Figure 7.29. The job time increases with the number of jobs submitted up to 1000 jobs, then decreases while the effective network usage decreases constantly with the number of jobs. LRU consistently has the slowest job times and highest effective network usage,

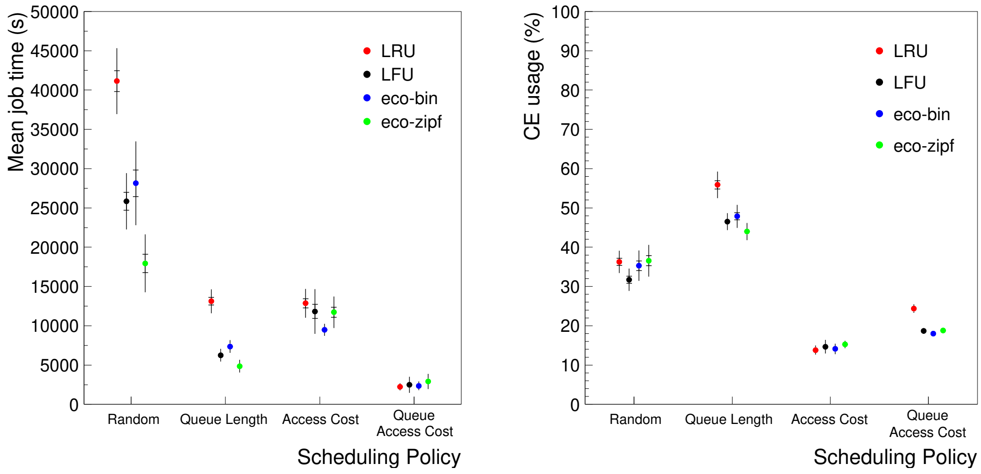


Figure 7.28: Mean job time and CE usage for CMS testbed 2002 with varying scheduling algorithms and CMS DC04 job submission rate.

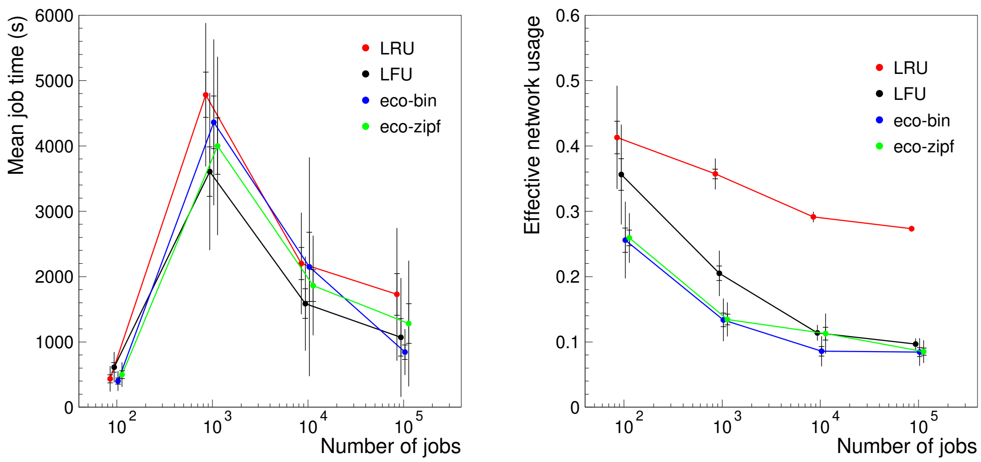


Figure 7.29: Mean job time and effective network usage for varying numbers of jobs submitted to CMS testbed 2002 by CMS DC04 users.

showing that it is poor at making replication decisions, although it does improve with a high number of jobs. The economic model optimisers use the lowest amount of network resources for all numbers of jobs because they make better decisions on which replicas each site should store. However, using the economic model to lower job times is only more effective than LFU when a large number of jobs are executed by the Grid.

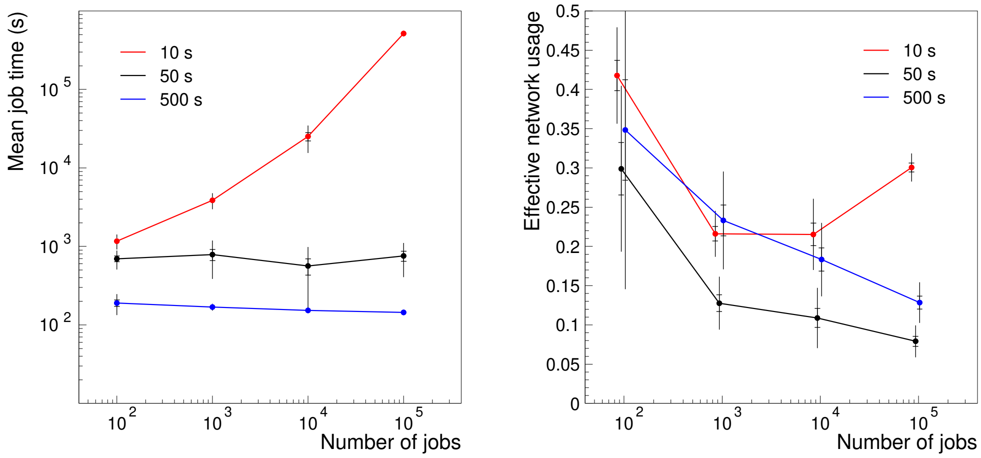


Figure 7.30: Mean job time and effective network usage for varying job submission rates to the CMS testbed 2002 using the binomial-based economic model.

The performance of any optimisation strategy is heavily dependent on the workload of the Grid, which is controlled by the rate at which jobs are submitted to the Grid. If the jobs are submitted at a rate faster than they can be processed by the Grid, we would expect the job times to increase steadily with the number of jobs, since queues will continually increase at each Grid site. On the other hand, if the job submission rate is very slow, after an initial period of heavy replication we would expect the job time to become constant. Figure 7.30 shows the performance of eco-bin with different numbers of jobs and intervals between job submission of 10 s, 50 s, and 500 s. With a fast job submission rate the mean job time increases almost linearly with the number of jobs, as the queues grow very large. The time remains roughly constant or decreases slightly with the lower rates, as we expect. The effective network usage decreases gradually for the slower job submission rates, but after a fall up to 1000 jobs, for the fast submission rate it rises with larger numbers of jobs. This is because although jobs are scheduled to the data, when they eventually come to run the data is not available locally anymore so overall more replication takes place.

Any Grid must therefore ensure that it has sufficient resources to deal with the expected job submission rate, otherwise job times will spiral upward and make the Grid unusable. If the resources can cope, a good choice of replication strategy means that after an initial period, job times and network usage will stabilise at reasonable levels.

7.5.3 Storage Element Sizes

Varying the SE sizes changes the value of D and for the EDG testbed lowering D had a very large effect on both the job times and effective network usage. Figure 7.31 shows the results of a similar test performed on the CMS testbed where the SE sizes varied from 3 GB ($D = 0.03$) to 100 GB ($D = 1.12$). As before, the linear decrease on the log-log scaled plot shows the job time varies with D following a power law. The rate of increase with lower D is very similar to the rate seen for the EDG testbed for all the strategies, showing that for these sizes of Grids (10 - 20 sites), the dependence of job time on D is independent of the number of sites. Eco-zipf is again the best performer at very low D, but as D increases, first LFU then eco-bin out-perform the others and so again we cannot say that varying the SE sizes affects any strategy more than any other apart from extremely low values of D. When D is very large, so that all the sites can store all the files for all the jobs, we see the same large increase in job time as before, caused by the queue access cost scheduling policy. The effective network usage follows a similar pattern to that which was seen for the EDG testbed, decreasing gradually down to almost zero. The relative performance of the optimisation strategies follows closely the relative differences in job time, showing two metrics are very closely linked.

7.5.4 Effects of Background Network Traffic

All simulation results shown so far for the EDG testbed and CMS testbed have included non-Grid network traffic. Here, some results are compared with simulations run with the same parameters, but without the inclusion of background network traffic. We would expect the job times to decrease because data can be replicated

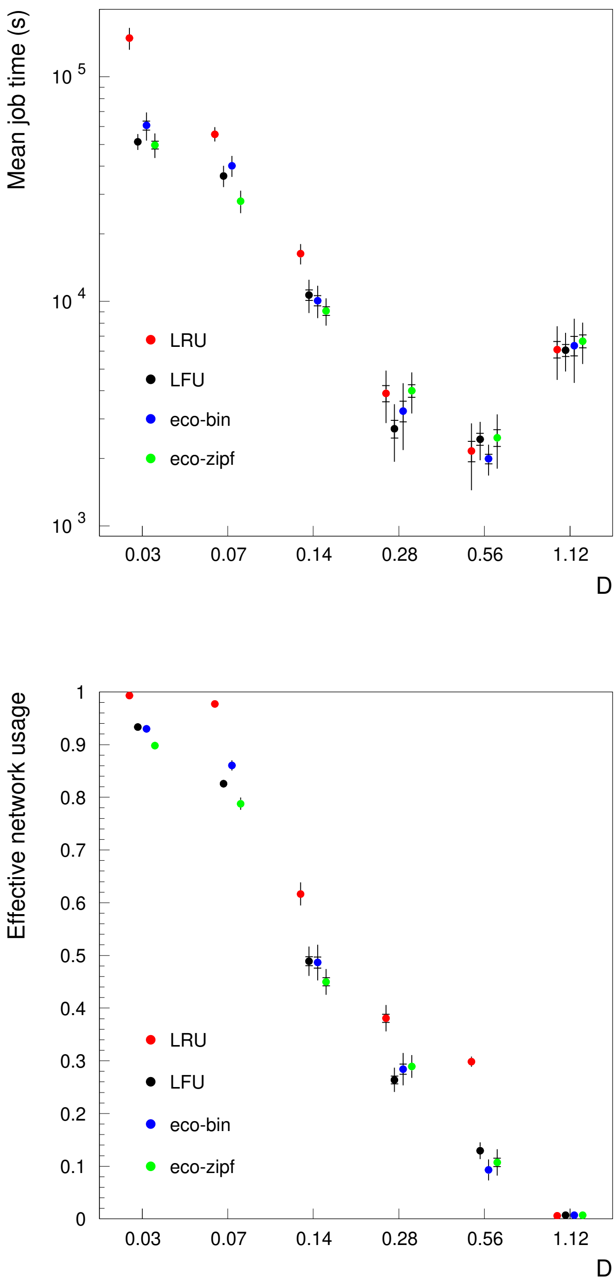


Figure 7.31: Mean job time and effective network usage for varying values of D for the CMS testbed 2002.

faster, but the effective network usage to stay constant, as this depends on the number of files copied across the network, which should remain the same.

