Measurement and Simulation of the Performance of High Energy Physics Data Grids

Paul Andrew Crosby

University College London

Submitted to the University of London

in fulfilment of the requirements

for the award of the degree of

Doctor of Philosophy.

June 2004.

Abstract

This thesis describes a study of resource brokering in a computational Grid for high energy physics. Such systems are being devised in order to manage the unprecedented workload of the next generation particle physics experiments such as those at the Large Hadron Collider. A simulation of the European Data Grid has been constructed, and calibrated using logging data from a real Grid testbed. This model is then used to explore the Grid's middleware configuration, and suggest improvements to its scheduling policy.

The expansion of the simulation to include data analysis of the type conducted by particle physicists is then described. A variety of job and data management policies are explored, in order to determine how well they meet the needs of physicists, as well as how efficiently they make use of CPU and network resources. Appropriate performance indicators are introduced in order to measure how well jobs and resources are managed from different perspectives. The effects of inefficiencies in Grid middleware are explored, as are methods of compensating for them.

It is demonstrated that a scheduling algorithm should alter its weighting on load balancing and data distribution, depending on whether data transfer or CPU requirements dominate, and also on the level of job loading. It is also shown that an economic model for data management and replication can improve the efficiency of network use and job processing.

Acknowledgements

Firstly I would like to thank my supervisor, Dave Waters, for three years of advice, encouragement and guidance during the studies described in the following pages.

Peter Clarke, Jon Butterworth and Tegid Jones were very helpful when I was establishing the direction that this work eventually took.

Many thanks to David Colling, whose assistance was enormously helpful in developing the Grid logging and monitoring techniques described in chapters 5 - 7.

Phil Adamson, Matt Isherwood, Will Curnow and Ryan Nichol have offered me varying degrees of advice, sarcasm, caffeine and alcohol over the years, and I would like to extend varying degrees of gratitude to them in return.

I have shared an office with many people during my course, so I would like to thank them all for being good company, as well as the rest of the UCL HEP group, for being friendly and helpful people.

I would like to thank my family for their support, and for generally being wonderful whenever appropriate over the last 26 years. The same goes to my friends who may not have seen much of me recently. And finally, love and thanks to Sarah, who has been superhumanly patient, and deserves more gratitude than will fit onto this page.

Contents

List of Figures

List of Tables

Chapter 1

Computational Grids

This chapter presents an outline of the thesis. The concept of a Grid is explained, the layers of hardware and middleware are described with examples, and examples are also given of the different kinds of application that can make use of Grid technology.

1.1 Introduction

This thesis explores ways of improving the efficiency of job and data management in computational Grids for High Energy Physics. Such Grids are being constructed to cope with the unprecedented workloads of new HEP experiments such as those at the Large Hadron Collider in CERN. They will be managed by decision making entities, or resource brokers, which will schedule computational jobs and control the flow of data between sites. These will operate according to certain policies, which may have to deal with a variety of usage patterns, and CPU and storage resource configurations.

In order to investigate different approaches to this problem, a simulation called EDGSim has been created to explore the ways in which the efficiency of job and data management can be improved. It models existing production Grid technology and job processing in the European Data Grid, as well as possible Grid structures for next generation HEP computing.

This first chapter gives some background to the Grid concept and its development, and outlines a variety of architectures and applications that make use of these ideas. Chapter 2 describes the requirements of HEP computing, and how Grid solutions have been developed to meet them. Chapter 3 describes the European Data Grid in detail, including its middleware components, and how it processes the computational jobs that make up its workload.

Chapter 4 describes how a simulation, EDGSim, was constructed for the work described in this thesis, in order to replicate the interaction and functionality of EDG components, and the submission and running of jobs. Chapter 5 details how real data were extracted from a production EDG testbed for comparison with and validation of the simulation, and Chapter 6 describes performance indicators used to measure Grid efficiency for real and simulated data.

Chapter 7 describes how the simulation was used to model the functioning and inefficiencies identified in the real Grid from the logging and monitoring data; test runs involving submission of simple data-independent jobs are carried out with the real Grid and the simulated one, in order to calibrate and validate the simulation. Chapter 8 then explores the effects of these inefficiencies in a larger Grid under heavier job loading, and shows how job scheduling can be improved given these constraints.

Chapter 9 reviews other work simulating Data Grids, then describes new performance indicators for the management of data transfers, and outlines the job scheduling and data management policies to be used. Chapter 10 presents the results from the Data Grid simulation in EDGSim, demonstrating how these policies perform with different loading conditions and middleware inefficiencies. Solutions are presented for job and data management in future Grid challenges, and the work is concluded in Chapter 11.

EDGSim and the simulated Grid models described were devised for the work detailed in this thesis, as were the data mining and monitoring techniques described in Chapter 5. The metrics E_{System}, E_{User} and $E_{Network}$ were developed with the assistance of David Colling at Imperial College London. The "Most Files + ETT" and "Adaptive" scheduling algorithms were devised for this work; other algorithms used were adapted from other Grid simulation projects. All results presented here were produced using EDGSim and the data extraction techniques mentioned above.

1.2 What is a Grid?

A Grid is an infrastructure which allows computing resources which are distributed geographically, and may belong to many separate organisations, to be managed and utilised efficiently, and shared between the members of the Grid. It should run in such a way that it is 'invisible' to the user who wishes to harness the resources available, utilising intermediary software known as middleware to bridge the gap between the user's applications, and the computational resources being harnessed to run them.

The name Grid is taken from the analogous development of national power grids in the last century. When electricity first became seen as a source of power for new kinds of machinery, light sources and so on, local generators were created to power the first light bulbs and other luxuries for people with access to these generators. However, demand began to grow for these new commodities, and eventually a national electricity network formed, able to aggregate the power generated across a nation, and supply it where needed. Lulls and surges in this demand at different times of day or year could be compensated for, providing a consistent supply of electricity for everyone, at a much reduced cost.

According to this analogy, we are currently at the stage of localised production, and the tools to build Grids of computing power are rapidly emerging. The development of the electrical power grid had a huge impact on society, allowing great leaps in technology and quality of life. Another analogy to the Grid which is often cited is that of the appearance of national train and road networks, which allowed the growth of much bigger centres of population. Some have suggested that the sharing of computing resources in such a way will have an impact upon society of a similar magnitude [1].

The Grid will not quite be a "next generation" version of the Web, despite some similarities. The World Wide Web, built on the HTTP protocol and the HTML language, allows users to share information around the world. A Grid goes much further than that, in that computing resources themselves are being shared. This is a much more complex task that requires research in a number of areas: security, as a reliable means of authentication is important when sharing one's computing resources; resource brokering, to find a suitable means of executing users' tasks; information sharing, to coordinate people and other Grid entities who are widely geographically distributed; and data management, reliably storing and transferring large volumes of data.

How precisely this will be achieved is a matter of ongoing research, and technology is emerging as many different potential users work to build the tools that they will need to make the Grid work for them. The way in which these technologies fit together, and some of the projects developing this middleware are described in sections 1.3 to 1.6.

There are several classes of application that will benefit greatly from the power that a Grid will make available to them, and in some cases it will be possible to run applications which would have been impossible with the computing power available at any one site. The requirements of these applications, and how the Grid can meet them, are described in section 1.7.

1.3 Structure of a Grid

The participants in a Grid will be widely geographically distributed, and may provide or require access to many different types of computational resource, storage, software, instrumentation and so on. These producers and consumers of resources will be groups of individuals and institutions with shared interests, and a set of rules governing their involvement in the Grid. Such a group is known as a Virtual Organisation (VO).

Since VOs will vary in size, and will have their own ideas of what they wish to put into and get out of a Grid, the means of sharing resources between them will have to be flexible, while still allowing owner control, and ensuring secure access to remote resources. This means ensuring interoperability between highly heterogeneous sets of resources and users. However, this complexity should ideally be hidden from the user, as Grid access should be as simple and transparent as possible. To facilitate this, standardised Application Programming Interfaces (APIs) are used, based upon appropriate Grid protocols, to allow a flexible and extensible architecture to be constructed.

The infrastructure of a Grid can be seen as a series of layers, connecting users' applications with the resources needed to run them. The layers of software making this connection are known as middleware. The hierarchy of layers in shown in fig. 1.1.

Components from each layer should be able to call those from any layer below them to carry out their function. They should be as modular and as flexible as possible, in order to future-proof the Grid structure.

In the following sections the role played by each of these layers is described, along with some examples of Grid software solutions.

1.4 Fabric

At the lowest level of the hierarchy is the fabric, consisting of the resources being contributed to the Grid. These could be a wide range of computing resources such as desktop PCs, batch farms or SMPs; storage media such as disk or tape; networks; or scientific instrumentation to be used collaboratively such as radio telescopes.

The upper layer of the fabric consists of local resource management, distinct from the middleware in the next layer. This includes operating systems such as Linux or Windows, local queueing systems such as PBS, and libraries of software available at this site. These systems should operate independently of the Grid middleware components, but be able to communicate with them via the appropriate protocols and APIs.

1.5 Core Middleware

The core middleware is responsible for communication between the components and sites of the Grid. This is where new Grid protocols must be defined

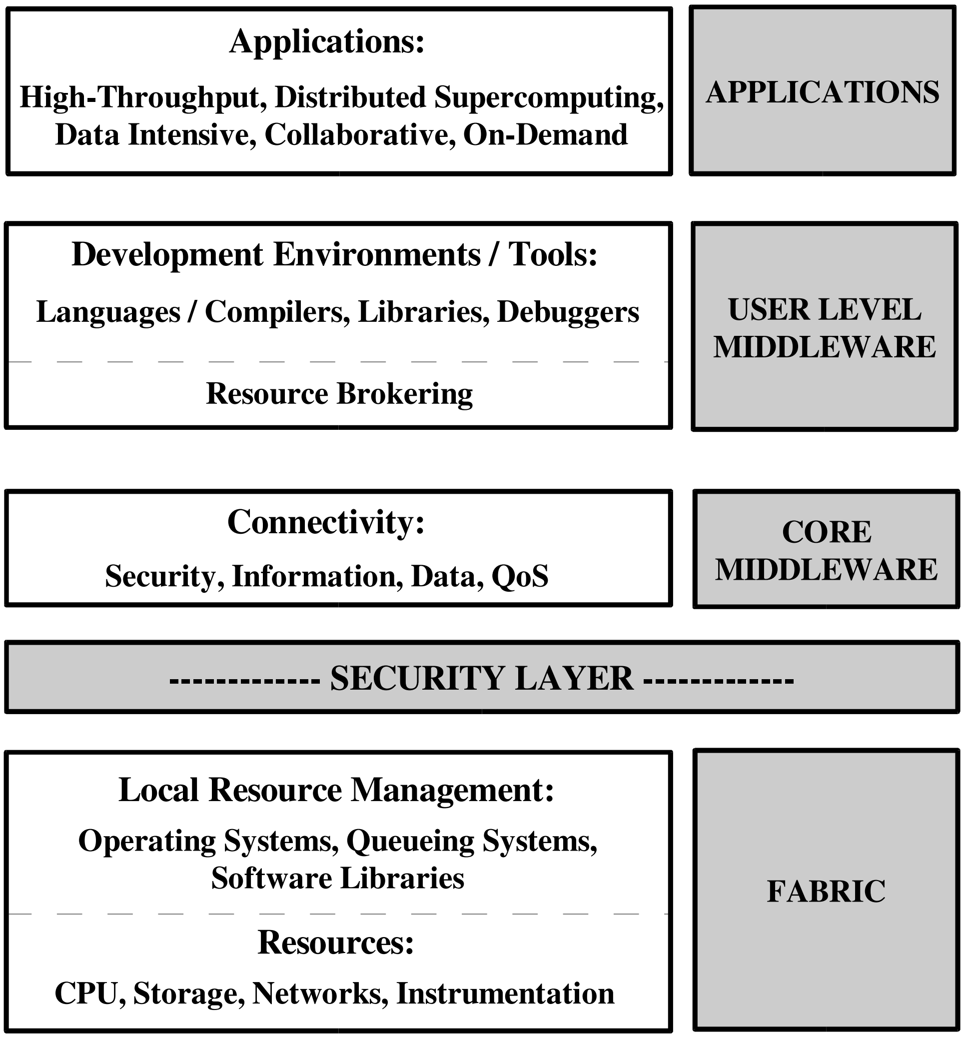

Figure 1.1: The layers of hardware and software that make up the architecture of the Grid.

and implemented in order to carry out Grid operations.

Information services will be required to pass status information from fabric components to aggregation and management middleware in the layer above. Transport protocols will be required for the transfer of data, whether it is scientific data or multicast communications. Quality of Service (QoS) information will also be conveyed through this layer, allowing network bandwidth, disk space and other services to be reserved, guaranteeing that a particular service can be carried out. Finally, this layer also manages Grid security and authentication.

1.5.1 Globus

The Globus toolkit [2] is a Grid software infrastructure designed to take advantage of existing services as much as possible. Globus is designed to work with heterogeneous resources, so it uses vendor-supplied protocols and interfaces where they are available at the fabric level, and supplies any other functionality where needed. The standard Internet protocols such as TCP/IP are used for communication. Fig. 1.2 shows how Globus' components fit into the Grid hierarchy shown in fig. 1.1.

The functionality needed to bridge the gap between the fabric level and the higher middleware functions are the responsibility of Globus. It takes information about the status of local resources, and packages them up in a form that can be used by resource discovery software, the Grid Resource Information Protocol (GRIP). This information is communicated via the Metadata Directory Service (MDS).

Security is handled with a public key based system, the Grid Security Infrastructure (GSI), which uses the X.509 authentication model within the

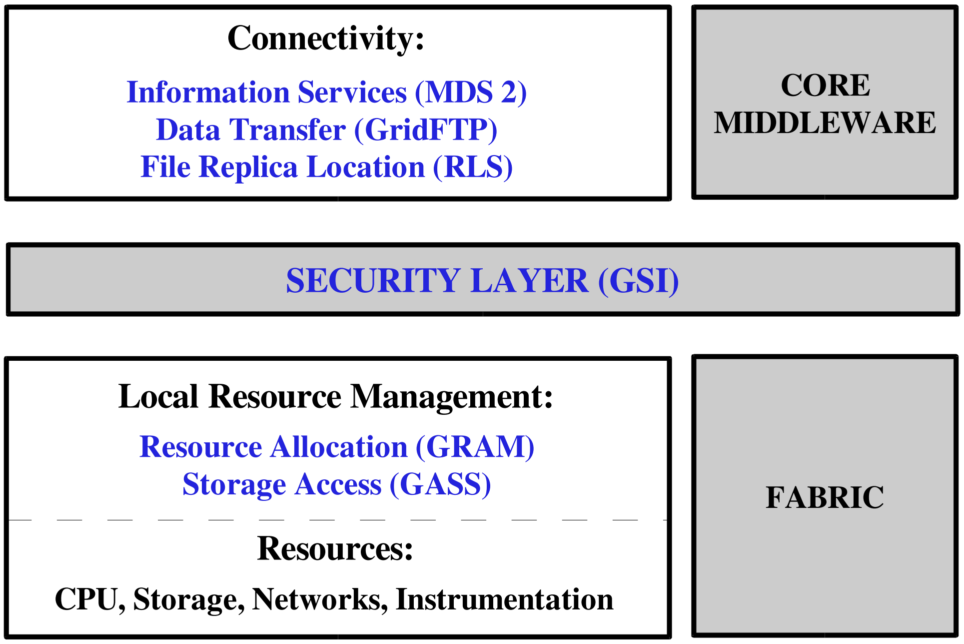

Figure 1.2: The places of the various Globus components in the Grid hierarchy. Globus' modular nature allows developers to choose the tools that they find useful.

OpenSSL framework [3]. The GSI interfaces with the local security solution at member sites. The resource owner has control over authorisation of users.

A strength of Globus is that it is not a single monolithic package. It is a set of services with well defined APIs, meaning that developers can choose those that are needed, and incorporate them into their applications as appropriate.

1.5.2 Legion

Legion [4] provides an object-based Grid architecture to create a single virtual machine from geographically distributed, heterogeneous computing resources. In the Legion model, all hardware and software components are represented by objects. These objects have a set of methods that can be called, and can perform operations on other objects using these methods, as in any object oriented system. Legion provides the APIs for this interaction to take place between components.

The classes used to describe the different types of component can be inherited from, and so that developers can override functionality according to their needs. Instances of classes are managed by class objects, which create, activate and deactivate objects of that class. Class objects are also responsible for providing details to other objects that wish to communicate with their instances. The hierarchy of basic Legion objects is shown in fig. 1.3.

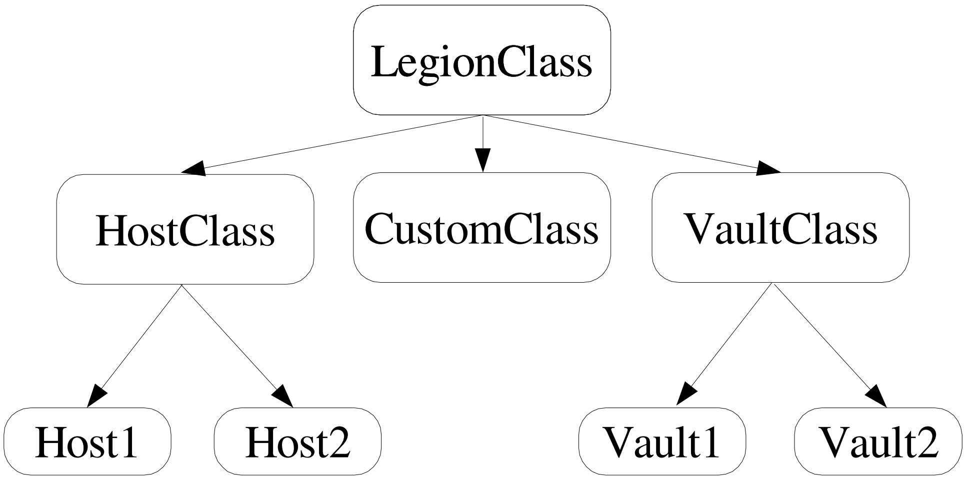

Figure 1.3: The hierarchy of basic objects needed to describe a Legion system.

The security model in Legion is designed to protect objects and communication between them. A user (or another object) wishing to make a call on an object has a certificate assigned to them describing the rights granted to them by the called object's owner. This certificate is checked for its scope and authenticity. The objects are also assigned public-key pairs to encrypt communication between them.

1.5.3 Netsolve

Netsolve [5] is an example of an application-specific Grid tool. It is a remote computing environment, in which the user sends the problem that they wish to solve to a remote server, which operates on the problem and then returns the results. It is intended to allow the user to access remote computing resources and take advantage of programming libraries (C++, Fortran etc.) and software packages (Matlab, Mathematica etc.) to carry out scientific computation. Standard communication protocols are used, and remote authentication is managed by Kerberos.

There are three main components of Netsolve: the agent, the server and the client. The relationship between these components is shown in fig. 1.4.

The user accesses the system via a client, which is a set of APIs which are used to describe the specific details of the request. The request is sent to an agent, which maintains a database of available servers, and determines those that are suitable for the task. The agent also keeps track of usage statistics and server failures, so that it can balance loading, and avoid unreliable resources. The server software makes the local software library routines available as a Netsolve service using its interface definition language.

Fault tolerance is also built into Netsolve, as the user has access to multiple agents and servers, so that a request should always be served unless failure occurs on a very large scale. If failure occurs, the last stage of a request can be retried at a different site.

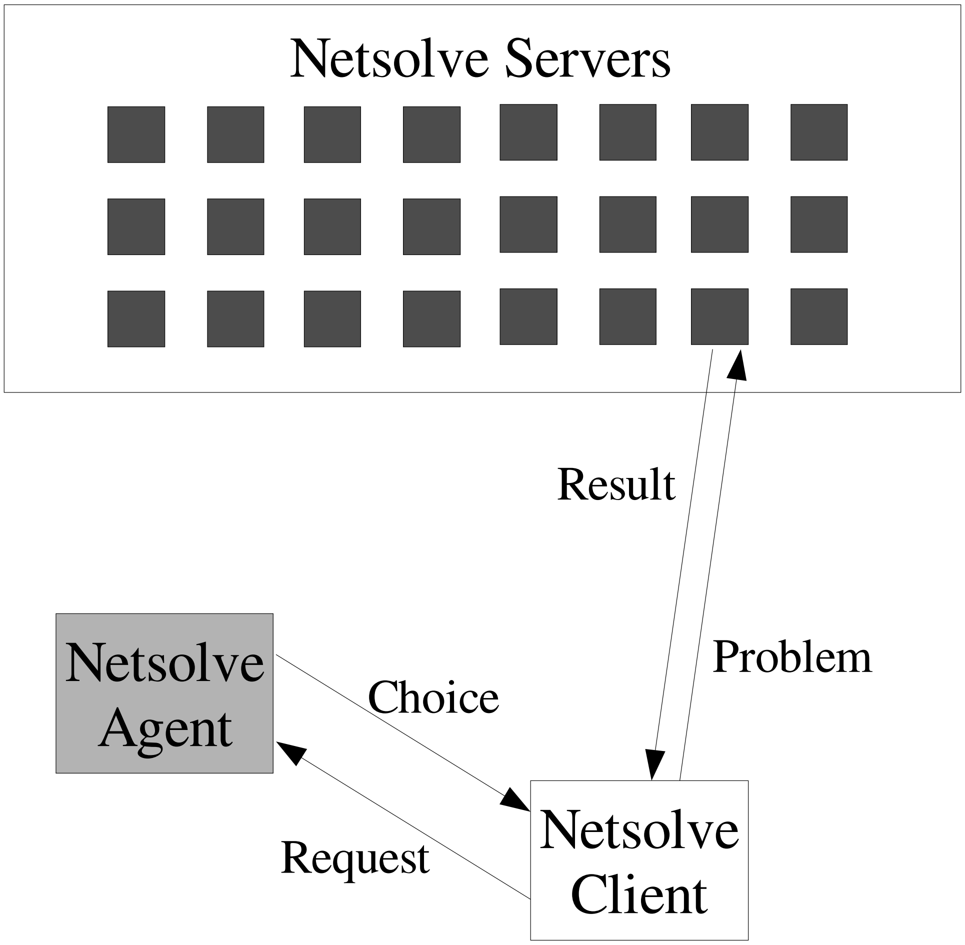

Figure 1.4: The relationship between client, agent and server in a Netsolve system, including the flow of communication.

1.6 User Level Middleware

The next layer up is concerned with resource brokering. This is where these resources, and the information about them, is aggregated in order to serve users' tasks. Based upon resource status information, and the requirements of the jobs to be scheduled, the brokering middleware must choose the most suitable match between job and resource. This could be cycles on a machine not currently booked or in use; the best network route to transfer a set of data, and the most appropriate location to copy it from; or free storage space for a new dataset produced in a scientific experiment.

Built on top of this layer are the APIs and Software Development Kits (SDKs) allowing the user to interface their applications with the middleware, and use the power of the Grid to solve their computational problems.

1.6.1 Nimrod/G

The Nimrod/G scheduler [6] is designed for use with high-throughput computing tasks, in which a large scale parametric study is required. It is based on a package called Nimrod, which was designed to manage a static set of resources. It has been integrated with Globus services in order to handle dynamic allocation of distributed resources across a variety of administrative domains. The architecture of Nimrod/G is shown in fig. 1.5.

The user submits their job via the client, defining the parameters of the problem they wish to solve in Nimrod's declarative modelling language. They can also declare their priorities as to how the job is run, influencing the decision made by the scheduler. Multiple instances of the same client can be run, allowing users to monitor their projects from different locations.

The task is then passed to the parametric engine, which is responsible for submitting and managing jobs. It parameterises the project and divides the workload between resources according to its scheduling adviser and status information supplied by the Globus information services. Jobs are passed to the dispatcher to be assigned to resources, and the parametric engine records the status of the project as a whole, in case of component failure in the Nimrod/G system.

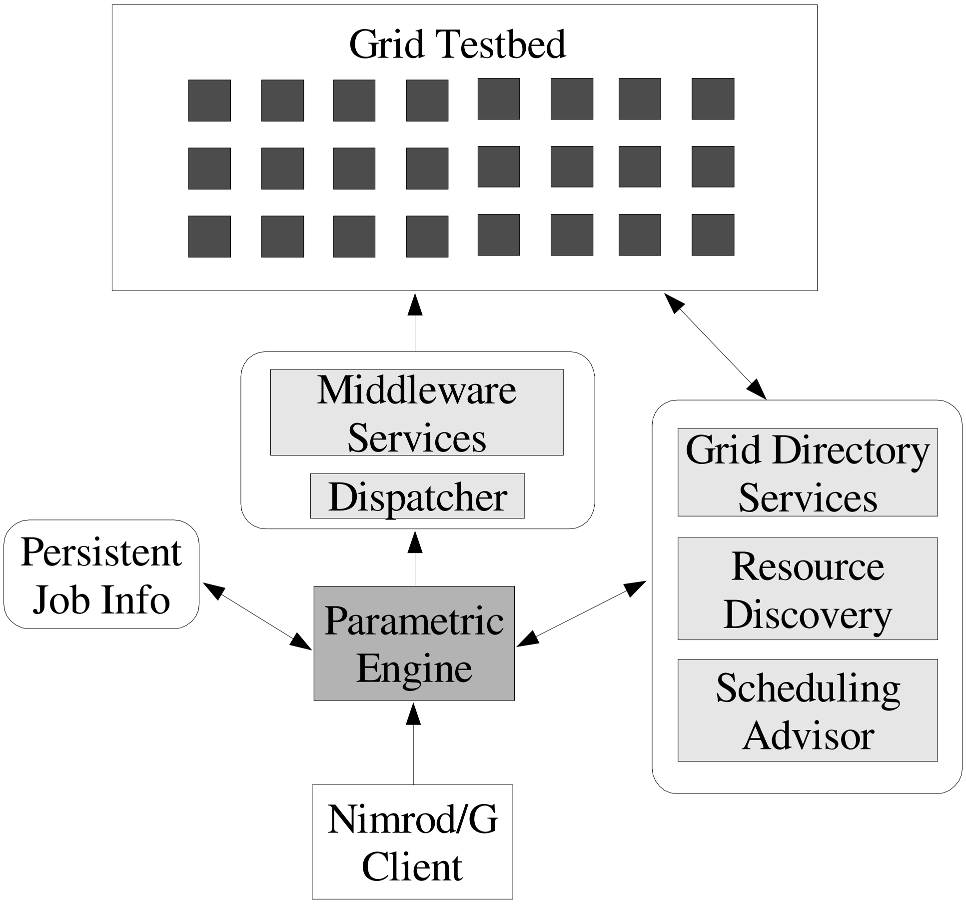

Figure 1.5: The architecture of the Nimrod/G scheduling system, showing how the parametric engine makes use of Grid services to find appropriate resources to run computational jobs.

The scheduling of the jobs itself depends upon the user's priorities. Nimrod/G regulates use of its resources by charging for access. The user decides whether they wish to minimise the time taken to run their project, or the cost of running. Dividing their project up will produce results more quickly, but will require a larger budget as more machines will be needed.

1.6.2 AppLeS

The AppLeS project has created a model for high throughput parameterisation applications, the AppLeS Parameter Sweep Template (APST) [7]. It caters for computational tasks that are numerous and independent (i.e. non-communicating) in nature, possibly with data requirements. It uses components from the Globus toolkit, particularly the GSI information services, and NetSolve's scheduling middleware. Information on network status comes from the Network Weather Service (NWS) [8].

The scheduler must be able to respond quickly and dynamically when dealing with the large volume of jobs involved in a parameter sweep application (PSA). It generates a plan to assign tasks to CPUs, and data transfers to network links. The scheduler creates a Gantt chart with a column representing each computational or network resource, with blocks added to each column to indicate time usage for this resource. When a new scheduling plan is created, different configurations are considered. A variety of heuristics can be used to compare these scheduling scenarios, and choose the one that is judged to be the best according to that policy.

The best heuristic depends upon the current workload, as well as the accuracy of the status information being passed to the scheduler. The scheduler monitors the state of the Grid, and chooses its policy accordingly. It can also choose how thoroughly it considers its choices - when the job submission rate is high, the time taken to consider all possible scheduling configurations can be prohibitive. To compensate, the scheduler chooses a configuration from a randomly chosen subset of parameter space.

1.7 Applications

There are many possibilities offered by Grid technology, including applications that would not have been possible with just the computing resources of a single site. Any system of classification for these applications will have overlaps, but this section will follow the system outlined in [1].

1.7.1 High Throughput Computing

High throughput computing requires a large amount of processing power, in order to deal with a load of many independent, or only loosely coupled jobs, like those described in section 1.6.2. Many applications of a scientific nature might take this form.

NOVA

The STAR experiment has created a tool called Networked Object-based Environment for Analysis (NOVA) [9] for high throughput Monte Carlo simulation, with I/O fault tolerance, as errors occurred most frequently at this stage. The tests were carried out on a distributed system (between BNL and LBNL on the two US coasts), using over 100 CPUs and generating more than 20 TB of data.

NOVA is built upon Apache web servers, and interfaces with the ROOT physics analysis package. It has a two level architecture to handle I/O problems such as a bottleneck in write-locks, when multiple processes attempt to write their output to the same file. Large tasks are broken up into smaller processes, which are handled by lower level agents. These are coordinated by the higher level agents, which manage distributed storage facilities, and control the low level agents' access to them.

Folding@Home

The mechanisms involved in protein folding have so far been simulated using conventional supercomputing technology, with many computational processes interacting with each other as the simulated system evolves. These processes involve a large amount of network I/O, and thus require tightly coupled CPUs in order to run efficiently.

However, in order to model protein folding in the detail required for continuing work, a much higher level of statistical sampling will be required, resulting in a workload that is too large for any single computing facility. In order to make use of distributed resources, it was necessary to develop a new computing model, as the existing algorithms would not have been workable with higher network overheads.

The Folding@Home project [10] (based on SETI@Home, which was developed to aid in the Search for Extra Terrestrial Intelligence) has successfully created such a model. It uses multiple, very loosely coupled simulation clients which are downloaded and run as a screensaver using idle cycles on workstations or home PCs. The server side demands are high, as large quantities of incoming data must be collated from heterogeneous resources, with fault tolerance, and bandwidth and latency issues to consider. Based upon the results returned by the clients, typically representing a 100 ps stage of a folding process that takes several hundred microseconds in total, the interesting parameter space for the next stage is determined, and new tasks are farmed out to the clients. In this way, 40,000 participants generated 10,000 CPU-years of data in 12 months.

1.7.2 Distributed Supercomputing

Distributed supercomputing aggregates computing power to run very large applications, such as simulations of stellar dynamics or complex chemical processes. Geographically distributed processes must be able to communicate with each other effectively, which means minimising latency and ensuring sufficient bandwidth between sites while the application is running.

SF-Express

This technology is of particular interest to the military, for the simulation of combat situations. The Synthetic Forces Express (SF-Express) [11] project was created to harness the power of distributed computing resources to handle the increasing complexity of the US Department of Defense's Modular Semi-Automated Forces (ModSAF) simulation. ModSAF simulates a battlefield scenario, with individual entities represented with a high degree of precision. Each workstation runs an independent execution of the SAFSim software, and represents 30-100 vehicles or other entities, which must interact with each other and the environment realistically.

The hardware requirements for a simulation involving 50000 vehicles are beyond the resources available at a single site. SF-Express is an attempt to run a ModSAF simulation on geographically distributed resources, maintaining the communication between simulated entities. The network of routers that manages this communication is shown in fig. 1.6.

Individual workstations communicate by exchanging Protocol Data Units (PDUs), which describe individual entities, environmental factors, weapons firing, etc. This information must be passed to all other workstations managing entities that might be affected by it. However, it is impractical to have

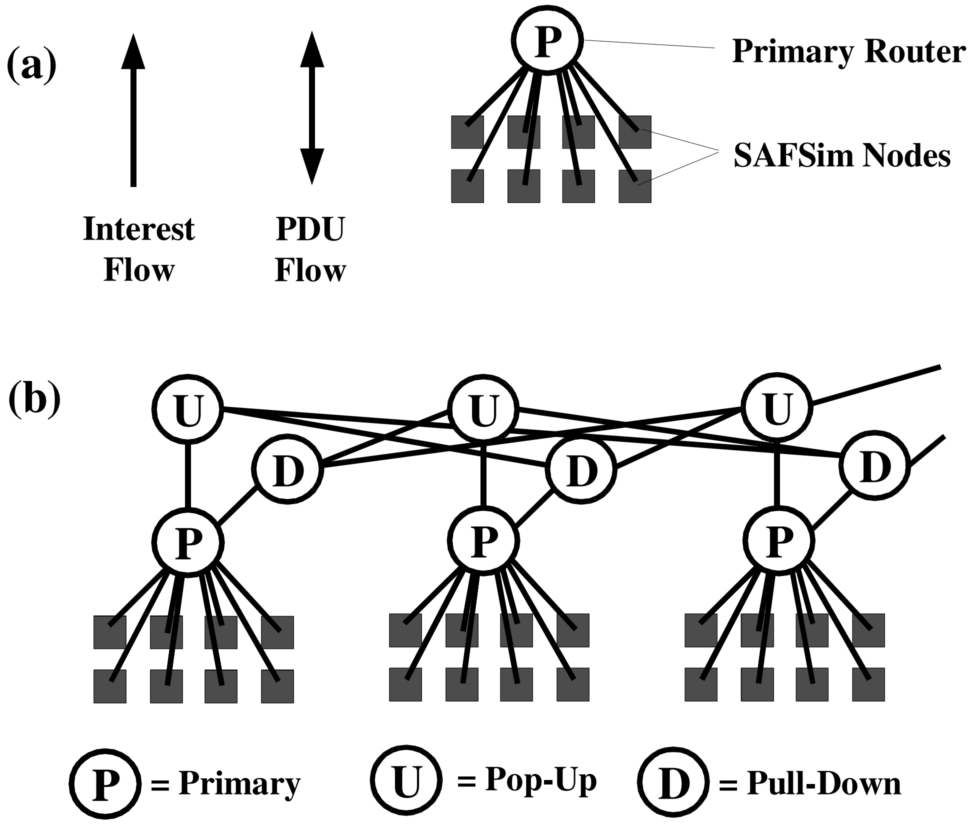

Figure 1.6: The router structure in the SF-Express network, showing (a) The communication between individual nodes and the primary router, and (b) the flow of information between routers at different sites.

all PDUs passed to all workstations, as this would not be scaleable for larger ModSAF scenarios.

In the SF-Express structure, all PDUs are passed to a Primary Router local to the workstations. The workstations also declare their interest states, i.e. the other entities from which they require the latest PDUs. The Primary Router passes the local PDUs to its Pop-Up Router, and the interest states to the Pull-Down Router. The Pull-Down Routers then have the task of acquiring the requested PDUs from the Pop-Up servers, and passing the results back to the Primary Router. This means that the information flow between sites is no larger than is necessary, and a set of communication protocols avoids deadlocks, or the arrival of unexpectedly large messages that might exceed the local buffer space.

Atmosphere-Ocean Circulation Modelling

A distributed computing system has been used to run a complex simulation of the interaction between the atmosphere and the ocean [12]. This heterogeneous system consisted of a Cray C-90 at San Diego Supercomputer Centre, and an Intel Paragon at Caltech, separated by roughly 200 km, and connected with a Gigabit network link. The simulation was decomposed into three parts - the physics of the atmospheric general circulation model (AGCM), the AGCM dynamics, and the oceanic general circulation model (OGCM).

The AGCM ran on the Cray at SDSC, whereas the OGCM, which was more suited to parallelisation, ran on the Intel machine. The AGCM dynamics component and the OGCM ran concurrently, and each part was broken into subsections that would send the resulting interaction data to the other site while the next section ran. A schematic of this model is shown in fig. 1.7.

It was estimated that during running the AGCM required 1 s per cycle for the physics, and 2s for the dynamics. The OGCM required 1.1 s in total. Running over the two sites, the simulation required 3 s in total per cycle, compared to 4.6 s when the whole model was run on the Paragon.

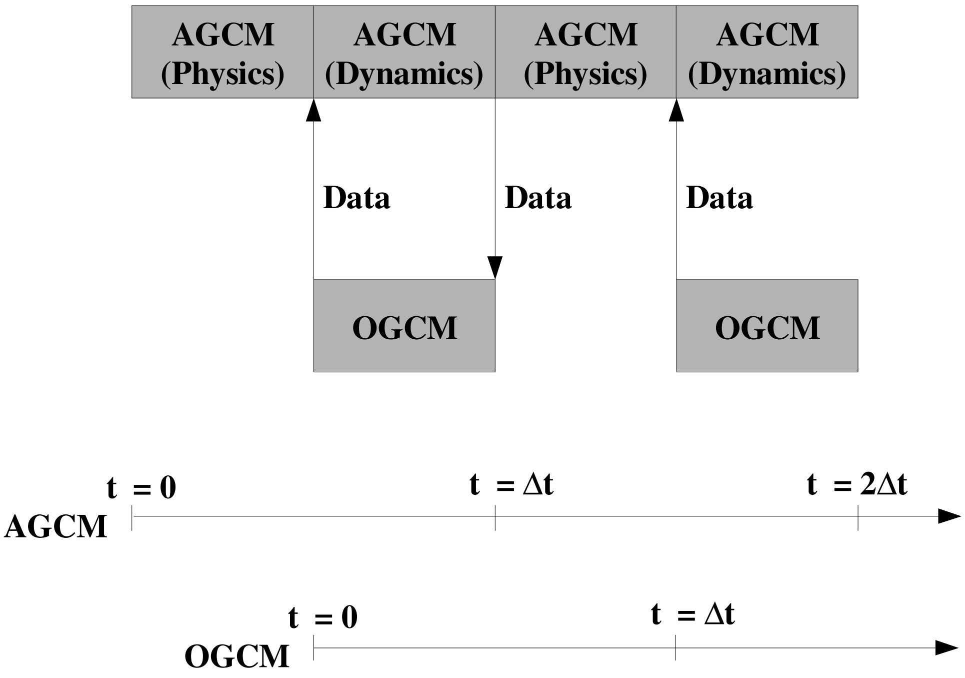

Figure 1.7: Schematic showing the interactions between the atmosphere and ocean components of the climate model, arranged so as to minimise time lost due to this communication.

1.7.3 Data Intensive Computing

Data intensive computing involves large volumes of data stored at many different sites. These data must be transported efficiently to the location where it is required, which means that accurate knowledge of available storage facilities and the state of the network is required, and computational resources must also be available to process these data too. The high energy physics data Grid model described in this thesis is an example of this type of technology.

1.7.4 Collaborative Computing

Collaborative computing is similar to distributed supercomputing, except that the collective computing power at the users' distributed sites is used to create a shared virtual space. This could be a virtual meeting room, which would be a significant advance in communication from standard video conferencing, or a means of collaboratively exploring a three dimensional model.

The Pervasive Collaborative Computing Environment

The Pervasive Collaborative Computing Environment (PCCE) [13] project aims to apply Grid techniques to share information in scientific collaborations. The first stage of this work has created the LBNLSecureMessaging system, which provides the means to communicate with whatever degree of formality is required. It uses similar technology to Internet Relay Chat (IRC), but with secure connections using X.509 authentication.

Conversations take place in virtual venues, which can be either private, accessible by invitation only, or public and open to anyone who wishes to join. Permanent venues can be established, which will exist whether they are in use or not, while temporary venues can be created for one-to-one conversations, with others joining as desired. Messages can be left for users who are offline. Information about user availability (available, busy, etc.), location (home, work, etc.) and venue membership is provided.