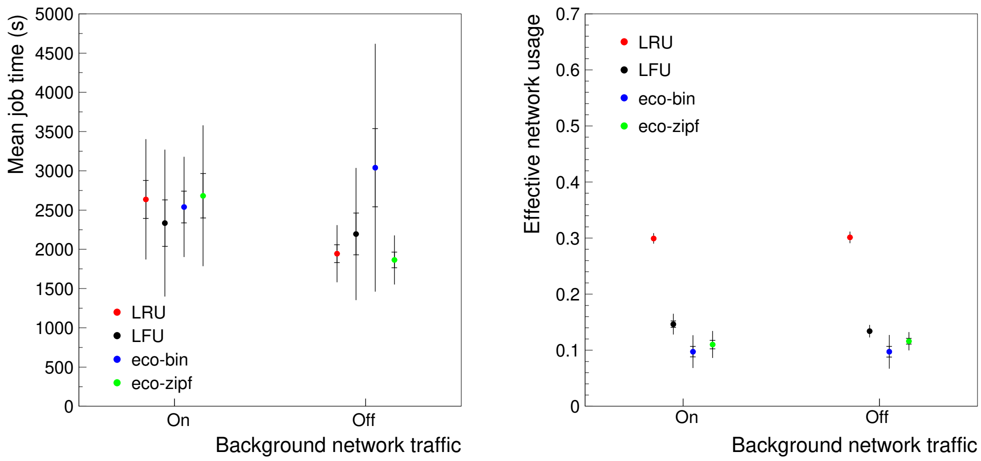


Figure 7.32: Mean job time and effective network usage with background traffic for the CMS testbed 2 on and off and $D = 0.56$.

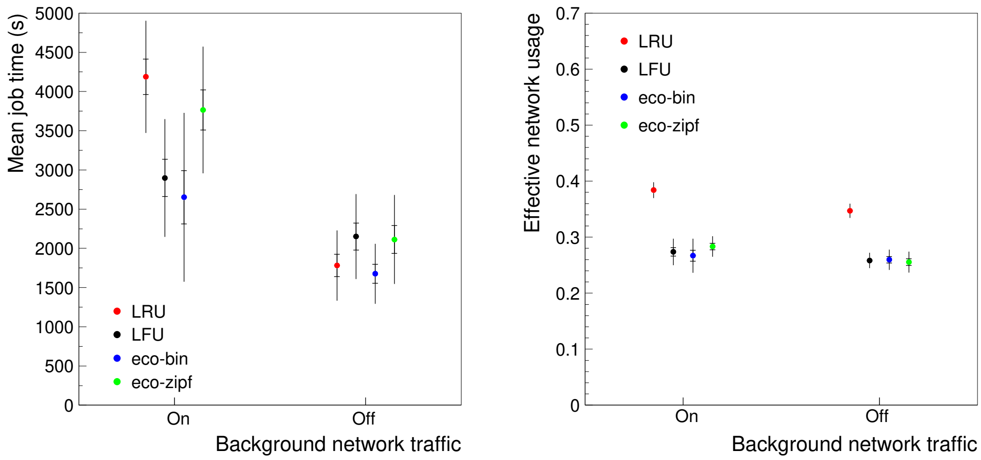


Figure 7.33: Mean job time and effective network usage with background traffic on and off and $D = 0.28$.

Figure 7.32 shows the mean job time and effective network usage for the four replica optimisation strategies when simulated background network traffic is on and off. As expected there is little difference in the effective network usage, but there is also little difference in the mean job time, although the standard deviation is smaller for all the optimisation strategies apart from eco-bin, as the background traffic is a large source of random variation in the simulation. The similarity in job times is due to the fact that for this configuration the value of D is relatively high (0.56), so not much replication takes place because the SE sizes are relatively large. This means the network bandwidth available has less effect on the time to run jobs.

The same comparison was done using a Grid where each site had its value of D reduced by a factor of a half (Figure 7.33). Here, the effective network usage has increased by up to a factor of three, due to the increased replication required by the reduced storage capacity. There is a clear decrease in the mean job time when background traffic is not included, and again the standard deviations are much smaller. Therefore we can conclude that for Grid configurations with low values of D, the presence of background network traffic has a large influence on the performance of the Grid, but does not affect any replication strategy more than any other.

7.5.5 Economic Model Analysis

The same analysis used for the EDG testbed was applied to the CMS Testbed and the results are shown in Figure 7.34, with the sites again ordered by the speed of their network connection.

The pattern of results is similar for both flavours of the eco model to the EDG configuration, but most of the sites with slower network connections almost exactly break even. This is because a low bandwidth connection makes it very unlikely that the AM will buy files from other sites or that the SB can sell files to other sites. The prices are simply too high and so the AM and SB on each site form an independent market, only trading between themselves. The sites with the higher bandwidth connections generally make a small loss apart from CERN and FNAL, which hold the master files. These sites do more trading among each other because it is fairly cheap to buy and sell files, although the net result is a loss for each site overall. The difference in prices for files at different sites is emphasised by the larger SB profits and AM losses for the lower bandwidth sites. Even though they run fewer

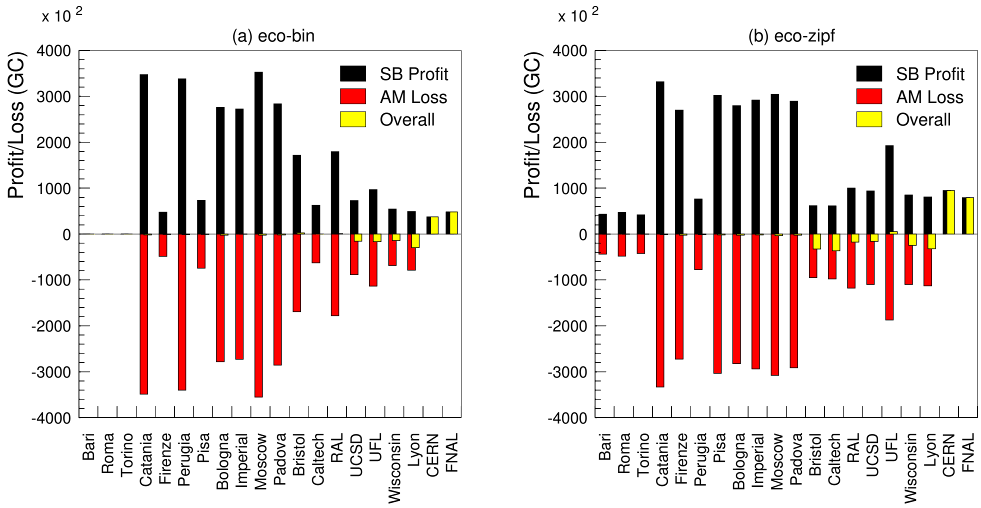


Figure 7.34: Total profit/loss for AMs and SBs at each CMS Testbed site using (a) eco-bin and (b) eco-zipf.

jobs, the file prices are so much higher for these sites that the difference can be up to 3-4 times that of the better connected sites.

Pricing information on the files bought and sold by each AM and SB in this situation are shown for eco-bin in Tables 7.4 and 7.5. Table 7.4 gives the numbers and average prices of files bought and sold by each SB along with the profit each makes, and Table 7.5 shows the number and average price of files bought by each AM and the net balance for each site (the SB profit minus the AM loss).

The first important point to note is that the numbers of files bought by SBs are far smaller than the numbers sold, which is because the probability that each job has of being submitted to the Grid does not change over the course of the simulation. The access patterns then remain fairly stable and so after some initial shuffling of files the SBs keep the same set for the duration, i.e. the initial conditions (the sites to which the first few jobs were sent) determine the sites to which all future jobs are sent and hence the files stored at each site. The queue access cost scheduling algorithm also ensures that jobs requiring similar data sets go to the same sites, therefore the files already on these sites are valued higher and are less likely to be deleted.

The number of files bought by AMs rises for sites with higher speed network

Site	Bandwidth (Mbit/s)	Files Bought	Ave. Cost (GC)	Files Sold	Ave. Gain (GC)	Profit (GC)
Bari	45	0	0	0	0	0
Roma	45	0	0	0	0	0
Torino	45	0	0	0	0	0
Catania	100	14	139	2834	123	347131
Firenze	100	8	146	390	124	474427
Perugia	100	12	141	2760	123	338333
Pisa	100	7	141	597	124	73505
Bologna	155	25	96.2	3460	80.4	276070
Imperial	155	14	94.9	3293	83.1	272603
Moscow	155	28	88.2	4385	81.0	352697
Padova	155	23	101	3530	81.0	283862
Bristol	622	33	19.5	8097	21.2	171691
Caltech	622	58	24.1	3120	20.5	62754
RAL	622	50	20.5	8693	20.7	179327
UCSD	622	57	18.1	4175	17.7	72866
UFL	622	56	19.3	5496	17.8	96917
Wisconsin	622	64	19.4	3195	17.5	54689
Lyon	2500	77	7.14	4679	10.5	48950
CERN	1000	0	0	2041	18.2	37305
FNAL	2500	0	0	2852	16.8	47923
All Sites	507	526	40.5	63597	43.8	2764050

Table 7.4: Storage Broker statistics for each site on the CMS Testbed.

Site	Bandwidth (Mbit/s)	Files Bought	Ave. Cost (GC)	Loss (GC)	Site Balance (GC)
Bari	45	0	0	0	0
Roma	45	0	0	0	0
Torino	45	0	0	0	0
Catania	100	2834	123	349078	-1947
Firenze	100	390	124	48597	-1170
Perugia	100	2760	123	340032	-1699
Pisa	100	597	124	74492	-987
Bologna	155	3459	80.4	278333	-2263
Imperial	155	3284	83.0	272888	-285
Moscow	155	4384	80.9	355002	-2304
Padova	155	3527	81.0	285779	-1917
Bristol	622	8059	20.9	169124	2567
Caltech	622	3150	19.8	62640	114
RAL	622	8696	20.5	178283	1044
UCSD	622	4938	17.9	88566	-15700
UFL	622	6330	17.9	113532	-16615
Wisconsin	622	3853	17.8	68816	-14126
Lyon	2500	6810	11.5	78890	-29941
CERN	1000	0	0	0	37305
FNAL	2500	0	0	0	47923
All Sites	507	63071	43.8	2764050	0

Table 7.5: Access Mediator statistics for each site on the CMS Testbed.

connections as we would expect because these sites have a higher throughput of jobs. The number of files sold by each SB is fairly similar to the corresponding AM, in fact it is exactly the same for sites with connection speeds lower than 155 Mbit/s, confirming that agents on these sites only trade within their own site. Sites with higher bandwidth trade much more freely with each other and so the correlation between AM and SB on the same site is not so strong. Average prices for files bought and sold fall rapidly with increasing network bandwidth, since prices are calculated on file transfer time which varies inversely with bandwidth. The SBs at sites holding the master files do not sell as many files as other similarly connected sites and much of this profit is gained at the start of the simulation when these two sites effectively have a monopoly on all the files since they hold the only copies. Once the files are replicated around the Grid these sites can still participate and win auctions but they do not have the same power to charge high prices and almost always a local SB will under cut them.

Taking all the SBs into consideration, the cost of buying a file is slightly less than the amount received for selling it, therefore any SB in general needs only to sell a file once to make a profit from it. Since all sites start from zero GC, the overall balance for every site combined is zero, with the SBs making around 2.7 million GC and the AMs spending the same amount. This corresponds to a cost of just over 550 GC to run a single job. In conclusion, this section and the analyses of other configurations has shown that an economy based on network costs can provide a fair, stable and efficient marketplace.

7.6 Larger Grids

Recent work by the other OptorSim developers has concentrated on testing optimisation strategies on larger Grid testbeds. It is expected that in the future Grids will grow up to the order of thousands of sites, and both the simulation and any strategies must be able cope with this scale of resources. In [28], the LCG testbed shown in Figure 1.6 was simulated, and it was found that the economic models only offered an advantage when the Grid was heavily loaded, as was the case in the EDG and CMS testbeds used here. This behaviour shows the optimisation strategies are stable and their effects are roughly consistent for different scales of Grids up to current sizes, but they have yet to be tested with the larger Grids that will be constructed in the future.

7.7 Summary

All the results presented here for different types of Grids have shown that the queue access cost scheduling policy significantly reduces the time taken by the Grid to run jobs, apart from some cases where SE sizes are very large and optimisation is perhaps not really necessary. LRU is usually the least effective replica optimisation strategy at lowering job run times and making best use of the Grid resources, because only considering the time a file was last accessed does not give an accurate picture of the file request pattern. The eco models consistently outperform LFU in very simple situations, however they are generally equal to LFU in more complex Grid environments, showing that the model perhaps does not scale as well as it should.

In general LFU is regarded as the simplest and best cache management strategy, and here we have shown that the eco models can equal its performance, although they have a much higher complexity. Therefore based on these results, any current Grid developments should use LFU, as it is very simple and easily understood, and the complex eco models do not offer any performance advantage. However, the eco-model is at an early stage of development, and part of our future work will be to assign limited budgets to the AM and SB agents and introduce charges for using other Grid resources such as computational power and storage space. This, and further refinement of the algorithms within the model, could push the performance past that of LFU far enough that it becomes advantageous to use in a real Grid deployment.

There is usually very little difference in the performance of the two flavours of eco model, even for different access patterns, suggesting that their good performance in some situations compared to LFU is due to the fact that they do not have to replicate every file, rather than the values placed on particular files. Analysis of the marketplace has shown that the economic model is generally stable for both flavours but the Zipf-based model may have a slight edge over the binomial-based model. The economic model we have developed so far only considers costs in terms of file transfer time and network usage, but should provide a good base to build upon for future investigations into this area.

Chapter 8

Conclusions

The concept of Grid computing has emerged as the solution to the needs of data-intensive scientific experiments such as the next generation of high energy physics experiments currently under development at CERN. Grid middleware enables distributed resources to be connected and aggregated together so they can seamlessly be exploited by scientists. The set of data management middleware services developed by the European DataGrid has been shown to scale well up to and beyond the expected limits of normal operation expected for high energy physics analysis in a Grid environment. The catalog services (RLS and RMC) can hold up to 10 million file mappings and handle up to 200 concurrent connections with relatively little degradation in performance, and the overhead of operations on these catalogs is very small compared to the transfer times of the average size of file ($\sim$ 1 GB). These services have been integrated into the LHC Computing Grid and have already successfully been used in a production context, during a CMS data challenge from March to May 2004, when the RLS was used to store information on over two million replicated files.

The effects of security increase the time to perform operations by a factor of 10 at least, but given the small overhead of the services compared to file transfer, this is not a major problem, and further development should reduce these effects further. The current main themes of development in the data management services are splitting the bulky Replica Manager client into several smaller lightweight services, using a virtual directory structure for logical file naming, and creating a combined catalog interface behind which lies a file catalog, a replica catalog and a (file) metadata catalog.

The Grid simulation package OptorSim, developed to evaluate replica optimisation strategies, has proven to be a very useful tool for simulating any generic Grid scenario. It allows any kind of Grid topology to be simulated and a multitude of parameters can be varied to compare many different situations. The modular code base allows easy implementation of any new optimisation strategy, and several external users have already adapted OptorSim to their own purposes. Here OptorSim has been used to validate our optimisation strategies on very simple Grids and test their performance on larger Grid configurations with up to 20 sites. It has recently been used to simulate around 100 Grid sites [28] and 10^5 Grid jobs, however further testing is required to find out if it can scale up to $\mathcal{O}(1000)$ sites and millions of jobs that are expected when the LHC starts operation in 2007. Future work will also allow different jobs to be simulated such as Monte-Carlo style physics data generation, where data is dynamically created throughout the simulation. Other dynamic effects such as the temporary unavailability of sites will be added along with more realistic simulation of components like the clusters of worker nodes which make up Computing Elements. Ultimately, the aim is to implement the strategies tested in OptorSim into the real Grid environment, and compare results on the performance of the strategies in both situations.

The optimisation of Grid resources such as network bandwidth, storage space, and computational power is important both for the consumer and the provider of each resource. It was shown that when these resources are limited, the Queue Access Cost scheduling policy, which takes into account both the location of data required by each job and the workload of each site, gave the best performance overall. As the scale of the Grid increases up to 20 sites, this policy reduces job times by up to a factor of 2 or 3 compared to other policies.

A comparison of replica optimisation strategies in a variety of situations has shown that Least Frequently Used (LFU) and two kinds of economic model strategies are between 20% and 100% better at reducing job times than the simple Least Recently Used (LRU) strategy. In a simple configuration with only two Grid sites the economic model was up to 20% better at reducing job times than LFU, but as the simulation testbed increased to 20 sites there was either no advantage using the economic models or only a very small advantage (around 5%). The variation of job times and effective network usage with the size of storage resources available at each site is approximately independent of the number of sites and the optimisation strategy (for non-trivial Grids), although the Zipf-based economic strategy can perform up to 25% better than the others when space is heavily constrained.

The 5% reduction in job time seen for the economic model in the larger testbed configurations probably does not outweigh the complexity involved in implementing this strategy, therefore LFU would be the best choice for any currently operational Grid. However, the current implementation of the economic model only takes into account network resources as a pricing mechanism, and future work will extend this to include charges for using other resources such as computational power and storage space, which should lead to an improved performance.

Appendix A

ScotGrid Website

This chapter gives an overview of work carried out on the ScotGrid website over the period January 2002 to June 2004, representing Phase 1 of the project. This work included the design and layout of the site, and the webpage-based monitoring of the resources used by each group of users.

A.1 The ScotGrid Project

ScotGrid is a prototype three-site Tier-2 centre for GridPP [66] for the analysis of data primarily from ATLAS and LHCb as well as from other experiments. Initially, cluster computational resources were largely based at Glasgow and data storage resources were largely based at Edinburgh. However in early 2004 a CPU farm in Durham University Institute for Particle Physics Phenomenology also became part of ScotGrid and the current three-site layout is illustrated in Figure A.1.

Phase 1 of the project consisted of a 128 CPU farm in Glasgow and a 5 TB data store in Edinburgh. As well as the high energy physics applications, ScotGrid was used by groups involved in Grid data management, bioinformatics, information retrieval and device modelling.