The intention is to apply these principles of collaboration in other areas. Extension into a videoconferencing system is one possibility. A system of collaborative document editing is also being devised. Two modes of editing are considered: synchronous and asynchronous. In the asynchronous mode, a change tracking system such as CVS might be employed. In this case the document would be locked until the author wished to check the new version into the repository to be further developed by others. Synchronous mode would require multiple users to edit the document simultaneously. Various facilities would be needed, such as a single, up to date view of the current version, and the ability to lock certain sections of the document to be edited in real time.

A Virtual Control Room for CMS

An ambitious project to create a virtual control room is under consideration by the CMS experiment at the Large Hadron Collider [14]. A modern high energy physics experiment involves hundreds of people, drawn from institutes all over the world, and during running they will all be assigned shifts to monitor the experiment. The travel costs involved in this can be high, and with many American members in the CMS collaboration, transatlantic travel will make up a large part of these costs. A virtual control room would be a means of cutting down the amount of travel required.

A shift will involve around five people working together in the CMS control room in Geneva, working closely together. Their communication will be verbal, along with other visual cues (eye contact, gesticulation etc). In order to allow someone at a remote site (e.g. in the US) to participate, this communication must be replicated sufficiently.

For the audio part of communication the most important consideration is latency. In order for completely natural conversation to take place, a round-trip time of 100 ms is needed. The current maximum with fibre optic cables is around 80 ms between Geneva and Fermilab, although measured values are often closer to 200 ms - not perfect, but still viable.

Bandwidth is not an issue for audio communication, but it becomes significant for the video component. A 3D representation of the control room for remote participants would be ideal (i.e. matching human perception), but a high resolution 2D picture may suffice, as long as important displays and instrumentation are shown as accurately as possible in the shared area. The image of the control room must also be updated with an acceptable frame rate. Tests running at around 60 frames per second required 40 MBits/sec for two way communication. When the LHC begins running in 2007 this bitrate should be readily available.

1.7.5 On-Demand Computing

On-demand computing caters for users with short term requirements that cannot be handled locally. This could be processing power, software, or access to a particular instrument or device. The user would book or hire the resource for a particular time duration.

Remote Control of X-Ray Microtomography

A demonstration at the SC'98 conference [15] applied Grid principles to the remote control and visualisation of an x-ray microtomography experiment. In such an experiment, the internal structure of a material is examined at the micron level by illuminating it with an x-ray source, and collecting data with a charge-coupled device (CCD). An image of this structure is then reconstructed, a procedure that is very CPU (and therefore time) intensive. Typically the rate of data production is a gigabyte per second, with the necessary computing power being around a teraflop per second. These requirements exceed the capabilities of local resources to deal with the load in real time. Since it is often necessary to alter the experimental parameters after visualisation to achieve the desired results, it can be a time consuming and expensive procedure.

Grid technology allows much greater computing power to be harnessed, so that the above can be carried out much more quickly. In addition, the experiment can be controlled from one or more remote locations. Fig. 1.8 shows the structure used.

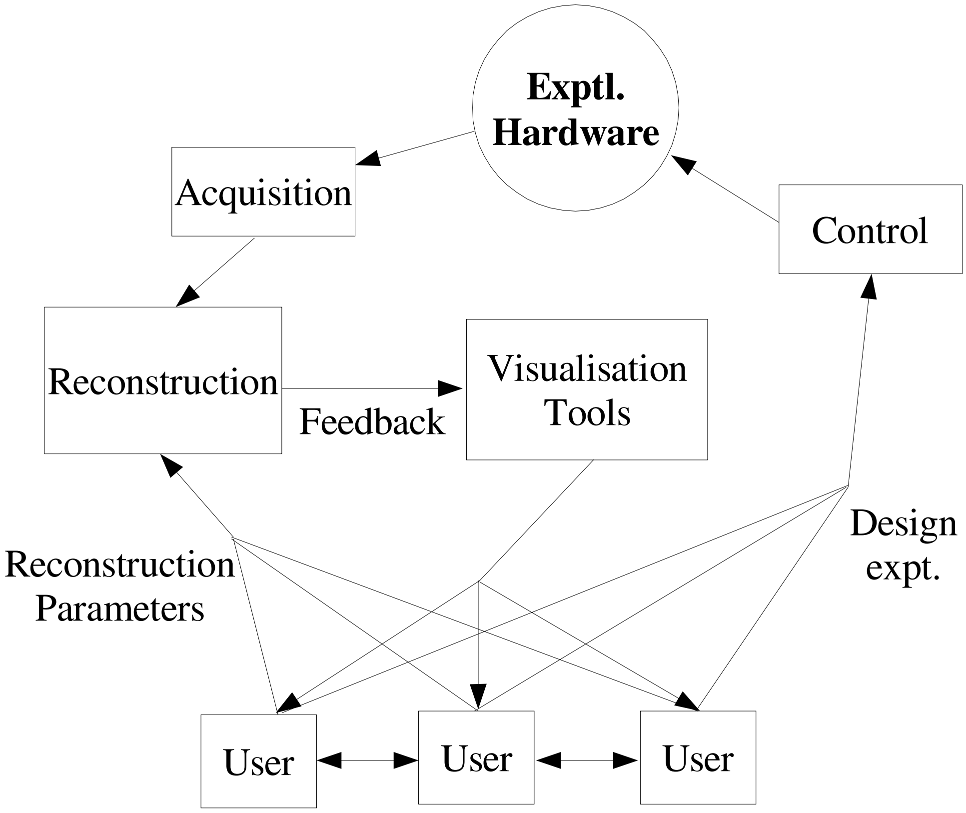

Figure 1.8: The structure of the distributed steering and visualisation SC'98 x-ray microtomography experiment, showing the communication between its components.

The users collaboratively design the experiment, communicating via video-conference if they are at different sites. These specifications are sent to the experiment, where they are executed by local technicians. The results are collected by data acquisition hardware, and passed to reconstruction software. At this stage, the images are not reconstructed at the maximum resolution - an initial, lower resolution image is sent to the user. This can be done with a variety of visualisation tools, from immersive viewing environments to ordinary desktop browsers. This less detailed image is then used to determine whether the experiment is producing useful results. If not, it can be stopped, and restarted after reconfiguration.

The users can then decide how to carry out the full reconstruction of the results. Based upon the received image, new parameters can be sent back to the reconstruction software, and a modified image returned, with the feedback loop continuing until a satisfactory result is achieved.

CosmoGrid

CosmoGrid [16] is a project created to make a several hundred CPU SGI Onyx machine at Cambridge available to users across the UK for remote collaborative visualisation of Cosmology data sets. The visualisation serving software used to enable this is called Vizserver. The structure of CosmoGrid is shown in fig. 1.9.

The cosmological data are all stored locally to the server, which carries out all of the data processing. The client issues commands which are executed on the server side. The results can be viewed on any client machine.

The images produced by the visualisation process are in high resolution, and thus can be many megabytes in size. However the SuperJANET IV

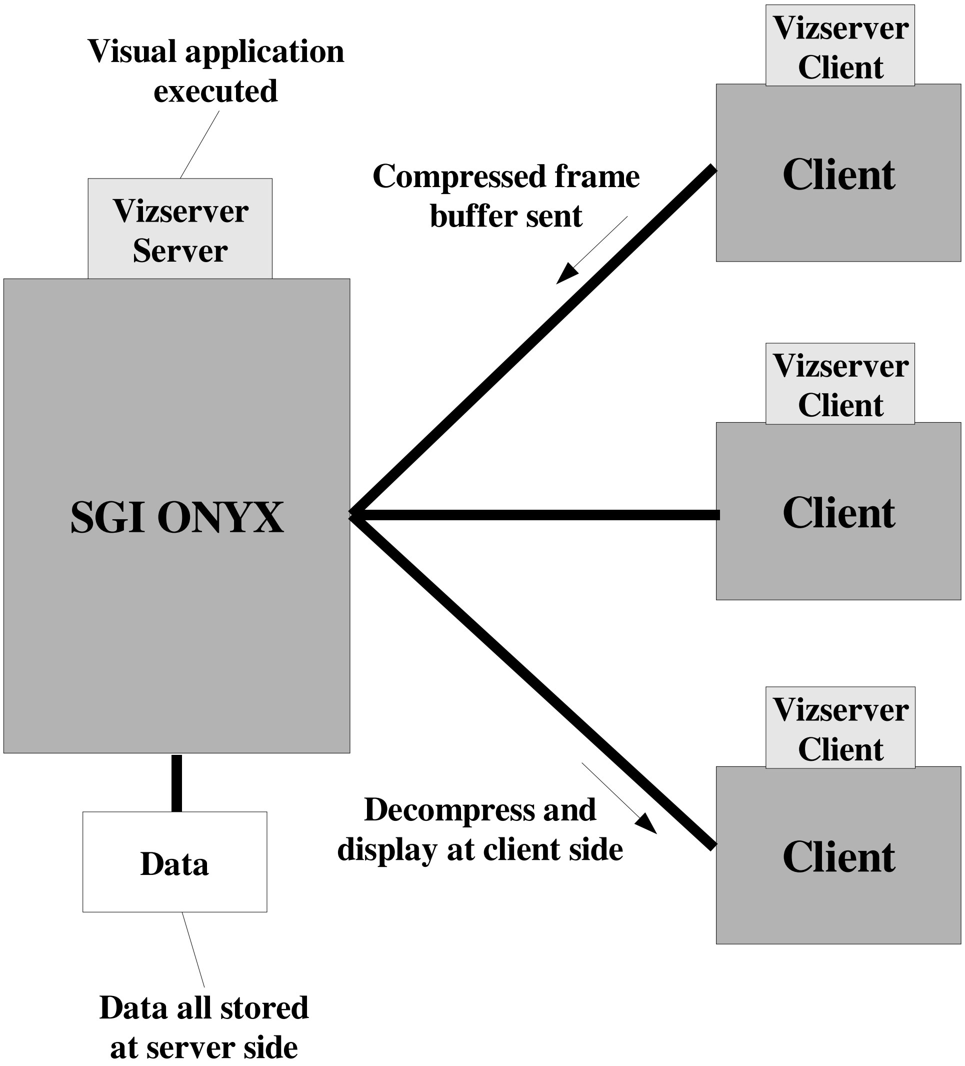

Figure 1.9: The Client-Server structure of CosmoGrid. Commands are issued from Vizserver clients and executed on the server side. The results can be visualised on any client machine.

Gigabit network has enabled successful tests over long distances, with visualisation carried out by clients as far away as Manchester and Durham.

1.8 Summary

In this chapter, the concept of a Grid, and its origin were explained. Its layered structure was described in terms of: the Fabric, which consists of computational, storage and other resources, as well as the local management software; Core Middleware, which governs communication protocols between sites, and security and authentication procedures; User Level Middleware, which is concerned with resource brokering at the site level, as well as the SDKs and APIs which allow users to interface with Grid middleware; and the Application layer, consisting of the applications that make use of this structure. Examples of middleware structures and Grid applications were detailed, with an emphasis on scientific uses of Grid technology.

CHAPTER 2

Data Grids for High Energy Physics

Expanding on the concept of data intensive Grid computing, defined in the previous chapter, the computing requirements of new High Energy Physics experiments are described here, taking the ATLAS experiment as an example. An example of the dataset size for a typical physics analysis channel is given, and the way in which such datasets and analyses can be handled with Grid technology is discussed. The configuration of resources being constructed on a Europe-wide scale is described, as are a variety of Data Grid projects around the world.

2.1 Requirements of a HEP Data Grid

It is the nature of collider-based high energy physics experiments that they are capable of generating many orders of magnitude more raw data than can be used in physics analysis. A typical collider experiment will accelerate bunches of charged particles (protons in the case of the LHC), and collide them in order to produce high energy interactions in a detector. These detectors must be as hermetic as possible in order to catch the cascades of particles produced by the particle interactions. They will be constructed from different sub-detectors which are arranged in layers radially outward from the beamline. For instance a detector is likely to have: a silicon vertex detector to reconstruct tracks and identify particles that decay very rapidly; drift chambers to find a particle's momentum; electromagnetic and hadronic calorimeters to determine particle energy; and muon chambers to identify muons which will typically not be stopped by the previous layers.

These components will all have their own readout channels, and will produce an output signal when they detect a particle. If tens of inelastic collisions occur in every bunch crossing, and there are millions of bunch crossings every second, a large flow of data will be produced, even if not all channels are constantly active. These data rate would quickly exceed the storage resources of the experiment, and thus it must be reduced in some way to a manageable level. This means that decisions must be made at run time to keep certain events and reject others, based on the physics that the experiment is investigating, triggering on signals from the products of interactions predicted by theory, as well as more generic signals such as particles with high transverse energy which indicate a hard inelastic collision. This must be done carefully, as there is a risk of rejecting the physics that one is searching for. A combination of hardware and software layers of triggering are used to reduce the flow of data, which will be appropriately buffered in order to minimise deadtime (i.e. time in which significant events could be missed because earlier events are still being processed).

ATLAS [17], along with the other LHC experiments, will produce an order of magnitude more data than any previous experiments. Bunch crossings will occur once every 25 nanoseconds, and running at a luminosity of 1×10^{34} cm^{-2}s^{-1} will produce on average 23 inelastic proton-proton collisions per crossing, with a centre of mass energy of 14 TeV. The signals produced by these interactions will be picked up by 148,119,000 readout channels, a figure dominated by the pixel components of the silicon detector. The pixels are 50x300 μm in size, as they are required to reconstruct hits with high spatial resolution.

The components will not constantly produce data, but will read out when they receive a signal. This occupancy will vary between different detector components. For instance, it will be a small fraction for the inner components (O(0.01) for the silicon pixels, O(0.1) for the silicon strips), but higher for outer tracking layers (up to 0.4), as fewer components cover larger areas. Even with this taken into consideration, the unprocessed output from the detector will be of the order of a petabyte per second.

The bunch crossing rate of 40 MHz is first reduced by the hardware of the Level 1 trigger. Event selection at this level reduces the data rate to roughly 75 kHz, or an estimated 160 GB per second of data. This is then passed to the High Level Trigger (HLT), which makes further cuts to leave O(100) Hz. Each event will be approximately 1.5 MB in size, which means that even after the triggering process around 13 TB of raw data will be produced in one day of running.

2.2 Physics at a Large Hadron Collider Experiment

The datasets eventually produced by ATLAS will be analysed by physicists to extract the information that is useful for a particular study. In the initial phase, studies will be conducted to place more precise constraints on QCD processes such as jet and heavy quark production, as well as photon physics and gauge boson production. The structure of the proton will be studied at higher energy than ever before, and the fractional momentum of the proton being carried by partons will be observed at very small scales ($< 10^{-5}$). These processes will form the background to searches for new physics, so they will have to be well understood.

Large numbers of $b\bar{b}$ pairs will be produced, which can be used for investigation of CP violation, and searches for rare decays. The top quark mass will be measured with 2 GeV precision, and single top production is expected to be seen, with high statistics allowing for observation of rare top decays. Many W production events will be seen, and an important goal will be to improve the precision of the W mass measurement, which has an effect on constraints of the Higgs mass. Investigations of quark compositeness will also be conducted.

If Supersymmetry exists at the electroweak scale, ATLAS will be able to find it, with squarks and gluinos with masses of up to 1 TeV being observable. From these observations it should be possible to identify the SUSY model underlying the events. Other exotic particles such as leptoquarks, monopoles and new gauge bosons can also be searched for. In many cases searches for rare events become possible because the volume of data generated at the

LHC will make it possible for them to be detected with sufficient statistical confidence.

One of the most important tasks for the LHC experiments will be to discover (or rule out) the Higgs boson, which will provide evidence for the Higgs mechanism which gives other particles the property of mass. The Higgs can be produced in a variety of high energy interactions, but the cross-sections of these interactions are small. The Higgs is also a very short lived particle and thus can only be observed indirectly via its decay products, which in many cases can be mimicked by other interactions. This means that many Higgs production channels are not viable, as the signal would be swamped by the background. The channels that are the most promising will also depend upon the mass of the Higgs, which has a lower constraint of 115 GeV from direct searches at LEP II, and an upper limit of 1000 GeV from theory. An example of such a Higgs channel is given below.

$\mathbf{H \rightarrow WW \rightarrow l\nu jj}$

In this channel, a Higgs particle produced in a qq $\rightarrow$ qqH interaction decays to a W^+W^- pair. (qq can be ud, $u\bar{u}$, $d\bar{d}$, etc.) One W decays into a lepton and a neutrino (producing a missing energy signal), and the other produces two jets of strongly interacting particles. There are three major backgrounds to these events: a W decaying to a lepton and a neutrino, with jet production; top pair production, containing real W $\rightarrow$ jj and W $\rightarrow$ $l\nu$ decays; and continuum production of W pairs which decay as above. Table 2.1 shows the number of events for Higgs production as well as the background channels for an integrated luminosity of 30 fb^{-1} and a Higgs mass of 1TeV [17]. This corresponds to an initial lower luminosity running period for the LHC of 3 years. The number of events before and after cuts to reduce background are shown.

Process	Events Before Cuts	Events After Cuts
W + jets	10,400,000	62
$t\bar{t}\rightarrow l\nu jjb\bar{b}$	2,250,000	85
$WW\rightarrow l\nu jj$ continuum	255,000	3
$H\rightarrow WW\rightarrow l\nu jj$ signal	486	73

Table 2.1: Comparison of no. of background and signal events for the $WW\rightarrow l\nu jj$ Higgs production channel at ATLAS, for an integrated luminosity of 30 fb^{-1} and a Higgs mass of 1 TeV, before and after cuts.

The cuts are made based upon factors such as the angle of particle trajectories in the detector, and energy thresholds for the deposits these particles leave in the calorimeters. In this case the cuts made are somewhat loose, so that the shape of a potential Higgs signal is not masked. As can be seen, these cuts should reduce the signal to background ratio from approximately 1:20,000 to 1:2.

While this might mean that a 1 TeV Higgs has a good chance of being detected by this method, the datasets involved will be very large. Assuming each event takes up 1.5 MB of storage space, this analysis will involve 19.4 TB of data. At least as much Monte Carlo data will be required in order to reduce statistical errors to an acceptable level.

In fact, even larger datasets would be required for a successful analysis in this channel in the case of a lighter Higgs. Table 2.2 shows signal and background for the same integrated luminosity as before, but with a Higgs mass of 600 GeV [17]. The Higgs can still be detected, albeit with a slightly smaller signal to background ratio of 1:3. However the total number of events involved is significantly larger - in this case the analysis will require an input dataset of 91.9 TB (again, with a Monte Carlo dataset of equivalent size).

Process	Events Before Cuts	Events After Cuts
W + jets	56,820,000	280
$t\bar{t} \rightarrow l\nu jjb\bar{b}$	4,440,000	45
$H \rightarrow WW \rightarrow l\nu jj$ signal	1,860	114

Table 2.2: Comparison of no. of background and signal events for the $WW \rightarrow l\nu jj$ Higgs production channel at ATLAS for a 600 GeV Higgs.

This is much larger than datasets produced in previous experiments. For example, the CDF experiment [18], currently running on the Tevatron collider in Fermilab, typically produces datasets of a few TB [19]. Volumes of data on such a scale require significant computational resources to process, but this can often be provided by the tape and disk storage and PC farms at a large facility such as Fermilab. However the datasets that will be produced by the LHC experiments are an order of magnitude larger, and will require a new computing model. The following section contrasts the current model with that of an appropriate computational Grid.

2.3 Data Analysis in High Energy Physics

At present, a typical experiment has the raw data stored centrally in tape archives, from which data can be staged to disk when required. When a certain data set is requested by an institute, possibly in an entirely different part of the world, the data are transferred to that remote site and the analysis is run locally. The institutes collaborating on the experiment operate more or less independently, so if a second institute in the same city required the same data set, it is quite likely that they would transfer it over again from the original site, as they would be unaware of a replica of that dataset in a physically much closer location.

However, this will no longer be possible on the scale of ATLAS and the other three LHC experiments. With tens of terabytes of data produced in one day of running by a single experiment, storage resources at any one site would soon be exhausted. The total output of data from the LHC is expected to be of the order of ten petabytes per year. Not only must these data be stored, but it must be made available to sites distributed around the world, and processing power must be available in order to analyse it. As well as this, large Monte Carlo jobs generate simulated data for testing purposes, a task which is even more CPU intensive. Analyses of datasets of the size of those described in the previous section would take a prohibitively long time to run using only the computational resources at a typical university, indicating that aggregation of resources at many geographically distributed sites will be desirable.

In order to compensate, computational Grid technology is being developed to improve the efficiency with which the resources available to the HEP community are used. When the LHC begins taking data in 2007, the task of the Grid will be to harness the computational power not just at CERN, but at member institutes around the world. This problem falls into the category of data-intensive computing, as described in the previous chapter. Some useful technologies have already been developed, while others have been designed specifically for scientific data handling.

There are certain properties that will be shared by any Grid set up for the analysis of HEP data, and arguably many types of scientific data analysis. The Grid must be able to accommodate large volumes of data as it is generated, and make it available to those who need access to it. The work to be done in processing these data, the individual jobs, will be large in number, but mostly independent of each other. That is, one job may depend on the results of a previous job, but the problem of complex programs composed of many processes all needing to communicate with each other over a wide area network will not apply here. This means that the HEP Grid scenario is similar in many ways to the high-throughput computing situation (see section 1.7.1), as well as the data-intensive one (section 1.7.3).

A system of data management must be created to keep track of all of the data in a HEP Grid, as not only must the data be registered when it is first made part of the Grid, but all of the copies, or replicas, of the data must also be registered, and identified with the original data from which it was copied. This description of data (or data about data) is called metadata, and a repository for this metadata will be needed, to answer queries about where a certain data set can be found, and most easily obtained from.

The resources available to users of the Grid can also be described using metadata, describing the specifications of storage and processors, as well as their current status in terms of available cache space or idle machines. This metadata, as well as that for the data, needs to be supplied effectively to those who need to use it. Therefore an information service is required to do this, and to keep the information as recent as possible, so that decisions are not being made based on old data.

Finally, and possibly most importantly, it must be decided how all of this information will be acted upon, and how decisions will be made on where submitted jobs will be sent. Should all users be allowed to allocate their own jobs, in an "every man for themselves" struggle for the best resources? Should there be a single resource scheduling entity that deals with all scheduling decisions? Or should the solution be somewhere in between, perhaps with a scheduler per institute, or per country? All of these possibilities will have their advantages and disadvantages, so it is up to those creating the Grid to make the best use of the available technology.

2.4 Regional Centres for HEP Computing

Consideration must also be given to the reality of how all of these resources will be coordinated. Users of the Grid may wish for experimental data to be distributed to the sites where they will be needed, but it will all start out at one point, for a given experiment. Thus the MONARC group working on computing for the LHC have devised the idea of Regional Centres [20].

The member sites of the Grid are categorised into Tiers, depending on their ability to supply data to the user. The intention is that the total resources at each Tier will be approximately the same. The figures quoted here are for ATLAS, and similar resources will be needed by the other LHC experiments. The Tier hierarchy is shown in fig. 2.1.

Tier 0

There is one Tier-0 site. This is CERN, where the data is acquired from the experiment, and initially stored. The first data reconstruction occurs here, and CERN shares the work of the Tier 1 sites. 520,000 CPUs will be sited here, along with 540 TB disk space, and 3 PB of tape.

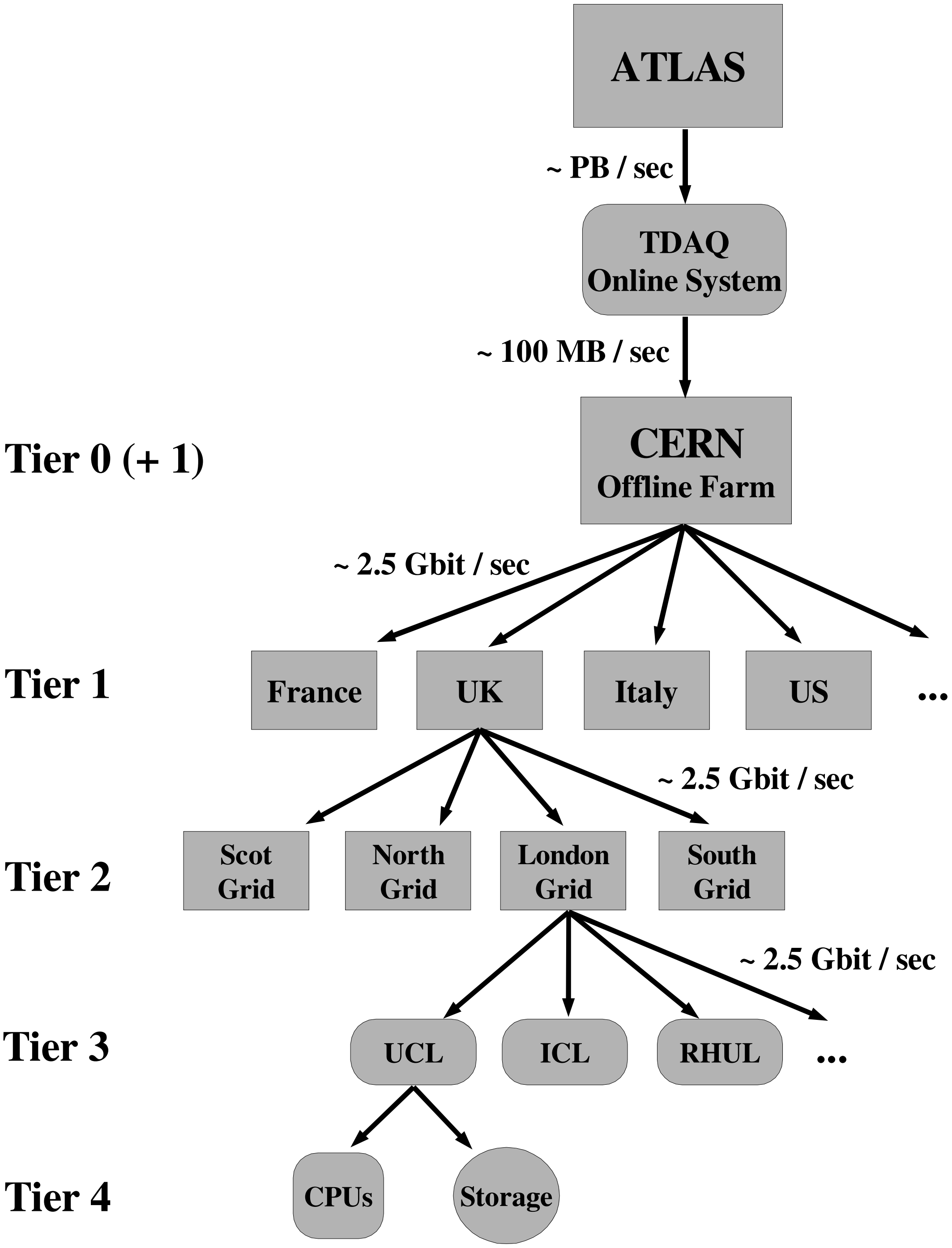

Figure 2.1: The hierarchy of Tiers, with typical network connections between the different layers.

Tier 1

Tier 1 Regional Centres will service a nation, or a group of nations (the UK's Tier 1 site will be at RAL). They are expected to replicate as much of the data stored at CERN as possible, in order to facilitate access to the data around the world. Their primary task will be to generate the following levels of the LHC data hierarchy: O(100 TB) of Event Summary Data (ESD), reconstructed from the raw data; O(10 TB) of Analysis Object Data (AOD) generated from the ESD; and O(1 TB) of Tag data providing event indexes. Each Tier 1 centre will have 10-20% of the resources of Tier 0.

Tier 2

Tier 2 centres will service single nations or regions, caching popular data in their local storage. They will concentrate more on data analysis. They will have 5-25% of a Tier 1 site's resources (or around 2% of those at CERN). ScotGrid, NorthGrid, SouthGrid and LondonGrid are the Tier 2 centres being set up for the UK.

Tier 3 / 4

The local computing resources at member institutions make up Tier 3 of the model, with Tier 4 consisting of individual machines.

In a sense this organisation of resources is separate from the development of Grid technology. It is more of an acknowledgement of the work that must be done to coordinate these resources, and make them available to become part of the Grid. Whatever form the applications and middleware that are eventually used will take, they will have to be mapped onto the real machines and storage facilities provided and used by the members of a HEP Grid.

2.5 Data Grid Projects

In the US, the Particle Physics Data Grid (PPDG) [21] is a DOE funded collaboration between high energy physicists and computer scientists to set up distributed data management services. iVDGL [22] is a similar project designed to build a data grid for physicists and astronomers. GriPhyN [23] is an NSF project that intends to set up a data grid infrastructure on the petabyte scale for the CMS and ATLAS experiments at CERN, the LIGO gravitational wave detector, and the Sloan Digital Sky Survey.

NEESgrid [24] is establishing a grid infrastructure for earthquake research, integrating data analysis and remote instrumentation. The Earth System Grid [25] is a data grid for climate modelling and other earth system simulations, also using collaborative technologies for exploration of these models. Similarly, the NASA Information Power Grid [26] is exploring data and modelling Grid technology.

The EU is funding several Grid projects. The European Data Grid project [27] is covered extensively in the next section. The LHC Computing Grid (LCG) [28] is being set up specifically to manage the data that will be produced by the Large Hadron Collider experiments, in part based upon the EDG's middleware. Other EU initiatives are: CrossGrid [29], investigating possible applications of Grid technology, and extending the Grid into other European states; GRIDSTART [30], which builds awareness of the possibilities of the Grid in the public and the commercial world; and Openlab [31], developing Grid middleware in collaboration with industry.

There are also initiatives which have been created to begin the process of uniting these separate Grids. The DOE Science Grid [32] is establishing a cross-disciplinary Grid for all US science research. The Astrophysical Virtual Observatory (AVO) [33] aims to establish distributed and interactive access to astronomical instrumentation, along with other partners in the AVO Alliance. DataTAG [34] are investigating the problems of building a large scale international Grid testbed, focusing on network issues, particularly in trans-Atlantic communication. The Global Grid Forum [35] is an international Grid community, merging US, European and Asian groups, formed to discuss distributed computing technologies and set international protocols and standards.

2.6 Summary

In this chapter, the requirements of HEP computing were outlined, in terms of the rates of data produced by the next generation of experiments. The ATLAS experiment at the LHC in CERN was used as an example, and the physics at this experiment was outlined. As an example of a typical dataset, a Higgs search channel was described, producing tens of Terabytes of data over three years of running. This was compared with the CDF experiment which is now running, and producing an order of magnitude less data. The current computing model was described, and compared with the Grid model. The Tier model of Regional Centres for HEP computing resources was explained, and examples of Data Grid projects were given.

This thesis is concerned with the middleware and structure of the European Data Grid. The EDG, its components, and how jobs are run within it are described in the following chapter.

Chapter 3

The European Data Grid and its Components

This chapter describes the structure of the European Data Grid, an example of a Data Grid for HEP computing as described in the previous chapter. It describes the EDG's middleware components, and how they interact with each other. It details the lifecycle of jobs in the current structure, and also in the more complex generation of middleware being developed to manage more complex jobs with input data requirements.

3.1 Structure of the European Data Grid

The European Data Grid [27] is an EU funded initiative to create a Grid of resources for computational and data-intensive scientific work. It has been created to aid not just the analysis of data from high energy physics experiments, but also that produced in biomedical research, and the European

Space Agency's Earth observation project. It is hoped that the EDG will be able to demonstrate the cross-discipline potential of Grid technology.

The EDG project is led by CERN, but with five other main partners (the European Space Agency, CNRS in France, INFN in Italy, NIKHEF in the Netherlands, and PPARC in the UK), and fifteen other associated partners from the Czech Republic, Finland, France, Germany, Hungary, Italy, the Netherlands, Spain, Sweden and the UK.

The not inconsiderable task of constructing the EDG is divided into twelve Work Packages, which make up four Working Groups: Testbed and Infrastructure; Applications; Computational and Grid Middleware; and Management and Dissemination.

The EDG structure is based on software components installed where appropriate at the member sites. Computing resources, which can be anything from a single machine to a farm of PCs, or larger multiprocessor machines, are managed by a Compute Element (CE). The storage space on disk and tape available for Grid use, containing permanently hosted files as well as temporary replicas, is managed by a Storage Element (SE). The data on the SEs are registered with a Replica Catalog (RC), which can supply information relating logical file names to the physical file names given to those data at various sites. Information about the CEs and SEs is regularly updated at the Resource Broker (RB), which uses that information, along with information about data file replicas, to pass a job, submitted using the User Interface software (UI), to the Job Submission Service (JSS).

The following sections will describe how how these components work, and how they interact with each other. A schematic diagram of these components and interactions is shown in fig. 3.1. This diagram shows the relationships between all of the middleware components, and all of the stages of communication between them as a job is submitted, scheduled and run. Simplified versions of this figure, showing the relationships that are significant to this simulation work are shown later in figs. 3.2 and 3.3.

3.2 The Middleware Components

This is the middleware structure as it stands for EDG v1.4. The EDG is implementing v2.1 at the time of writing, but the older version is used for the initial studies here in chapters 5 to 8, as this was the structure during the comparison tests between real and simulated Grid data described in these chapters. Section 3.3 describes job submission for v1.4. In section 3.4 the more complex submission process for jobs with input data dependencies is described, for the v2.1 middleware with its more sophisticated data management capabilities.

3.2.1 User Interface

The User Interface is the software used by the end user to submit his job to the Grid. This user must write a JDL (Job Description Language) file, which is in the form of a ClassAd (see section 3.2.2). This will include the details of the binary to be run, any input data files required, and instructions on how to deal with output, if any. It also contains the requirements of the job for the machine it will run on, and the user's preferences as to the properties of these machines. This file is submitted using the Condor-G [37] interface to the Grid, and this information, the binary, and any local configuration files are passed on to the Resource Broker.

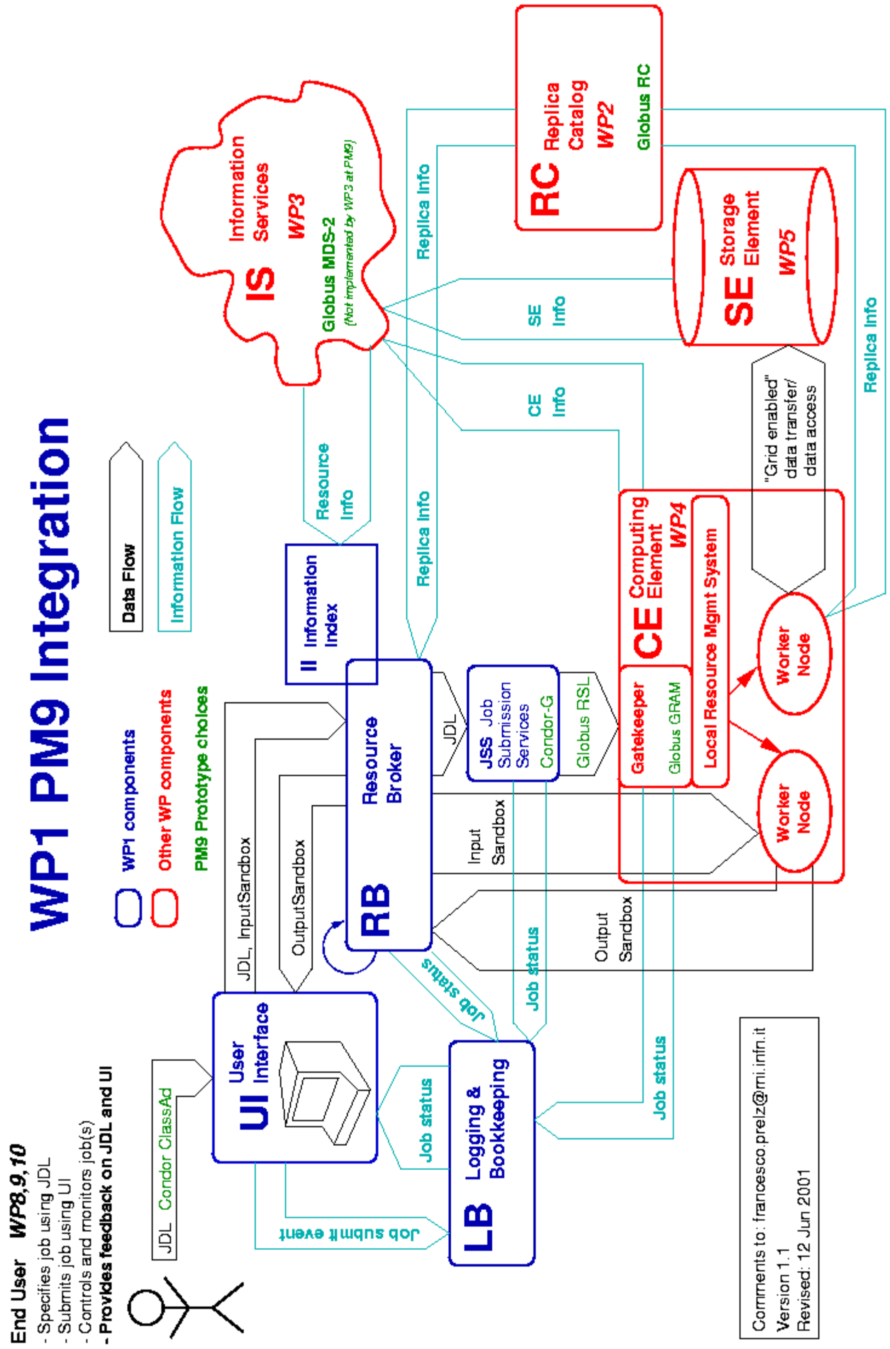

Figure 3.1: Schematic showing the relationship between middleware components of the EDG, and the information passed between them [36].

3.2.2 Resource Broker / Job Submission Service

The Resource Broker is the nerve centre of the Grid, with information from all of its components being passed to the RB so that it can make job scheduling decisions. The information sent to it by resources, and the user's JDL file, are in the form of ClassAds. These are compared by the Condor software, which attempts to find the most suitable resource to run the job on by matching the most suitable jobs to resources.

ClassAds contain a list of their attributes in the form:

```
Property = ''Value''
```

Followed by a description of the requirements and preferences for the properties of the ClassAd they are to be matched to, in the form:

```
Requirements = (other.Property1 > 5.0) &&
                      (other.Property2 == 1)
```

These ClassAd statements always evaluate to a value, in the case of the one above a boolean. However a quantity prefixed with "other" requires information from the ClassAd that this one is being matched with. If the statement evaluates to true in the context of the other ClassAd, then the two can be matched, otherwise it is not a valid combination.

In the job / resource matching case, the user will have supplied a list of the job's requirements, such as machine architecture and operating system. If the attributes of the system do not meet these requirements, that resource will be dropped from the list of candidates for the job. Similarly, the resource will supply a list of requirements of the jobs to be run on it, for instance specifying the maximum memory usage allowed.

The second phase of the matching is the ranking of ClassAds in order of preference. The user supplies a list of preferred attributes in a way similar to the Requirements statement, except that it evaluates to a number:

```
Rank = other.Property3 + (5 x other.Property4)
```

Evaluating the Rank statement in the context of the candidate resources will produce a different value for each, allowing the candidates to be put in order of preference. The job will be sent to the resource that is (in the user's eyes) the most favourable for that job.

If the user does not include a Rank statement in their JDL, the default will be used. For the EDG's RB, this is the minimum value of a quantity called Estimated Traversal Time. A job's queue traversal time is the time it takes to pass through the local queue at a CE and start running (i.e. this quantity will be zero if the CE has a free CPU when the job arrives). The Estimated Traversal Time for a job being scheduled is the traversal time for the last job to run at that site, normalised with the length of the queue at that time and the current time, as shown in equation 3.1 (and further explored in section 8).

$$\text{Estimated TT} = \text{Last TT x} \frac{\text{Current Queue Length}}{\text{Queue Length At Arrival}} \tag{3.1}$$

The resource ClassAds are brought to the RB by the Information Services, updated when the previous information's lifetime has expired. This information is stored in the RB's Information Index, which the RB will consult when it is sent a job to schedule.