The main particle physics activities were Monte Carlo generation and data challenges in preparation for the LHC starting up in 2007. Monte Carlo generation involves simulating many events and the consequent output of the detectors in order to understand the detector performance and select trigger strategies for interesting

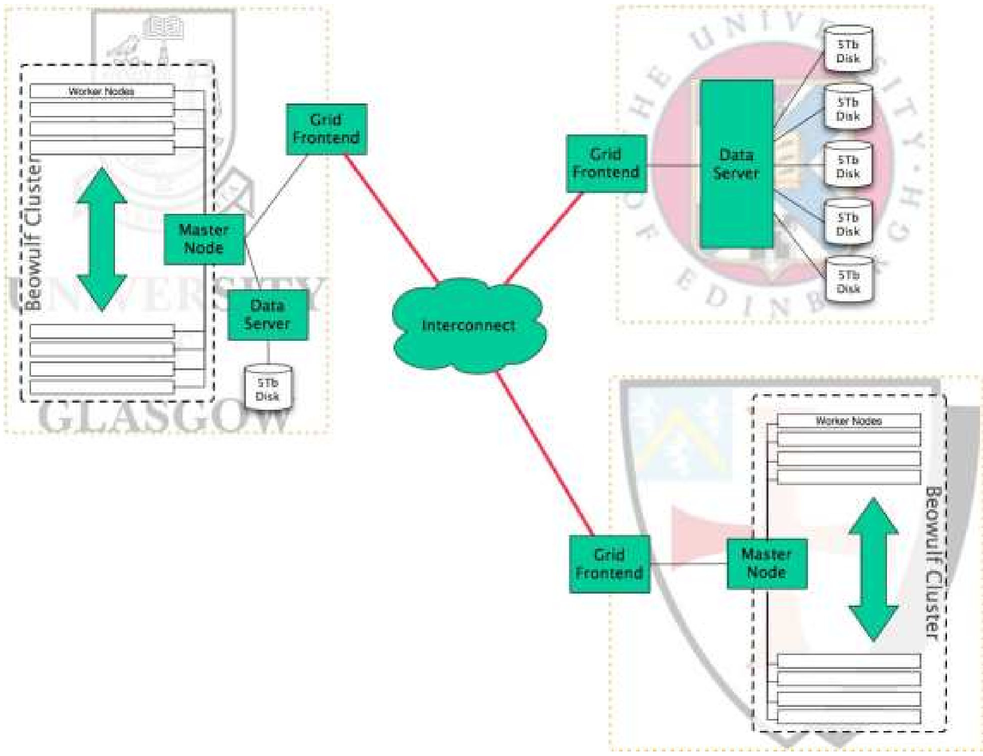


Figure A.1: The layout of ScotGrid, showing the resources at Glasgow, Edinburgh and Durham.

events when the real detectors come on-line. Data challenges occur periodically and are essentially scaled-down practise runs of the running experiment designed to test the entire distributed computing framework. Each data challenge uses more data than the previous one, aiming eventually to reach the real levels produced by the experiments. Members of the ATLAS and LHCb experiments have used ScotGrid heavily for these activities.

The Grid Data Management group use ScotGrid for high-volume runs of the OptorSim simulation tool, to investigate replica optimisation in a Grid environment. Most of the results from OptorSim presented in this thesis were obtained using ScotGrid. The UKQCD collaboration uses ScotGrid for lattice field theory calculations through simulation of Quantum Chromodynamics (QCD). This aims to increase our understanding of the strong force and hence the predictive power of the Standard Model. Researchers at the Biomedical Research Centre in Glasgow have been using ScotGrid to assist with their research into developing new statistical methods to analyse complex genetic data currently being produced by life science experiments.

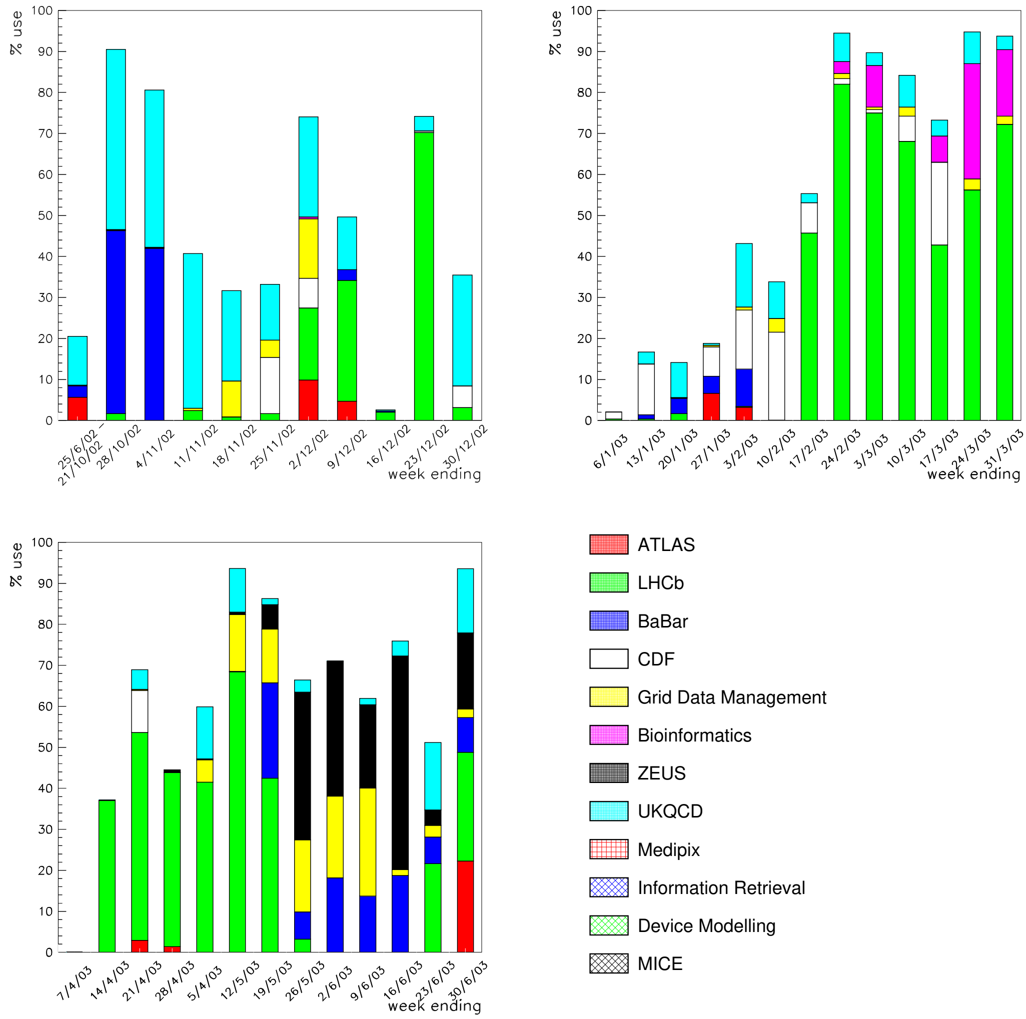


Figure A.2: ScotGrid use by group from June 2002 to June 2003.

Information Retrieval looks at ways of effectively gaining information from masses of data, including from the World Wide Web using techniques such as link analysis. The Medipix group is developing a single photon counting chip and use ScotGrid for detailed simulation studies. Simulations of semi-conductors are also run on ScotGrid by the device modelling group which use Monte Carlo techniques to model components such as transistors. The MICE experiment at Rutherford Appleton Laboratory aims to test a possible cooling mechanism for a muon beam used in a neutrino factory, and ScotGrid has provided computing power for beamline simulation studies.

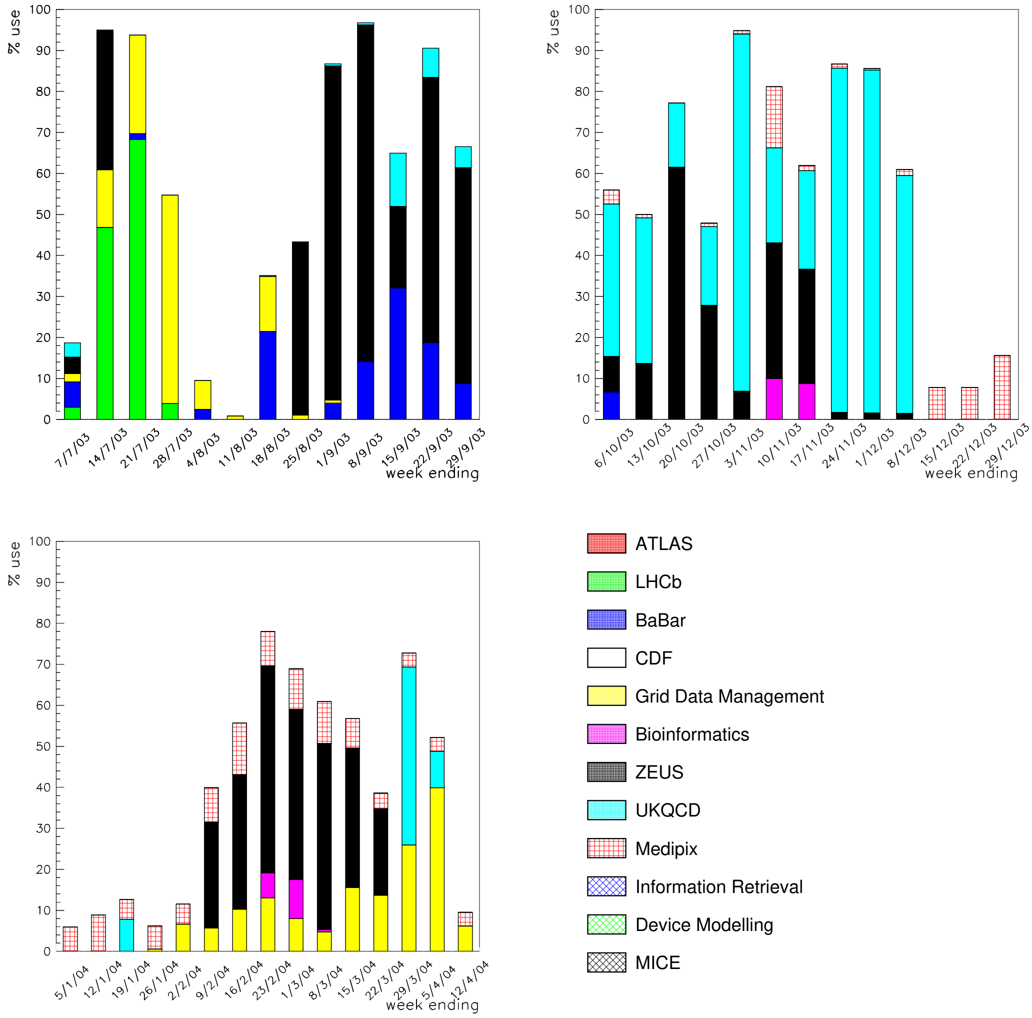


Figure A.3: ScotGrid use by group from July 2003 to April 2004.