If the job has data dependencies, the RC will be consulted to determine the location of these data (see section 3.2.3).

Once a job has been scheduled to run on a resource, it is passed to the Job Submission Service. The JSS handles the actual submission of the job to its chosen resource, and once it is there it is responsible for monitoring that job on behalf of the user. If for any reason the job is no longer able to run on that resource, the JSS can retrieve it and return it to the RB for rescheduling. If the resource fails and the job is lost, the JSS can resubmit it when the resource is functioning again.

When the job has completed, the output is returned to the RB and placed in a cache for the user to collect, unless the user has specified an SE on which to store the output.

3.2.3 Replica Catalog

Every data file on the Grid must be registered with the Replica Catalog. It is given a logical filename (LFN) representing the file itself, and physical filenames (PFNs) represent instances of that file on specific SEs.

When a particular data file is requested, it will usually be in terms of its LFN. A request is then sent to the RC which returns the PFNs of all of the existing instances of that file, and the SEs where they are stored.

3.2.4 Compute Element and Worker Nodes

The Compute Element acts as an interface and gatekeeper between the Grid at large, and a local system. It determines who should be allowed the use of the resources it governs, and what type of jobs should be allowed to run there. These user requirements and preferences are passed to the Resource Broker along with the details of the resource in a ClassAd.

Once a job has been accepted, it is passed to whatever queueing system is used at the site. However it is still monitored by the Job Submission Service at this point, and when the job has finished running, its results will be returned to the Resource Broker to be collected by the user.

The CE manages one or more Worker Nodes, which may be simple desktop PCs, or more powerful multiprocessor machines. The WNs have a minimum of Grid middleware installed on them, so that they can be dedicated to running jobs, with the CE handling interactions with other Grid entities.

3.2.5 Storage Element

The Storage Element controls disk and tape space available to Grid applications, and provides information about the data stored in it. Again, it acts as an interface, as an application requesting a file does not care how that file is physically stored, only that it gets the file. (The SE's storage space is distinct from space available at individual Worker Nodes for job running, which makes it possible for a site with a CE to function without an SE.)

An SE can have permanently hosted files, and cache space for temporary replicas of files hosted permanently elsewhere. Descriptions of these data, known as metadata, are passed to the Replica Catalog.

When a job needs an input file to run, it contacts the SE at which the file is located, and the file will be copied over to the WN on which the job is running. Data files can also be copied from one SE to another, if it would be useful for a replica of the data to be stored locally to a given CE. There is no provision for automatic replication of files, in order to optimise performance, in version 1.4 of the EDG middleware.

3.2.6 Information Services

The purpose of the Information Services is to make sure that information about resources (CEs and SEs) is made available as needed. Access to up to date information is important for effective scheduling of jobs.

The IS is built on the Globus Metadata Directory Service (MDS). This consists of a hierarchy of Lightweight Directory Access Protocol (LDAP) servers, which store information about the Grid entities below them in the hierarchy, and can be queried in order to obtain it. This is the status information for the CEs and SEs in the Grid, and it is expressed in the Condor ClassAd schema described in section 3.2.2.

Each member site of the Grid with one or more CEs or SEs will have a Grid Resource Information Server (GRIS), which stores status information about the local Grid entities, and sends it upward in the IS hierarchy. This information is timestamped, and has a lifetime associated with it. Once this expires, a status update should be supplied.

A Grid could potentially be large with many member sites, so the status information from the GRIS at several of these sites can be grouped together under a Grid Information Index Server (GIIS), which will collect updates from the sites below it in the LDAP tree structure, and pass them in turn to a GIIS above them. At the top of the hierarchy is the Information Index belonging to the RB, which will use the information to make job scheduling decisions.

3.2.7 Logging and Bookkeeping

The Logging and Bookkeeping services have the task of recording the activity of the Grid as it runs jobs. The LB server will normally be located at the same site as the RB. It consists of an SQL database, which stores events in the lifetime of Grid jobs, such as submission, arrival at a CE or completion, along with a timestamp.

Since this job running data are stored in the LB, this is where user queries are sent when they execute a UI command to give their job's status. The information returned is the most recent job event stored in the LB.

As well as this active role in Grid running, the LB will continue to store job information after a job's completion, and the return of results to its owner. These stored data can be used for Grid monitoring and troubleshooting.

3.3 Running Jobs on the EDG

Typically a job submitted to the EDG as described above will have no input data dependency, as the necessary data management middleware is not yet in place. If a user wishes to use data stored at an SE, they must either request a copy of these data to be transferred to a suitable location beforehand, or have their job run on a CE local to the SE with the data. Thus even in these cases, the input data are a consideration before job submission, rather than during the process of job scheduling. The job's lifecycle is shown in fig. 3.2.

The job is submitted from a UI and sent to the RB, which makes a scheduling decision based upon the job's stated requirements, and any preferences expressed by the user, as well as status information sent to its II from the resources via the Information Service. The job is then delegated to the JSS, which submits the job to the chosen CE and monitors it as it runs. Once it has completed, its output is returned to the RB, to be collected by the user.

If the job produces any sizeable output, it will be impractical (and antiso-

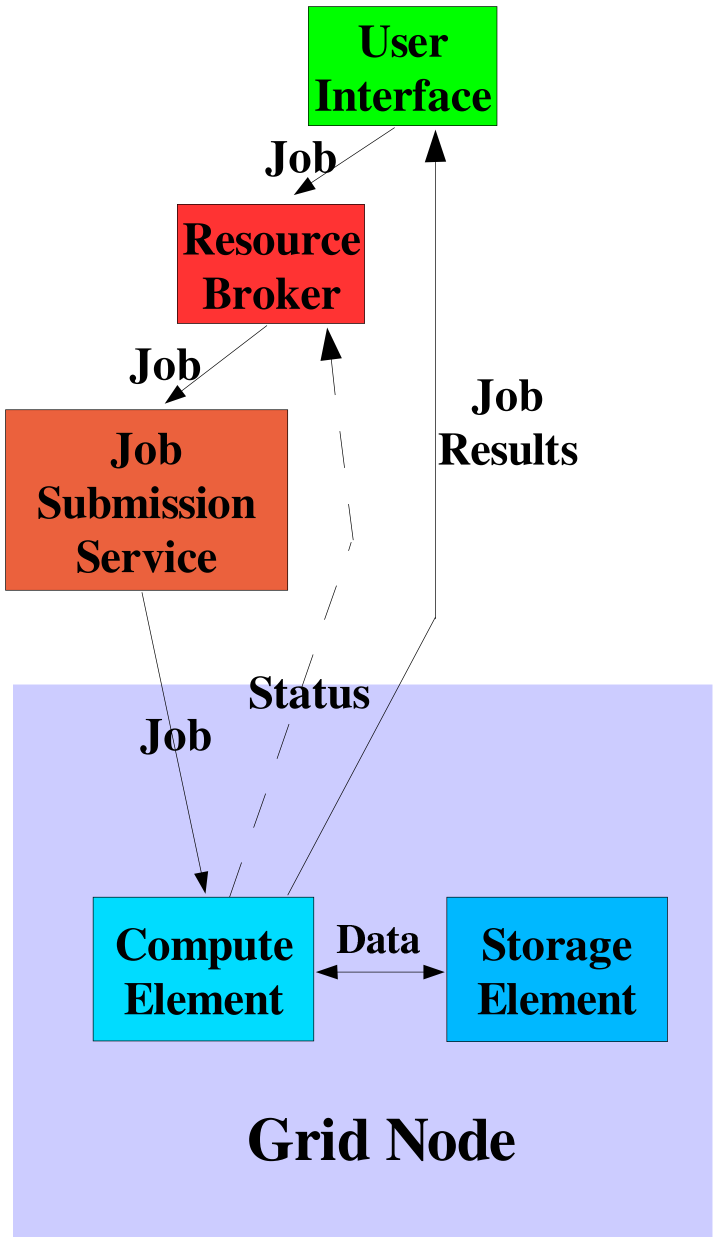

Figure 3.2: The progression of a job through the middleware components of the EDG, as well as other information passed between those components in order to facilitate job running.

cial) to leave it in the RB's cache. In this case the user can specify an SE at which the output data can be stored. However, the limited data management capabilities of the middleware can cause problems here too, as the job will be constrained to run on a CE local to the chosen SE.

3.4 Data Dependent Jobs

The middleware for the next generation of the EDG (version 2.1) will allow for jobs requiring input data. The Resource Broker will now have the task of factoring this requirement into its scheduling decisions, by consulting the Replica Catalog. In addition, the information delivered to the RB via the IS will be more sophisticated, under the new Relational Grid Monitoring Architecture (R-GMA) structure [38]. This will include network status updates between sites, supplied by Network Monitors which carry out tests of the connections between sites. The slightly more complex chain of events is shown in fig. 3.3.

In this case, a job sent to the RB will also include a list of logical filenames (see section 3.2.3), which it passes to the RC to be resolved into physical filenames resident on specific SEs. These sets of PFNs will allow the RB to identify CEs local to SE storing the required files. This information can then be factored into the scheduling decision for the job.

Once the job has been submitted to a CE via the JSS, it can begin running on any data that are already present on the local SE, copying it to the local disk of the worker machine. Any other data must now be copied from remote sites. This can be done preemptively, so that the data is ready for the job to run over when it is done with the previous file.

If a file is copied from a remote site, it is also available for storage on

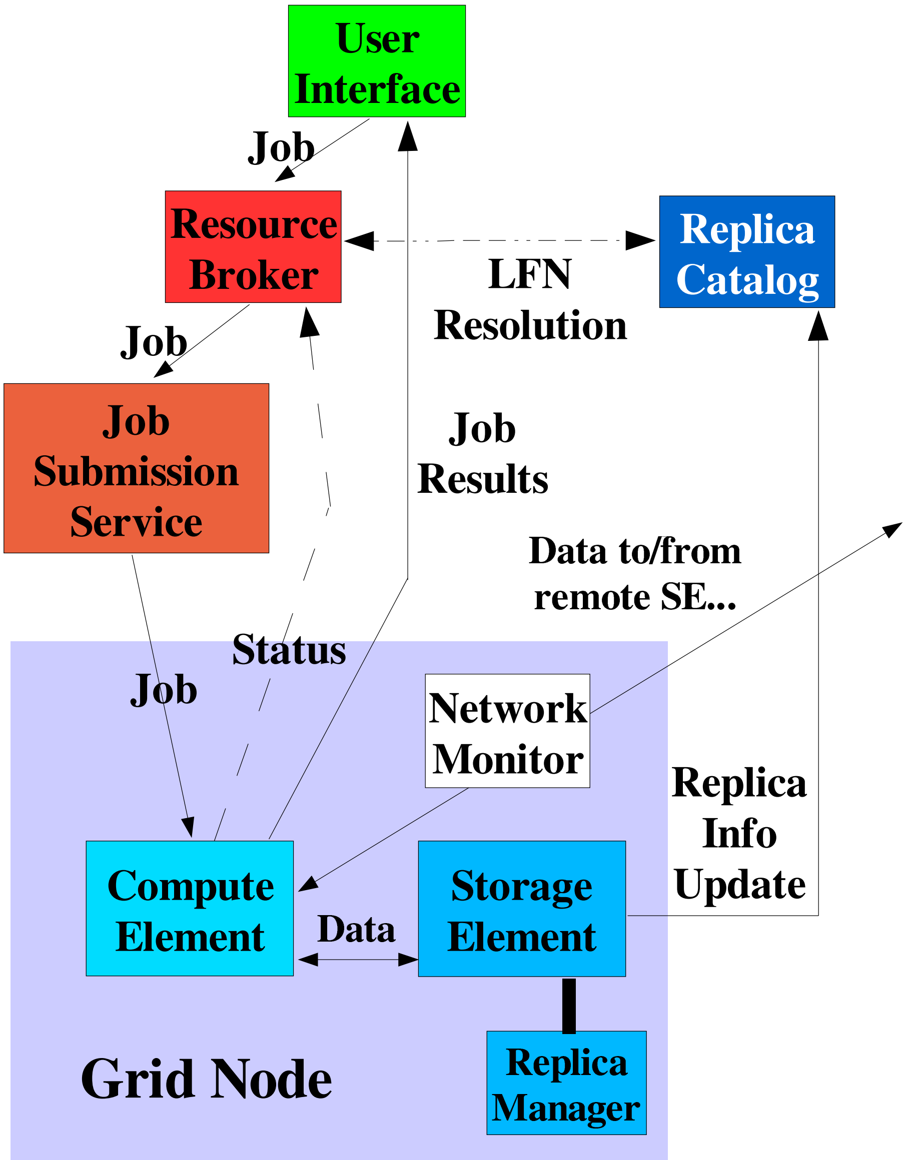

Figure 3.3: The progression of a job with data dependencies through the middleware components of the EDG, as well as other information passed between those components in order to facilitate the necessary movement of data and job running.

the local SE. The decision whether or not to keep the file will be made by a Replica Manager, which monitors usage of the SE disk cache, and makes its decision based upon data access patterns.

3.5 Summary

A potential crisis in computing for high energy physics has been averted with the application of Grid technology. The problem is not that the necessary resources are not available, but that previously they have not been used as efficiently as they might. However, advances in communication technology have now made distributed computing and resource sharing a practical solution.

These techniques are promising, but they are still being developed, and will require further work before they can be applied to service a running particle physics experiment. Part of the difficulty in this work is due to a lack of information about how these systems will behave on the scale necessary for such a task. The patterns of usage by the physicists working on the experiment are also uncertain, and can only be based on extrapolations from current experiments which are an order of magnitude less powerful than the LHC. A HEP Grid will have to be robust and reliable, but also flexible enough to cope with the demands that will be placed upon it.

In this chapter, an attempt to construct such a Grid, the European Data Grid, was described. The middleware components that make up its structure were detailed, as well as their interactions as they process the job load. The lifecycles of the simple jobs running in the current (v1.4) structure, as well as those with data dependencies that will run in the next generation of Grids, were described.

In order to study the current EDG's functionality, how it can be improved upon, and also how it can be developed for future eventualities, a simulation has been constructed to model the communication of its middleware components, and the movement of jobs as they are processed. This simulation is described in the following chapter.

Chapter 4

Simulating the European Data Grid

This chapter explains the benefits of simulating a Grid to investigate its running efficiencies, and gives examples of existing simulation tools. It then describes the rationale behind the methods chosen in simulating the European Data grid with EDGSim, details the simulation of the middleware components, and explains how a simulated test run is conducted.

4.1 Data Grid Simulations

It is clear that a Grid setup would be very beneficial for any organisation needing to get the maximum usage possible out of its computing resources. However, exactly how to do this is not quite so clear. A great many factors influence how well a system as large and complex as a Grid will perform, and it is a difficult task even to determine how best to measure such a system.

A software simulation saves valuable time in determining the optimal parameters and configuration for a Grid, answering important questions without resorting to trial and error with a real testbed. By the nature of such a system, it can only be tested with a network of well (even globally) distributed sites with a realistically heavy workload to manage, and this obviously takes a great deal of manpower, cooperation, and time. If this network can be simulated to a sufficient degree of complexity, some of this effort can be saved by testing ideas out on the simulation instead.

For instance, the choice of scheduling policy can have a dramatic effect on the efficiency with which the Grid processes jobs. The choice of algorithm will affect not just the running of the submitted job, but also that of the system as a whole. The Condor ClassAd matching mechanism used by the EDG Resource Broker (see section 3.2.2) provides a framework for job scheduling, but no overall policy, as users can choose how their job is scheduled by selecting their favoured resource attributes. In a busy production Grid it may become necessary to regulate the job load in a more formal way. A simulation provides an opportunity to test different resource selection policies, and measure the effect that they have on system and job performance.

The topology of the Grid itself will alter the way in which jobs are handled. The distribution of computational and storage resources will have an effect on system performance, whether they are spread evenly across the member sites, or grouped into larger facilities. Network capacity must be considered when moving large quantities of data.

The functionality of the middleware can also be investigated, to see where improvements might be made. For instance, the importance of the regularity of Information Services updates, or the mechanism for replication of data can be examined using a simulation, avoiding the need for extensive alteration of real life middleware.

Several Grid simulation tools have been created by other projects. The MONARC group at CERN [39] developed a package for the simulation of distributed computing for LHC experiments. GridSim [40] simulates resource brokering in the style of Nimrod-G, with deadline and budget constrained scheduling. MetaSimGrid [41] has been constructed with the SimGrid toolkit for building simulations of distributed computing systems, in which CPUs, networks etc. are all represented as generic resources, and the user decides the manner in which these resources are occupied by tasks. Bricks [42] simulates Grid systems using a client-server model, concentrating on system performance. MicroGrid [43] specifically simulates Globus-based Grids, allowing real applications to be run over a virtual testbed. ChicSim [44] is more specifically designed for the simulation of data Grids, although it is still under development. OptorSim [45] simulates an EDG middleware setup, but has been constructed with more of an emphasis on replica management and optimisation, with recent work adding job scheduling to the model.

For the simulation work described in this thesis, the Ptolemy II modelling package [46] was chosen. The existing projects were either too specialised for a particular Grid model, or were not yet developed enough to be of use. Ptolemy II is a design and modelling tool which was not created specifically with Grid simulation in mind, but its discrete event model handles concurrent processes and events in a way that is well suited for this purpose. The following section describes how a simulation is constructed using this toolkit.

4.2 The Building Blocks of Ptolemy II

In Ptolemy II (PTII) actions are carried out by Java objects called Actors, which can either be abstract entities to do arithmetic, or generate signals, and so on, or they can represent more solid entities such as the Actors designed for the scheduling simulation described later. These Actors communicate by exchanging Tokens, which are wrapper objects containing information that can be as simple as a double precision number, or a Java object in its own right. A simple schematic of an Actor is shown in fig. 4.1.

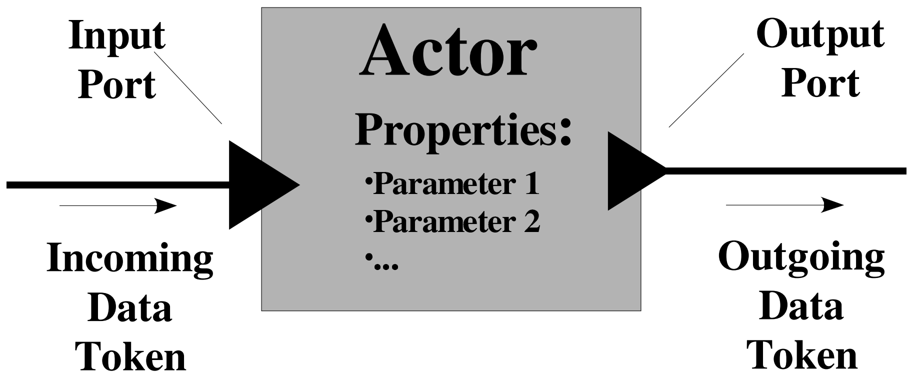

Figure 4.1: Schematic diagram of a Ptolemy II Actor, and its interactions with other Actors via data Tokens.

Any time an Actor carries out an action, or sends or receives a Token, it is considered an event. The events are placed in chronological order in the event queue, and then processed one by one in that order. The state of the simulation only needs to be updated at these event times, minimising the running duration of the simulation. Many events will result in another event at some point in the future; this future time is calculated, and an event is placed in the event queue, requesting that the appropriate Actor or Actors are woken up at that time to carry out the necessary action. Events cannot be removed from the queue, but if a queued event is rendered obsolete by other preceding events, the Actor can determine that it has been unnecessarily activated, and remain dormant.

The Actors are connected together in the GUI (called Vergil). Each Actor has one or more ports, which can send Tokens, or receive them, or both. Restrictions can be placed on the type of Token that the port can handle. Ports can be set up as single ports, connected to one other Actor, or as multiports, able to connect to many Actors.

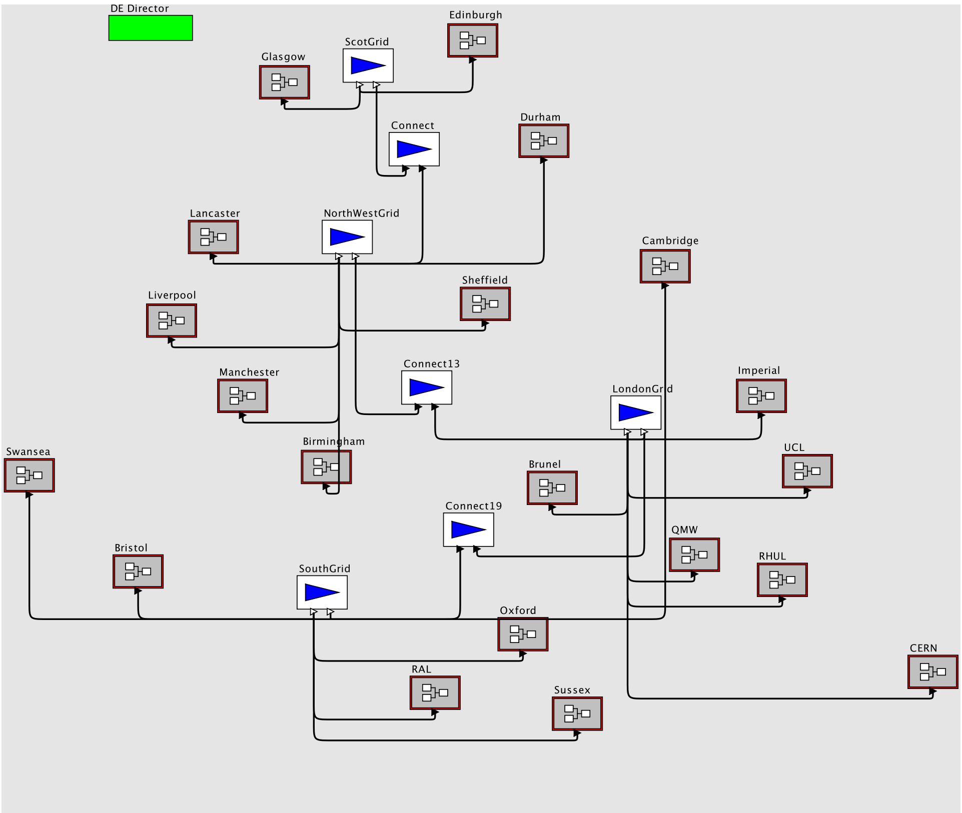

Figure 4.2: View using the Ptolemy II GUI (Vergil) of a simulated Grid. Composite objects representing member sites, objects governing network properties, and the connections between them are shown.

The GUI representation of a simulated Grid built using Ptolemy II is shown in fig. 4.2. The member sites (labelled by the names of the sites that they represent) are represented by composite Actor objects. The individual Actors shown here (blue triangles enclosed by a white box) govern the network properties of the Grid, passing simple status messages, or larger data transfers requiring significant network bandwidth and time. Connecting lines represent network connections, and also the allowed paths for communication in simulation terms. In this figure the member sites of the Grid are connected to Router objects, and the Routers are linked to each other by Actors representing the properties of their network connection. The Director object (in green) controls global properties, and the running of the simulation.

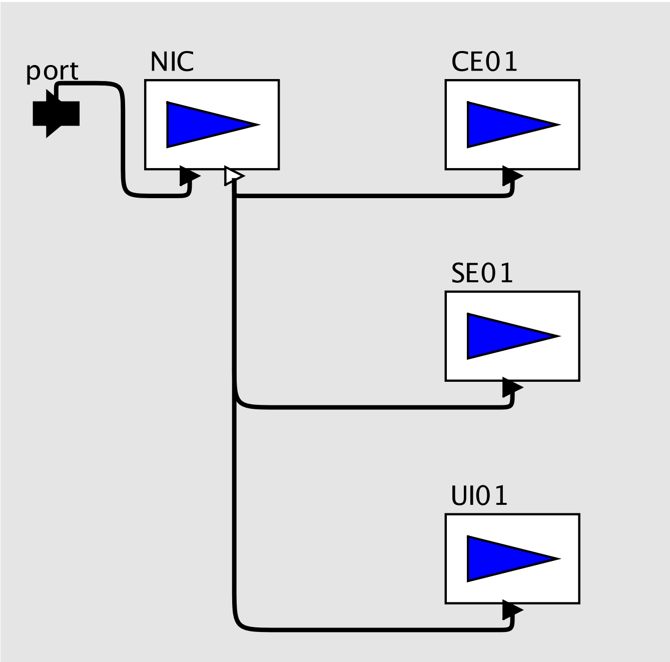

Figure 4.3: Vergil representation of a member site of the simulated Grid, showing the middleware components present there.

Fig. 4.3 shows the contents of one of the composite Actors representing a Grid site. Actors representing Compute (CE01) and Storage (SE01) Elements, and a User Interface (UI01), can be seen here, as well as a NIC Actor which governs outward network connectivity through a port object. The ports belonging to the Actors themselves can be black or white in colour. The former indicates a port that can connect to one other Actor only; the latter can connect to multiple other entities.

4.3 Architecture of EDGSim

Ptolemy II has a large selection of Actors, designed to be generic and usable for a wide range of purposes. These supplied Actors are simple in function on the whole, intended to perform a small simple task in combination with many other Actors, and communicating by passing simple messages to each other as mentioned above.

This simulation, called EDGSim, is intended to represent the interaction of the various entities in an EDG setup. PTII presents two possibilities for this representation: composite Actors, which would be made up of the simplistic prewritten Actors in a specific configuration (with these composites then being combined), or new customised Actors.

The composite Actor route would save time in writing new code, but configuring them would be a sizeable task in itself. Reconfiguring them for a new experiment with the simulation might involve repeating much of this work. Passing a message between these composite Actors would also be difficult, as the supplied Actors are designed to be as general as possible, and thus can only pass simple message types (integers, strings etc.) that can be handled by another generic Actor type.

Writing new customised Actors solves these problems, as a single Actor representing an entity would be easier to configure, and they can be designed to pass objects of the desired complexity. In this case even a simple number being passed would also have to have information describing its destination attached, equivalent to a URL to guide the message to the correct place. For these reasons, it was decided that new Actors would be written. The following sections describe how the functionality of real Grid middleware, and the network communication between them, were simulated.

4.3.1 User Interface

The User Interface is the Actor that generates the jobs for the simulated Grid to process. These Job objects have a certain memory requirement, need to run for a certain number of CPU cycles, and can have a requirement for one or more input data files stored at the SEs of the Grid. Jobs can be generated singly or in batches, and are then sent to the Resource Broker to be scheduled.

The UI can be modified to change the size of the jobs generated, in terms of how much memory they require and how many CPU cycles they will need to run, or the distribution of submissions in time. The size of the dataset can also be specified, and the dataset will be chosen from the files distributed between the simulated Storage Elements according to a specified access pattern (as discussed later in section 9.1.1).

4.3.2 Resource Broker

Job scheduling is carried out using a simpler regime than the ClassAd matching used by the real RB (see section 3.2.2). The RB is configured with a particular scheduling algorithm, and all jobs in a given run are assigned to resources according to this policy. Job scheduling will still take the job's requirements and the current status of the Grid's resources into account.

If the job has any data input requirements, the RB takes the requested logical file names and sends them to the Replica Catalog. The job will wait at the RB until the resolved PFNs arrive back from the RC. At this point the RB will consider the suitability of the available resources, based upon the most recent status updates, and then choose one for the job according to the scheduling policy. A schematic representation of the communication between Actors during job scheduling is shown in fig 4.4.

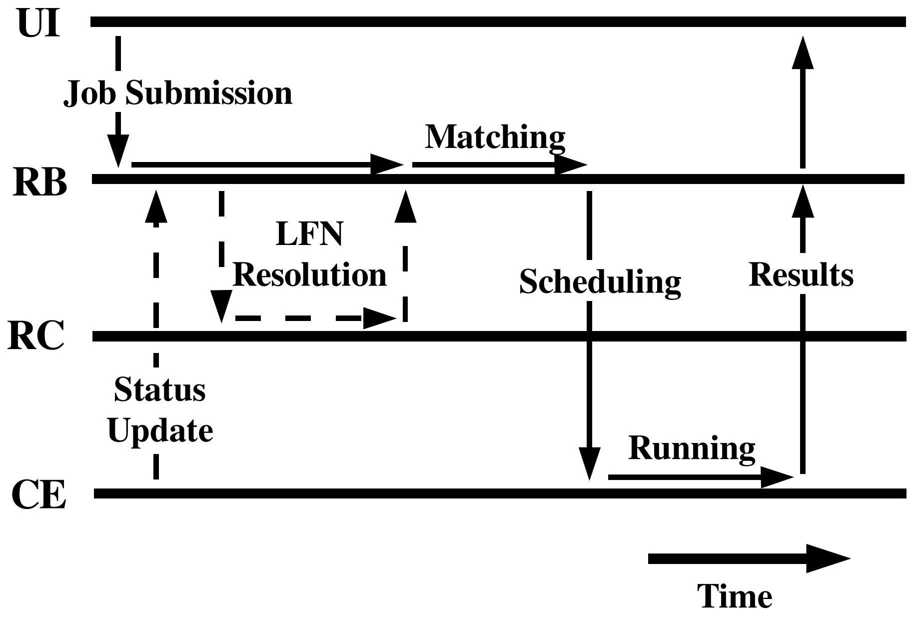

Figure 4.4: A schematic of the flow of communication between Actors during job scheduling in EDGSim. The movement of the job can be seen, as well as the information supplied to the RB to aid in scheduling decisions.

The job is sent to the chosen resource, along with the list of most accessible physical versions of required input data. The RB acts as the JSS as well in the simulation for simplicity's sake. As the JSS must run at the same site in a real EDG setup, this is a reasonable simplification.

4.3.3 Replica Catalog

As the name suggests, the Replica Catalog stores information about the data files on the Grid, and any copies of them that have been made. It stores information about permanent and replica files, and can resolve logical file names into physical file names. The LFN of a file is the name of the original permanent file. When a replica is made, it is given a new PFN, which in the simulation consists of the LFN prefixed with the name of the SE it is cached on.

When the RB requests LFN resolution, it returns the appropriate list or lists of PFNs. The RC has no decision making or data management responsibilities, as these functions are controlled at site level.

4.3.4 Compute Element

The Compute Element Actor represents the middleware that mediates between the Grid and the local software and hardware, the queueing software that manages the local processing power, and the CPUs themselves. When jobs arrive, sent by the Resource Broker Actor, they are assigned to a CPU if one is available, or treated on a first come, first served basis by the queueing policy, assigned to a CPU when one becomes free in the order of the jobs' arrival.

The simulated machines of the CE all have the same CPU speed, and run a maximum of one job at a time. When the job is assigned to a machine, it requests any required data that are not already present on the local SE. It can begin running if any of these data are present, and if not, the job will wait until it arrives. Data can also be copied preemptively, i.e. data that will be required later by the job can be copied over while the job is running over the first file. If the job has no input data requirements it can begin running immediately.

When a job begins running, the time at which it will be completed, or will finish reading the current data for a data dependent job, is calculated based on the CPU's speed, and the number of CPU cycles required by the job. A request for the CE Actor to be woken up at this time is placed in the event queue. When the specified time elapses, a data dependent job will start running over the next data file, or wait for more data to arrive if it was not possible to transfer some requested files preemptively. Data independent jobs (or dependent jobs that have run over all requested data) are now deemed to have completed, and timing data relating to the job's lifetime are returned to the RB Actor. Output data are not considered, as its size is assumed to be negligible in comparison to the large input data files for the jobs simulated here.

CEs will also periodically send status information to the RB. The hierarchy of the simulated Information Services is the simplest possible, with all sites sending information directly to the RB Actor.

4.3.5 Storage Element

Storage of simulated data files is managed by the Storage Element Actor. It can have a set of permanently hosted files, and a cache in which replicas can be stored and requested by a CE.

When a job requires data that are not already present on the local system, a request for the most accessible replica will be sent to the appropriate SE, based upon network status data from tests periodically carried out between different sites. The transfer of files between sites is handled by the Network Control Actor (see section 4.3.6 below). SEs can also decide to store a new replica that has been transferred to a local CE, or preemptively replicate a file. If the cache is full, a replica may be removed to create space according to the SE's management policy. Such mechanisms will be discussed later in chapter 9.

If the cache is full, a replica may be removed to create space according to the SE's management policy. If none can be deleted, because the files are in use or reserved for future use, the request will be queued until disk space can be found. Once a transfer to an SE has completed, the new replica will be registered with the RC (as will the removal of replicas).

4.3.6 The Network Model

There are several specialised Actors which control the simulation of the Grid's wide area network. It is a simplified model, with no representation of packet-level behaviour in data transfers. Transfer of data is controlled by Network Control Actor, using information supplied by the Actors that make up the path of the transfer. Other messages, such as jobs being submitted or status updates, are considered to be negligible in size compared to data files, and are passed either instantaneously or with a small fixed delay. All communication between Actors is sent in a customised wrapper object called a Message Token. These can contain any kind of Java object, and three additional String objects. These represent the type of message being passed, the name of the Actor the message is being sent to, and the node at which the Actor can be found.

At the site level, all local entities are connected to a NIC Actor, so called because it acts as a network interface card, coordinating communication with remote Grid entities. It registers the local Actors, and can pass incoming messages to the appropriate place. It also knows about the maximum bitrate available in its connection to the WAN, as well as the ongoing data transfers passing in and out of the site.

The NICs, and therefore the local sites, are all connected to a Router Actor. In a similar manner to the registration of local Actors with their NICs, all NICs register with their connected Router. The Routers also register with any other Routers that they are connected to via a Connect Actor (see below). When the simulation starts, the Routers read in a lookup text file, which tells them which Router to forward a message to if the destination Actor is not found at a site registered with that Router. In this way, a message will be forwarded on until it reaches the correct Actor.

Routers are connected to each other via a Connect Actor. These are configured with a maximum bitrate for their particular leg of the WAN. They also keep track of the file transfers passing through them in both directions.

Data transfers are coordinated as shown in fig. 4.5. When an SE or CE requires a file, it sends a request for that file to the file's home SE (these will be referred to as the destination SE and the sending SE respectively, for simplicity's sake). If the requested file is not available (i.e. replica information from the RC is now out of date), the destination SE will request the file from another source.

If the file is found, the sending SE sends a confirmation message back. This passes through all of the NICs and Connects which the file transfer will require. As it does so, the bandwidth that will be available to the transfer can be determined. The bitrate available to each transfer through a NIC or Con-

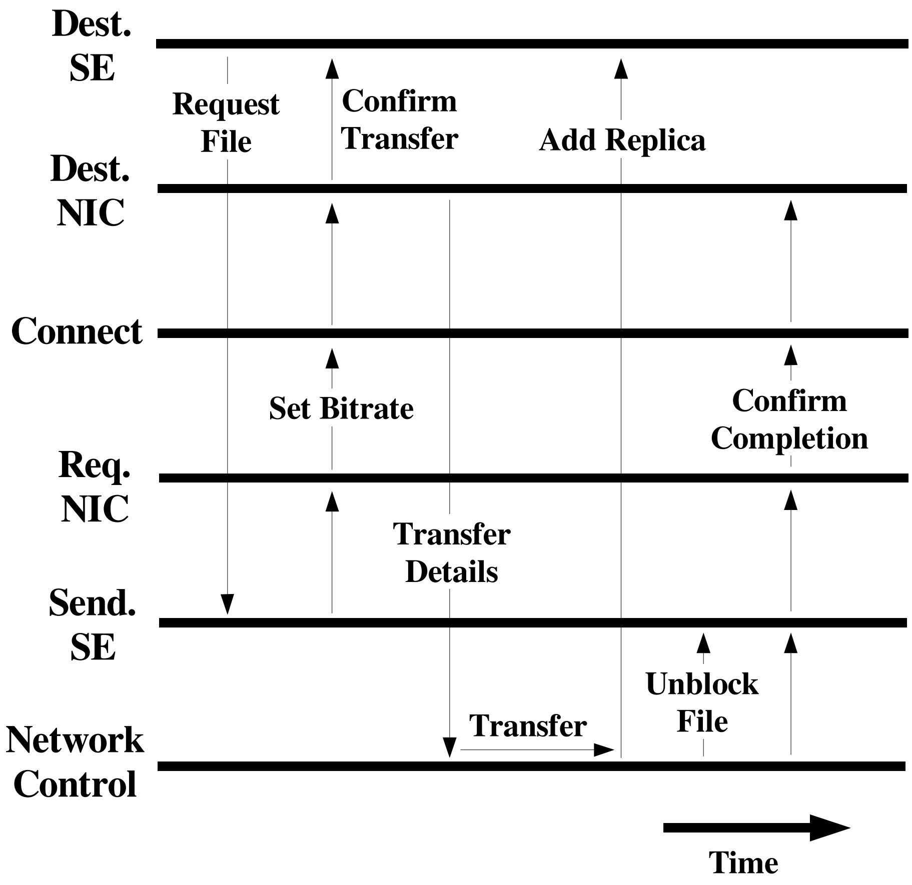

Figure 4.5: A schematic of the flow of communication between Actors during data transfer in EDGSim.

nect is the total bitrate divided by the number of transfers. The bandwidth of the transfer is constrained to be the smallest of the bitrates available. The commencement of the transfer is then registered with all concerned entities, with the file at the sending SE being locked for the duration of the transfer, and an appropriate amount of cache space being reserved by the destination SE. The transfer details are also passed to the Network Control Actor.

The transfer times are managed by Network Control. When a new transfer starts, it has the task of recalculating the timings of existing transfers, as the extra flow of data through network connections restricts the available bandwidth. Completion times are placed in the event queue in much the same way as the job completion times described in section 4.3.2. When a transfer has completed, notification is sent to all Actors involved, and bandwidths are again recalculated. The destination SE adds the new replica to its cache, and the file on the sending SE is unblocked.

This method of updating transfer times means that the network model is a reasonably realistic one, with busy periods resulting in longer transfer times. However, one simplification made here is that 100% of bandwidth is shared out between transfers, whereas in real TCP/IP this would not normally be the case. This means that the performance of the network modelled here is a slightly optimistic one.

4.3.7 Running EDGSim

All Actors require configuration in order to run the simulation. The User Interface and Resource Broker Actors generate random numbers for some operations (selection of input data, choice of CE, etc.), so these Actors read in seeds for their random number generators from input files. SE Actors can be given their initial store of permanent files in a similar manner. The Router lookup file was discussed in the previous section.

Some configuration information can be added using Ptolemy II's GUI, Vergil. The variable parameters of an Actor, such as its name, the number and speed of CPUs at a CE, total cache space at an SE, etc. can be entered in a dialogue box. The GUI is also the fastest way of assembling a simulated Grid testbed, configuring the resources and components at the member sites, and establishing the network connections between them. Ptolemy uses this information to generate an XML file describing the Grid, which is straightforward to parse if the setup needs modifying later.

When the simulation begins running, initialisation messages are sent between Actors. This allows them to register so that NICs know which other Actors are found locally, and Routers know which NICs are attached to them. All files are registered with the Replica Catalog, and the logical file names of all registered files are sent to the User Interfaces so that they can generate jobs with input data dependencies.

Once a job has been generated, assigned, queued, given its input data, run, and completed, it produces results. These are sent back to the Resource Broker, which writes them out to a plain text file, with each line corresponding to a job. It writes its own status information to another logging file at regular intervals, and the Compute Elements also write logging files recording their job processing during the runs. Network Control records network usage in another file. These text files are read into ROOT [47], a C++ based analysis package, in order to produce the results shown in the following chapters.