Graphs showing the use by each group of users are shown in Figures A.2 and A.3. By the end of Phase 1 in April 2004 the CPU farm had processed over 100,000 jobs which had consumed just under 900,000 CPU hours, giving an average use of around 50% of the resource capacity. Phase 2 started in May 2004 when the centre was upgraded to 256 CPUs at the Glasgow site and 24 TB of storage in Edinburgh. Also at this time an 80 CPU farm based at Durham was connected and large-scale integration within the larger UK-wide Grid infrastructure began (up to then the computational facilities were only available to locally submitted jobs).

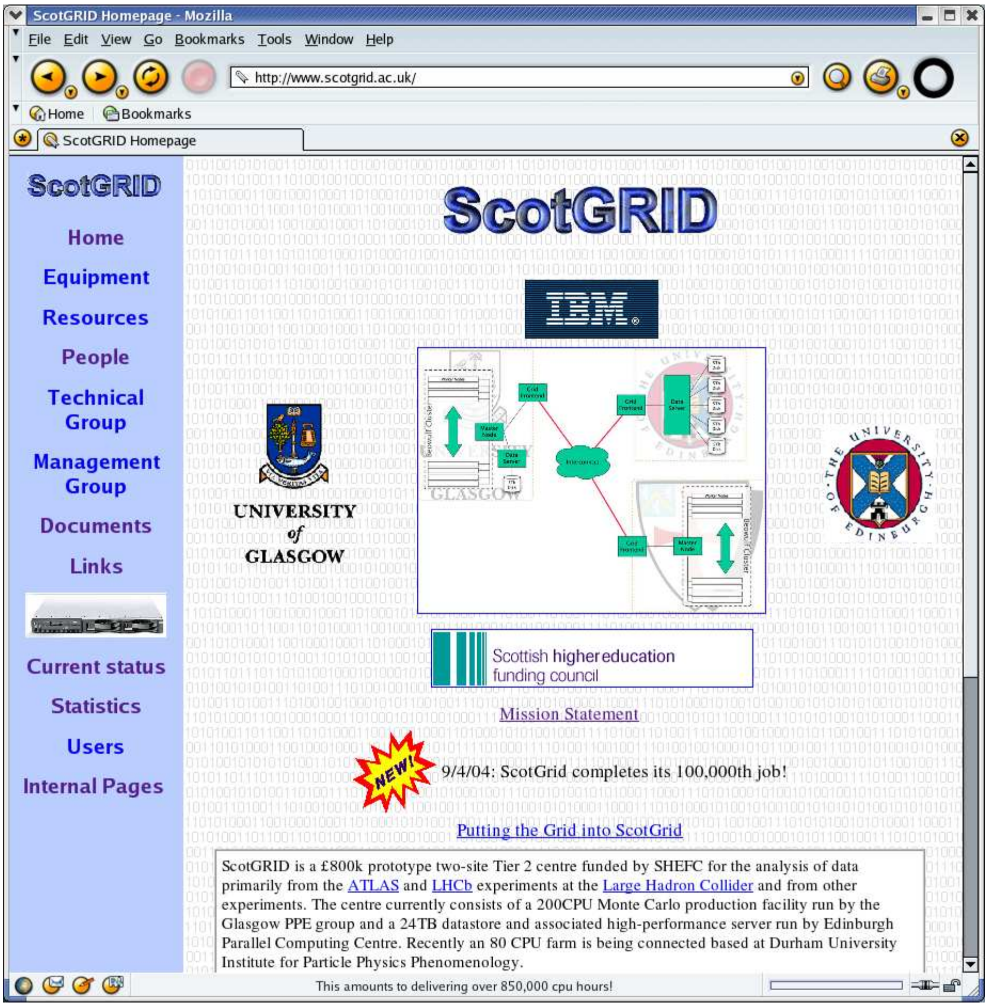


Figure A.4: ScotGrid home page.

A.2 Website Monitoring

The website for ScotGrid (`http://www.scotgrid.ac.uk`) gives information on the hardware resources, members of the project and various sub-groups within the project, papers and presentations given concerning ScotGrid and links to other related web sites. Via the website it is also possible to monitor the current status of jobs running on the Glasgow CPUs and view statistics of use by the different groups over the lifetime of the project. A screenshot of the home page is shown in Figure A.4.

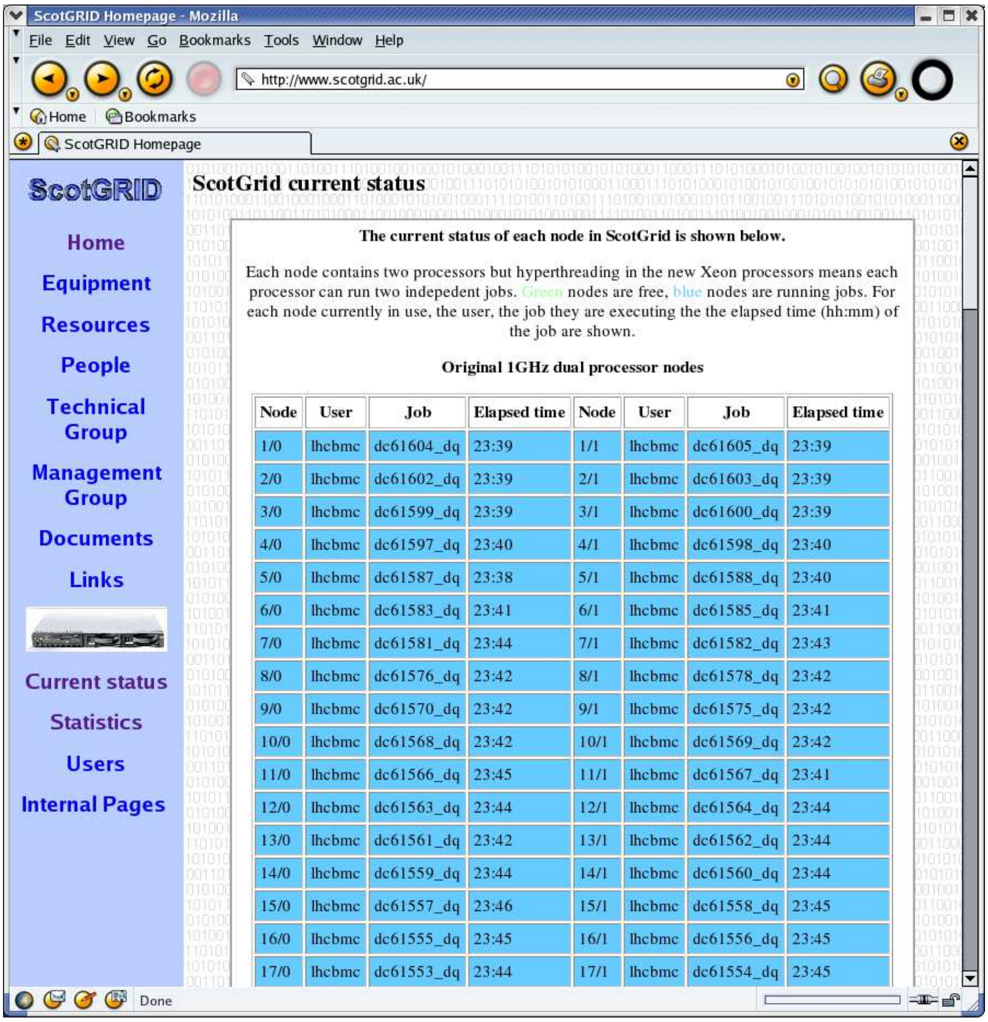


Figure A.5: ScotGrid current status web page.

The web page shown in Figure A.5 gives information on the jobs currently executing on the Glasgow CPUs, updated every ten minutes. This table is produced from the output of the Portable Batch System (PBS) command `qstat -n` which gives the status of every job submitted to the scheduler that is either running or waiting to run. This information is processed by a Perl script to output the HTML page showing for each node the owner of the job, the name of the job and the length of time for which it has been running.

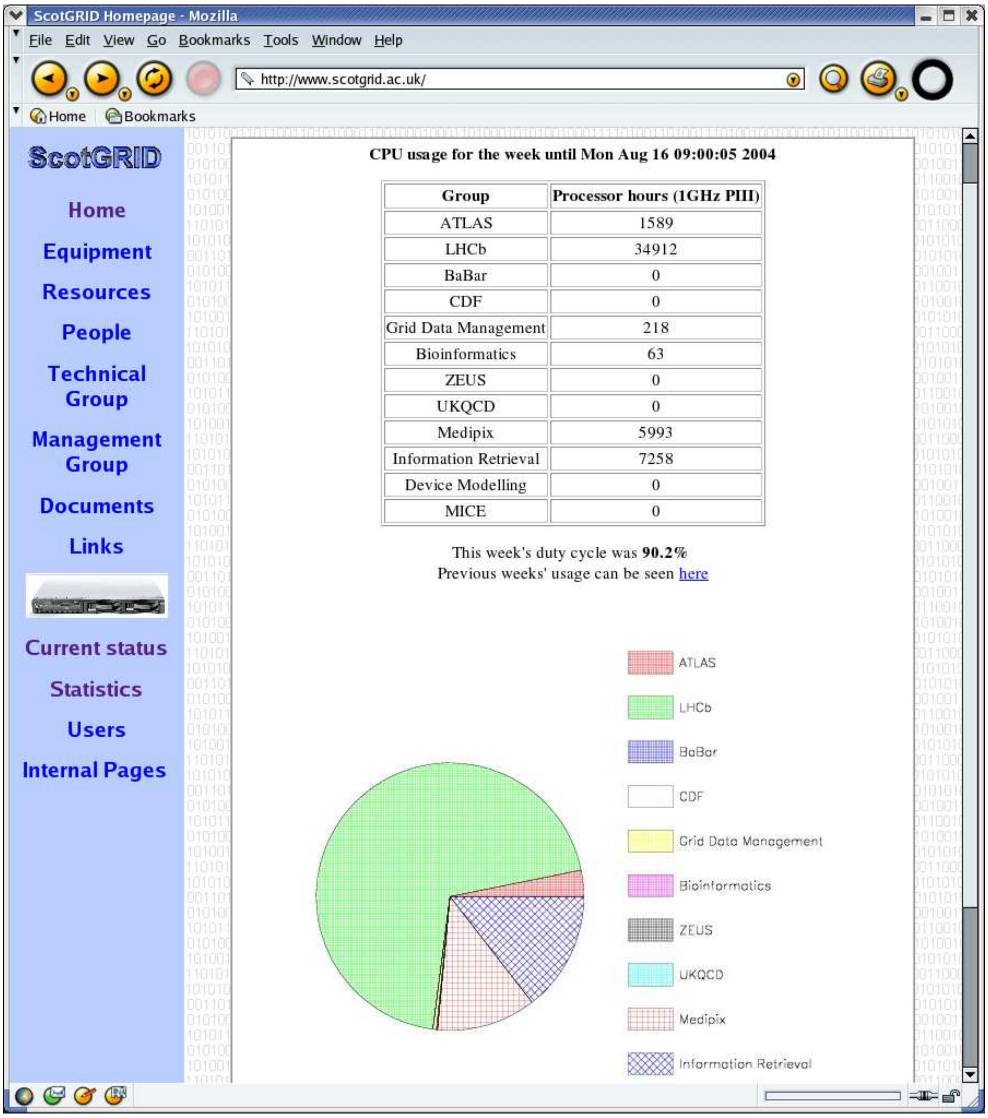


Figure A.6: Weekly use of ScotGrid by group for the week up to 16th August 2004.