The running time for a Ptolemy II application running in discrete event mode varies greatly, as it is dependent on the number of events that occur during the run. In order to run for long enough to produce a set of results that are not dominated by an initial period of instability (usually 1000 or more jobs), a simulated EDGSim run with around twenty member sites will take roughly an hour to run on a 1 GHz PC.

4.4 Summary

In this chapter, the motivation for simulation of Grids is outlined, and examples of other simulation tools are given. The structure of EDGSim is then explained in detail, including the Discrete Event model in which job running, data transfers etc. are broken down into a series of chronologically ordered events. The mapping of middleware functionality and communication onto Java objects in the Ptolemy II modelling framework is described, as is the manner in which test runs are conducted in EDGSim.

In order to calibrate and validate this simulated model, its results can be compared with those of jobs running on a real EDG testbed. Such a testbed has been constructed between academic sites in the UK by the GridPP collaboration. Logging data relating to the running of jobs can be extracted from the GridPP's Logging and Bookkeeping broker, and the running of individual computational resources can be monitored by querying the GRIS at each site, which serves status information to the Resource Broker. These data mining techniques are described in the next chapter. Comparisons can then be made between job submission in the real and simulated Grids, and this will be explored in the chapters following that.

Chapter 5

Monitoring Grid Resources

This chapter describes the two main methods employed to extract job and resource data from the real Grid. In the first, the Logging and Bookkeeping database is queried in order to reconstruct the lifecycle of jobs, from submission to retrieval of results. In the second, metadata relating to resource status is obtained by querying the Grid Resource Information Server at each site, with a picture of resource use over time built up by regular queries. The strengths and weaknesses of both methods are discussed, as is the useful information that can be obtained using these methods.

5.1 Monitoring with the Logging and Bookkeeping Services

All of the events in the lifetime of a job as it passes through the Grid are recorded in the Logging and Bookkeeping server (LB). The information is passed to the LB by various Grid components (User Interface, Resource Broker, Job Submission Service, a CE's Globus Job Manager), and stored in a relational database.

This database consists of tables representing the users registered with the Resource Broker, the jobs submitted in the LB's history, the events generated by those jobs, and further tables containing more detailed information on those events. All events are timestamped with millisecond precision. Events may not be registered in strictly chronological order, due to lags in the arrival of information, and in some cases the same event can generate several entries, as it is registered by local CE middleware, the JSS, and the RB. These multiple entries will have different time stamps, as they correspond to the middleware component registering the event, rather than the event itself. The schema for this database, and the events corresponding to the LB codes, are shown in appendix B.

Using this information, it is possible to reconstruct the lifetime of a job. This lifetime may consist of a great number of steps, whether the job is eventually successful or not. A job that runs smoothly with no hitches (failure, matching, resubmission etc.) will produce of the order of ten events; some jobs that are resubmitted many times can produce several hundred events.

Fig. 5.1 represents the stages in the life of a job. The job can fail at various stages in the submission process, or even while running, and be returned to the RB to be matched and resubmitted. A job that keeps failing, or cannot be run, will be aborted. It should be pointed out that these failures are those associated with the Grid middleware; a job failing due to badly written code or other user errors will be counted as a successful run in Grid terms.

The useful entries for this work are those corresponding to jobs arriving at and leaving the various components of the Grid. These can be used to

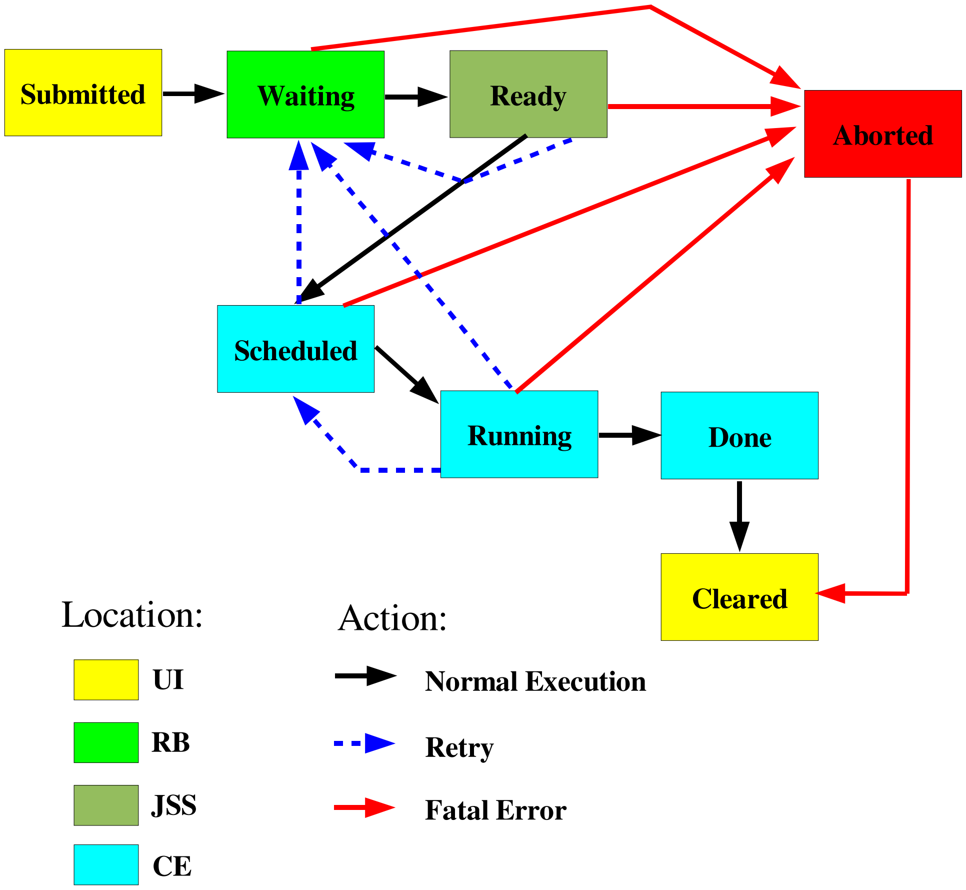

Figure 5.1: The lifecycle of an EDG job, showing the states through which it may pass, and the allowed transitions between them.

find the time spent by the job in the various stages of its lifecycle, and also to determine the load on those components from jobs submitted to the RB with which the LB is associated.

Logically, the number of jobs arriving at a component and those leaving it again should be equal. In practice, determining the true number of jobs passing through a Grid component can be complicated. For instance, jobs submitted to the Resource Broker can either be matched with a resource and passed on to the Job Submission Service, or they can be rejected and returned to the User Interface from which they were submitted. The job might arrive at the JSS, but be rejected and returned to the RB. This is registered not as a job arriving at the RB, but as an error code in the JSS. Thus significant job events can be recorded in a somewhat non-intuitive way, and care must be taken not to "lose" jobs by missing relevant event codes.

The reliability of the middleware components when updating the LB is also significant. The same event can be entered multiple times (by the same component, as opposed to the entries duplicated by different ones as mentioned above). Events can sometimes fail to be registered, particularly those supplied by the CEs' Globus job management software.

A more challenging problem arises from the instability of the Resource Broker. It uses a relational database (separate from the LB database) to store information on the jobs currently active in the Grid, and keep track of their movements. Failures in this database require it to be reset, leading to the loss of all jobs in the Grid system, whichever stage they are at. This does not cause any error codes to be recorded in the LB, so the job disappears from the system, leading to an apparent build up of jobs over time.

This unreliability can however be compensated for. Two approaches can be taken to deal with incompletely registered jobs.

The first approach is to remove all codes corresponding to these jobs. If a job does not register a bare minimum set of important events (for instance job submission time, scheduling time, the times at which it started and finished running, etc.), all other events associated with it can be removed. This method is useful if one is more interested in the timings of jobs and the different parts of their lifecycle. These data can be gathered without being affected by the removal of anomalous code.

If one is looking at the loading of Grid components, removing codes in this way will give too low a value for the number of jobs at a component. Instead, timings for the missing events can be estimated using values from other jobs. If code A is missing for a given job i, the mean interval between A and the next significant code B for all correctly logged jobs can be determined. The estimated timing of the missing code is:

$$\mathrm{t}_A^i = \mathrm{t}_B^i - < \mathrm{t}_B - \mathrm{t}_A > \tag{5.1}$$

Where a particular time code is missing, and there are no clues to be gained from other codes for that job, the best estimate is an average of the times for other jobs in the database. In this way one can build a more accurate picture of the job load (with the caveat that the timings are now slightly approximate).

Plots can be created representing the number of jobs at various stages in the Grid. By counting the number of codes representing jobs entering or leaving a component in a given time bin, a running total of the number of jobs at that stage can be created. Fig. 5.2 shows the results of this process for jobs queued at the RB; queued at the JSS; queued locally at the CEs; and running on a worker node (over the period of a month).

Two problems are immediately apparent here. Job totals drop below zero for the graphs relating to CE residency, and the lines seem to diverge steadily from a queue size of zero after time t=0.

The first problem is partially due to the zero point being arbitrary. This plot contains no information on the status of the Grid prior to the chosen starting time. In fact the Grid was in more use at the beginning of this period than at the end, so the zero point for all four lines is likely to be slightly lower than where it appears here.

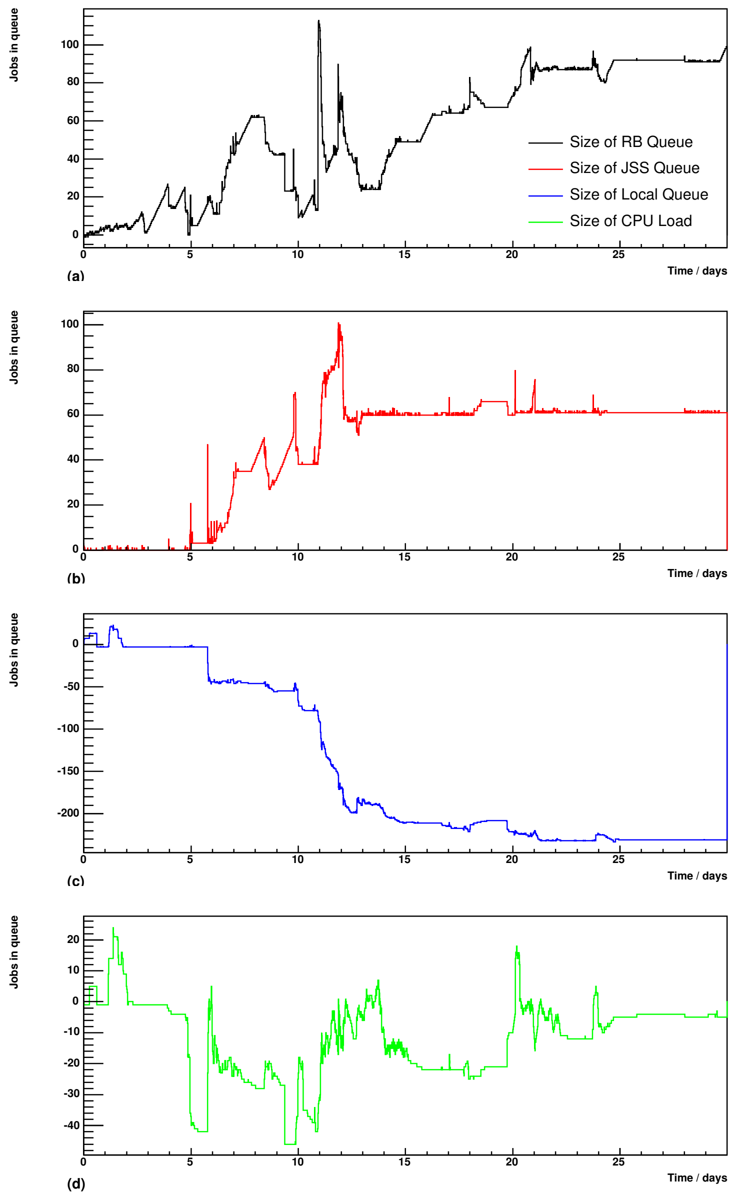

Figure 5.2: Jobs resident at various Grid components over time, as recorded in the LB database: (a) the Resource Broker, (b) the Job Submission Service, (c) queued locally at a CE, and (d) running on a Worker Node. These totals have not been corrected for inefficiencies in the recording of events by the LB.

However this cannot entirely account for such a large drop. The rest is due to logging inefficiencies. Some job events seem not to be registered on occasion, and if this happens more often with, for instance, the code corresponding to a job starting its run, a simple count of jobs (starting and completing) will begin to dip below zero.

This also partly accounts for the increase in the RB load, and possibly the smaller increase in the JSS, as codes corresponding to jobs leaving these components are not entered into the LB. However a large contribution comes from the instability in the RB's Postgres database mentioned earlier. A pile up of jobs can sometimes occur, leading to the RB being restarted, and thus losing all jobs being processed without an exit code. This leads to a step increase in a running total of jobs. This is best seen in the JSS total, which shows the plateau of the zero point being shifted upward by around 60 jobs by the end of the month depicted.

In order to produce a more accurate plot of Grid component loading, the second correction method described above, that of estimating the missing code timings from the measured mean average, was implemented. The result is shown in fig. 5.3.

The shifting zero problem has been solved, with the job load dropping back to the correct zero point when the Grid is not in use. Periods of intensive job submission show up as large spikes here, with jobs passing through the RB and JSS before queueing and running at the CEs, or stalling at a particular stage and being removed from the system.

However this approach is not appropriate when attempting to accurately determine the behaviour of the system, and jobs within it. Unless stated otherwise, in this thesis the first method (that of omitting corrupted job data) is used.

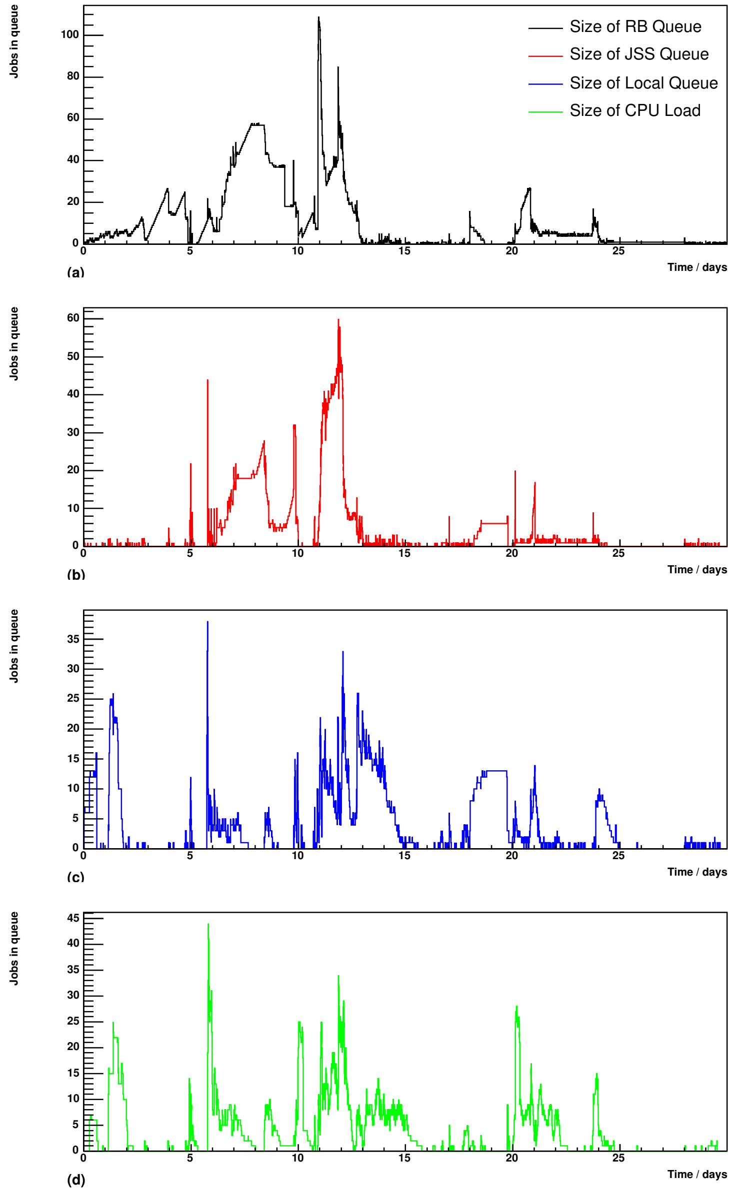

Figure 5.3: The same job loading data with time from the LB as in fig. 5.2, but corrected for missing or duplicated event codes.

5.2 Site Monitoring

All Grid resources provide status information via the Information Services for use by the Resource Broker when making decisions on job scheduling. This information is published by a hierarchy of LDAP servers, which can be queried at any level for the status of all entities below that point in the hierarchy. In the simple structure of the GridPP testbed, there are two levels: the Information Index of the RB, and below that the GRIS (Grid Resource Information Server) servers at the member sites.

A Compute Element advertises a large number of attributes, some of which are largely constant over time, such as the number of processors it manages and the address of its local SE, and others which change more dynamically, such as the number of jobs currently running or an estimate of the time a job would spend queueing at the site. These quantities can be translated into a Condor ClassAd and matched with requests made by users based on these quantities. They can also be accessed with a simple LDAP query.

The IS can be used to construct an up to date picture of the resources currently available in the Grid, with sufficient compensation for the manner in which the information is presented. The main complication is due to different queues at a site getting separate entries in the information provided. Thus the total number of processors, those currently available, the current number of resident jobs and so on will be advertised separately for each queue. These figures may be the same for each queue, but will sometimes vary if different subsets of machines are made available to them. The queues will only publish figures for the jobs that they manage, but jobs run by other queues will affect the number of available machines.

In some cases a resource may have other queues not made visible to Grid users. This can make the advertised numbers of resources seem somewhat misleading. For instance, a CE claiming to have a total of 50 processors and no jobs currently queued or running might still advertise 0 available CPUs because the farm is being used by non-Grid users. A reasonable compromise is to redefine the total processors available to be the currently free CPUs (as advertised), plus the number of jobs running at the CE (on all Grid queues). Any measures of Grid usage will arrive at a more meaningful figure in this way. (This method assumes that all queues at a site share the whole pool of CPUs, which is supported by observation of the logging data).

The jobs shown in the IS data are a different subset of the total job load on the EDG to those recorded by the LB. Whereas the LB stores data on the jobs that pass through the UK RB, the IS data pertain to the job load on the resources being monitored. This includes jobs scheduled by other RBs, and excludes jobs scheduled by the UK RB to non-GridPP resources.

By monitoring these sites with regular LDAP queries to the local GRIS, the job load at the selected resources can be monitored. The GridPP CEs were queried every 15 minutes for their currently available CPUs and currently resident jobs. The jobs submitted to these resources for the same month shown in fig. 5.3 are plotted in fig. 5.4.

As seen in the LB data, the Grid has at least residual activity throughout this month, with some busy periods, often showing a daily cycle of job submission. Empty bins are due to inefficiencies in the LDAP querying script (including the gap of several hours at around 4 days). The unusually large spike at 20 days may be due to an error in status reporting by a local GRIS.

Although the two data sets do not match completely, they can be compared to look at broad trends in job submission. An equivalent plot to fig. 5.4,

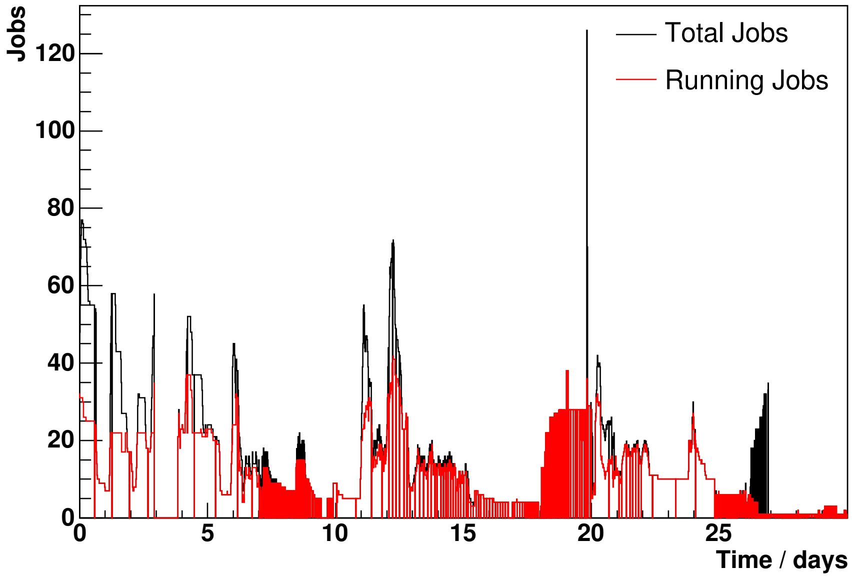

Figure 5.4: Jobs resident at CEs with time. The total jobs as well as those running are shown. The data were collected using monitoring scripts which queried the LDAP based IS servers every 15 minutes.

but using LB data, is shown in fig. 5.5.

Several similar structures can be seen, although there are some differences, such as some of the peaks of activity shown in days 1-5 in fig. 5.4, the IS plot. Possible inefficiencies in logging can also be seen here, too - a batch of jobs shown to be running at 18-20 days in the IS data seems to be reported as arriving in fig. 5.5, but the jobs' running does not appear to be registered in the LB. This could be a failure in the middleware managing the job, or a problem with updates via the IS.

Another use of the IS data is that it allows determination of the available computational resources, and their stability over time. In fig. 5.6, the CPUs available at nine GridPP CEs over the same month is shown. They are those

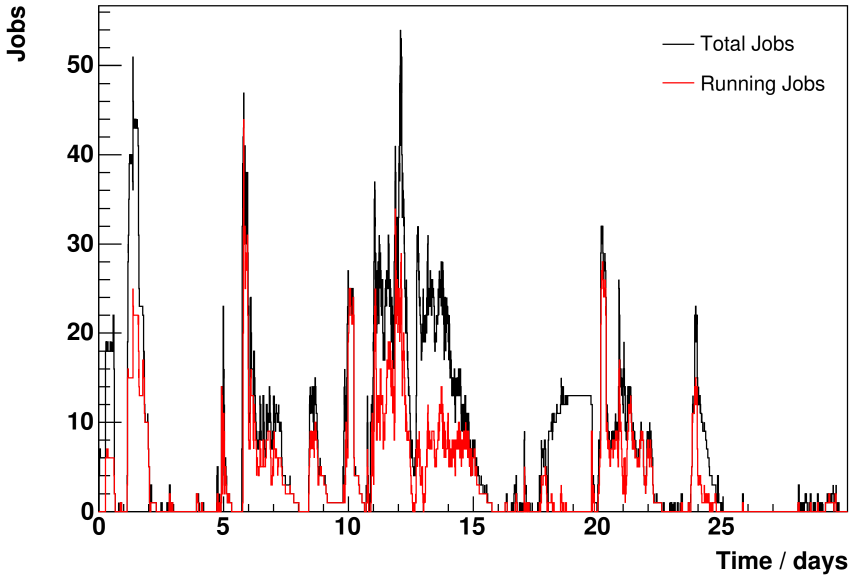

Figure 5.5: Jobs resident at CE with time from LB data, covering the same time period as fig. 5.4.

listed in table 7.2, along with a second CE at Bristol (no. 9 in this plot) which was omitted from the tests in chapter 7 due to its unreliability.

Using these data, the consistency of these sites as resource providers can be monitored. In some cases a site becomes unavailable due to failures; in others, it is because the CPUs are being used by a job queue that has not been made visible to the Grid, which also accounts for periods where the number of available CPUs is temporarily reduced.

The Information Services can be used to determine the state of Grid resources in a way that is not possible with the LB data alone. However, it has some shortcomings. While the numbers and distributions of jobs across the Grid can be determined in this way, it is not possible to monitor the progress of individual jobs. Thus when submitting batches of test jobs, they

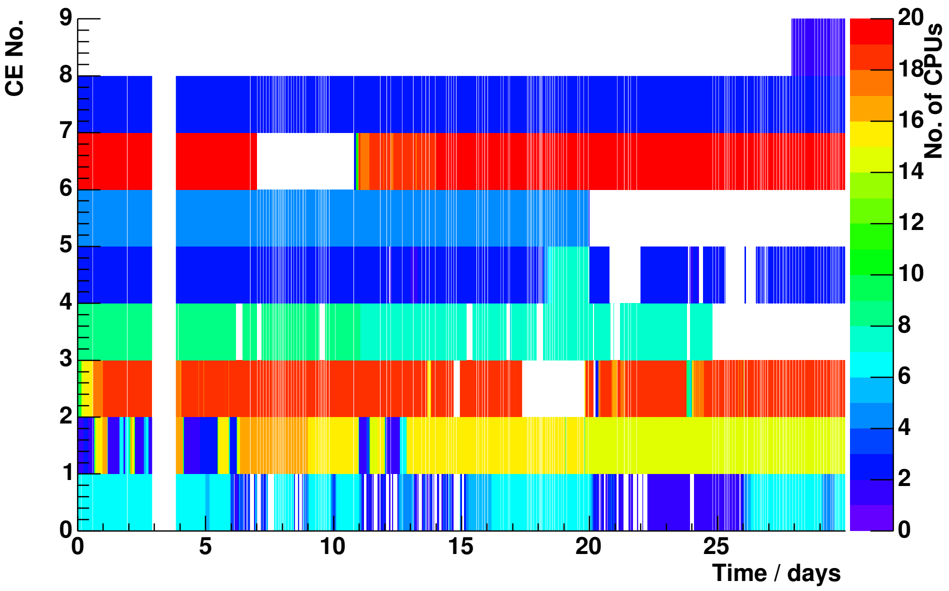

Figure 5.6: The distribution of available CPUs at 9 CEs, taken from the IS data over the same period shown in fig. 5.4.

cannot be distinguished from the background of jobs submitted by other users, and the effectiveness of job scheduling cannot be determined.

Jobs that have been submitted but not scheduled are also not shown in the IS data, as it only knows about jobs that have reached resources, and not those that are being processed by the RB and JSS. This means that the efficiency of the brokering process cannot be determined, as the rate at which the Grid middleware is dealing with incoming jobs is unknown.

5.3 Possibilities with EDG Monitoring Tools

The techniques discussed above make it possible to monitor the progress of jobs, as well as the state of the Grid while they run. This allows for a measure of how effectively the jobs are being scheduled, both in terms of how rapidly results are returned, and how fairly the load is being shared over the Grid.

The biggest missing piece is the activity of the Resource Broker itself. We are able to see the outcome of its decisions, as the jobs are submitted and scheduled to CEs, but the decision making process itself is not recorded. If a bad choice is made it could be due to shortcomings in the scheduling policy, old and inaccurate information from the IS, other inefficiencies in the middleware, or a combination of problems.

The RB's behaviour can be inferred to a certain extent by looking at the outcome of its decisions with the LB's records of job lifetimes, and comparing with the state of the Grid at the time using IS monitoring data. However the latter can only be relied on to give an accurate picture of the static qualities of resources, such as total available CPUs. Information (advertised through the IS) on the job load, which changes rapidly when the Grid is busy, can often be somewhat inaccurate as it is updated infrequently. The RB itself may have even older information, as it periodically updates its own records from the GRIS at each CE (the default time for this is every 10 minutes). However from observation, the update time for the information advertised by local sites appears to be shorter than the RB's 10 minute intervals, so this latter update time is the more significant one.

Information about the Grid users is also not stored in the LB, other than their ID and authentication details. Each user belongs to a Virtual Organisation, which will have access to a different subset of the total EDG resources. This means that if two jobs belonging to two different users are submitted simultaneously, the RB will be choosing a CE for them from different lists. Without knowledge of the CEs available to each VO at a given moment, it is difficult to determine how good the RB's decision was. The CE selected for each job is recorded in the LB, but the list from which it was chosen, and the status information used to do so, is not.

5.4 Summary

In this chapter the methods which can be used to extract job and resource data from the Grid are discussed. The process of querying the Logging and Bookkeeping database to extract job event data is described, as are the inefficiencies in the logging process. In the case of jobs with missing events, it is explained how these events can be estimated, or the jobs removed from calculations.

A method of monitoring Grid resources is then described, involving regular queries to the GRIS at each member site. The difference between the subset of jobs seen here and in the LB data is explained, as are the limitations to the information that can be gained from these queries.

Finally there is a discussion of how this information can be used, how one must be cautious about the interpretation of data produced in this logging and monitoring process, and how the behaviour of the RB can only be inferred indirectly, as its decisions and the information on which they are based are not recorded.

Having established how data can be extracted from the real Grid, and generated using the simulated one, the next task is to compare the results. The following chapter describes the performance indicators which will be used to carry out this comparison.

Chapter 6

Performance Indicators

This chapter describes the metrics chosen to measure the performance of the Grid. After a discussion of the different perspectives from which Grid efficiency can be seen, two performance indicators are defined: User Efficiency, measuring Grid performance from the point of view of a user submitting jobs; and System Efficiency, from the perspective of the owners of computational resources on which the jobs will run. Some other performance indicators are also described, followed by a discussion of the optimisation of Grid performance.

6.1 Measuring Grid Efficiency

A Grid is a complex system, due to both the number of entities involved in it (hardware, software and human), and its distributed nature. Measuring the efficiency of its performance will be a complex task, as all of these factors will be an influence to a certain extent. Even defining this efficiency is not simple.

The performance of the Grid can broadly be seen from two perspectives: that of the user watching job performance, and the resource owner who is more concerned with effective use of the Grid. The user will want their jobs to be scheduled quickly, and begin running as soon as possible, on the resource that is most suitable for their job (fastest CPU, most memory etc.). Individual user needs must be considered here too - multiple users with similar requirements will be competing for the same resources.

The owners of the Grid's resources will want the workload of jobs to be shared as fairly as possible between them, so that one site does not have a long queue of jobs waiting to run, while another has idle CPUs. Often these two aims will be complementary, as more efficient use of the Grid's resources will lead to faster processing of jobs, thus satisfying the users too. However in some situations, for instance when several users require the same resource or particular datasets, satisfaction of individual user requirements by completing their job as quickly as possible may come at the expense of the system.

The relationship between these two perspectives is not an obvious one. However in considering the GridPP testbed described here, data management issues can be discounted, and efficiency can be described in terms of access to CPU power for the user, and an even distribution of jobs across CPU resources from the system perspective. To reflect this, two metrics are presented: User Efficiency, indicating how quickly a user's jobs are completed; and System Efficiency, reflecting how effectively the job load is shared between resources [48]. Other useful metrics are also discussed, as well as a method of estimating the effect of an ideal scheduling regime.

6.2 User Efficiency

From the user's perspective, the ideal situation is one in which a job is submitted, matched promptly, and then scheduled to a site where it can begin running immediately, completing without any problems and successfully delivering its results. An efficiency of 1 would correspond to a situation in which time associated with overheads is reduced to zero, and the job's lifetime consists purely of the time it takes to run, on the fastest machine available to it. Clearly this is impossible to achieve, as there are always overheads associated with the batch system, etc. However a comparative measure of User Efficiency can still be made based upon this principle:

$$\mathrm{E}_{User} = \frac{\text{Running time on fastest CPU}}{\text{Total job lifetime}} \tag{6.1}$$

This measure takes all delays into account, including those incurred in the resource matching process, the time taken to transfer the job between sites, and time spent in a local queue at a CE. This is useful, as the origin of a delay cannot always be determined from the LB data due to ambiguities or errors in the recording of job events.

E_{User} will also be a function of job complexity, as a job that requires a large dataset and is I/O intensive would make it difficult to determine the actual running time of a job. Jobs that can be checkpointed or split up and run in parallel also complicate matters. However, the EDG middleware setup described here does not easily allow for complex or data dependent jobs, so these issues need not be considered in this case.

There is a dependency on job lifetime in E_{User}, which could distort measurements in cases in which jobs of many different CPU time requirements are involved. To eliminate this problem, only jobs submitted as part of the test run with identical CPU requirements are considered.

The CE which eventually receives the job, and the CPU that runs it, will also have some effect, as the job running time will vary, affecting E_{User} as described above. Unfortunately the characteristics of the Worker Nodes and the local queueing policies are not advertised by the CE via the IS, and not stored in the LB, so this information is not available. However the job running data indicates that the spread of CPU speeds across the GridPP testbed did not appear to vary greatly (see fig. 7.8 for an example of running times from a test run). The numerator in equation 6.1 can be taken to be the running time of the job as recorded in the logging data.

Thus despite its simplicity and some limitations, E_{User} is an appropriate performance indicator for the tests carried out here, giving a useful measure of how well a Grid user's requests are being handled.

6.3 System Efficiency

An idealised efficiency of 1 from the system's perspective is more difficult to identify. The distribution of jobs across resources must be considered, but so must the actual job loading at the time. One could take an instantaneous measure of system efficiency to be a ratio of the number of jobs running at that time, over the total number of jobs submitted, which would include those involved in the brokering process, or queued by a CE. This is given by:

$$\text{Instantaneous } E_{System} = \frac{\text{CPUs Delivered}}{\text{CPUs Requested}} \tag{6.2}$$

This is effective for an underloaded system, i.e. one in which there are enough CPUs to run all existing jobs. However when the job load increases beyond this point, some jobs will have to be queued. In order to accommodate this, the denominator of E_{System} can be modified, to instead represent all of the CPUs that could be delivered:

$$\text{Instantaneous } \mathrm{E}_{System} = \frac{\text{CPUs Delivered}}{\text{Min(CPUs Requested, CPUs Available)}} \tag{6.3}$$

In order to determine system performance for a period of time, the mean value of this quantity over that period can be calculated:

$$\mathrm{E}_{System} = \frac{\int_0^T \text{Inst. } \mathrm{E}_{System}\, dt}{\int_0^T \mathrm{dt}} \tag{6.4}$$

As with E_{User} this is a comparative measure, as a perfect efficiency of 1 would correspond to all jobs being instantaneously matched and scheduled to a CE with a free CPU.

One must be careful when determining E_{System}, however, as there are complications. Not all of the information necessary to make a universal calculation of E_{System} for the GridPP testbed is available. The GridPP is a subset of the EDG, and its RB is used by other virtual organisations with access to resources that are not part of the GridPP. This complicates the calculation of the quantity 'available resources' in the E_{System} formula.

For each Virtual Organisation, a different set of resources is available, and a job may be sent to an overloaded resource simply because its VO does not have access to a CE with idle Worker Nodes. Strictly speaking, to calculate a universal value of E_{System} for the GridPP testbed, one would have to include jobs submitted through other RBs, but this data is not stored in the GridPP LB.

To cut out these problems generated by unavailable information, the calculation was restricted to data from the test jobs, using a known and fixed subset of the resources. While this cannot be seen as a universal measurement of the system efficiency, it still acts as a useful measure of how effectively the RB is matching jobs to resources.

6.4 Other Performance Indicators

While the two metrics described above will be the main ones used here, there are other means of determining Grid performance which can be useful in some situations.

In an unstable Grid, for instance an infrastructure such as the EDG, there is no guarantee that all jobs will be successfully completed. (N.B. failure due to user error, such as poorly written code, is considered a success in job management terms, as long as the job was delivered to a resource of the type requested.) In such a situation, a slightly more pessimistic measure of efficiency from a user's perspective is a simple percentage of jobs completed, or Crude Efficiency:

$$\mathrm{E}_{Crude} = \frac{\text{No. Jobs Completed}}{\text{No. Jobs Submitted}} \text{x } 100 \tag{6.5}$$

An alternative way to handle an imperfect Grid is to discount jobs that have been lost in such a manner, and base calculations upon successfully scheduled jobs only. If one is investigating the effectiveness of a particular scheduling policy or Grid management scheme, then inefficiencies due to a specific middleware implementation are less interesting. As Grid middleware is under development and is constantly being updated, these inefficiencies are unlikely to be more than temporary problems.

Another simple performance indicator is applicable for Grid test runs consisting of a fixed number of jobs submitted in one or more batches. This is the time taken for all jobs to have run and completed, a quantity which will be of interest to the user who wants to have the job results back as soon as possible. This quantity can be called Last Completion Time (LCT), as it corresponds to the time interval between the beginning of job submission, and the completion time of the last job to finish running.

The main problem with Last Completion Time as a performance indicator is again associated with middleware inefficiencies. Jobs can be delayed at various stages in their lifecycle, in some cases indefinitely. In these cases a success or failure event may be recorded much later than those of the other jobs, giving an exaggerated value of Last Completion Time, or not recorded at all, making the value of this metric somewhat ambiguous. In this case a cut-off point can be chosen, and data relating to jobs delayed by more than this value can be discarded.

6.5 Optimising Performance Indicators with Idealised Scheduling

While it may not be possible to achieve a perfect efficiency in a Grid environment with job running overheads and other inevitable problems, it should be possible to maximise efficiency to some degree. In this chapter the ideas of user and system perspectives on efficiency have been discussed, and one can work to maximise either of these efficiencies. The Grid setup and its workload may allow for both quantities to be maximised with the same strategy.

In order to produce this maximised efficiency, an idealised approach to scheduling is needed. This requires perfect knowledge of the system, and the jobs being introduced to it, so that the scheduler can make a fully informed decision. It could therefore be said that this is idealisation of information, rather than idealisation of the scheduling policy.

An accurate information service will be required, one that is up to date and not some minutes behind the current status of Grid resources. It also needs more information about the jobs being scheduled. The current EDG schema supplies little beyond the requirements of the job in terms of hardware and running conditions as well as the user's scheduling preference, and that only if the user chooses to include this in the JDL file. Thus the RB schedules a job in the same way whether it is a "Hello World" job that will take less than a second to run, or a Monte Carlo job that will run for twenty four hours. One possibility would be a sampling approach, in which a small section of the job is run in order to estimate the CPU cycles needed to complete the whole task [49]. However the EDG schema has no provision for this technique.

In addition to this, the RB might also require the timings of job submissions, and the order in which jobs arrive at the RB for scheduling. This would allow an ideal distribution of jobs to be determined, so that they can be spread between resources in such a way as to maximise efficiency. Even in a very heterogeneous Grid, with CPUs of many different speeds, the CPU time a job requires at a given resource can be calculated with this information available, so the future time for which that resource will be booked is known.

This scheduling problem is different to many in conventional computing. In some cases, processes running on a computer cluster will be moved from one CPU to another, optimising the system during runtime as jobs begin and finish running [50]. However this is not practical for a distributed scenario such as a Grid, because the overheads involved in transferring a job to a geographically remote site would cancel out any advantage that might be gained.

Much scheduling work concentrates on efficient means of sharing processors between tasks [51], which is a concern for local resource management rather than Grid middleware.

Some distributed applications are closely coupled, and require significant network I/O in order to run. In these cases the scheduling policy will be concerned with arranging the processes to minimise time lost during communication between distributed nodes (as with the examples in section 1.7.2). Again this is not applicable here, as the jobs discussed in this work run independently of each other.

Scheduling is greatly complicated when jobs require access to input data (and, to a lesser extent, access to free cache space to store their output). The ideal situation for a data dependent job is a free CPU at a site with local access to all of the requested input data. In order to facilitate this for as many incoming jobs as possible, the scheduler would need foreknowledge of the requirements of future jobs, so that it could arrange for the data to be distributed appropriately, and timeslots to be reserved for these jobs at local resources. This is a very complex task, with no simple means of determining the optimum configuration of data and jobs. The scheduling of data dependent jobs will be explored more fully in chapters 9 and 10.

However for the jobs in the first set of tests here, these complications need not be considered. The jobs are independent, and do not require access to data, which means they can be dealt with as they arrive - no foreknowledge is needed, because actions taken on behalf of previous jobs will have no influence. One can determine the total load on a given resource with knowledge of that resource's properties, and the requirements of the jobs already resident there. The following section demonstrates how this knowledge can be used to schedule jobs.