The use statistics were summarised at the end of each week in the form of a table and pie chart, as shown in the web page in Figure A.6. This information was obtained from the scheduler command `showstats -u`, which shows the total number of processing hours used by each user since ScotGrid started operation. A record is kept of these figures each week and the difference between this week's and last week's figures is used to find the current week's use for each user. The users are then mapped to a group and another Perl script is used to create the HTML page from this information. The pie chart is created using PAW (Physics Analysis Workstation) [107].

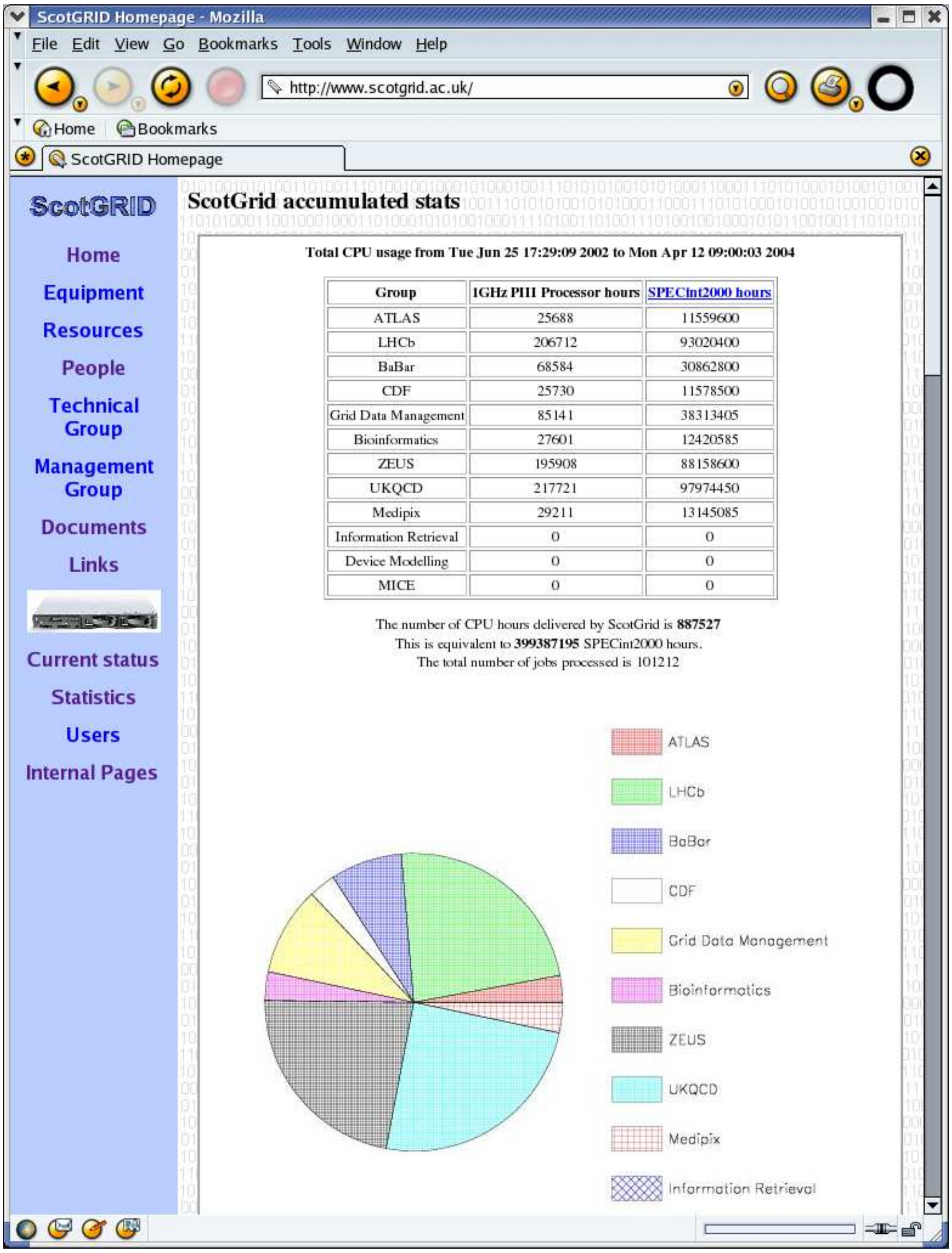


Figure A.7: Total use by group for Phase 1 of ScotGrid.

The same command is used to calculate the total use for each group since ScotGrid started operation, and a similar procedure creates the web page shown in Figure A.7.

A.3 Summary

ScotGrid has provided valuable computational power not just for particle physics, but a wide range of scientific applications. The website for Phase 1 of ScotGrid contained pages to display statistics on the use of the CPU farm by the different groups. These pages were automatically generated using accounting information from the local batch scheduler and gave information on the current jobs running, the use by each group per week and the total use per group since ScotGrid began operation. The applications had used just under 1 million hours of CPU time in the first two years of the project. The centre has increased recently increased in size and it can provide even more processing power for these scientific applications, however, the main focus is now to integrate a significant fraction of these resources into the Grid.

Appendix B

Glossary

Many concepts and components involved in the field of Grid computing have their names abbreviated to short acronyms. An alphabetical listing of all the acronyms used in this thesis is provided in this appendix.

ALICE	A Large Ion Collider Experiment
AM	Access Mediator
AOD	Analysis Object Data
API	Application Program Interface
ATF	Architecture Task Force
ATLAS	A Toroidal LHC ApparatuS
CA	Certificate Authority
CAS	Community Authorization Service
CDF	Collider Detector at Fermilab
CE	Computing Element
CERN	European Organization for Nuclear Research
CLI	Command Line Interface
CMS	Compact Muon Solenoid
CPU	Central Processing Unit
DAES	Data Access Estimation Service
DMTF	Distributed Management Task Force

DNS	Domain Name System
EDG	European DataGrid
EGEE	Enabling Grids for E-science in Europe
ESD	Event Summary Data
ESG	Earth Science Grid
FBR	Frequency Based Replacement
FNAL	Fermi National Accelerator Laboratory
GB	Giga Bytes (10^9 Bytes)
GC	Grid Credits
GDMP	Grid Data Mirroring Package
GGF	Global Grid Forum
GMA	Grid Monitoring Architecture
GRACE	GRid Architecture for Computational Economy
GRAM	Globus Resource Allocation Manager
GridPP	Grid for Particle Physics
GriPhyN	Grid Physics Network
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
GUID	Globally Unique IDentifier
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPG	Information Power Grid
IRR	Inter-Reference Recency
JDL	Job Description Language
JVM	Java Virtual Machine

LAN	Local Area Network
LCAS	Local Centre Authorization Service
LCG	LHC Computing Grid
LCMAPS	Local Credential Mapping Service
LFN	Logical File Name
LFU	Least Frequently Used
LHC	Large Hadron Collider
LIGO	Laser Interferometer Gravitational Wave Observatory
LIRS	Low Inter-reference Recency Set
LRC	Local Replica Catalog
LRFU	Least Recently Frequently Used
LRU	Least Recently Used
MB	Mega Bytes (10^6 Bytes)
MDS	Metacomputing Directory Service
MONARC	Models of Networked Analysis at Regional Centres for LHC Experiments
MP3	MPEG audio Layer-3
NASA	National Aeronautics and Space Administration
NEESgrid	Network for Earthquake Engineering Simulation Grid
OASIS	Organization for the Advancement of Structured Information Standards
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
P2P	Peer to Peer
P2PM	Peer to Peer Mediator
PAW	Physics Analysis Workstation
PB	Peta Bytes (10^{15} Bytes)
PBS	Portable Batch System
PPDG	Particle Physics Data Grid

QAG	Quality Assurance Group
RAL	Rutherford Appleton Laboratory
RB	Resource Broker
RDBMS	Relational DataBase Management System
RFD	Ready For Download
RFT	Reliable File Transfer
R-GMA	Relational Grid Monitoring Architecture
RLI	Replica Location Index
RLS	Replica Location Service
RM	Replica Manager
RMC	Replica Metadata Catalog
RMS	Resource Management System
RO	Replica Optimiser
ROS	Replica Optimization Service
RPC	Remote Procedure Call
RTT	Round Trip Time
SAM	Sequential Access to Metadata
SB	Storage Broker
SE	Storage Element
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SRM	Storage Resource Manager
SSL	Secure Socket Layer
SURL	Storage URL
TB	Tera Bytes (10^{12} Bytes)
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
UML	Unified Modelling Language

URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
VO	Virtual Organisation
VOMS	Virtual Organisation Membership Service
WMS	Workload Management Service
WP	Work Package
WSDL	Web Services Description Language
XIO	eXtensible Input/Output
XML	eXtensible Markup Language

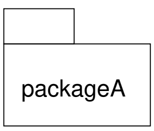
Appendix C

UML Notations

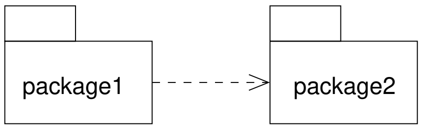
The Unified Modelling Language (UML) [126] provides a standard way of describing the structure and operation of object-oriented systems. In this thesis three types of UML diagrams are used to explain the OptorSim simulation and some of the notation used in these diagrams is presented here.