6.5.1 Conflicts in Scheduling Optimisation

The total load L_{CPU} is defined by:

$$L_{CPU} = \frac{\text{Total remaining required CPU cycles}}{\text{CPU speed x No. of CPUs}} \tag{6.6}$$

This quantity represents the time needed for a resource to run all of its jobs (assuming an efficient local scheduling policy at the resource, as this is out of the RB's control). It can be determined for all resources, as can the increase that would be caused by the addition of an incoming job. A possible ideal scheduling policy would assign the job to the resource that gives the lowest L_{CPU} after the addition of the job, i.e. leads to the most even distribution of the job load. This means that this technique will distribute jobs so as to minimise LCT.

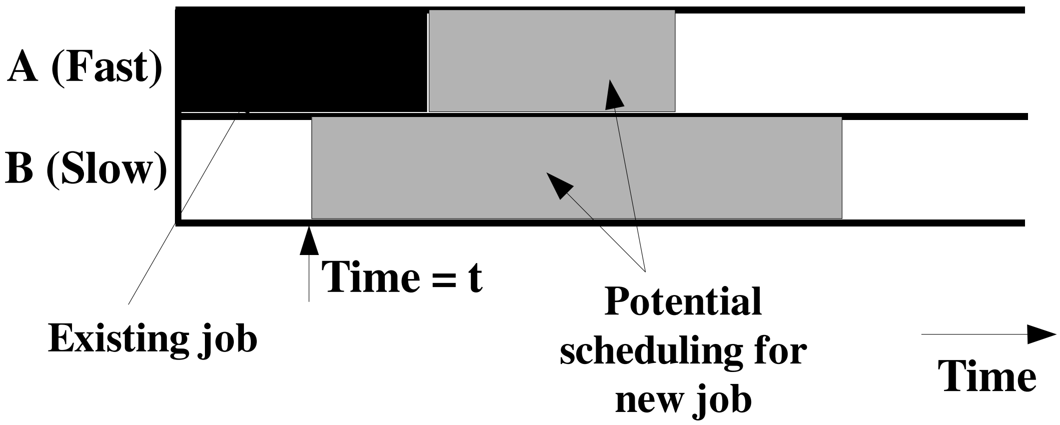

Figure 6.1: A potential conflict between different idealised scheduling policies. If the incoming job is assigned to A it will complete more quickly; if it is assigned to B the jobs will be more evenly distributed.

This is not a universally ideal policy, however, as situations exist in which conflicts arise between system and user requirements. Figure 6.1 shows such a situation. A job is submitted at time t, and the scheduler finds two suitable CPU resources, A and B. A is a fast machine, but already has a job running there, whereas B is much slower, but is currently free. The scheduler determines that the job would still complete most quickly at A, and if it is attempting to minimise LCT, this is where the job will be scheduled. E_{User} would also be minimised by this action, as the job's results would be returned to its owner more quickly.

But if the scheduler were attempting to minimise instantaneous E_{System} it would instead send the job to B, distributing the jobs more fairly from the resource owners' point of view. This minimises a different quantity, which can be called U_{CPU}, the CPU usage, defined as:

$$U_{CPU} = \text{No. of idle jobs} \tag{6.7}$$

Obviously in the scenario presented here, summarised in table 6.1, L_{CPU} and U_{CPU} cannot be minimised simultaneously, but in practice this conflict seems to arise very rarely. As discussed in section 7.4, the available CPUs do not seem to vary significantly in speed. This means that LCT and E_{System} optimisation will be in agreement for the majority of scheduling decisions.

Action	**LCT**	$\mathbf{E}_{User}$	$\mathbf{E}_{System}$
Minimise L_{CPU}	Optimised	Optimised	Not Optimised
Minimise U_{CPU}	Not Optimised	Not Optimised	Optimised

Table 6.1: Optimisation strategies for data independent jobs in a heterogeneous Grid.

The obstacle that prevents these idealised policies from being implemented is the absence of information on jobs' running times (or more strictly, the number of required CPU cycles). Thus the EDG default scheduling policy, Estimated Traversal Time (as described in section 3.2.2), attempts to determine the loading of resources by observing how long jobs take to begin running at each of them. However this policy will be less effective when dealing with the submission of jobs with a large distribution of CPU requirements. A good scheduling policy must take advantage of information which is available, and attempt to compensate for information that is not.

Despite the difficulty in implementing idealised scheduling policies in a real world Grid, they can be employed in a simulation. This can be useful in comparative tests to see how well other schedulers are performing. An example of this will be seen later, in section 8.6.

6.6 Summary

In this chapter, various measures of Grid performance were described. After a general discussion of efficiency measurements, two key metrics were defined. The first, User Efficiency, gives the performance of the Grid from a user's perspective, based upon the ratio of the job's running time, and the total job lifetime. The second, System Efficiency, views performance from the resource owners' perspective, and is based upon the number of jobs that have been supplied with the requested CPU resources. The limitations of both metrics were discussed. Some other performance indicators were also outlined.

In the final section the optimisation of Grid performance with idealised scheduling was discussed. The limitations to the available information that make this optimisation unfeasible in practice were explained. Different methods of optimisation were described, and potential conflicts between these ideals were discussed.

These metrics can be applied to both real and simulated Grid data, in order to determine how both systems handle different job loading scenarios. In this way the simulation can be validated with the real data. The next chapter describes how this comparison was carried out, and how the simulation was modified to behave more like the real Grid.

Chapter 7

Comparison of Simulation with Data

In order to simulate the GridPP testbed setup, various parameters of the simulated Grid have been set by hand according to observations of the running of the real Grid. This chapter begins by detailing these parameters, and the reasons for the choice of values. It then describes a simulated Grid testbed, based upon a real set of resources, and the background of jobs observed in the real Grid from data obtained from the LB database. Jobs are then submitted to the real Grid, and simulated job submissions are carried out to mirror them. Delays caused by middleware inefficiencies are observed, and are then implemented in the simulation in order to more effectively match the real data.

7.1 Grid Component Parameters

7.1.1 User Interface / Job Characteristics

Worker Node Specs: It was assumed that jobs would not fail due to shortcomings in the WNs such as insufficient available memory, or the wrong operating system version. The limiting of middleware compatibility to a specific architecture and operating system (PCs running Linux Red Hat 6.2, for EDG version 1.4), as well as the simple nature of the jobs being run in the tests, lends weight to this assumption.

Scheduling Options: All jobs generated use the default scheduling option (Estimated Traversal Time, see section 3.2.2). This is true for the majority of jobs submitted to the GridPP Resource Broker, as shown in table 7.1.

Scheduling Policy	Usage
Estimated Traversal Time	87.7%
Max. Allowed CPU Time	7.0%
No. of Free CPUs	3.4%
Other	1.9%

Table 7.1: Usage of Scheduling Policies for GridPP RB, Nov. 2002 - Sep. 2003, taken from the GridPP Logging and Bookkeeping Database. The total number of jobs submitted in this time was 40460.

Job Submission Rate: A User Interface submits a batch of jobs at a rate of one every nine seconds, this being approximately the time between execution of the job submission command, and the return of a message indicating success or failure of the submission. The distribution of these intervals during test runs is shown in fig. 7.1.

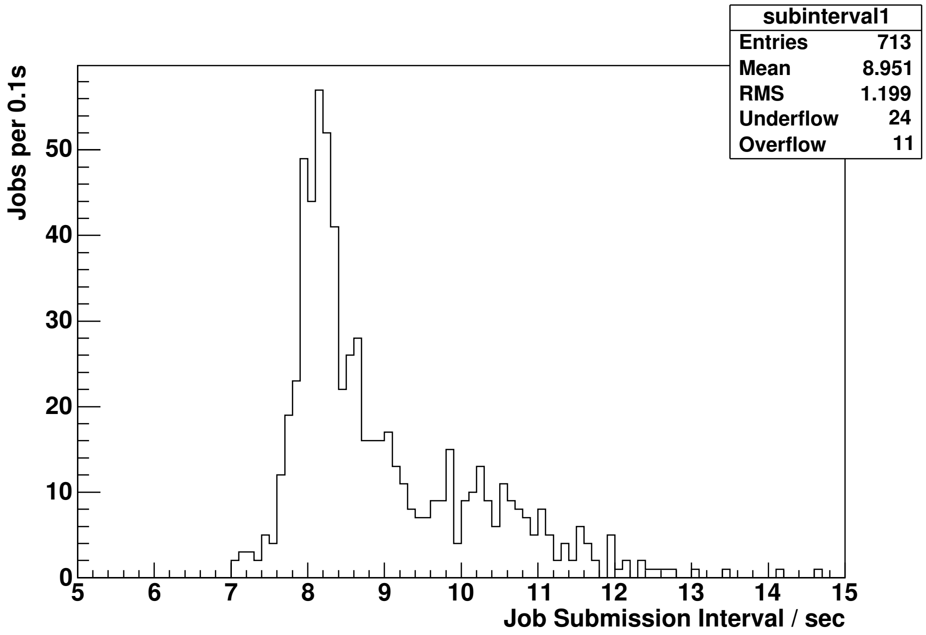

Figure 7.1: Distribution of time intervals between successively submitted jobs during test runs.

7.1.2 Resource Broker Characteristics

Fixed Resource List: The simulated RB has a fixed list of resources from which it chooses an appropriate CE for a job according to a scheduling policy. Thus if two or more resources are given the same ranking, the same CE will be chosen every time, leading to an uneven distribution of jobs. Observation of rapidly submitted jobs in the LB data supports this assumption. If the rate of status updates delivered to the RB is slow, this can result in a CE being overloaded, even if there are others with free CPUs.

Information Services Update Time: This is the interval between status updates, delivered by the resources to the Resource Broker. This is set at 300 seconds, an approximate figure arrived at through observation of the RB logs. This timing is difficult to quantify, however, as it varies widely in a manner that is not easy to predict. The stated value is in fact 10 minutes, but inefficiencies in the Information Services complicate matters. When the RB has made its resource matching decision, it queries the chosen CE to ensure that it is currently available. If the CE is already dealing with several incoming jobs, it may not respond. This causes the RB to pick the next CE in its ranked list. Thus a batch of jobs submitted to the real world RB in rapid succession is likely to be distributed between several resources, even if the RB would prefer to send them all to the same site. This is approximately equivalent to a reduction in the IS update time, which would also cause the RB to distribute jobs between resources, as it would become aware of overloading at a resource and attempt to compensate.

Scheduling Interval: The time taken by the RB to match a job to a resource. This varies according to the complexity of the job in terms of its data requirements. However replica management is not fully implemented in this version of the EDG middleware, and the majority of jobs run have no data dependency. When the RB is functioning correctly, data independent jobs take $\sim$10 seconds to be matched [52], so this is the figure used in the simulation.

Jobs Dealt With Sequentially: The simulated RB deals with jobs sequentially, in the order that they arrive. The real RB creates a separate process for each job that it is engaged in matching, so theoretically is dealing with them in parallel. However bottlenecks in the matching procedure mean that this does not happen in practice - these parallel processes must queue for the resources needed to match the jobs. As will be shown in section 7.3, the GridPP testbed is rarely used heavily enough to make the two models perform significantly differently.

7.1.3 Compute Element Characteristics

Jobs Never Rejected or Unsuccessful: Data on job running can only be based upon those jobs that run successfully, so resource failures have not been factored into the simulation. In fact CEs that consistently rejected or failed test jobs were cut out of the GridPP test runs in order to reduce occurrences of job failure.

All CPUs Are Identical: The characteristics of Worker Nodes are difficult to determine accurately, as they are not advertised by their CE. A job submitted to a site with a heterogeneous cluster of CPUs will be assigned to one of them according to a policy which is also not publicly available, so the structure of that node is hard to discover even by submitting jobs querying its properties. However in this Grid model, CPU speeds are not a factor in the assignment of jobs to CEs. As this study is concentrating on the resource brokering aspect of the Grid, WNs across the virtual Grid were set to be identical.

7.2 Virtual GridPP

The GridPP is part of the larger European Data Grid, but is also a testbed in its own right, with a Resource Broker based at Imperial College London. However the resources in the GridPP testbed are also accessible by the other users and RBs of the EDG, so the records of the Imperial RB do not give the whole picture of GridPP usage.

The alternate view of the GridPP is through the LDAP-based Information Services. The RB's Information Index contains (nearly) up to date status information on the resources it can currently see. More reliably, the servers at the resources themselves can be queried for the latest Grid usage. This too has a disadvantage, in that you can only observe usage of the resources in terms of the number of jobs at each site. You cannot monitor jobs individually, or identify jobs submitted via a particular RB. Some sites also have local queues not visible to the Grid middleware, often resulting in the site advertising no jobs currently running, but no free CPUs. This makes it difficult to calculate the usage of CEs via the IS.

The best compromise was to choose the set of resources which appeared to have the least traffic from other sources, accessible by a member of the GridPP VO, and carry out tests upon these. For instance, the large CE at Glasgow (over 100 CPUs) was often in use via a non-Grid queue as described above, but when occasionally idle, it would almost double the number of CPUs available on the GridPP testbed. In order to select a stable set of resources (i.e. approximately constant number of CPUs), CEs like this were excluded in the Job Description File for the test jobs. A list of eight CEs, with between one and twenty Worker Nodes each providing a total of 59 CPUs, was selected for the tests, and for representation in the simulated version of the GridPP. The distribution of CPUs is detailed in table 7.2, the simulated Grid is shown in fig. 7.2, and a simulated node with a configuration screen for the local CE is shown in fig. 7.3.

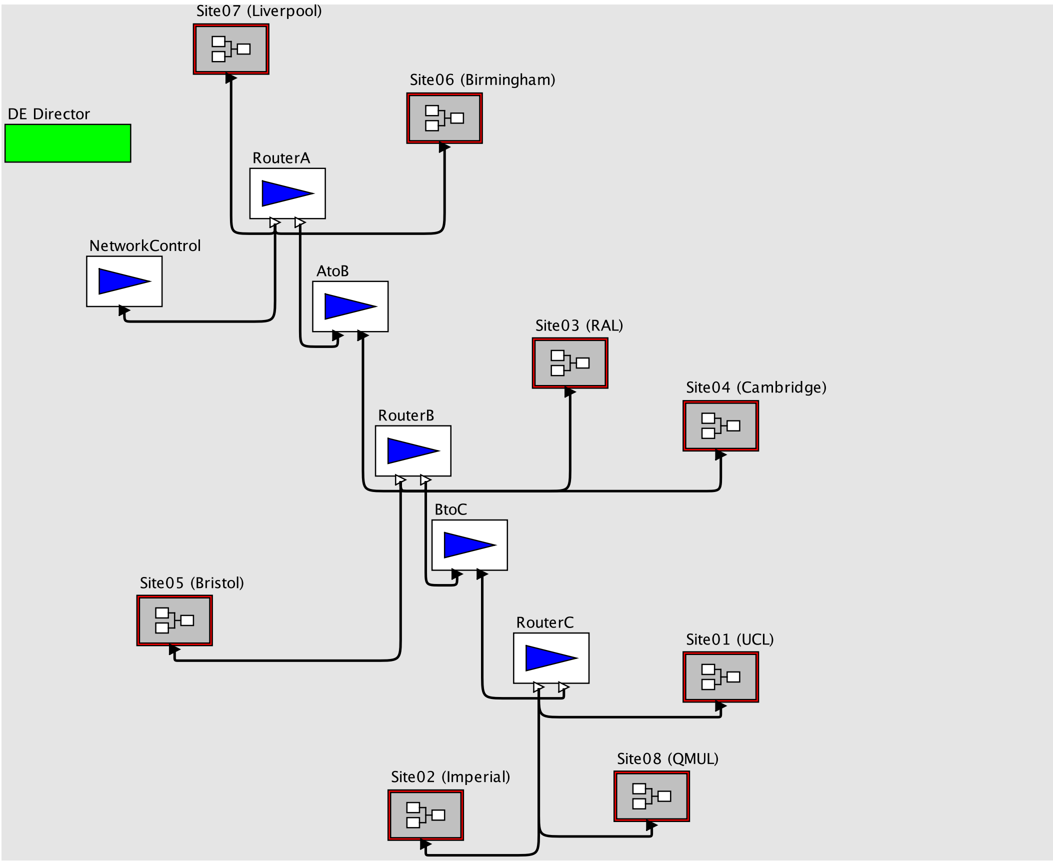

Figure 7.2: The topology of the simulated GridPP testbed in the PTII GUI, showing the eight sites represented.

The calculation of E_{System} (see section 6.3) depends upon whether the system is in an under or overloaded state, i.e. whether there are more jobs than CPUs or less, as the definition of the denominator differs in these two cases. Monitoring the resources using the LDAP Information Services provides data on resource availability and currently running or pending jobs.

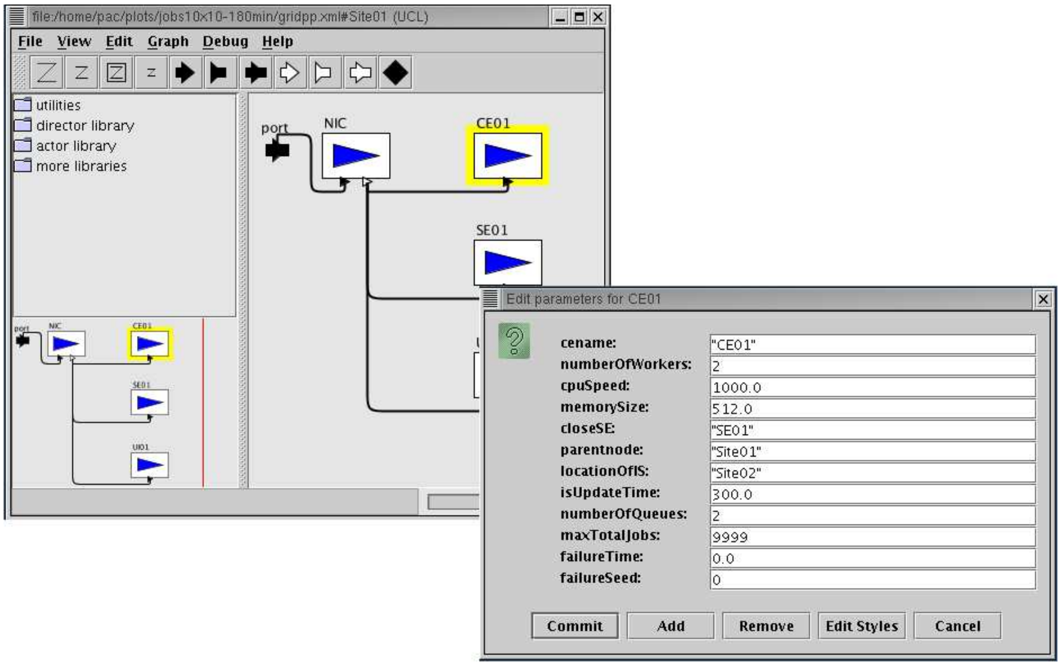

Figure 7.3: The Actors present in a simulated Grid node in the PTII GUI, showing a dialogue box for configuration of a CE.

No. / Location of CE	No. of CPUs
1. Birmingham	6
2. Cambridge	16
3. RAL	20
4. Imperial	8
5. QMUL	2
6. Liverpool	4
7. UCL	2
8. Bristol	1

Table 7.2: Number of Worker Nodes at simulated CEs

Fig. 7.4 shows the number of CPUs and jobs for the selected resources over a 16 day period.

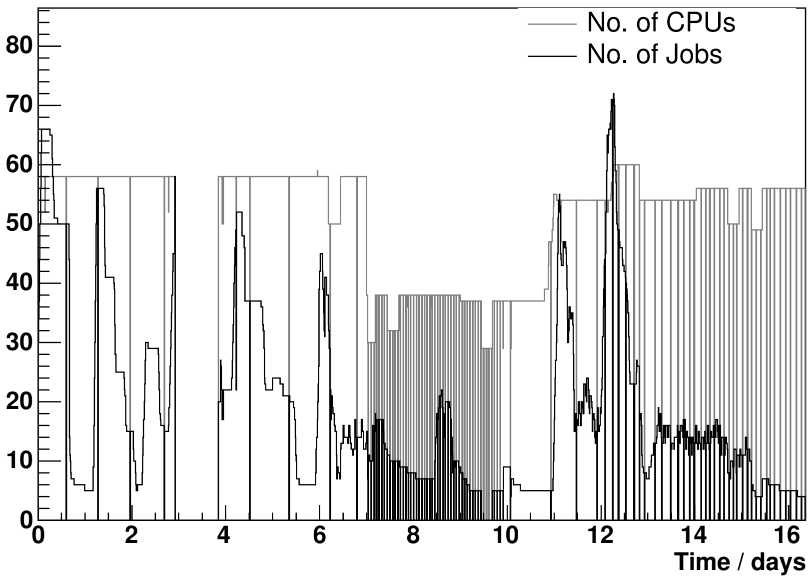

Figure 7.4: The CPUs available on the GridPP testbed over a 16 day period, and the jobs running on them.

The x axis of this plot corresponds to time, with each bin representing 15 minutes, the time between queries to the IS by the monitoring script, and zero corresponding to 09:00 on the morning of day 1. As can be seen some of the time bins are empty. This is due to the varying time taken for the LDAP servers to return the requested data. A larger gap at 3 - 4 days represents a temporary failure in the monitoring script.

The plot includes data on the 8 chosen GridPP CEs, and the total available CPUs can be seen to vary over time as described above. The job total shows a series of daily peaks as might be expected, with users submitting jobs some time in the morning, and the jobs being scheduled and completing over the next day or so.

While most of the time the job load is significantly less than the number of CPUs available, there are spikes in activity which mean that the two quantities are much closer. However, this is misleading as a guide to scheduling decisions and E_{System}. Firstly, there are several Resource Brokers (up to 4) with access to these CEs. Secondly, users can submit jobs directly to resources, bypassing the brokering process entirely. This is often the reason for these spikes. Fig. 7.5 shows the distribution of jobs across the 8 chosen resources listed in table 7.2.

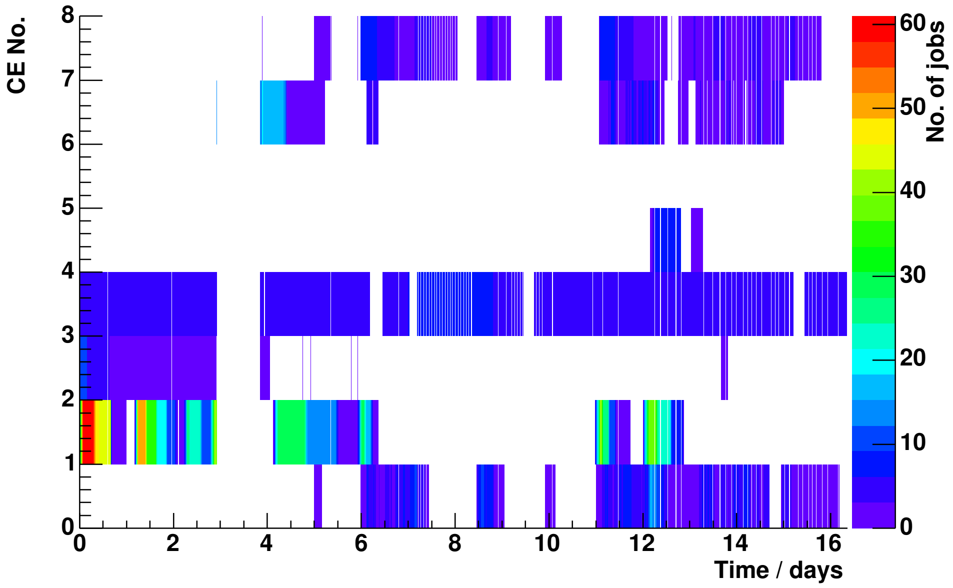

Figure 7.5: The distribution of jobs across the eight resources chosen for job submission tests over a 16 day period.

Again the x axis represents time, but here the job load is shown for individual CEs, the colour scale indicating the jobs at a site at a given time. The jobs in the large spike at the beginning of fig. 7.4 are almost all queued or running on the second CE shown here. This is a 16 CPU CE in Cambridge, which was observed via LDAP monitoring to run several of these local submissions of large numbers of jobs.

Even during intensely busy periods, some CEs remain completely empty, suggesting that the uneven job distribution cannot be entirely due to the RB's decisions, as even with a very slow information update time the other sites would eventually begin filling up. It seems likely that user specifications are coming into play here. The jobs may have been submitted with instructions that they go to a specific site, or restrictions placed on the jobs regarding CE characteristics (minimum memory, queueing restrictions etc.) might mean that only one resource is deemed viable.

This shows that background job running on the Grid is quite complex, and E_{System} should be calculated with some caution, as not all of the information about other users' jobs is available. However, monitoring data show that there were at least some CPUs available at any point. As long as test jobs were submitted at a rate that would not overrun the chosen resources, the system could be regarded as underloaded when determining the denominator of E_{System}.

7.3 Job Background

The job background for the purposes of these studies are any jobs submitted to the Grid other than test jobs during a test run. These must be parameterised with reasonable accuracy, because they can impinge on the results of the tests. A heavy background load will reduce the number of free CPUs, and will also affect the performance of the RB, giving it more decisions to make based upon imperfect information.

The majority of jobs recorded in the LB database use a negligible amount of CPU time. These are often of the “Hello World” type, as users try out the mechanism of job submission, or test how well various middleware components are functioning. Other Grid usage does not conform to a particular pattern, as users test the Grid’s handling of more complex tasks. Jobs of a day or even more in duration are observed, although these are in the minority.

In general, these jobs do not have a dependency on input data. This is a matter of practicality, as the data replica management middleware was still somewhat rudimentary at this stage of its development. Any data that need to be moved for a job to be run would require explicit commands to be issued by the user, rather than the automated processes that later versions of the middleware will use.

Monte Carlo style jobs produce output data, sometimes in significant volumes that cannot be simply returned to the user along with the logging data that are supplied when the job has completed running. This must be stored at the local SE, or transferred to another at a remote site where storage space is available. However, with the low levels of usage during the period of monitoring and testing described here, these issues of data storage will not be significant.

It should be noted that the situation in a production Grid, heavily used by members of active large scale HEP experiments, will be quite different. Real and Monte Carlo data will be produced in large volumes, and will require storage in an organised and accessible way. These datasets will be in frequent demand, and automated data management, as well as optimising replication of the data at sites where it is needed, will be implemented to compensate, as explored in chapters 9 and 10.

Fig. 7.6 shows the running duration of all background jobs recorded in

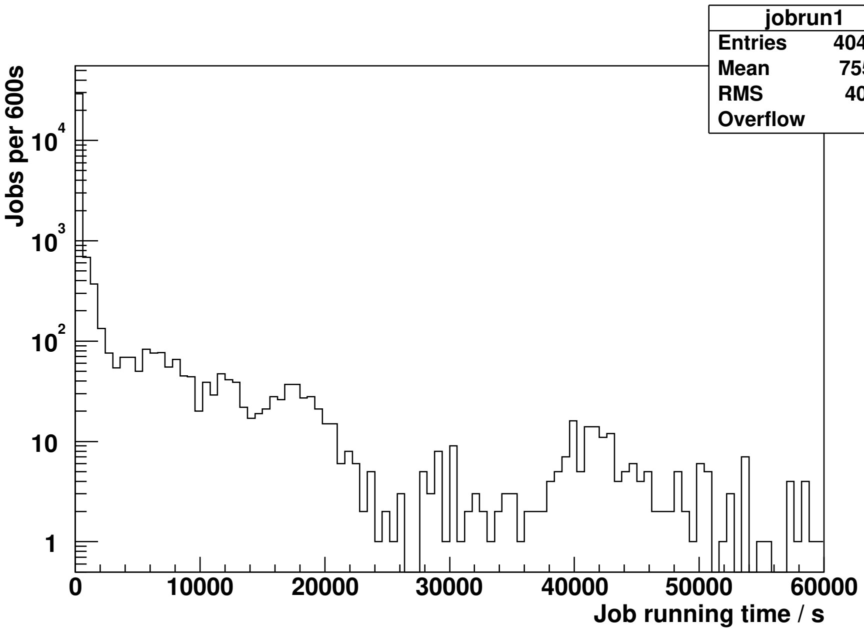

Figure 7.6: The distribution of times taken for all background jobs recorded in the LB database to run.

the LB database, between November 2002 and September 2003. The data are dominated by a large spike near zero, as expected. The rest of the data are in a long tail, dropping off slowly toward zero. Other than the very short jobs, the number of jobs passing through the RB is not large - ~10000 jobs over ten months (run on all sites, not just the GridPP CEs selected for this work).

The distribution in time of background jobs is shown in fig. 7.7. A period of two weeks is shown, for clarity of detail in daily Grid activity.

There is no consistent daily cycle of job submission visible here, as usage is low. However when spikes of activity appear, for instance in the last four days shown here, a spacing of around one day for the peaks can be seen. A distribution of single jobs at regular intervals can also be seen, with gaps in

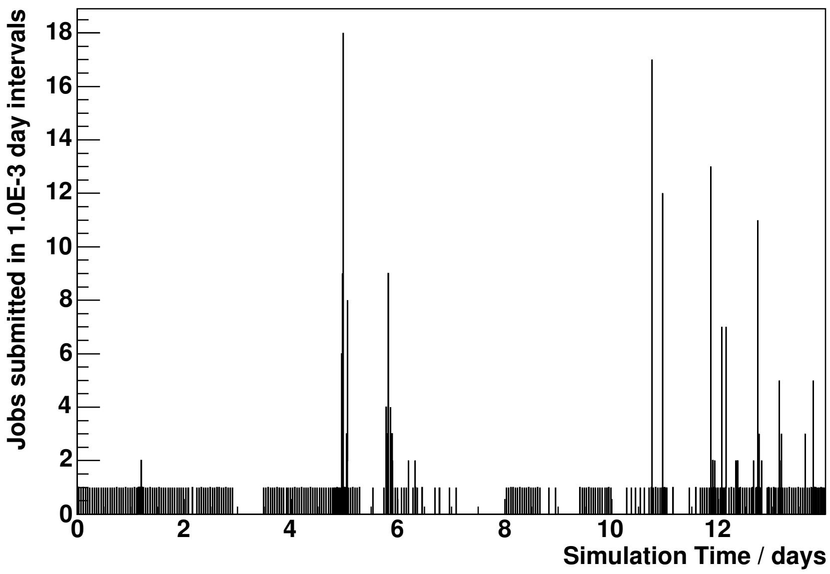

Figure 7.7: Submission times for all background (i.e. non-test) jobs over a two week period.

places but often continuing for periods of twenty four hours or more. These are likely to be automated status monitoring jobs.

To produce a job background similar to this in the simulation, a two part distribution was used. The first part was a steady submission of jobs with negligible running time, representing "Hello World" and monitoring type jobs. The second represented the smaller component of longer jobs. A truncated Gaussian distribution (cut off at 60000 seconds) with a mean of zero and an RMS of 30000 seconds represents the bulk of the longer jobs. The Gaussian jobs were submitted at the same time in each simulation day, to replicate the cycle seen in fig. 7.7.

7.4 Test Job Submission

The EDG middleware is inevitably somewhat unstable, as it is technology under development. The Resource Broker in particular, as the centre of decision making, has inefficiencies which can delay job submission, or even lock the submission process up. Corruption of the job registration database can lead to jobs being lost entirely. This means it can be difficult to conduct any long term tests involving submission of multiple jobs, without the results being distorted by problems unrelated to the intended functioning of the middleware.

It was decided to submit test runs of 100 jobs, submitted in batches of varying sizes with varying time intervals between batches. The times of the various stages in the test jobs' lifecycle were extracted from the Logging and Bookkeeping database. These data were then used to calculate values of System and User Efficiency for the Grid with this loading.

The test jobs were simple, in that they required no input data to run, and produced output of a negligible size when completed, thus factoring out data management and access to Storage Elements. They were constrained to the eight chosen CEs, by specifying in the Job Description File that they should not be submitted to the resources that had been determined to be unstable or prone to overloading with locally submitted jobs. Job batches were submitted at evenly spaced intervals, and monitored via periodic requests to the Logging and Bookkeeping services. The jobs' output was retrieved when an 'Output Ready' status message was obtained. If a job was delayed in the scheduling process for more than a day or so beyond the completed running of all other jobs, it was determined to have failed. The jobs ran for between three and eight hours, depending on the CPU speed of the WN on which they ran.

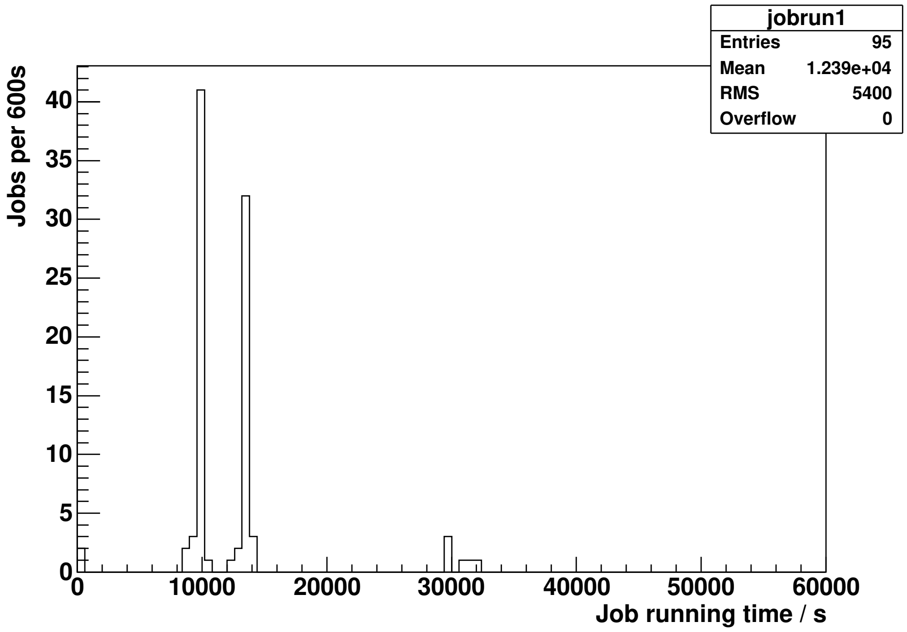

Figure 7.8: Running duration for test jobs during run with 20x5 job batches at 90 min. intervals.

Fig. 7.8 shows the duration of test jobs submitted via the Imperial RB during a submission of 100 jobs in batches of 5 every 90 minutes. Most job durations are in a cluster with two distinct peaks near 10000 seconds, as most of the test jobs ran at two sites with slightly different CPU speeds. A few jobs ran on much slower machines, taking around 30000 seconds to complete. Two jobs appear to have run for a very short time; this indicates job failure, which is still recorded as a success, since the Grid middleware successfully delivered them to a CE. In total 95 jobs are recorded, as 5 failed at the submission stage.

In fig. 7.9, the x axis represents time during the same test, and when jobs are submitted through the RB. The test job batches are shown in red as a series of spikes at regular intervals. Some of these spikes show less than

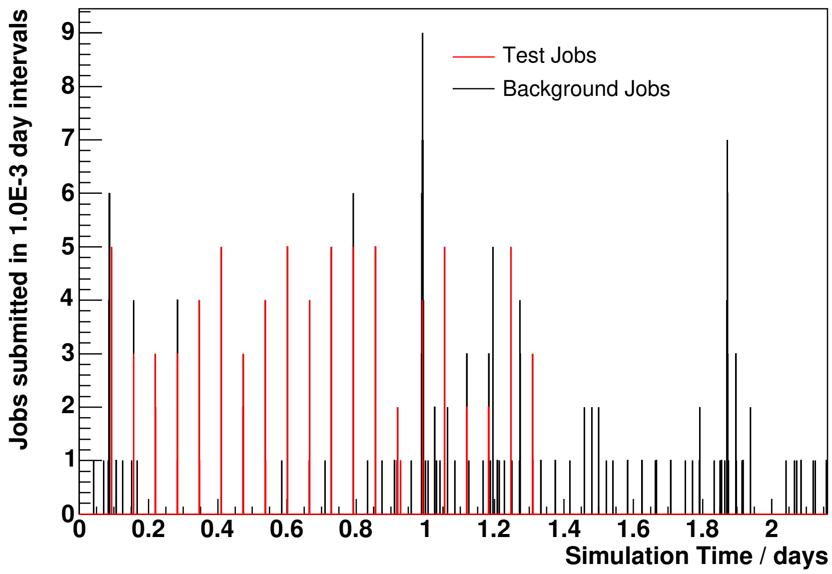

Figure 7.9: Job submission times for all GridPP jobs during test run with 20x5 job batches at 90 min. intervals.

five jobs, as jobs are delayed, or the time code for the jobs submission is registered with the LBB server some time later. In some cases adjacent bins are filled, so the five jobs are split between them.

7.5 Preliminary Runs

Four test runs of 100 jobs were carried out, with the jobs submitted in batches of different sizes and at different intervals. These were 20 batches of 5 jobs with 90 minute intervals, 10 batches of 10, with 180 and 90 minute intervals, and 5 batches of 20 with 180 minute intervals. These distributions were chosen to place a significant load on the selected Grid system, while keeping it in the underloaded regime to simplify calculation of E_{System}, as explained in section 6.3.

Using background distributions as described in section 7.3, four simulated test runs were carried out with similar input distributions. The simulated results were compared with the real data using the E_{System} and E_{User} metrics (see fig. 7.10).

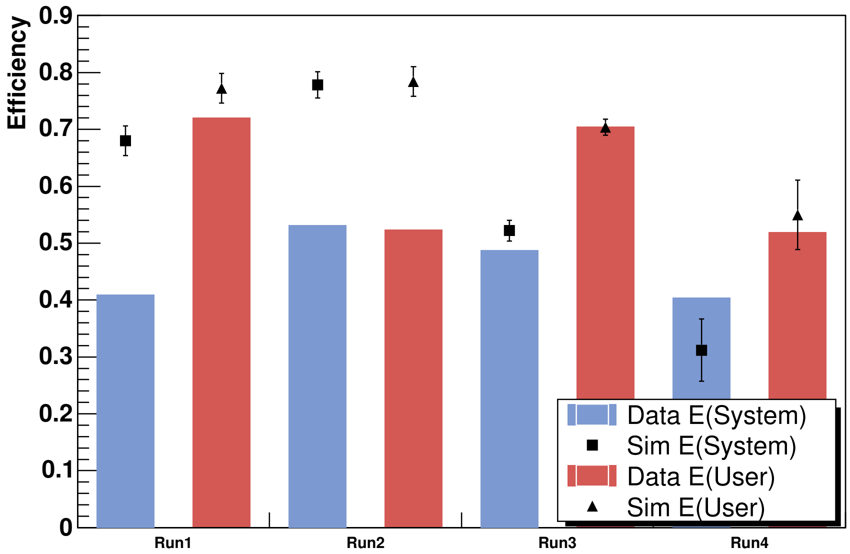

Figure 7.10: Comparison of E_{System} and E_{User} for data and preliminary simulation runs, with four different input job batch distributions: (a) 20 x 5 jobs at 90 minute intervals; (b) 10 x 10 jobs at 180 minute intervals; (c) 10 x 10 jobs at 90 minute intervals; (d) 5 x 20 jobs at 180 minute intervals.

The data results are shown by the solid histograms. It should be pointed out that these histograms indicate single runs on the real Grid, hence the lack of error bars. This is because of the unstable nature of the Grid and its middleware at the time of testing. In order to carry out a run of 100 jobs submitted in batches, two days or more of running time were often needed. During such a period the Resource Broker would sometimes need to be restarted, causing a loss of all jobs, as described in the previous chapter. Frequently on other occasions, too many jobs would be indefinitely delayed or lost at various stages in their lifecycle for the data of that run to be of use. (A cut-off of 90 out of 100 completed jobs was used to select valid runs.)

For this reason, there are no meaningful errors that can be shown for the real Grid runs. Without errors for the real data, precise comparisons cannot be made between them and the simulated runs, but they can still be used as a guide to ensure that the simulation is behaving as it should, in line with the real system. The simulated attempts to match the real runs are the corresponding data points. The error bars are the spread of values over ten simulated runs. In the case of E_{User}, the value used for each run is the mean of the individual E_{User} values per job. For E_{System}, it is the average value over the run, for the duration of that run, taken from a distribution such as that shown in fig. 7.12. A distribution of such mean E_{System} values (taken from Run 1) is shown in fig. 7.11.

The match is reasonably good for the more rapid job submission in runs 3 and 4, but for the other two runs it is poor. The simulation predicts increasing E_{System} and E_{User} for slower job submission, but it appears to stay roughly the same. A comparison of instantaneous E_{System} values during the run with 10 batches of 10 jobs at 180 minute intervals, for data and simulation, is shown in fig. 7.12.

The shapes of these two plots are very different. The simulation deals with the 100 test jobs in 1.3 days, and a regular structure can be seen in the E_{System} values as the ten job batches arrive and are dealt with (the regular drops in E_{System} showing the RB scheduling the jobs imperfectly).

In the real data, however, the structure is much less regular and distinct.

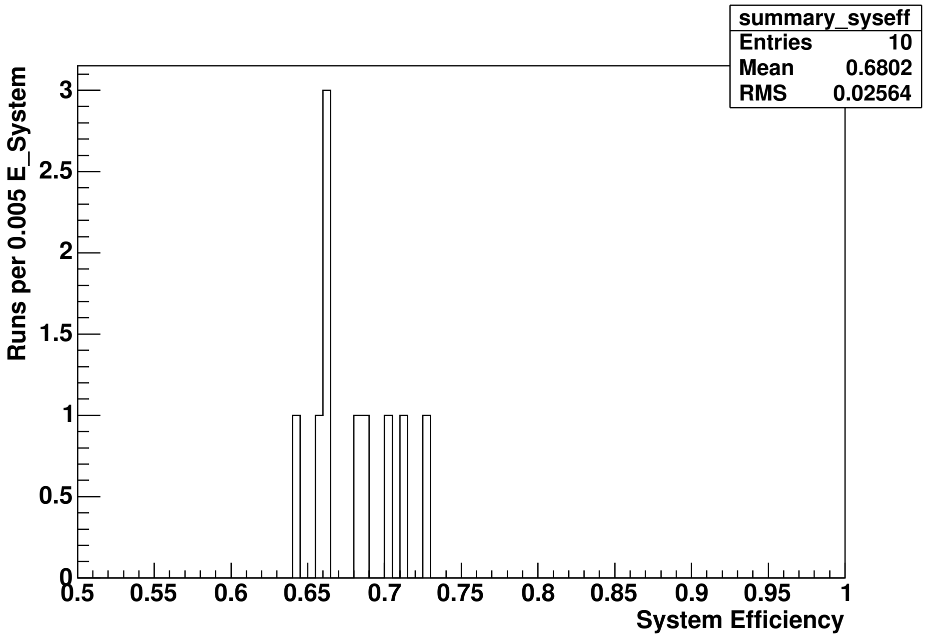

Figure 7.11: The spread of average E_{System} values over 10 runs

The E_{System} values are very uneven, with a drop to little more than zero at one point. The Last Completion Time is just over two days in the real data. Inefficiencies are being incurred in the real Grid which have not yet been factored into the simulation.

Fig. 7.13 shows the total number of test jobs (without background jobs included) at the CEs during the run on the real Grid. One might expect to see ten increases in the number of jobs at regular intervals, and then a drop off as they are eventually all run and then complete. Instead a much more erratic arrival of jobs is seen, with a large peak indicating more than twenty jobs arriving at resources at once. This explains the large drop in E_{System} 0.7 days into the run, as jobs are being delayed in the scheduling process, and they eventually arrive at the CEs some time after submission. The duration of these delays are shown in fig. 7.14.

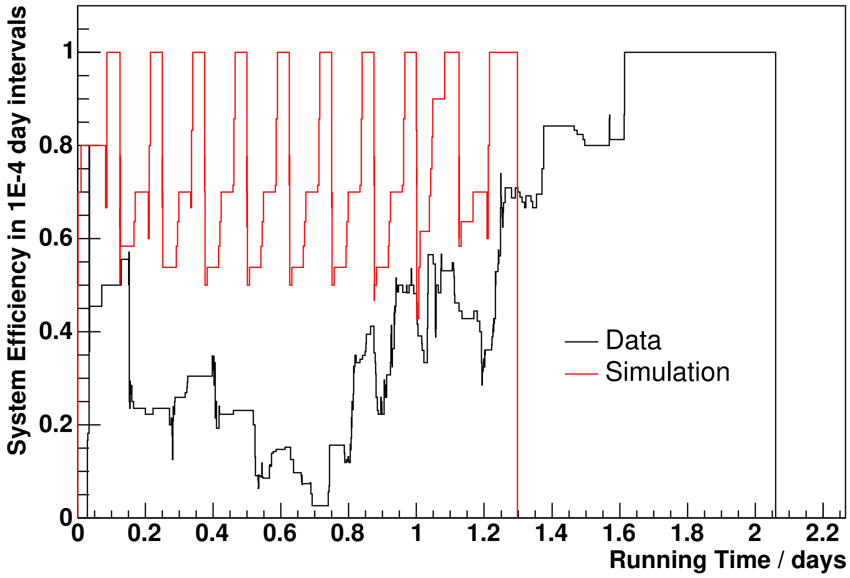

Figure 7.12: Instantaneous E_{System} during test run with 10x10 job batches at 180 min. intervals

More than half of the jobs are being delayed, some by almost a day, while they are being matched and scheduled. This explains the discrepancy between the real and simulated data, as the simulation allows for 10 seconds job matching time, and the same again to send the job to the chosen resource. This will make a significant difference to the running of the Grid - E_{System} will be reduced, as will E_{User} for the jobs held back in this way. The differing Last Completion Times are explained by the delays.

The delays incurred by all jobs in the LB database's lifetime during the matching and scheduling phase can be plotted in the same way. In fig. 7.15 these delays are shown, plotted separately for the RB and the JSS.

Significant delays are incurred in both middleware components, although RB delays are more frequent. For the purposes of this work, these delays can

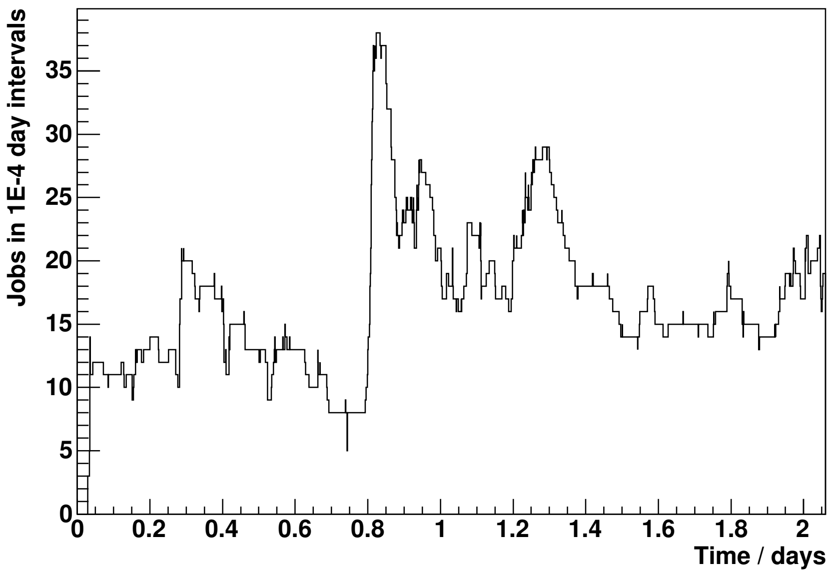

Figure 7.13: Number of jobs at GridPP sites during test run with 10x10 job batches at 180 min. intervals

be dealt with together, as an inefficiency in the brokering process as a whole.

These delays are not constant with time, and were seen to vary between test runs. In order to reproduce the effects of these delays in the simulation, a two part distribution was created. 40 per cent of jobs processed by the RB were not delayed during scheduling. The rest incurred a delay generated according to a half Gaussian distribution with a mean of zero, and an RMS given by that of the real data from the run (n.b. this RMS does not include the peak at zero for the undelayed jobs).

The resulting distribution is shown in fig. 7.16(b), with a spike near zero for the undelayed jobs, and the Gaussian delayed jobs in red. For any given run, the width of this distribution was that of the real data e.g fig. 7.16(a), with a cut for jobs delayed for less than 1000s. An example of the resulting

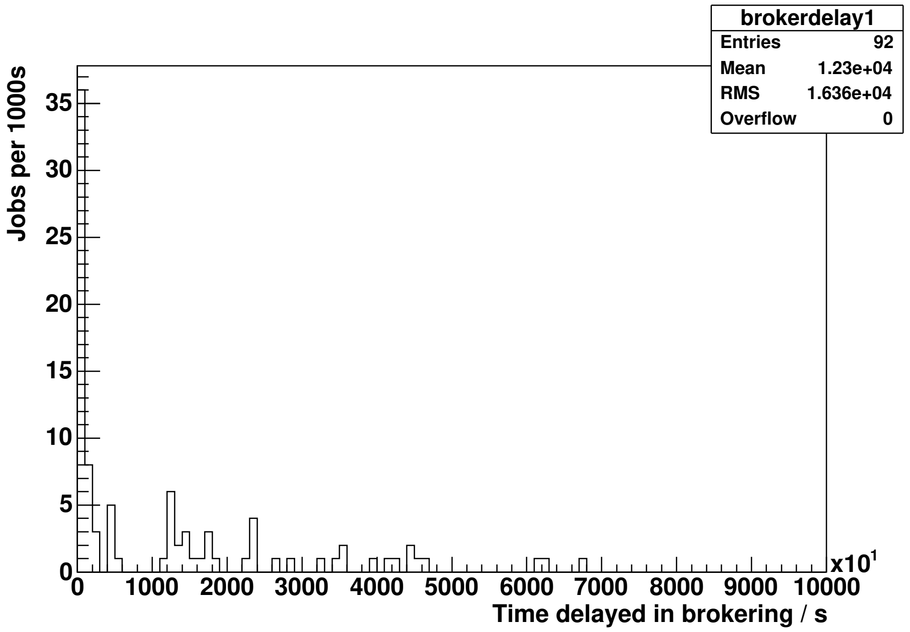

Figure 7.14: Distribution of delays in the brokering process during test run with 10x10 job batches at 180 min. intervals

simulated delay distribution is shown in fig. 7.16(c). For the runs considered here, the simulated match to the RMS of the real data delays is shown in fig. 7.16(d). The real measured values are shown by the solid bars, with the simulated values given by the points. The error bars give the spread of values generated over ten runs of the simulation.

7.6 Runs with Brokering Delays

The simulated runs described in section 7.5 can be carried out again, with the brokering delays implemented. The results are now closer to the real data. The uneven E_{System} structure for the run shown in fig 7.12 is once again compared to the simulation in fig. 7.17.

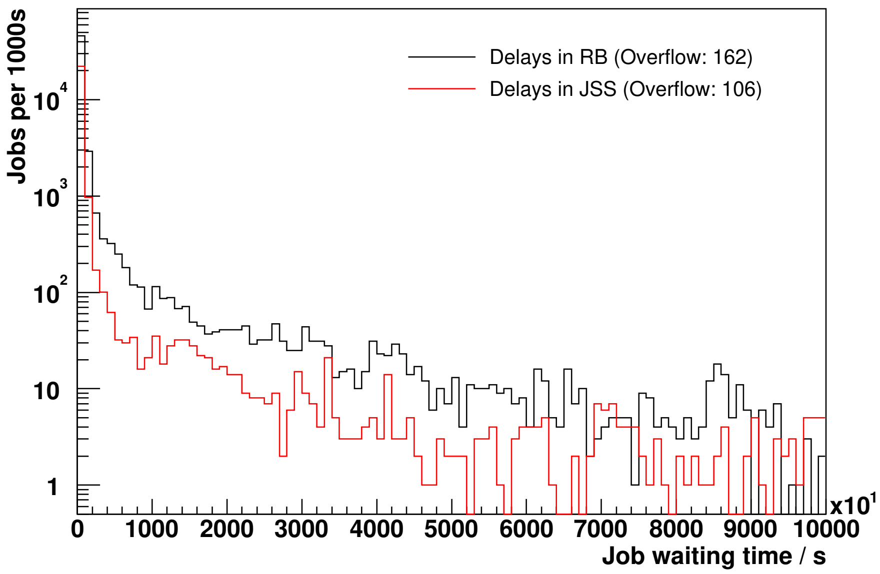

Figure 7.15: Delays in the brokering process for all jobs in LB's lifetime, showing RB and JSS delays

The shape of the simulated data are different to that of the real data, but the mean is now lower, and closer to the measured value. The discrepancy in the shape is due to complexities in the structure of the brokering delays. The decrease in E_{System} to little more than 0.02 at one point in the real data suggests that the brokering inefficiencies were larger at this point than at others. The distribution used to generate delays in the simulation remains constant during a run, producing the much less variable shape shown here.

Fig. 7.18 summarises the results of the simulated runs and compares them to the real data as before. This time the E_{System} values are much closer to those measured. E_{User} is matched less well, generally underestimating the measured value because of the simplified distribution of delays.

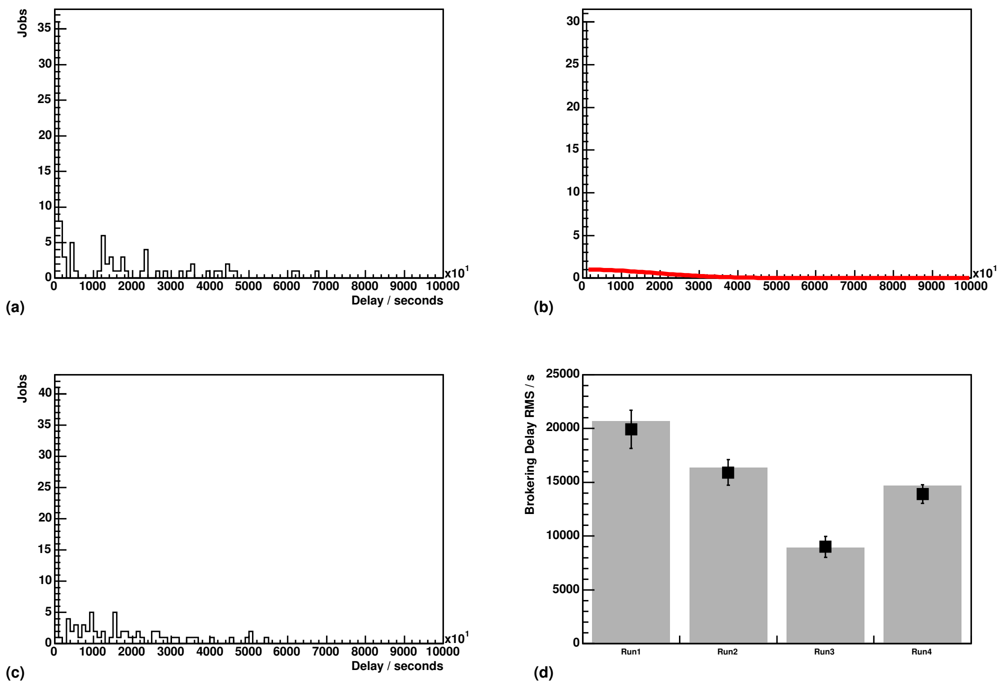

Figure 7.16: The generation of a simulated brokering distribution to match that observed in the real data: (a) delays observed during a test run from fig. 7.14; (b) the two part distribution used to generate the simulated delays; (c) a delay distribution for a simulated run; (d) comparison of brokering delay RMS between the simulation and data.

7.7 Discussion of Simulation Validation

The simulation has been shown to produce results that are broadly the same as the measured data from jobs run on the real GridPP testbed, across a range of different running conditions, despite some simplifications having been made.

The differences are largely due to the delays in brokering, as the results are

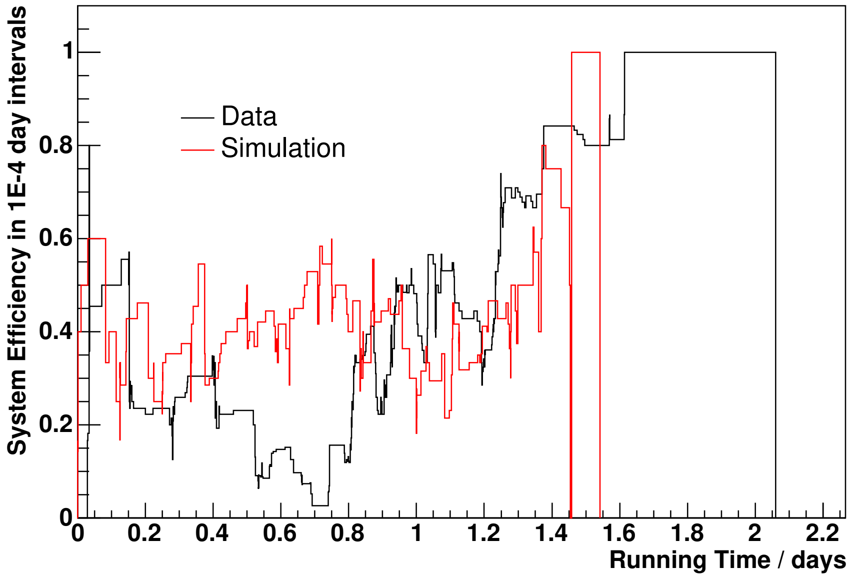

Figure 7.17: Instantaneous E_{System} during test run with 10x10 job batches at 180 min. intervals, corrected for brokering delays.

dependent upon the distribution of their duration, as well as their distribution in time. Without an extensive study of an operating RB and other associated entities the precise nature and causes of these delays cannot be determined. In any case, these delays are a product of the specific middleware in operation during these studies. While it is likely that any Grid middleware will have inefficiencies to some extent, those seen here have no long term significance for this work, and it would not be fruitful to investigate them further. The simulation's output is close enough to the measured results that it can be used to investigate alternative configurations for an EDG Grid, and how variations in certain parameters or the job workload affect the Grid's efficiency.

This simulation configuration, including the simulated brokering delays, will be used for the tests in the next chapter unless otherwise stated.

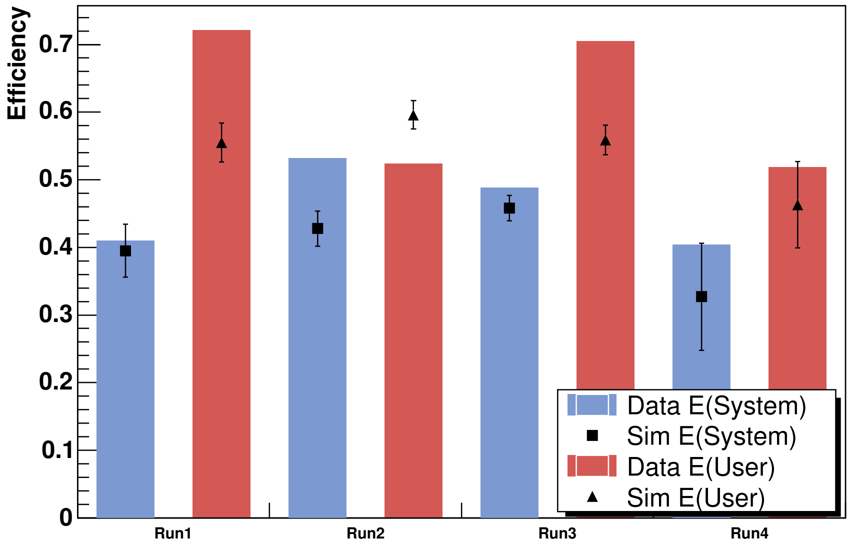

Figure 7.18: Comparison of E_{System} and E_{User} for data and simulation runs with brokering delays implemented, with four different input job batch distributions: (a) 20 x 5 jobs at 90 minute intervals; (b) 10 x 10 jobs at 180 minute intervals; (c) 10 x 10 jobs at 90 minute intervals; (d) 5 x 20 jobs at 180 minute intervals.

7.8 Summary

In this chapter the configuration of EDGSim to match job running in the real Grid was described. Parameters were set according to observations made of the real Grid, and were detailed here with reasons for the chosen values. The selection of real GridPP resources chosen to be simulated was described, as was the monitoring of these resources in order to observe the distribution of background jobs. A similar loading of background jobs, which was low and mainly consisting of short jobs, was implemented in the simulation.

Several runs of test job submissions were conducted, in batches with varying sizes and intervals, in both the real and simulated Grids. The simulated results were overly optimistic, and investigation of the real test jobs using the LB data revealed that delays in brokering were being incurred because of middleware inefficiencies. These delays were modelled and implemented in the simulated RB. New simulated runs were conducted, which produced results that were much closer to the real ones. The chapter concluded with a discussion of the implications of these delays.

Having established that EDGSim can produce realistic results, by comparing it with real data from a small testbed, it can be used to investigate more complex setups. In the next chapter, the middleware functionality modelled here is applied to larger simulated Grid testbeds, in order to investigate the effect of varying the configuration of the Grid on the efficiency of job and resource management.

Chapter 8

Scheduling for Simple Jobs

Having established the validity of the simulation, the next step is to apply it to a range of Grid scenarios. In this chapter the EDG's Estimated Traversal Time scheduling policy is tested to examine its strengths and weaknesses under different conditions, and improvements to this policy are suggested. Parameters of the simulated Grid are varied, in order to investigate their effect upon performance, from resource owner and user perspectives.

The effect of varying the interval between Information Services updates is investigated; the size of the Grid testbed, the number of Resource Brokers and the distribution of CPU resources are altered to determine the effect on efficiency; a number of different job submission strategies are used to see how this affects job processing; and the results from a realistic scheduling policy are compared with an ideal scheduler.

8.1 Information Services Update Speed

Using the simulated Grid based upon the GridPP resources used in chapter 7, the load upon the Grid was increased to the point that the resources stayed mostly busy, while remaining in the underloaded regime, by adding a heavier load of background jobs running for a period of several hours. The scheduling policy remained the same (Estimated Traversal Time, see section 3.2.2), as did the properties of the 1000 test jobs, submitted in 100 batches of 10 at 90 minute intervals. The longer duration of these tests, as compared to the comparison tests, was intended to produce a steady state of job processing once the test run was established.

The update time for the Information Services was varied between 300 seconds (approximately the real value) and a more ideal 30 seconds, in order to determine its effect on E_{User} and E_{System}. The results are shown in fig. 8.1.

As can be seen, the Grid's efficiency is uniform until the IS update time is reduced to around 100 seconds. Further reductions lead to an increase in both E_{User} and E_{System} as the RB is able to make decisions based upon more up to date information. This interval corresponds roughly to the time taken for a batch of ten jobs to be submitted, such as the simulated test batches here. Previously almost all of a job batch would be matched using the same information which becomes increasingly out of date (i.e. a resource with a single free CPU receives an entire job batch before the RB can be informed that the CE is now busy, as observed in batch submissions to the real testbed). With the availability of more current information, the RB becomes more able to make effective decisions.

This increase in efficiency is not as large as might be expected - with an IS update time of 30s, the RB should be making much more effective decisions.

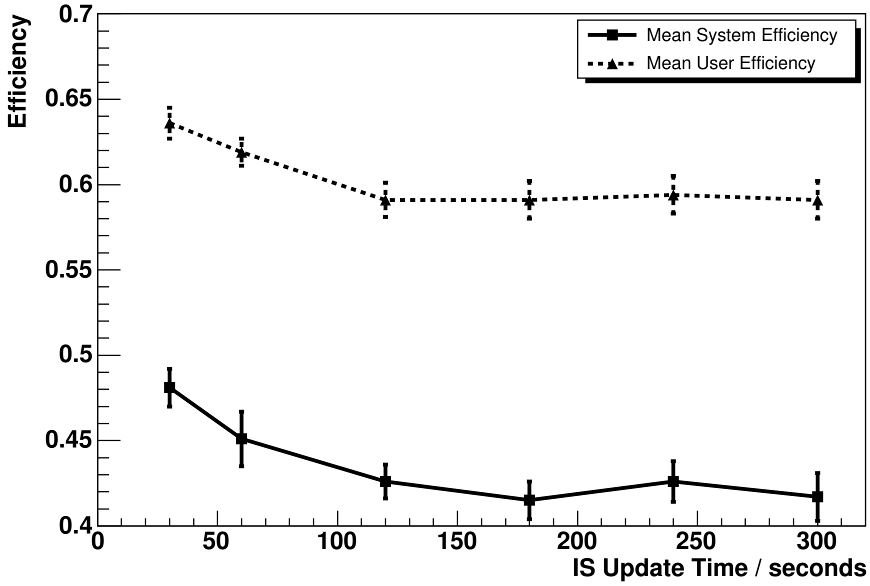

Figure 8.1: Change in E_{User} and E_{System} in the simulated Grid as the interval between Information Service updates is varied. These figures are calculated from 1000 job tests in a small but moderately loaded Grid.

This is due to the delays being incurred in the brokering process. In fig. 8.2 the same runs have been carried out again, but this time without the delays.

The improvement with the faster update times is now much greater. The RB's ability to make scheduling decisions is no better than before, but now the implementation of those decisions is more effective.

8.2 Scalability of Scheduling Policy

Tests were carried out with simulated Grids of different sizes in order to investigate the effectiveness of scheduling as the availability of resources is restricted. The Grid consisted of homogeneous CEs of 10 CPUs each, from

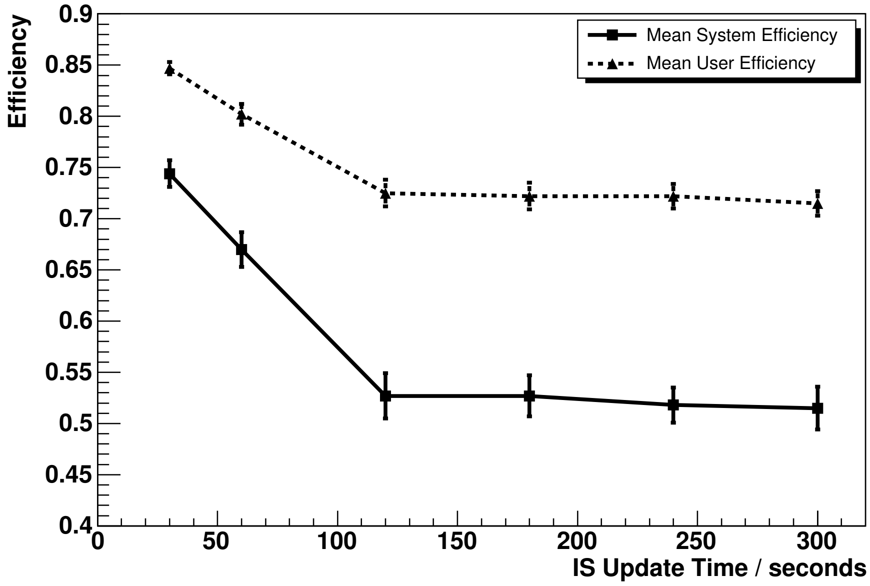

Figure 8.2: Change in E_{User} and E_{System} in the simulated Grid as the interval between Information Service updates is varied, as with fig. 8.1, with no RB delays.

a maximum of 400 CPUs in total down to 100 CPUs. The background job load was kept constant between runs, with a combination of very short jobs, and longer ones running for a period of hours. 100 batches of 10 test jobs were submitted at 900 second intervals. The total job load reached a plateau at just under 250 jobs after a day or so of simulation time (in the larger Grid configurations with more CPUs than active jobs). The Information Services update time was 300 seconds.

These tests were carried out with two scheduling policies. The first was the default EDG policy of Estimated Traversal Time. The second was a modified version of this. In standard ETT scheduling, the RB will make an arbitrary choice in the case of a tie between CEs, for instance several underloaded CEs with an ETT of zero. The RB's list of tying CEs is fixed in order, and it always chooses the first entry in this list. Thus in an underloaded Grid, all jobs will be sent to the same site, rather than being shared between them. This modified policy (Randomised ETT) instead makes a random choice from the list in the case of a tie. The results of the two sets of runs are shown in fig. 8.3.

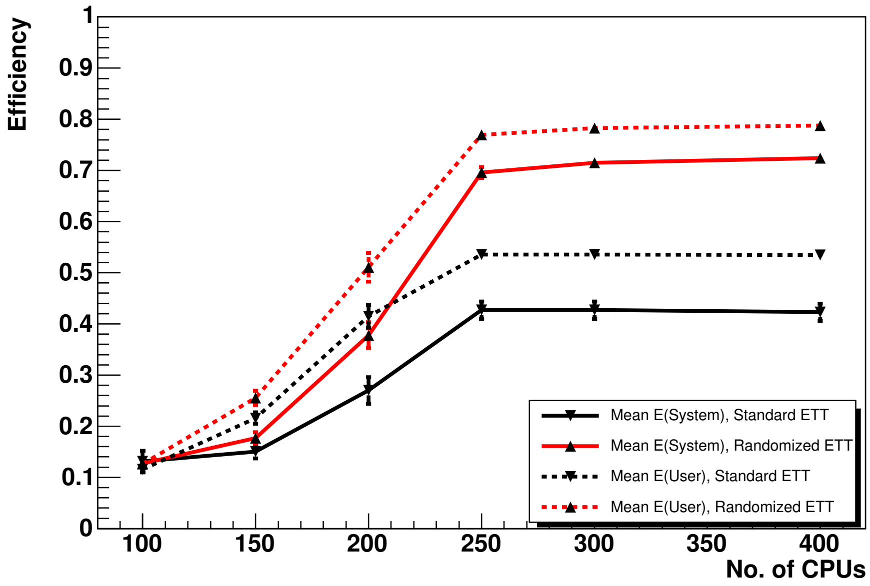

Figure 8.3: Change in E_{User} and E_{System} as a fixed input job load is submitted to Grids with differing numbers of resources. The measurements were carried out using the standard Estimated Traversal Time scheduling policy, and a modified version.

In both cases, E_{User} and E_{System} remain constant for larger numbers of CPUs. In these cases, the randomised ETT scheduler does much better than the standard algorithm, as it compensates for the uneven loading caused by CE status information being out of date. Efficiency starts to drop once there are fewer CPUs than jobs, and the system enters an overloaded state.

The performance of the two schedulers converges, as with loaded CEs there are fewer ties in the comparison of ETT values, and the advantage of the randomised algorithm is lost.

This difference is a significant one for the real GridPP testbed described in the previous chapter. For most of the time during monitoring, the Grid was in an underloaded state, meaning that the scheduling policy was operating inefficiently for any batch submission of jobs. Job submission records showed that the RB repeatedly sent jobs to a single site even when others were idle, only targeting other sites when the IS data were updated, or the site in question stopped responding as it attempted to cope with the workload.

8.3 Multiple Resource Brokers

Until now, the simulated Grid has had only one RB to make all of the decisions. In a large scale production Grid such as the EDG, multiple RBs will share the job load, and will all be able to submit to the same resources. Using a Grid like that of the largest in section 8.2 with 40 identical CEs, the same job load was tested with different numbers of RBs. Five simulated Grid users simultaneously submitted batches of 10 jobs at 180 minute intervals; the total job load was moderately heavy but less than the total capacity of the Grid. The number of RBs was varied from one to five (i.e. an RB per test user). Both the standard ETT scheduler and the modified version were used. The results can be seen in fig. 8.4.

There appears to be an optimal number of RBs for this Grid setup, with 3 brokers giving the best performance in terms of both E_{System} and E_{User}. Up to this point, adding more RBs to the Grid shares the job load between them, allowing the jobs to be scheduled more efficiently. Adding a fourth RB de-

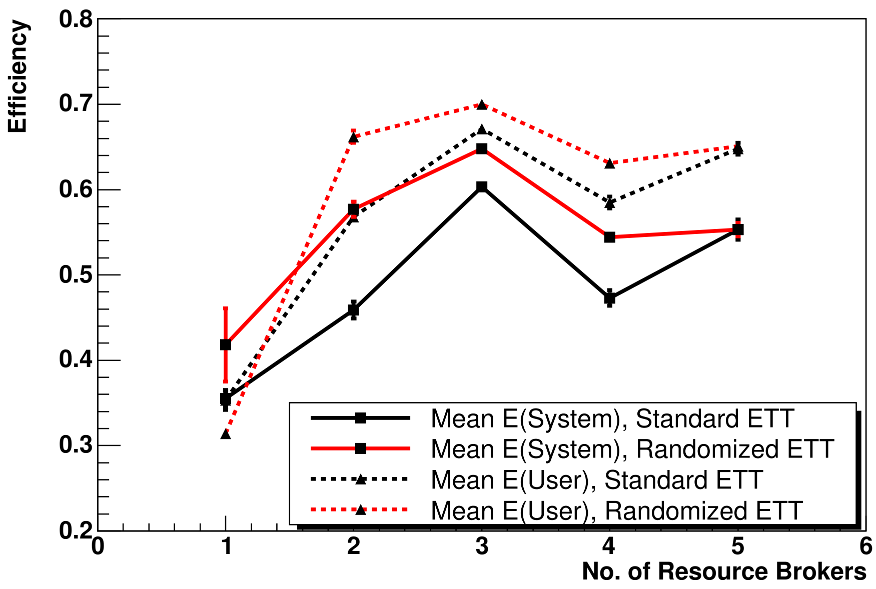

Figure 8.4: E_{System} and E_{User} for varying numbers of RBs, with standard and randomised ETT scheduling.

creases the performance, as the inefficiencies of the brokers are compounded. They are all making bad decisions based upon the same obsolete information supplied by the Information Services, so more jobs are inefficiently scheduled before the next IS update can arrive.

Efficiency increases again for five RBs. This is the case in which each user submitting test jobs effectively has their own RB to handle their job requests. This more symmetric system gives a better performance, although not as good as the 3 RB set up.

Again the randomised ETT scheduler is more efficient than the standard version. The behaviour of E_{System} and E_{User} is similar over the range of RB configurations.

8.4 Scheduling for Heterogeneous Grids

Two different simulated Grid distributions were compared, with the same total no. of CPUs (400): the homogeneous Grid with 10 CPUs at each of 40 resources, as before; and a heterogeneous Grid, with CPUs assigned as described in table 8.1. The input job distribution was as in section 8.3, and the number of RBs was again varied, using both the standard and randomised ETT scheduling policy. The results are shown in fig. 8.5.

No. CEs	1	2	10	12	5	10
No. CPUs each	100	50	10	5	4	2

Table 8.1: Distribution of CPUs in heterogeneous Grid.

Only E_{System} is shown here, as once again the behaviour of E_{User} is similar. The results for the homogeneous Grid are those from section 8.3, shown again for comparison alongside those from the heterogeneous Grid.

The performance of both scheduling algorithms is generally better for the homogeneous resource distribution as might be expected, since assigning jobs across an asymmetric Grid is a more difficult task. The schedulers have no knowledge of the number of CPUs belonging to the CEs, and make their decisions based upon observation of the CEs' performances only.

The notable exception is in the case of two RBs scheduling jobs for a heterogeneous Grid. There is a large peak here for the randomised scheduler, and it gives a significantly better performance than in most of the other scenarios, even outperforming the randomised scheduler for the homogeneous layout. This is where the ETT part of the algorithm shows its strength - once the first job results begin to return, it is able to determine that the site with

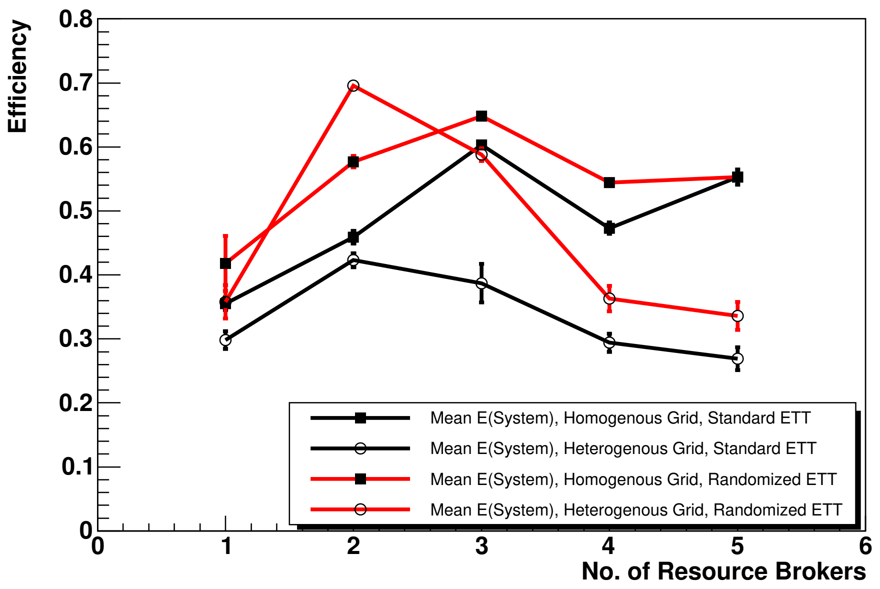

Figure 8.5: E_{System} for varying numbers of RBs, with standard and randomised ETT scheduling, with homogeneous and heterogeneous Grid topologies.

100 CPUs is able to process the larger share of the jobs. The standard ETT scheduler is also able to determine this, but by the time it does so, it will already have made a number of bad scheduling decisions, and its performance overall is less good.

The optimal number of RBs appears to vary for different distributions of the same number of resources, 3 for a homogeneous Grid and 2 for a heterogeneous Grid. This may be because load balancing is more difficult in the heterogeneous case, so brokering conflicts appear more rapidly. There is no increase in efficiency for 5 RBs in the heterogeneous case as there is with the homogeneous one, although the fall in efficiency slows slightly.

8.5 Job Submission Strategies

Similar input job distributions were used in each of the simulation tests described above. These distributions were now varied, to see how this would affect the efficiency of job processing. Again using the 40-node homogeneous Grid, jobs were submitted in batches of 10 as before, but with varying CPU time requirements, numbers of batches, and time intervals between batches. The total CPU time needed for all test jobs in each run was the same, as was the time period over which job submission occurred, i.e. the more numerous batches of smaller jobs were submitted more rapidly to compensate. The E_{System} and E_{User} values for these runs are shown in fig. 8.6, using the standard and randomised ETT scheduler.

Intuitively, one might feel that splitting the job load up should improve performance, but these results appear to show that the opposite is true. E_{System} falls off as the number of batches increases (and the size of individual jobs decreases). This is due to the inefficiencies in the RB, such as those relating to slow IS updates as in section 8.1. As it has to deal with more and more jobs, the effects of these inefficiencies become larger. With small jobs, the system performance is very poor. With the larger jobs the randomised ETT scheduler performs better than the standard version, but this difference almost disappears with a larger load of small jobs. This is because the randomised scheduler only has an advantage when dealing with an underloaded Grid. With many small jobs running on most of the Grid nodes, this advantage is lost.