C.1 Package Diagrams

In Java^{TM}, classes with closely related functionality can be grouped into packages. This means that any functionality that is shared by members of the package need not be exposed outside the package and allows easier code management. Package diagrams show the dependencies between each package, i.e. which packages are used by other packages. In general it is a good idea not to have cyclic relationships between several packages.



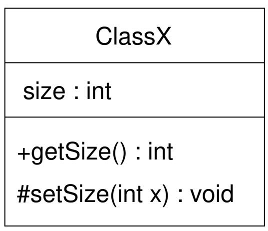
The folder defines the name of the package, and optionally the classes contained within it.



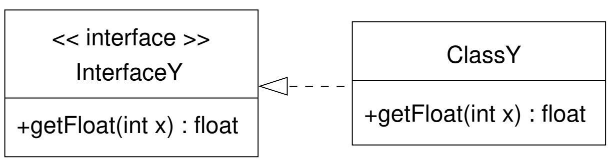
The dotted arrow describes a dependency, i.e. package1 uses functionality provided by package2.

C.2 Class Diagrams

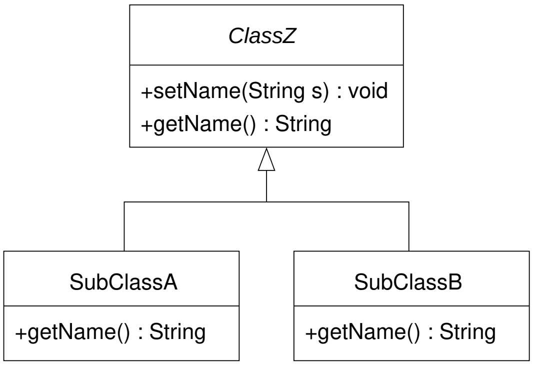
A class defines a set of attributes and methods. Objects in an object-oriented system are created as instances of a class. Attributes are variables associated with each class and methods are functions of the class that can be called by other classes. Interfaces cannot be instantiated but provide template methods that classes implement with their own functionality and they are used to hide or encapsulate different implementations behind a common set of methods. Subclasses inherit all the methods of their superclass but can also provide their own implementation of any of these methods. Class diagrams show the attributes and methods defined for each class and the relations between classes, subclasses and interfaces.



The name of this class is ClassX and it has one attribute, size, of type int and two methods: getSize() which takes no parameters and returns an int and setSize() which takes an int as a parameter and does not returns anything. Methods marked + are public, so any other class can call them, and methods marked # are protected, so they can only be called from classes in the same package or subclasses.



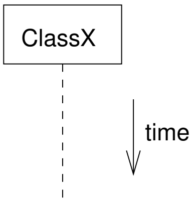
ClassY implements InterfaceY, providing its own version of the getFloat() method.



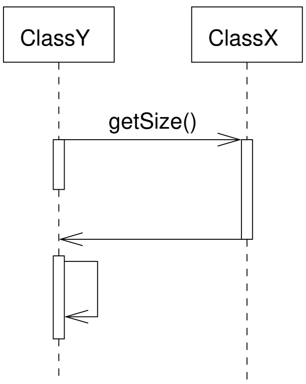
The abstract (denoted by italic name) ClassZ cannot be instantiated but the two subclasses can, and they each provide their own implementation of getName(). The subclasses both use setName() defined by ClassZ.

C.3 Sequence Diagrams

Sequence diagrams aim to show the sequence of events over time as a set of messages or method calls between classes.



A class or object is represented by a box with a dotted line going downward showing the interactions of the class over time.



ClassY calls the getSize() method of ClassX, which returns the size. ClassY then performs some internal processing.

References

[1] S. Acharya and S. B. Zdonik. An Efficient Scheme for Dynamic Data Replication. Technical Report CS-93-43, Brown University, 1993.

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

[3] A. H. Alhusaini, V. K. Prasanna, and C.S. Raghavendra. A Unified Resource Scheduling Framework for Heterogeneous Computing Environments. In *8th Heterogeneous Computing Workshop*, San Juan, Puerto Rico, April 1999.

[4] ALICE - A Large Ion Collider Experiment.
`http://cern.ch/Alice/`.

[5] D. Anderson. *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter SETI@Home. O'Reilly, 2001.

[6] Apple iTunes.
`http://www.apple.com/itunes/`.

[7] ATLAS - A Toroidal LHC ApparatuS.
`http://atlasexperiment.org/`.

[8] M. Baker, R. Buyya, and D. Laforenza. The Grid: International Efforts in Global Computing. In *International Conference on Advances in Infrastructure for Electronic Business, Science,and Education on the Internet (SSGRR 2000)*, Rome, Italy, August 2000.

[9] A. Baranovski et al. SAM Managed Cache and Processing for Clusters in a Worldwide Grid-Enabled System. Technical Report FERMILAB-TN-2175, FNAL, May 2002.

[10] A. Baranovski, G. Garzoglio, L. Lueking, D. Skow, I. Terekhov, and R. Walker. SAM-GRID: A System Utilizing Grid Middleware and SAM to Enable Full Function Grid Computing. In *Beauty 2002*, Santiago de Compostela, Spain, June 2002.

[11] S. J. Baylor, M. Devarakonda, S. Fink, E. Gluzberg, M. Kalantar, P. Muttineni, E. Barsness, R. Arora, R. Dimpsey, and S. J. Munroe. Java Server Benchmarks. *IBM Systems Journal - Java Performance*, 39(1):57–81, 2000.

[12] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Design of a Replica Optimisation Framework. Technical Report DataGrid-02-TED-021215, CERN, Geneva, Switzerland, 2002.

[13] W. H. Bell, D. G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *Int. Journal of High Performance Computing Applications*, 17(4), 2003.

[14] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, P. Millar, K. Stockinger, and F. Zini. Evaluation of an Economy-Based File Replication Strategy for a Data Grid. In *Int. Workshop on Agent Based Cluster and Grid Computing at CCGrid2003*, Tokyo, Japan, May 2003.

[15] T. Berners-Lee. Information Management: A Proposal. CERN, May 1990.

[16] S. Bethke, M. Calvetti, H.F. Hoffmann, D. Jacobs, M. Kasemann, and D. Linglin. Report of the Steering Group of the LHC Computing Review. Technical Report CERN/LHCC/2001-004, CERN, 2001.

[17] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of ACM*, 13(7):422–426, 1970.

[18] R. F. Boisvert, J. Moriera, M. Phillipsen, and R. Poz. Java and Numerical Computing. *Computing in Science & Engineering*, 3(2):18–24, Mar-Apr 2001.

[19] D. Bosio, J. Casey, A. Frohner, L. Guy, P. Kunszt, E. Laure, S. Lemaitre, L. Lucio, H. Stockinger, K. Stockinger, W. Bell, D. Cameron, G. McCance, P. Millar, J. Hahkala, N. Karlsson, V. Nenonen, M. Silander, O. Mulmo, G. Volpato, G. Andronico, F. DiCarlo, L. Salconi, A. Domenici, R. Carvajal-Schiaffino, and F. Zini. Next-Generation EU DataGrid Data Management Services. In *Computing in High Energy Physics (CHEP 2003)*, La Jolla, California, USA, March 2003.

[20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *IEEE INFOCOM'99*, New York, NY, USA, March 1999.

[21] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman. Benchmarking Java Against C and Fortran for Scientific Applications. In *2001 Joint ACM-ISCOPE Conference on Java Grande*, pages 97–105, June 2001.

[22] V. Bush. As We May Think. *The Atlantic*, 1945.

[23] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing.* PhD thesis, Monash University, Melbourne, Australia, April 2002.

[24] R. Buyya, D. Abramson, and J. Giddy. Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, 2000.

[25] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1507–1542, 2002.

[26] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Journal of Concurrency and Computation: Practice and Experience*, 14(13-15), 2002.

[27] D. G. Cameron, R. Carvajal-Schiaffino, P. Millar, C. Nicholson, K. Stockinger, and F. Zini. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, USA, November 2003.

[28] D. G. Cameron, R. Carvajal-Schiaffino, P. Millar, C. Nicholson, K. Stockinger, and F. Zini. OptorSim: A Simulation Tool For Scheduling And Replica Optimisation In Data Grids. In *Computing in High Energy Physics (CHEP 2004)*, Interlaken, Switzerland, September 2004.

[29] L. Capozza, K. Stockinger, and F. Zini. Preliminary Evaluation of Revenue Prediction Functions for Economically-Effective File Replication. Technical Report DataGrid-02-TED-020724, CERN, 2002.

[30] M. Carman, F. Zini, L. Serafini, and K. Stockinger. Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid. In *Int. Workshop on Agent based Cluster and Grid Computing at CCGrid 2002*, Berlin, Germany, May 2002.

[31] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputing Applications and High Performance Computing*, 11(3):212–223, 1997.

[32] C. Catlett. Standards for Grid Computing: Global Grid Forum. *Journal of Grid Computing*, 1(1):3–7, 2003.

[33] V. Cerf and R. Kahn. A Protocol for Packet Netwoek Intercommunication. *IEEE Transactions on Communications*, 22(5):637 – 648, May 1974.

[34] CERN - The European Organization for Nuclear Research. `http://cern.ch/`.

[35] 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.

[36] A. 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 and Distributed Computing (HPDC-13)*, Honolulu, Hawaii, USA, June 2004.

[37] M. Chetty and R. Buyya. Weaving Computational Grids: How Analogous Are They with Electrical Grids? *Computing in Science and Engg.*, 4(4):61–71, 2002.

[38] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[39] climate*prediction*.net.
`http://www.climateprediction.net/`.

[40] CMS - The Compact Muon Solenoid.
`http://cmsinfo.cern.ch/`.

[41] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework. White Paper, March 2004.