E_{User} also falls, although not to the same extent as E_{System}. This is partly due to the reasons given for the fall in E_{System} above, but the drop may also be showing a shortcoming in E_{User} as a measure of efficiency. As shown in

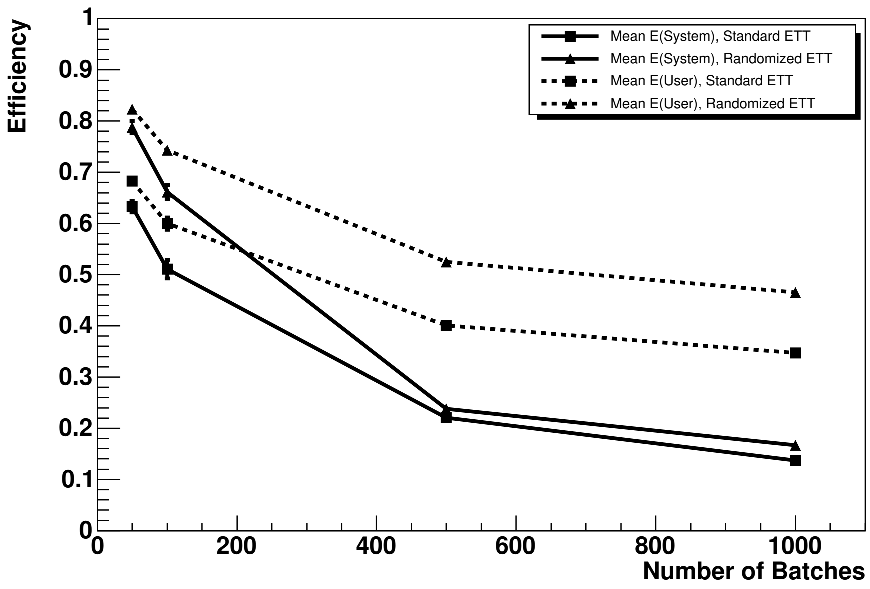

Figure 8.6: E_{System} and E_{User} for job loads submitted in varying numbers of batches of 10 jobs each, using standard and randomised ETT scheduling: 50 batches of 40000s jobs; 100 batches of 20000s jobs; 500 batches of 4000s jobs; and 1000 batches of 2000s jobs.

equation 6.1, it is calculated as a ratio of the running time of the job with its total lifetime. This causes problems when comparing E_{User} for jobs of different running times. Jobs with a shorter duration will give a lower value for this metric, compared with a job that took the same time to be scheduled and queued, but ran for longer.

In order to look at the results from the user's perspective in another way, the metric Last Completion Time is used, measuring the total time taken to run all jobs (see section 6.4). This value for the runs described above is shown in fig. 8.7.

This plot shows that despite the fall in the other two metrics as the

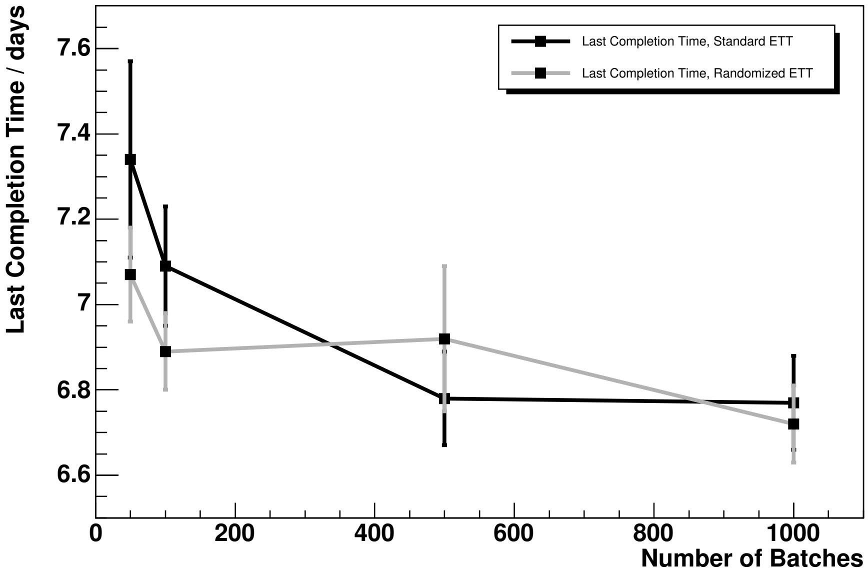

Figure 8.7: Last Completion Time for job loads submitted in varying numbers of batches, using standard and randomised ETT scheduling, as in fig. 8.6.

job load is split up, the user will actually benefit as their jobs complete more quickly, with bad scheduling decisions becoming less important when applied to smaller jobs. This adds weight to the suggestion that E_{User} is less useful as a metric when comparing jobs of different duration. However, the benefit does not appear to be very great, with an improvement of less than 10% with E_{System} being degraded to unacceptably low levels.

This is due to the simulated inefficiencies in the RB, as seen in the performance of the real RB. In some cases, a job may be delayed for as much as a day before it is even sent to a CE to be run. This means that the Last Completion Time will be distorted, if even one job is held back in this way. Delays of this kind also account for the large errors on the points in fig. 8.7, as they occur in some runs and not in others.

8.6 Ideal Scheduling for Simple Jobs

In order to give a reference point for the efficiency figures given here, the ideal scheduling approach described in section 6.5 can be used, minimising L_{CPU}. The simulated runs carried out in section 7.6 were repeated, but using the ideal scheduling algorithm rather than Estimated Traversal Time. The RB is able to use the most current information (i.e. there is no problem of IS update time), and knows the CPU requirements of the incoming job, in contrast with the real scenario. The results are shown in fig. 8.8.

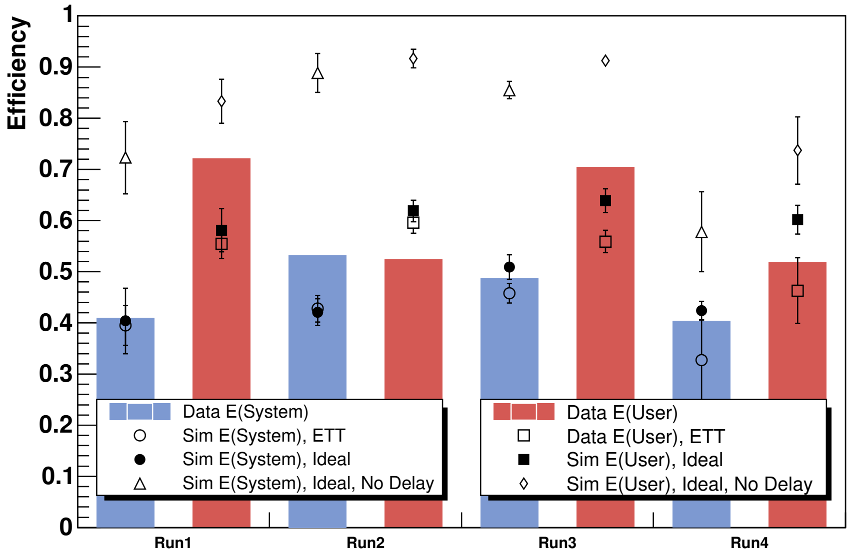

Figure 8.8: Comparison of simulation using ideal scheduling with real data from chapter 7. The real data are represented by the solid bars; the points represent the simulation with Estimated Traversal Time scheduling, with ideal scheduling, and with ideal scheduling when no delays are incurred in the brokering process.

The improvement supplied by the supposedly ideal scheduler is negligible or non existent for the first two runs shown here, with less rapid job submission rates. A slight improvement appears in the third and fourth runs, as the RB struggles to cope with so many jobs using out of date information.

The reason why no significant improvement is made is the delays incurred in the RB and JSS. The RB is able to make more effective decisions with the ideal scheduling regime, but the delays interfere with their implementation. The runs were carried out once more, but with brokering delays switched off. The results are also plotted in fig. 8.8.

This time, the improvement in efficiency is much larger, particularly for the less rapid job submission runs. This again highlights the problems caused by delays and inefficiencies at this stage of job processing.

8.7 Summary

In this chapter a number of parameters in the simulated Grid were varied, in order to test their effect on efficiency measures. Reducing the IS update time from 300 to 30 seconds produced an improvement in both E_{User} and E_{System} of around 10%; this increased to 25% for E_{User} and 50% for E_{System} without the delays in the RB investigated in the previous chapter.

The EDG's Estimated Traversal Time algorithm was tested against a modified version, with varying sizes of Grid testbed under similar loading times. For the larger (and thus less busy) Grids, a 50% advantage in E_{User} and E_{System} was seen for the randomised ETT algorithm, with this advantage diminishing for the smaller testbeds.

Both algorithms were tested with varying numbers of RBs scheduling a heavy job load. The randomised ETT algorithm again performed more effectively in terms of E_{User} and E_{System}, and an optimum number of RBs was found, with 3 RBs giving a much better performance than just 1 (60% or more E_{System}, 100% E_{User}). More RBs than this decreased efficiency, although a system in which each user had their own RB produced a slight improvement.

Similar results were observed when tests were carried out with a heterogeneous CPU distribution across the Grid. In general, scheduling is more problematic than for a homogeneous Grid, but the randomised ETT scheduler continues to produce better results than the standard version. The modified Grid structure results in a reduction in the optimal number of RBs to 2.

Different submission strategies were compared, with the same CPU-hour load divided into different numbers of jobs. A small improvement (10%) was seen in job processing time when the workload was divided into many small jobs, but E_{System} drops to negligible levels. This variation in job running times highlights a problem with the E_{User} metric.

Finally the simulation validation runs from the previous chapter were repeated using an ideal scheduler, minimising job running time. Only small improvements in performance (around 5% for E_{User} and E_{System}) were seen unless the simulated RB delays were switched off, in which case improvements of 50% or more were seen, highlighting the problems caused by middleware inefficiencies.

Despite these inefficiencies, improvements in performance can be made with a simple modification to the ETT scheduling algorithm. By introducing a random choice in the case of deadlock, jobs are distributed more effectively between resources. This is especially true in the case of low job loading.

Modifying the regularity of Information Service updates seemed to produce only small improvements in performance, because of the middleware inefficiencies, and dividing the job load into smaller jobs was ineffective for similar reasons. The introduction of extra RBs can significantly improve efficiency, but there appears to be an optimal number of RBs beyond which efficiency drops again. The dependency of this optimal number on the setup of a particular Grid could be clarified by further work. In all cases investigated here, E_{System} and E_{User} could be optimised simultaneously.

Chapter 9

Data Dependent Jobs

Simulating a fully functioning Data Grid is a much more complex task than the simple CPU-only models described in the previous chapters. As well as finding appropriate machines to run CPU bound jobs, the middleware must now make sure that jobs have access to the data that they require in order to carry out their analysis. These data may not all be found at the same site, making it necessary for files to be transferred, creating a new replica. In some cases it may be best to send the job to a site with the required data; in others it might be preferable to move data to a less busy site. The problem of scheduling becomes a multidimensional one, and there will not necessarily be a clear-cut way of determining the best scheduling choice.

A solution for this data management problem is to copy data between sites independently of specific requests for files by jobs. Unlike the RB's job scheduling algorithms discussed so far, this will require predictive behaviour, copying certain files between sites based upon their popularity. This can be controlled at the site level by the Storage Elements. Allowing the SEs to determine when it is best to create new replicas decentralises control of data management, increasing reliability.

This chapter outlines the management policies that will be used in simulation of data dependent job processing. Other Data Grid simulation work is reviewed. The modification of existing performance indicators is explained, new metrics are defined to account for the more complex system being simulated. The four scheduling algorithms to be used are described; and two replica management policies, and two preemptive replication policies are introduced.

9.1 Other Data Grid Simulations

This section will briefly outline other work that simulates a Data Grid structure, highlighting features relevant for the further development of the simulation described here.

9.1.1 OptorSim

OptorSim [45] is the project closest to this work, as it is specifically based upon the EDG middleware. It has been developed by members of EDG Work Package 2, which is responsible for replica management and optimisation, and the emphasis is on this area. The network model is very sophisticated, representing bottlenecks and competition for bandwidth. OptorSim can also introduce background network activity, modelled on daily variations measured over a real network.

Simulated sites can have Compute Elements, Storage Elements or both, and the sites are connected to routers which make up the structure of the network. CEs run one job at a time, i.e. they have the equivalent of one CPU each. Jobs are data dependent, requiring one input file each to run. The file required by the job can be chosen according to one of several probability distributions, reflecting the non-uniform popularity of datasets in HEP analysis.

The favoured access pattern for OptorSim is the Zipf distribution, as World Wide Web file access has been modelled in this way [53]. It is defined to be:

$$\mathrm{P}_i \propto i^{-\alpha} \tag{9.1}$$

P_i is the frequency of requests for the i^{th} ranked file in the dataset, and $\alpha < 1$. α is set to 0.7, which results in a small number of files being requested frequently (i.e. small values of i), and a long tail of rarely requested files.

Replica management is carried out locally, so that control of replication is distributed, and decisions are made according to an economic policy, described in [53]. Each data file has a value based upon its popularity, and the more a particular file is accessed, the more profitable it is deemed to be for the SE hosting it. When a file is required by a job at a CE, the local SE must make a decision as to whether the file is copied across and a replica stored locally, or if the job must access the file remotely instead. If the file is judged to be more profitable than a currently resident replica, (according to a prediction function based on previous data requests), the existing replica will be deleted, and the new file will replace it.

Economic thinking is also employed when determining which replica of a requested file is transferred. The site requiring the file initiates an auction, in which sites possessing the file make bids to supply it, based upon current network conditions. The winning bidder will be the site that is able to supply the file the most quickly. In addition, sites can bid even if they do not currently own a copy; they can initiate a secondary auction to acquire the file themselves, and then make a bid in the original auction. In this way, replication of popular data files can occur independently of direct requests by jobs.

Job scheduling has also been added to OptorSim in recent work. The RB can make scheduling decisions based upon CE loading data. It can also access network status information, and determine the CE which will be able to access the file most quickly.

9.1.2 ChicagoSim

Although not created with the EDG specifically in mind, ChicagoSim [44] is a simulation of a high energy physics Data Grid. The structure is similar, with Compute and Storage Elements managing local resources, and an External Scheduler that decides how jobs are assigned. CEs can have multiple CPUs, and multiple External Scheduler entities can be implemented, but the network model is less sophisticated than that of OptorSim.

Jobs also have a single file dependency in this model, and the files are again chosen according to a probability distribution. Replica management is carried out at the local level, but the policies used are somewhat different. Instead of monitoring the popularity of datasets across the Grid, the Replica Managers instead keep a record of the popularity of the replicas stored at the local SE. When the job load at the site's CE passes a certain threshold, replication of the most locally popular files is triggered.

In [54] ChicagoSim is used to investigate how well a range of scheduling and replication policies complement each other. The scheduling policies vary in sophistication, with decisions made on site loading or data location. The latter category produced the best results when combined with independent replication. Perhaps surprisingly, the choice of replication policy made little difference. The performance of the system was almost identical whether the files were copied to a site chosen for its relatively low job load, or entirely at random.

9.1.3 Bricks

Data Grid simulation has been carried out using an extension of the Bricks Grid simulator [55]. The emphasis here is slightly different, concentrating more on the topology of the Grid than the management of jobs and data. This work has also been carried out with LHC applications in mind, and the middleware framework is similar to those described above. Comparisons are made between a fully Grid-like distributed network of computational and storage resources in a Tier structure as described in section 2.1, and a large centralised computing farm with less resources but none of the overheads incurred by the former setup.

Different types of HEP job (e.g. reconstruction and analysis) are represented here, occurring with different frequencies and with different resource requirements (size of input dataset, running time). It is assumed that these tasks can be parallelised and split into individual processes for each input file, resulting in a single file dependency for each job.

Several scheduling algorithms are used in the distributed case, based on loading and data considerations. The replication policies are similar to the ones used by ChicagoSim above, but they are implemented by a global replica manager rather than local managers at each site. While this would make it easier to coordinate data usage information over the Grid as a whole, the decentralised approach is likely to lead to greater stability, as it removes the single point of failure. As Work Package 2 is developing a system of local replica management (as shown by their investigations with OptorSim) for the EDG, it seems more relevant to concentrate on the decentralised model for work with EDGSim.

9.2 Metrics for Data Dependent Jobs

A simulated Data Grid is significantly more complex than the CPU-only models considered so far. Previously, only computational resources were involved in job running. Now there are storage resources that must be managed, and limited network bandwidth to consider. The use of these resources cannot be treated separately, as jobs may need to use all of them in order to run. A measurement of system performance must take all of these factors into account.

The performance indicators used so far will still be useful, although they may require modification for use in this new context. In addition, they will need to be complemented by additional metrics relating to these other resources. It is not straightforward to define a single metric describing a universal efficiency measure for resource usage, as different types of resource are only loosely coupled, and it may not be possible to optimise use of all resources simultaneously. In order to study this complexity, several metrics providing different views of a Grid model's running will be used.

9.2.1 Modifying User and System Efficiency

The results in section 8.5 indicate that E_{User} has weaknesses as a metric for job performance under certain conditions. These problems occur under two conditions: when the CPU speeds of the Grid are heterogeneous, and when the jobs themselves have different demands for CPU cycles. The former case did not seem to apply to a significant degree for the testbed studied in chapter 5, and the homogeneous CPU speeds simulated here mean that this issue can be ignored here.

The second problem will apply when a user breaks up a job load into differently sized sections. In the context of data dependent jobs, this corresponds to the fraction of a dataset assigned to individual jobs, where those jobs in combination make up a larger project. As equation 6.1 shows, E_{User} has a dependency on job lifetime. This means that jobs with a shorter running time will seem to perform less well than a longer running job that has queued for the same amount of time.

In order to compensate for this, E_{User} can be modified for use in the context of data dependent jobs. Under the existing definition, E_{User} can be expressed as follows:

$$\begin{aligned} E_{User} &= \frac{\text{Running Time}}{\text{Total job lifetime}} \\ &= \frac{\text{Running Time}}{\text{Scheduling Time} + \text{Waiting Time} + \text{Running Time}} \end{aligned} \quad (9.2)$$

Note the use of the phrase "Waiting Time" rather than "Queueing Time", as this should now include periods when a job has been assigned to a CPU, but is waiting for input data to arrive. Assuming that a job's running time is proportional to the number of files that it consumes, where all files are the same size as in this simulation, E_{User} for data dependent jobs is as follows.

$$E_{User} = \frac{\text{(Running Time / No. Files)}}{\text{Scheduling Time} + \text{Waiting Time} + \text{(Running Time / No. Files)}} \tag{9.3}$$

By normalising the job running time in this way, the problem can be avoided. This should make it possible to compare the results from runs with jobs of differing duration.

E_{System}, as defined in equation 6.3, is still a useful metric for efficiency of CPU use. However, it needs clarification in the case of data dependent jobs. The numerator of the E_{System} expression is a quantity described as "CPUs Delivered", which is straightforward in the case of jobs with no data requirements, as it is simply the number of jobs assigned to CPUs. A data dependent job may be assigned to a CPU, but not be running at a particular moment, as its input data may not yet be available. For the purposes of E_{System}, this should not be counted as a CPU delivered until the data arrive and the job can begin running.

9.2.2 Average Response Time

Another metric that measures efficiency from a user's perspective is Average Response Time, used in evaluating performance in other simulation work (e.g. [54],[55]). The response time for a job is:

$$T_{Response} = \text{Scheduling Time} + \text{Waiting Time} + \text{Running Time} \tag{9.4}$$

In other words, it is the total job lifetime. The mean $T_{Response}$ is a useful indicator of how well a user's jobs are being executed. However, it has a dependency on the running time of the job, as well as any overheads incurred due to matching for input data, and delivery of these data. It should only be used in comparisons of similar Grid scenarios (e.g. comparisons of the performance of different algorithms with the same Grid setup and input job distribution).

9.2.3 Network Efficiency

A metric will also be needed here to describe how efficiently the network resources are being used. There is no meaningful way to express this as an instantaneous value, in the same way that E_{System} gives the proportion of CPU resources being used at a particular moment. It is not necessarily ideal to minimise network usage, as some data must be transferred in order that jobs can have access to the files that they need.

This problem can also be viewed in terms of the number of file accesses made or requested during a run. In [53] a quantity called Effective Network Usage is defined as follows:

$$\mathrm{r}_{ENU} = \frac{\text{No. Remote Accesses } + \text{ No. Replications}}{\text{No. Local Accesses}} \tag{9.5}$$

This quantity will be smaller in the case where the number of local file requests is proportionally larger than the number of remote requests. This is a useful way of determining the efficiency of network use, but it will need to be modified slightly to fit the form of E_{User} and E_{System}. An idealised efficiency of 1.0 could be defined as a situation where all file requests can be handled locally, without the need for file transfer to occur. Network Efficiency is then defined as:

$$\mathrm{E}_{Network} = \frac{\text{No. Local Accesses}}{\text{Total File Accesses}} \tag{9.6}$$

As with the other metrics defined in chapter 6, a perfect efficiency measure of 1.0 is very difficult or impossible to attain here. If all files are initially stored in one location, such as CERN in the case of high energy physics experiments, then all data will have to be replicated at least once to spread them around for load balancing purposes. Jobs requesting multiple files are likely to require further replication, as the members of a requested dataset may well be spread over several sites.

9.3 Scheduling Data Dependent Jobs

Scheduling data dependent jobs is now no longer simply a matter of satisfying user requirements while balancing the job load between the available computational resources. A successful scheduling algorithm will also be able to minimise the number of transfers, as well as the time spent by idle jobs waiting for data to arrive. This section will describe the algorithms to be tested in Data Grid scenarios.

9.3.1 Estimated Traversal Time

The benchmark to which the performance of more sophisticated algorithms can be compared is the minimum Estimated Traversal Time algorithm described in section 3.2.2, using the randomised choice between equally ranked resources as described in section 8.2. This algorithm performed well for jobs without data dependency; however it will be unaware of network and data considerations, and any algorithms designed to handle data dependent jobs should be able to outperform it when these become significant factors.

9.3.2 Most Files + ETT

This is intended to be the most simplistic data-aware scheduling algorithm, and it is a simplified version of an approach taken in [54]. It has no awareness of the status of the network, and relies only on information about computational resources, and replica locations supplied by the RC.

Two quantities are taken into consideration - the Estimated Traversal Time of the CE, and the number of requested input files stored locally to it. A good candidate resource will minimise the former, and maximise the latter. For this algorithm, they are given equal weighting. The RB will choose the resource that minimises the following:

$$\mathrm{M}^{i}_{Data+ETT} = \frac{\mathrm{F}^{Max} - \mathrm{F}^{i}}{\mathrm{F}^{Max}} + \frac{\mathrm{ETT}^{i}}{\mathrm{ETT}^{Max}} \tag{9.7}$$

Here F^{Max} and ETT^{Max} are, respectively, the most required files at any one site, and the largest ETT value given by a candidate CE. F^{i} and ETT^{i} are the values of these quantities for site i.

9.3.3 ETT + Queue Access Time

The algorithm that was found to give the best results by the OptorSim project uses a combination of estimated queue traversal time, and a quantity called Queue Access Time, described in [53]. In order to determine this quantity, the Resource Broker must be able to take advantage of network monitoring services.

The Access Time for a job that has been assigned to a particular site is the total time that will be required to transfer any required data that are not already stored locally to that resource. The Queue Access Time is the sum of the Access Times for all jobs queued at the resource. If there are multiple replicas of a file required by a job, the time for the replica that can be accessed the most quickly will be used, as it is assumed that this replica will be copied to the site.

The RB then calculates the Access Time for the new job if it were assigned to each of the available resources, resulting in a new total for each site. The best site is the one that minimises Queue Access Time plus Estimated Traversal Time:

$$\mathrm{M}_{QAcc+ETT} = \text{Queue Access Time } + \text{ New Job Access Time } + \text{ ETT} \quad (9.8)$$

9.3.4 Adaptive Scheduling (DataLoad)

The previous algorithms apply the same policy to all incoming jobs, regardless of the current conditions of the Grid. However, the best choice for a job may vary depending on how busy the Grid is at the time of submission, as well as the availability of replicas of the data requested by the job. For instance if the job requests files that are not available at any SEs close to a CE that can run the job, then an algorithm that makes decisions based upon data location will be less useful.

The DataLoad algorithm devised for this project, shown in fig. 9.1 is designed to be adaptive, and respond to Grid status updates. It checks the number of requested files available at SEs with a local CE, the free cache space available at the SEs, and the current loading of the CEs. Based upon this information, it chooses a scheduling policy. Its preference will be for a CE with a free CPU plus at least some of the requested data. If this cannot be fulfilled, a CE with a free CPU and some free cache space will be selected. Failing that, the job will be assigned to the CE with the smallest job load.

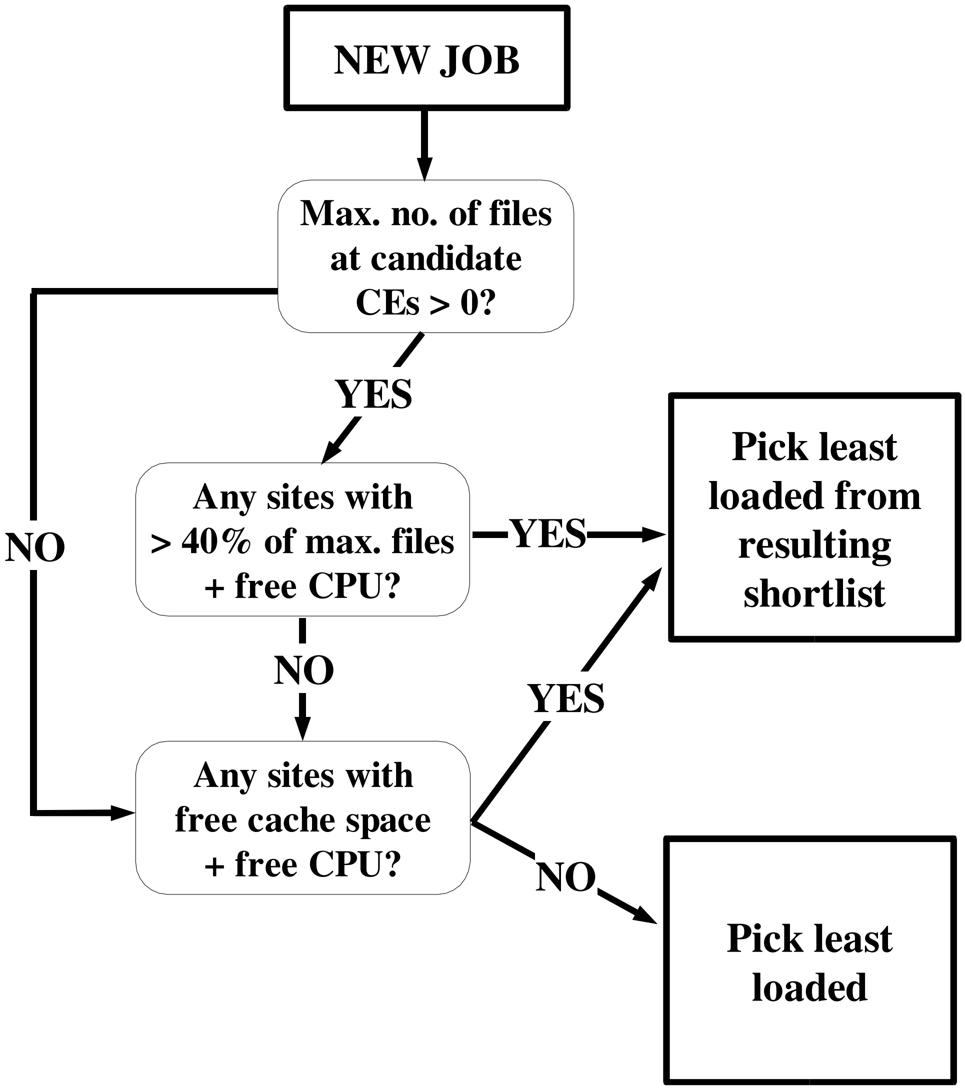

Figure 9.1: The DataLoad adaptive scheduling algorithm, which modifies its approach to scheduling depending on current loading, and distribution of the requested dataset.

9.4 Replica Management and Dynamic Replication

A job scheduling policy can only do a limited amount to ensure that data transport and network usage are efficiently handled. At most they can minimise the amount of replication carried out for specific job requirements, copying data only when it is not practical to move the job to the location at which it is stored. In order to minimise unnecessary file transfers, a replica management policy for SE caches is needed. Data can even be transferred independently of job requests by a system of preemptive replication based upon observation of the number of requests for files.

This section describes two replica management algorithms, as well as two replication policies which have been simulated in order to compare their performance. These are executed by a Replica Manager agent at each site, rather than by a single global manager. They are used in combination with one of the scheduling algorithms described in the previous section.

9.4.1 Least Recently Used

This algorithm can be seen as a "bare minimum" approach to replica management. When a new file is copied for a job to begin running, the least recently used replica is deleted to make space for it. This has the advantage that it does not require knowledge of file usage patterns, although it will be accordingly unable to respond to such feedback and improve its performance.

9.4.2 Economy

The Economy replica management algorithm implemented by the OptorSim project in [53] bases its decision on how profitable a replica is for an SE, i.e. how often it is likely to be needed locally in the future. The popularity of datasets (the number of times they have been requested) is shared between SEs, to ensure that Replica Managers at each site are aware of the global popularity of files. When a new file is transferred for a job its popularity is compared with that of the least popular file currently cached. If the new file is more popular than the old file by a certain profit margin, the old file will be deleted to make room for the new one. This profit margin is the difference in number of requests between the most and least popular files currently stored, with a minimum profit margin of 5.

An assumption is made here that the profitability of a given file remains constant with time, which is true within the limited time of a run with the simulation. In the long term adjustments would have to be made to compensate for changes in the popularity of datasets, e.g. a cut off in access pattern statistics older than a month.

9.4.3 DataRandom Replication

The ChicagoSim project in [54] investigated combinations of scheduling and replica management policies to discover which were the most complementary. They found that use of a replication policy improved performance significantly, but that all policies seemed to produce improvements of roughly the same magnitude.

In their approach, the local Replica Manager monitors the popularity of replicas stored locally, as well as the loading of the CE. When the job load passes a certain threshold, replication is triggered. The most requested replicas on the SE are offered to other sites on the Grid. The sites chosen for this offer, whether based upon site loading or a random choice, seemed to make little difference, so a random selection is used here (the DataRandom algorithm). This approach has the advantage that the Replica Managers do not need to share information about resource status, as the decision to replicate is based solely upon local monitoring.

In the implementation simulated here, the job load at each CE is checked every 360 seconds (simulation time). If the number of jobs is twice the number of CPUs belonging to that CE, replication is triggered. The most popular replica is offered to a randomly chosen site. If the site does not already have a replica of the offered data, and can clear space in their cache for it, it will copy the file. In this way popular data can be migrated from heavily loaded sites to others. This should make it easier for scheduling algorithms that take the location of requested data into account to balance the job load.

9.4.4 Economy Replication

The Economy Replication policy implemented in EDGSim is a simplified version of the model described in [53], without the complex negotiation procedure used in that work. The Economy algorithm monitors the popularity of files across the Grid much like the replica management algorithm described in 9.4.2, and determines whether it would be profitable to make a local copy of regularly requested data.

OptorSim uses an auction system, in which a site that needs a particular file announces its request, and other sites make bids to supply the data.

Secondary auctions can be initiated by sites that do not have the file, but wish to acquire it in order to participate in the primary auction. These secondary auctions propagate popular data across the Grid.

The OptorSim bids are based upon network status, so the best bid will be the site that can supply the file most quickly. This process differs from that of EDGSim, but the end result should be similar, as the decision is still made on the basis of access times. Instead of the secondary auctions, a slightly different economic system is used to spread popular data between sites.

As with DataRandom, the algorithm acts once every 360 seconds in simulation time. If cache space is available at a site, the most requested file on the Grid is transferred there. If not, the Replica Manager considers the popularity of replicas already locally resident to determine whether the least profitable should be deleted to make room for a more popular file. Much like the Economy replica management process, the file must be more popular than a locally resident file by a profit margin equal to the difference between the number of requests for the most and least popular files stored locally. However there is a minimum margin of 20 requests here, to avoid an excess of preemptive replications.

There is no secondary auction system here, as there is in OptorSim. Rather than a series of negotiations between the storage brokering entities in the simulation, taking a certain period of simulation time to complete, a decision is made at a single site to transfer a file from another.

9.5 Summary

This chapter described the management policies that will be used in the runs in the following chapter. Other Data Grid simulation work was reviewed. Four scheduling algorithms were described: the randomised ETT algorithm from the previous chapter; a simple algorithm that considers ETT as well as the location of replicas of the requested files; a more complex algorithm that minimises ETT and the necessary transfer times for requested files; and an adaptive algorithm that modifies its behaviour according to job loading conditions.

Two replica management policies were described: one which deletes the least recently accessed files to make way for new data; and another which decides which data should be retained based upon an economic consideration of the data's popularity with users. Additionally, two replication policies were defined: one which replicates frequently requested data to other sites in heavy loading conditions; and a second which makes economic decisions as to whether popular data at other sites should be replicated to a particular SE.

Chapter 10

Job and Replica Management in a Data Grid

There is no fully functional, production Data Grid that can be used for comparison with the simulated version. Instead, experiments can be conducted to compare the output of EDGSim with that of the other simulation work described in the previous chapter. This chapter describes how EDGSim was configured to run similar tests to those conducted by OptorSim in [53] in order to verify their results, before moving on to other configurations and topologies. Initial runs are conducted to make comparisons with the OptorSim results. The effect of varying the delays in the Information Services, and the time taken by the brokering process, are investigated. Next, the relative performance of the management policies is studied when the job running times are increased from the very short times used in OptorSim. Heterogeneity of CPU and storage resources is introduced, and the running times of jobs are further extended to be closer to those of HEP jobs, and the loading level of the Grid is also varied to monitor the effect on efficiency.

The error bars shown in all plots here represent the spread of results over ten simulation runs.

10.1 Configuration of EDGSim with OptorSim Parameters

The OptorSim Grid testbed is a simple, homogeneous one with 17 nodes corresponding to the member sites of the GridPP. One of these nodes represents RAL, the Tier 1 site, which has an SE storing all of the data files in the Grid, but no CE. The CE at each of the other sites can run a maximum of one job at a time (i.e. they effectively have a single CPU). The SEs at the non-RAL sites have the same amount of cache space which can store a fraction of the total dataset.

Jobs arrive at the single RB at a rate of one every five seconds, and request files based upon a Zipf distribution as described in section 9.1.1. The running time of these jobs is short compared to the transfer time for the data. Although OptorSim is capable of simulating background usage of the network (effectively a time-dependent modification of the available bandwidth), it is not used here, i.e. the files requested by test jobs are the only network traffic. There are no delays in the brokering process, and the RB has accurate information about the testbed resources, effectively resulting in no IS delays.

Using EDGSim, a simulated Grid was created to replicate the above configuration as closely as possible. Stored at the virtual RAL site was a dataset of 1000 files, each 1 GB in size. The network structure had a 10 GBit / s "backbone" running between the routers, with smaller connections between the routers and the member sites (typically 1 GBit / s). Two SE configurations were used, varying in the local cache size, with 100 GB (10% of the size of the total dataset) and 200 GB (20%).

The input job distribution consisted of jobs requesting 10 files each, and running for 1 second per file. These files were sequential, but the first in the sequence was picked according to the Zipf distribution (e.g. if the first file generated from the distribution was number 463 of 1000, files 463-472 would be requested).

Four scheduling policies, described in chapter 9, were used to manage the job load: "Estimated Traversal Time"; "ETT + File Location"; "ETT + Queue Access"; and the "Adaptive" algorithm. These were used in conjunction with: the "Least Recently Used" replica management policy; "Economy" replica management; and "Economy + Replication", replica management with data replication. The results are shown in fig. 10.1 for cache sizes of 100 GB, and in fig. 10.2 for 200 GB.

With smaller SE caches, the "ETT + Queue Access" algorithm performs significantly better than the others according to all metrics. With the very short job running times (1 second per file), the jobs' lifetimes are often dominated by the time taken to serve them the requested data, so this algorithm is well suited to the task, as it minimises the time taken in data transfer. In contrast, the "ETT" algorithm gives very poor results, as it has no awareness of the jobs' data dependencies. The "ETT + File Location" and "Adaptive" algorithms perform slightly better, as they both attempt to minimise data transfer, albeit in a less sophisticated manner than "ETT + Queue Access".

With the smaller cache sizes, sites can only store 10% or the total dataset,

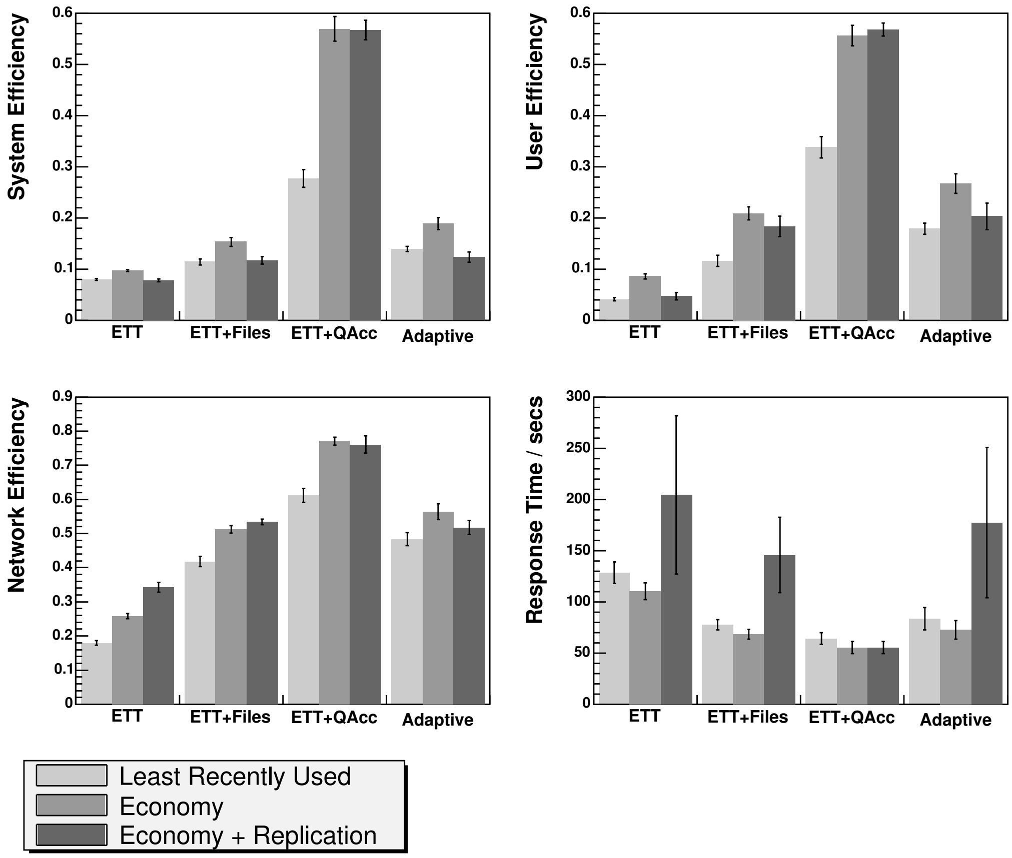

Figure 10.1: Results of EDGSim runs with configuration like that of OptorSim, and SEs configured with 100 GB cache space.

and the task of replica management becomes very important. In almost all cases here the "Economy" algorithm outperforms the more basic "Least Recently Used" policy, as by observation of access patterns it is able to determine the more popular files and retain them for future use.