[42] H. Dail, F. Berman, and H. Casanova. A Decoupled Scheduling Approach for Grid Application Development Environments. *Journal of Parallel Distributed Computing*, 63(5):505–524, 2003.

[43] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[44] Distributed Management Task Force.
`http://www.dmtf.org/`.

[45] Distributed Particle Accelerator Design Project.
`http://stephenbrooks.org/muon1/`.

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

[47] Earth System Grid.
`http://www.earthsystemgrid.org/`.

[48] Architecture Task Force (ed: L. Momtahan). Software architecture models. Technical report, European DataGrid, February 2004. ATF Final Architecture Report.

[49] Enabling Grids for E-science in Europe.
`http://www.eu-egee.org/`.

[50] Enterprise Grid Alliance.
`http://www.gridalliance.org/`.

[51] C. Ernemann and R. Yahyapour. *Grid Resource Management: State of the Art and Future Trends*, chapter Applying Economic Scheduling Methods to Grid Environments, pages 491–506. Kluwer Publishing, 2003.

[52] European DataGrid.
`http://www.edg.org/`.

[53] J. Ferguson. Visualization of Replica Optimization in Data Grids. Master's thesis, University of Glasgow, 2004.

[54] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-performance Distributed Computations. In *IEEE Symposium on High Performance Distributed Computing*, Portland, Oregon, USA, August 1997.

[55] Folding@home.
`http://www.stanford.edu/group/pandegroup/folding/`.

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

[57] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[58] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, 35(6):37–46, 2002.

[59] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum, 2002.

[60] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid. *International Journal of Supercomputer Applications*, 15(3), 2001.

[61] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Fransisco, California, August 2001.

[62] W. Gentzsch. Response to Ian Foster's "What is the Grid?". *GRIDToday*, 1(8), 2002.

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

[64] Globus Alliance.
`http://www.globus.org/`.

[65] M. Greenberger. The Computers of Tomorrow. *The Atlantic*, May 1964.

[66] Grid for UK Particle Physics.
`http://www.gridpp.ac.uk/`.

[67] Grid Physics Network.
`http://www.griphyn.org/`.

[68] Grid3.
`http://www.ivdgl.org/grid3/`.

[69] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr. Legion: The Next Logical Step Toward a Nationwide Virtual Computer. Technical Report CS-94-21, August 1994.

[70] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. In *1st IEEE/ACM International Workshop on Grid Computing*, Bangalore, India, December 2000.

[71] B. Huberman and T. Hogg. Distributed Computation as an Economic System. *Journal of Economic Perspectives*, 9(1):141–152, 1995.

[72] B. T. Huffman et al. The CDF/D0 UK GridPP Project. CDF Internal Note CDF/DOC/COMP_UPG/5858, February 2002.

[73] Internet Engineering Task Force.
`http://www.ietf.org/`.

[74] H. A. James, K. A. Hawick, and P. D. Coddington. Scheduling Independent Tasks on Metacomputing Systems. In *11th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS'99)*, Cambridge, MA, USA, November 1999.

[75] S. Jiang and X. Zhang. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 31–42, June 2002.

[76] Job Statistics for CMS Data Challenge 2004.
`http://cmsdoc.cern.ch/cms/LCG/LCG-2/dc04/fakeanalysis/pierro/`.

[77] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *20th International Conference on VLDB*, pages 439–450, 1994.

[78] W. E. Johnston. A Different Perspective on the Question of What is a Grid? *GRIDToday*, 1(9), 2002.

[79] B. Kahn and V.Cerf. ARPANET Maps 1969-1990. *Computer Comunications Review*, 1990.

[80] G. Kan. *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter Gnutella, pages 94–122. O'Reilly, 2001.

[81] N. H. Kapadia and J. A. B. Fortes. PUNCH: An Architecture for Web-Enabled Wide-Area Network-Computing. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 1999.

[82] LCG Grid Deployment Board. Resource Allocation and Planning Q2 2004. CERN Note, April 2004.

[83] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies. In *1999 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, pages 134–143, May 1999.

[84] E. A. Lee. Overview of the Ptolemy Project. Technical Report UCB/ERL M03/25, University of California, Berkeley, July 2003.

[85] V. Lefebure and T. Wildish (editors). The Spring 2002 DAQ TDR Production. CMS Internal Note. 2005/000, Geneva, Switzerland.

[86] W. Leinberger and V. Kumar. Information Power Grid: The New Frontier in Parallel Computing? *IEEE Concurrency*, 7(4):75–84, 1999.

[87] LHCb - The Large Hadron Collider beauty experiment. `http://cern.ch/lhcb/`.

[88] LHC@Home.
`http://cern.ch/athome/`.

[89] D. O. Litvintsev. The CDF Data Handling System. In *Computing in High Energy Physics (CHEP 2003)*, La Jolla, California, March 2003.

[90] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *8th Int. Conference on Distributed Computing Systems*, pages 104–111, 1988.

[91] D Lucking-Reiley. Vickrey Auctions in Practice: From Nineteenth Century Philately to Twenty-first Century E-commerce. *Journal of Economic Perspectives*, 14(3):183–192, 2000.

[92] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, 19(5):395 – 404, July 1976.

[93] P. Mockapetris. Domain Names - Concepts and Facilities. RFC 1034, November 1987.

[94] MONARC Project.
`http://cern.ch/MONARC`.

[95] Napster.
`http://www.napster.com/`.

[96] NASA Information Power Grid.
`http://www.ipg.nasa.gov/`.

[97] Z. Nemeth and V. Sunderam. Characterizing Grids: Attributes, Definitions and Foundations. *Journal of Grid Computing*, 1(1):9–23, 2003.

[98] Network for Earthquake Engineering Simulation Grid.
`http://www.neesgrid.org/`.

[99] C. Nicholson. Simulation of a Particle Physics Data Grid. Graduate Report, University of Glasgow, September 2004.

[100] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *1993 ACM SIGMOD Conference*, pages 297–306, 1993.

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

[102] A. Oram, editor. *Peer-To-Peer: Harnessing the Benefits of a Disruptive Technology*. O'Reilly, 2001.

[103] Organization for the Advancement of Structured Information Standards.
`http://www.oasis-open.org/`.

[104] E. Otoo, F. Olken, and A. Shoshani. Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids. In *2002 ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, USA, May 2002.

[105] S-M. Park, J-H. Kim, Y-B. Ko, and W-S. Yoon. Dynamic Data Grid Replication Strategy Based on Internet Hierarchy. In *Grid and Cooperative Computing, Second International Workshop (GCC2003)*, Lecture Notes in Computer Science, pages 838–837, Shanghai, China, December 2003. Springer.

[106] Particle Physics Data Grid.
`http://www.ppdg.net/`.

[107] Physics Analysis Workstation (PAW).
`http://cern.ch/wwwasd/paw/`.

[108] The Globus Project. GridFTP: Universal Data Transfer for the Grid. White Paper, 2000.

[109] Project JXTA.
`http://www.jxta.org/`.

[110] R. Raman, M. Livny, and M. Solomon. Matchmaking: An Extensible Framework for Distributed Resource Management. *Cluster Computing*, 2(2):129–138, 1999.

[111] R. Raman, M. Livny, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Seattle, Washington, USA, June 2003.

[112] K. Ranganathan and I. Foster. Simluation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[113] K. Ranganathan, A. Iamnitchi, and I. Foster. Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany, May 2002.

[114] F. Schintke, T. Schtt, and A. Reinefeld. A Framework for Self-Optimizing Grids Using P2P Components. In *14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, Prague, Czech Republic, September 2003.

[115] SETI@Home.
`http://setiathome.ssl.berkeley.edu/`.

[116] A. Shoshani, A. Sim, and J. Gu. *Grid Resource Management*, chapter Storage Resource Managers. Kluwer Academic Publishers, 2003.

[117] L. Smarr and C. E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[118] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A Scientific Tool for Modeling Computational Grids. In *2000 ACM/IEEE Conference on Supercomputing (SC2000)*, Dallas, Texas, USA, November 2000.

[119] D. Stickland. CMS Input on Computing Capacities and Services for the Computing MoU taskforce. CMS Computing Model Note, April 2004.

[120] H. Stockinger, F. Donno, E. Laure, S. Muzaffar, P. Kunszt, G. Andronico, and P. Millar. Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project. In *Computing in High Energy Physics (CHEP 2003)*, La Jolla, California, USA, March 2003.

[121] H. Stockinger, A. Samar, S. Muzaffar, and F. Donno. Grid Data Mirroring Package (GDMP). *Scientific Programming Journal - Special Issue: Grid Computing*, 10(2):121–134, 2002.

[122] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests. In *11th IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, Edinburgh, Scotland, July 2002.

[123] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, San Fransisco, California, USA, August 2001.

[124] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the Well: Creating Communities for Grid I/O. In *2001 ACM/IEEE Conference on Supercomputing (SC2001)*, November 2001.

[125] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure (OGSI) Version 1.0. Technical report, Global Grid Forum, 2003.

[126] Unified Modelling Language Version 2.0.
`http://www.uml.org/`.

[127] H. R. Varian. Economic Mechanism Design for Computerized Agents. In *First USENIX Workshop on Electronic Commerce*, New York, USA, 1995.

[128] S. Vazhkudai, S. Tuecke, and I. Foster. Replica Selection in the Globus Data Grid. In *1st International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001.

[129] W. Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance*, 16(1):8–37, March 1961.

[130] W3C - The World Wide Web Consortium.
`http://www.w3.org/`.

[131] W3C. "Web Services Activity".
`http://www.w3c.org/2002/ws/`.

[132] Web Service Definition Language.
`http://www.w3.org/TR/wsdl`.

[133] O. Wolfson, S. Jajodia, and Y. Huang. An Adaptive Data Replication Algorithm. *ACM Transactions on Database Systems*, 22(2):255–314, 1997.

[134] R. H. Zakon. Hobbes' Internet Timeline, 2004.
`http://www.zakon.org/robert/internet/timeline/`.

[135] G. K. Zipf. *Selected Studies of the Principle of Relative Frequency in Language.* Cambridge, MA.: Harvard University Press, 1932.