As might be expected, the performance of all scheduling algorithms is improved when the cache size is increased to 200 GB. However, the relative performances, at least for "LRU" and "Economy" replica management, stay roughly the same. The improvement gained by using "Economy" manage-

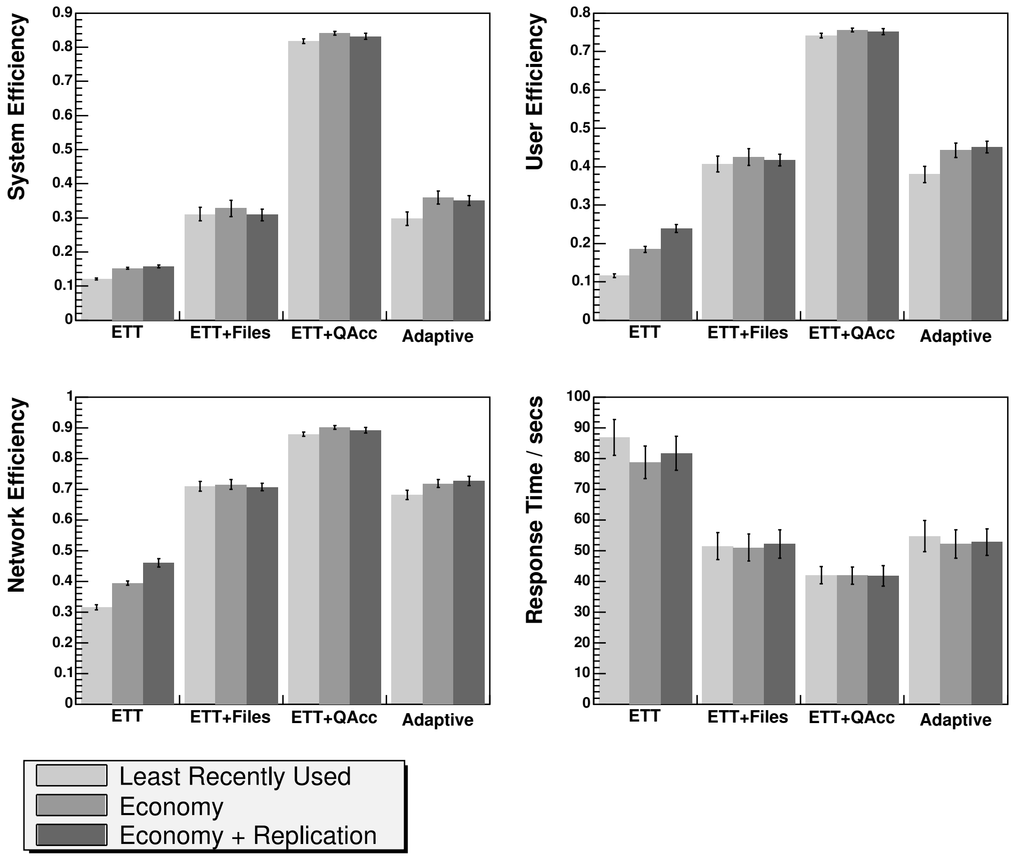

Figure 10.2: Results of EDGSim runs with configuration like that of OptorSim, and SEs configured with 200 GB cache space.

ment is most evident with 100 GB caches, with less storage space to spare.

The file transfers during a run for the "LRU" and "Economy" algorithms are shown in figs. 10.3 and 10.4 respectively.

Fig. 10.3(a) shows that the number of file transfers stays more or less the same during a run of 5000 jobs (i.e. 50000 file requests), even if (b) shows some fluctuations in the number of transfers at any one time. The "LRU" algorithm has no capacity to learn, and so its performance does not improve.

In contrast, fig. 10.4(a) shows the "Economy" algorithm begins with a

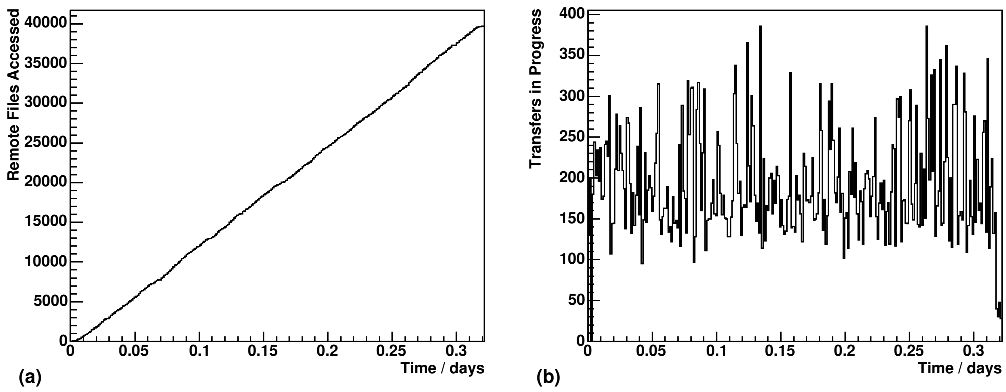

Figure 10.3: File transfers with "Least Recently Used" replica management: (a) shows the cumulative file transfers during a simulation run; (b) shows the transfers in progress at different points during the run. "ETT + Queue Access" scheduling was used, and the SEs had 100 GB cache space. Each bin here represents 100s.

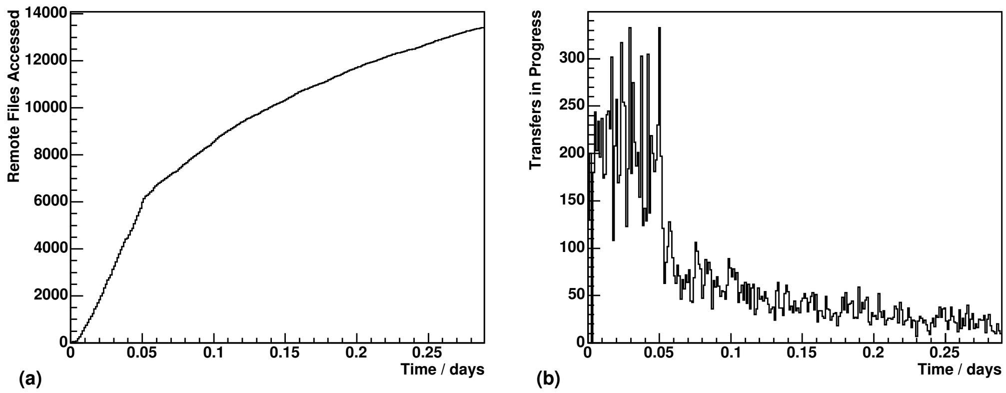

Figure 10.4: File transfers with "Economy" replica management: (a) shows the cumulative file transfers during a simulation run; (b) shows the transfers in progress at different points during the run. "ETT + Queue Access" scheduling was used, and the SEs had 100 GB cache space. Each bin here represents 100s.

steady rate of transfers, but is then able to start reducing the number of remote file access as it learns which files are requested more often, and keeps these in the cache. This can be seen clearly in (b), with an initial surge as the caches fill up, followed by a steady decrease in running transfers as the algorithm improves its knowledge of the file access patterns. In the case shown here, there is 1600 GB cache space available, so assuming a typical bandwidth of 0.2 GB / s, the cache space will take 8000 seconds to fill up, or 0.09 days, assuming all files are transferred from the originals at RAL. Fig. 10.4 gives a value of 0.05 days, and a lower value like this might be expected, as files can be transferred from other caches after they have been requested once, thus speeding the process up.

The introduction of "Economy + Replication" seems to have no positive effect in most cases, with E_{User} and E_{System} staying roughly the same, or even decreasing. In the 100 GB cache runs, all algorithms except "ETT + Queue Access" suffer. The latter is the only one which is specifically concerned with minimising file transfer times. The others, even those with some input data considerations, are attempting some form of load balancing, which is less useful in this context, as job running times are less significant. Jobs requesting data are in competition with preemptive replication of data, which restricts the available bandwidth, and thus the jobs wait longer for data to be delivered. An improvement in $E_{Network}$ for the "ETT" algorithm is more due to this scheduler's lack of data awareness, leaving the "Economy + Replication" algorithm to do the work of distributing data across the Grid. The penalty paid by the jobs is shown in the $T_{Response}$ plot, with the network contention caused by the preemptive replication leading to erratic job behaviour.

Although the "Economy + Replication" algorithm is not producing the

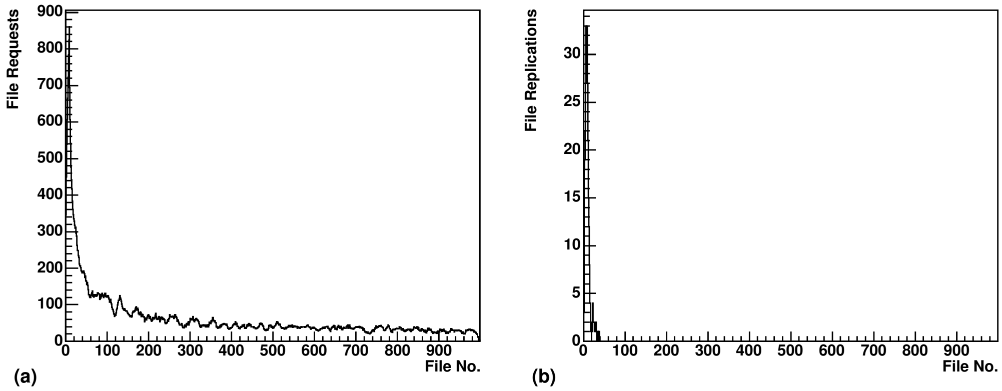

Figure 10.5: The effect of a preemptive replication policy: (a) shows the popularity of files, requested by jobs according to a Zipf distribution; (b) shows the replication of the most popular files by the "Economy + Replication" algorithm. In this run the "ETT + Queue Access" scheduling algorithm was used, and the SEs had 200 GB cache space. Each bin here represents 100s.

desired improvement in efficiency, it is nevertheless carrying out its intended task of replicating the most popular files independently of job requests, as can be seen in fig. 10.5. The relative popularity of files, chosen by 5000 jobs in batches of 10 according to the Zipf distribution, is shown in (a). The replication algorithm running at local sites has made requests for the most popular files, as shown in (b). However, this procedure does not help the user to get job results back more quickly, at least in this scenario.

The results of using the "DataRandom" replication algorithm are not shown here, because it has negligible effect in this context. It is activated when the workload passes a certain threshold at a site, namely twice as many queued jobs as running ones. The job load here is relatively light, so the "DataRandom" replication algorithm is not activated.

The success of the "ETT + Queue Access" algorithm bears out the results of the OptorSim work, as does the improvement of performance when using "Economy" replica management. In this case "Economy + Replication" does not have the beneficial effect shown in the OptorSim work, although it has been demonstrated to be transferring popular files as intended.

However the Grid setup used here is a somewhat simplistic one, and the jobs run for a much shorter time than typical physics analysis jobs might. In the following sections the simplified aspects are expanded upon, and jobs of more typical sizes are submitted.

10.2 Effect of IS and Brokering Delays

The runs conducted so far have included no delays incurred in middleware processes, although as seen in chapter 7, Grid processes often take a non-negligible period of time to complete in practice. This section investigates the effect of less frequent updates from the Information Services, and longer job scheduling times, upon performance of the Grid described in the previous section.

10.2.1 Frequency of Information Service Updates

So far the IS update period has been small, shorter than the time taken for RB scheduling decisions to be made. The update time was now increased to 300 seconds, the estimated figure used for earlier results in chapters 7 and 8. The effect of this change on the Grid testbed used in the previous section can be seen in figs. 10.6 and 10.7, for cache sizes of 100 GB and 200 GB respectively. The results without the longer delay are represented by the dashed histograms.

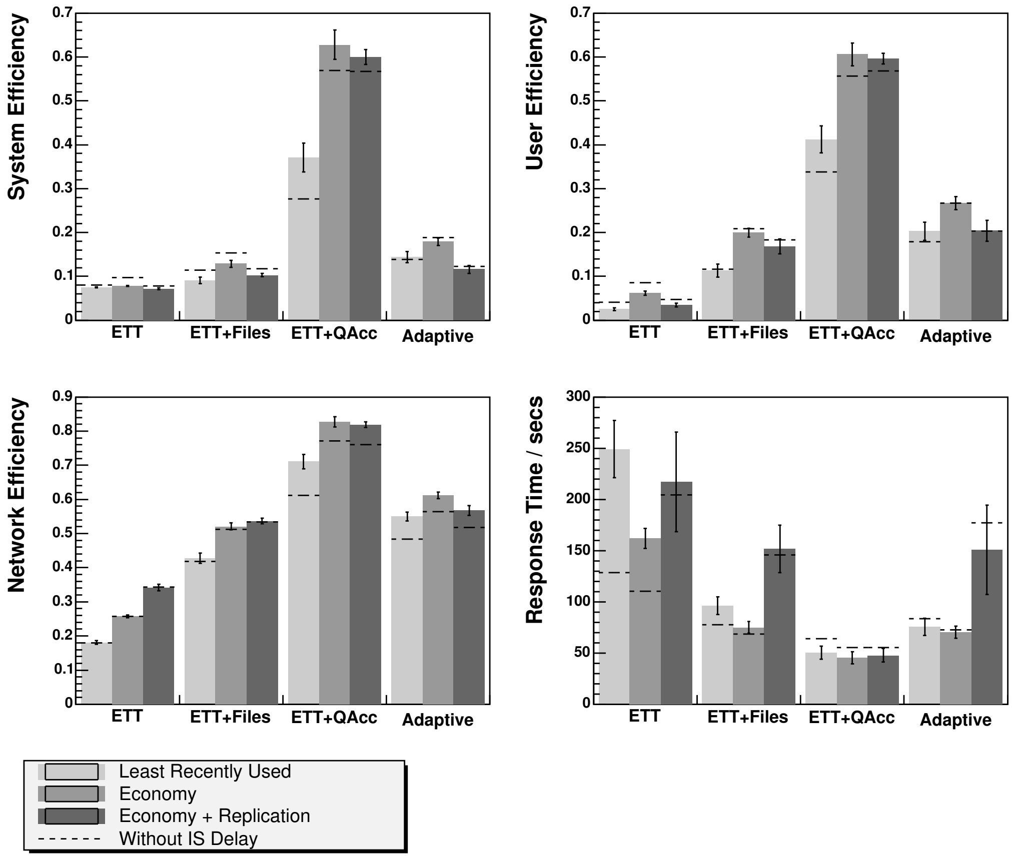

Figure 10.6: Runs with OptorSim-style homogeneous Grid with 300s IS update time, and SEs configured with 100 GB cache space. The dashed histograms represent the results without IS delays.

The results for the "ETT" algorithm are as one might expect - all efficiency measures decrease, as the RB makes its scheduling decisions based upon old information. This decrease is correspondingly smaller for "ETT + File Location", as this algorithm has a 50% weighting for Estimated Traversal Time values and the distribution of requested data.

In contrast, and slightly counter-intuitively, the performance of "ETT + Queue Access" improves with the slower IS. This is a result of the very short

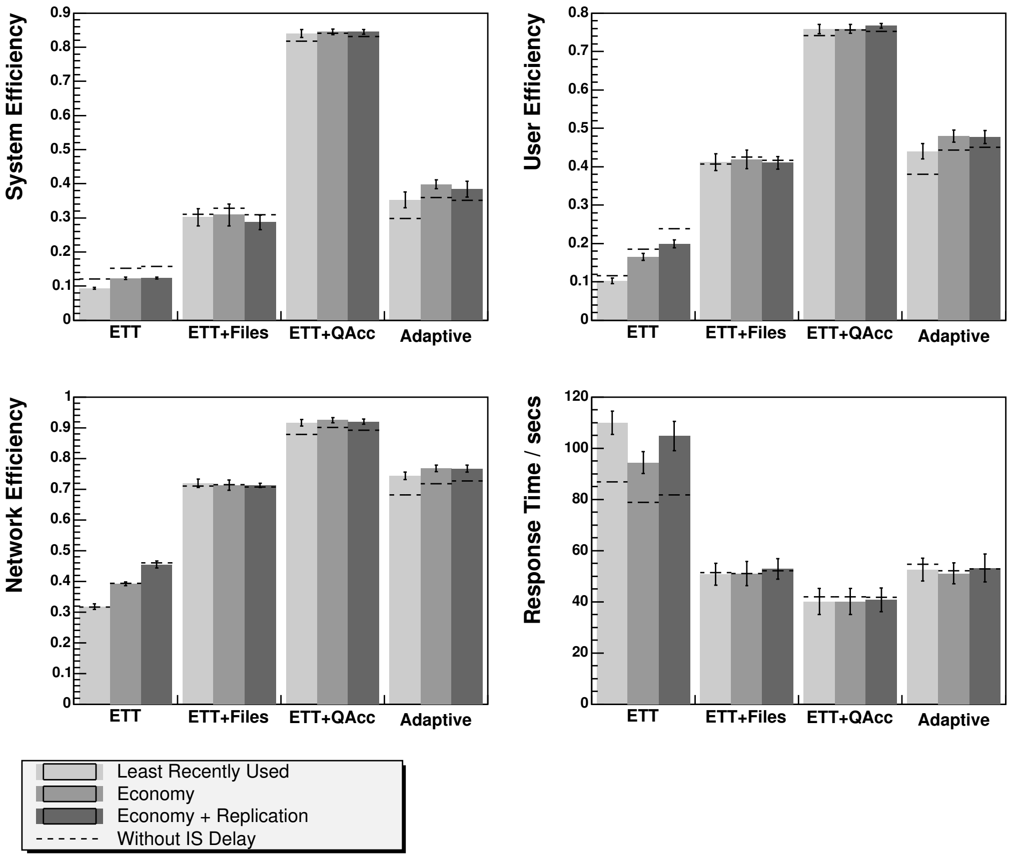

Figure 10.7: Runs with OptorSim-style homogeneous Grid with 300s IS update time, and SEs configured with 200 GB cache space. The dashed histograms represent the results without IS delays.

running time of the jobs, and their sharply peaked file access pattern. As the RB must now schedule several jobs using the same information before the next update arrives, they will be sent to the same site. Due to the nature of the Zipf distribution, they are likely to have at least a few of their requested dataset in common, meaning less file transfers will be needed. As transfer times are the dominant part of a job's lifetime here, the imbalance in the job load will not significantly affect efficiency. Thus $E_{Network}$ increases because less transfers take place; E_{User} increases (and $T_{Response}$ decreases) because the jobs complete more quickly; and E_{System} increases because jobs spend less time waiting for data to arrive, so resources are delivered to the jobs more effectively. This effect is greater with the larger SE caches shown in fig. 10.7, as more of the commonly requested files can be retained at each site.

The "Adaptive" algorithm also shows improved efficiency for large cache sizes. This is again because jobs will be sent to sites with a proportion (40%) of the requested data, combined with attempted load balancing based upon old status information, resulting in a clustering of jobs. With the smaller cache sizes (fig. 10.6), the "Adaptive" algorithm concentrates on load balancing as sites are less likely to have 40% of the required data, so the advantage is lost.

In any case, the difference in results is not a large one, even for large intervals between updates. Ordinarily a less frequent IS will lead to decreases in efficiency from both a user and resource owner perspective. The results presented here highlight a peculiarity of Grid scenarios where the lifetime of a job is dominated by the time it spends waiting for data to be delivered.

10.2.2 Job Scheduling Time

The Resource Broker has been observed to take longer to schedule jobs with a dependency on input data than those without [52]. This is due to the extra complexity of matching jobs with the location of requested data files taken into consideration, as well as the overheads associated with the RB's communication with the Replica Catalog and network information services. These lengthier scheduling processes take approximately four minutes to be completed. The runs in section 10.1 were repeated, but with this job scheduling time extended to 240 seconds. The results, for 100 GB and 200 GB of SE cache, are shown in figs. 10.8 and 10.9.

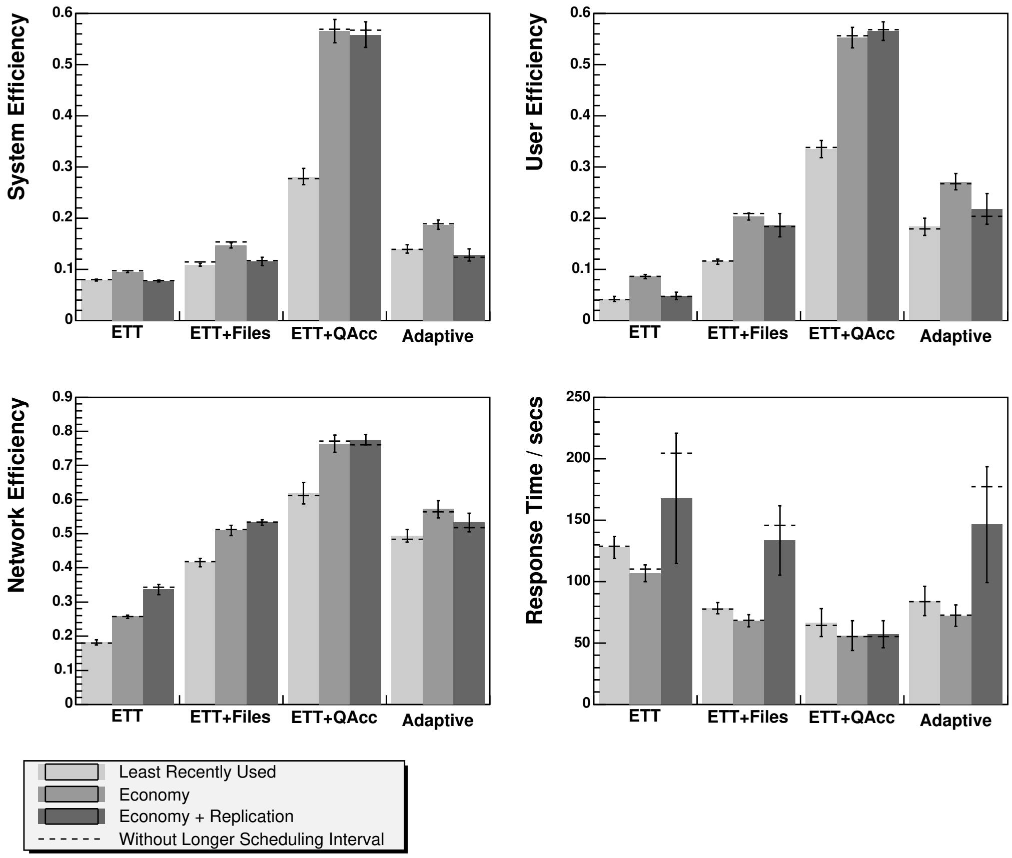

Figure 10.8: Runs with OptorSim-style homogeneous Grid with 240s job scheduling time, and SEs configured with 100 GB cache space. The dashed histograms represent the results with a negligible scheduling time.

These results indicate that the time taken to schedule jobs does not significantly affect the outcome of this scheduling. The effect of these intervals is to offset the timing of the passing of jobs to the chosen resources by four minutes, as each job is handled by a parallel process. This means that by

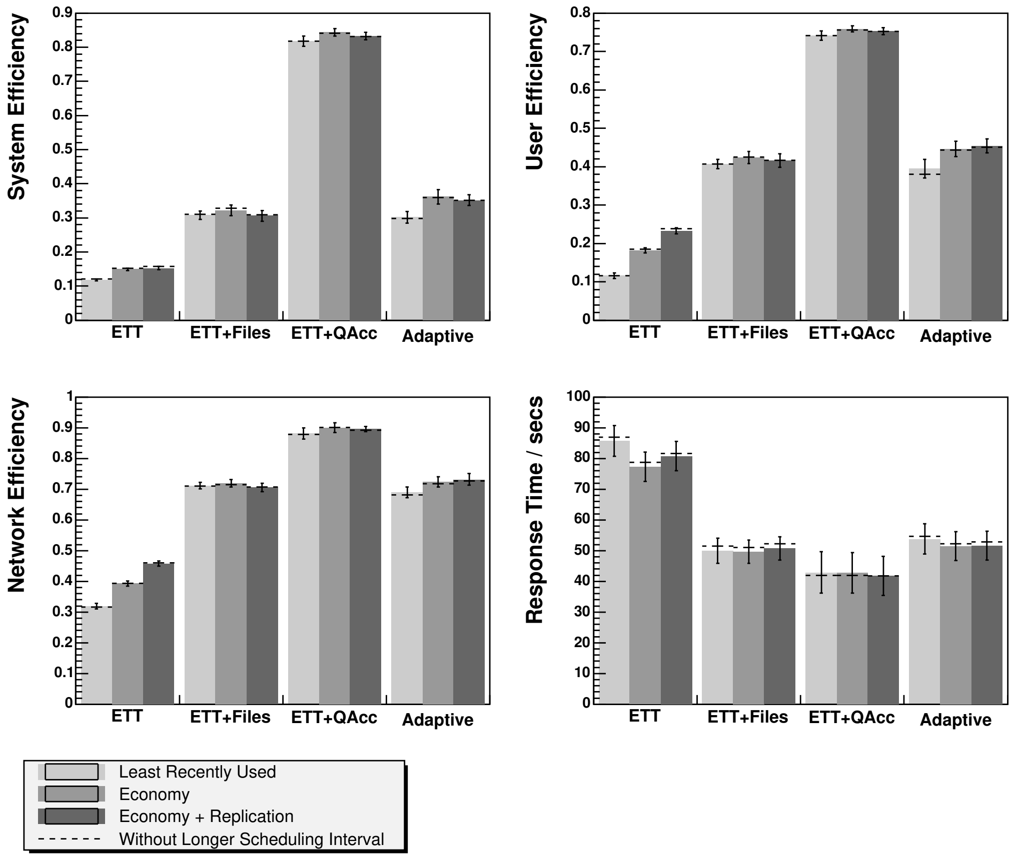

Figure 10.9: Runs with OptorSim-style homogeneous Grid with 240s job scheduling time, and SEs configured with 200 GB cache space. The dashed histograms represent the results with a negligible scheduling time.

the time the job arrives, the CE, replica and network information used to schedule it is out of date; however this does not alter the outcome of the run to any great degree. This scheduling time is significantly smaller than the time spent by a file in an SE cache, particularly a popular file. Job lifetimes here are dominated by file transfers, and the distribution of data across the Grid does not change quickly enough for the performance of the scheduler to be degraded.

10.3 Longer Job Running Times with a Simple Grid

The results presented in the previous sections describe jobs with a running time which is small in comparison with the time taken to transfer a data file. However a typical particle physics data analysis job will take much longer than one second to run over a 1 GB file, and times of the order of several minutes or longer might be more typical. The effect of even a small increase in running time on the scheduling strategies in this simplistic Grid setup are shown in fig. 10.10 for 5 seconds per file, and fig. 10.11 for 10 seconds per file. In order to normalise the results for comparison with the 1s / file runs, the $T_{Response}$ figures have been divided by the total job running time.

Increasing the running time to 5 seconds has dramatically reduced the gap in performance between "ETT + Queue Access" and the other algorithms. E_{System} has almost levelled out, because the jobs are now running for a longer proportion of their lifetimes, and all four algorithms will be attempting to balance the load over a simple and homogeneous Grid.

E_{User} and $T_{Response}$ reveal that performance from a user's perspective is more erratic. The loading of the Grid has effectively been increased (the same number of jobs submitted at the same intervals, but running for longer), so jobs are being queued more often before they can begin running. This is why with even the more effective algorithms, the $T_{Response}$ is more than twice the required running time of the job. With only a "Least Recently Used" replica management policy in place, performance is erratic, and often significantly less efficient than with "Economy" management. In general preemptive replication improves $E_{Network}$, but its effect on job running varies.

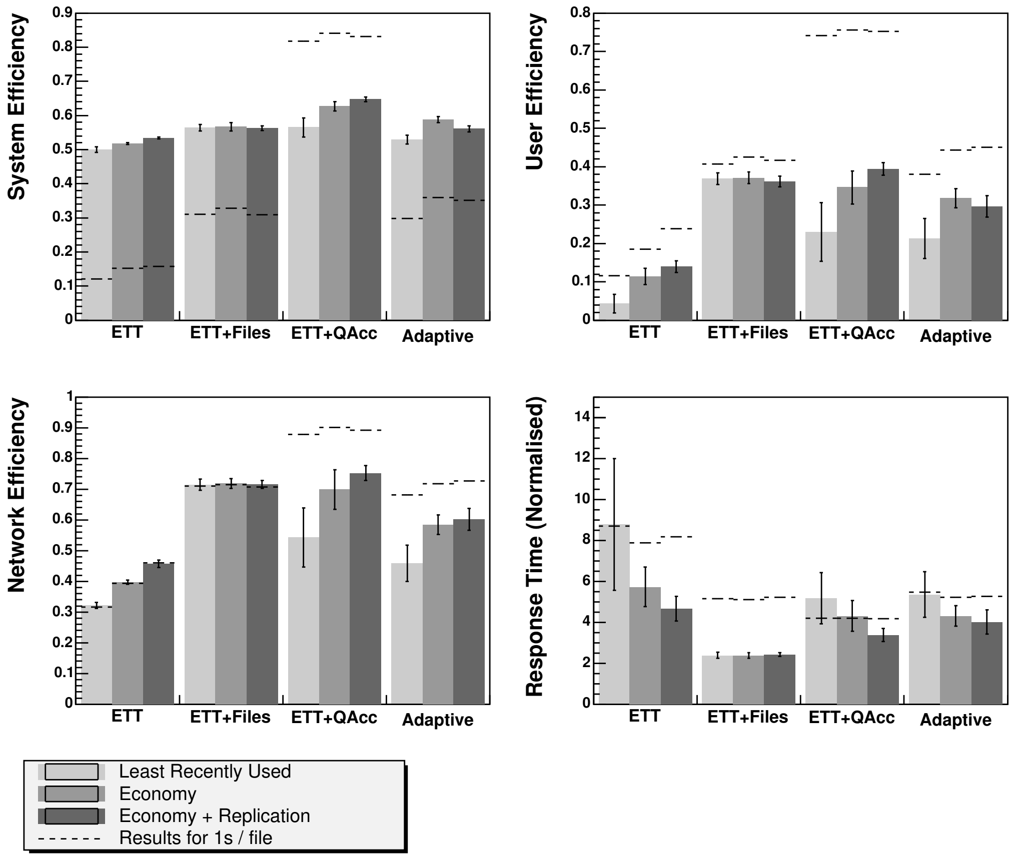

Figure 10.10: Runs with OptorSim-style homogeneous Grid with 5s running time per file, and SEs configured with 200 GB cache space. The dashed histograms represent the results with 1s / file.

The most consistent algorithm is "ETT + File Location", which produces similar results with any replica management policy. "ETT + Queue Access" is less successful now that job lifetime is not dominated by file transfer times. With jobs spending more time queueing, this algorithm's knowledge of current network properties at the time of scheduling becomes less useful. "ETT + File Location" instead looks at groupings of data, which change less rapidly with time.

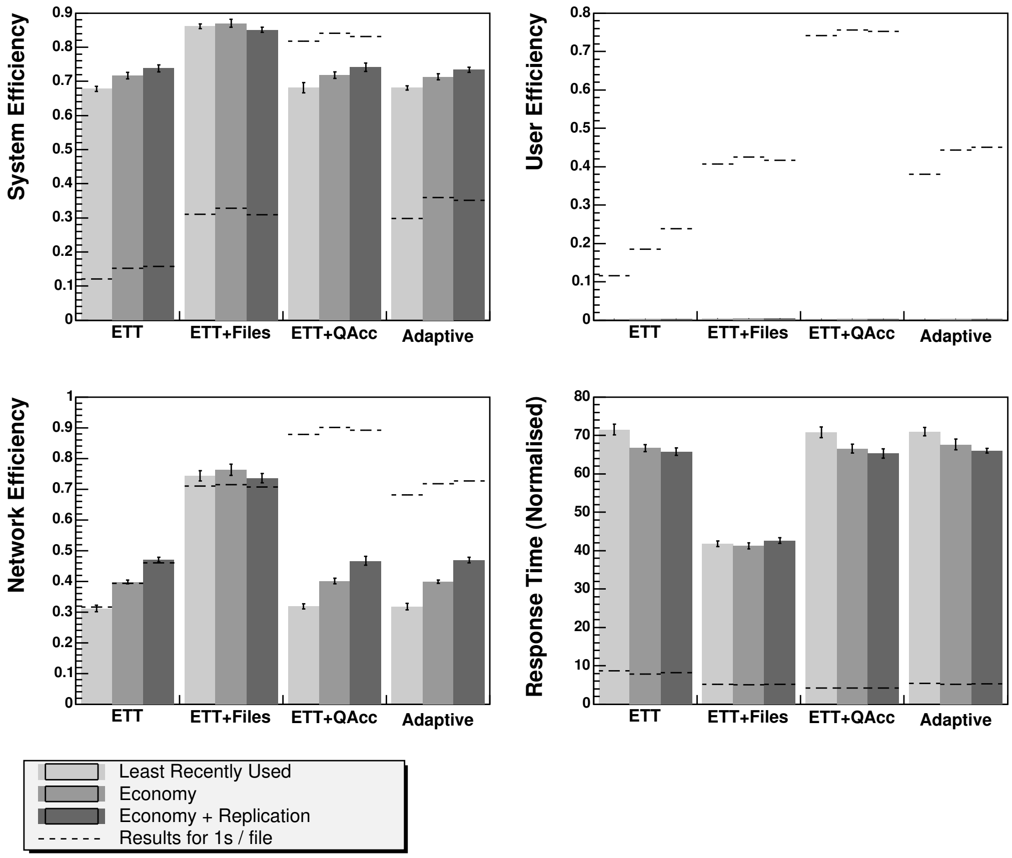

Figure 10.11: Runs with OptorSim-style homogeneous Grid with 10s running time per file, and SEs configured with 200 GB cache space. The dashed histograms represent the results with 1s / file.

The effect is exaggerated further when the running time is increased to 10 seconds. E_{User} has now dropped to almost zero, as the Grid is now very overloaded. The $T_{Response}$ plot shows that jobs are often queued behind tens of jobs before they are assigned to a CPU. E_{System} is higher for most algorithms, as they balance the load as effectively as they can. The exception is “ETT + Queue Access”, for which the results are converging with the “ETT” algorithm now that running times are longer compared to file transfer times. "ETT + File Location" continues to give the best $E_{Network}$ results, and job performance results in general.

10.4 Heterogeneity of CPU Resources

The results in the preceding sections have been generated with a very simplistic Grid, in which all member sites are identical, other than the RAL site with the permanent copies of data files. These sixteen sites have identical amounts of cache space, and one CPU each on which to run jobs. The Grid will eventually make many times more resources available in order to handle the workload of the LHC experiments. This resource set has been estimated by the LCG for its European members [56], and in the GridPP 2 proposal for the UK sites [57], projecting the increases in available CPU power and storage in the years leading up to full LHC running.

The simulated testbed was now updated to include a CPU distribution in the same proportions as those for the 2004 GridPP testbed [58], scaled down by a factor of ten, as shown in the second column of table 10.1. For this run the storage distribution was homogeneous, with 500 GB at each site (this is the mean of the cache sizes, scaled down by a factor of 100 for simulation feasibility). The "DataRandom" preemptive replication algorithm is also used here along with the "Economy" management policy, as there is now some asymmetry in the Grid, unlike previous runs.

Jobs run for 300 seconds per 1 GB file, which is more representative of the duration of HEP analysis jobs. File access was again based upon a Zipf distribution, with 6000 data files available. 5000 jobs were submitted requesting 10 data files as before, at 10 second intervals, resulting in busy but not overloaded resources. The results are shown in fig. 10.12.

Resource	No. of CPUs	Cache Size / GB
Site01	15	150
Site02	110	400
Site03	0	1000
Site04	5	33
Site05	30	200
Site06	15	90
Site07	5	1630
Site08	22	200
Site09	20	100
Site10	4	640
Site11	23	100
Site12	32	3640
Site13	8	53
Site14	37	136
Site15	41	380
Site16	31	280
Site17	12	60

Table 10.1: Number of CPUs and Cache Sizes at GridPP sites

In many cases here, the efficiency measures are close to 1.0. This is partly because the loading of the Grid is not heavy enough to require much queueing of jobs, and partly because of the longer running times per file, compared with the earlier runs with a more simplistic Grid. File transfer times are no longer a significant part of job lifetimes, so the time spent by jobs assigned to a CPU but not running is comparatively short. This is why the "ETT"

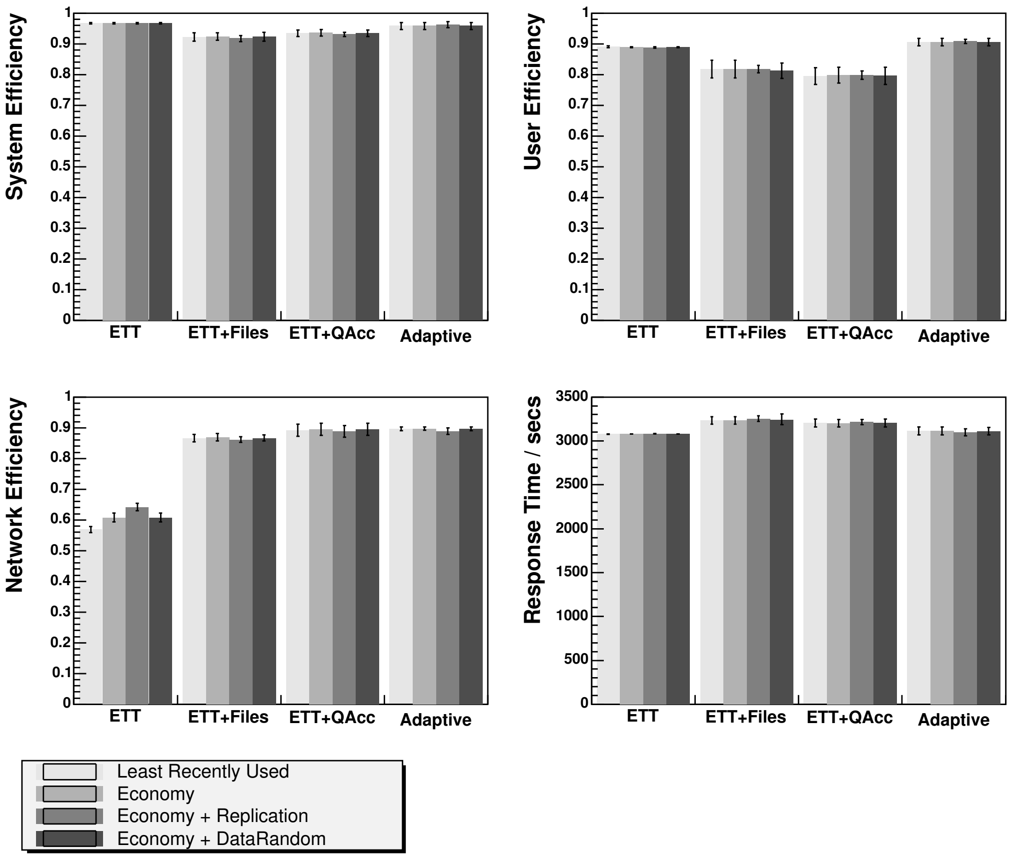

Figure 10.12: Results for Grid with heterogeneous CPU resources, and 500 GB cache at each site. Jobs run for 300s per file, and there are 6000 files available.

algorithm, which concentrates purely on load balancing, results in the best job performance and CPU occupancy. However the price paid for this is a much poorer $E_{Network}$ value.

The other algorithms perform similarly in all respects, with the "Adaptive" algorithm achieving slightly better results. The other two are concerned more with data distributions, and with a homogeneous cache space distribution there is little to choose between sites in this respect.

10.5 Heterogeneity of Storage Resources

To further improve the realism of the simulated Grid, the storage resources were now configured in a heterogeneous manner, again in proportion to those in the real GridPP testbed. The total cache space was the same as in section 10.4, but ranging from 33 to 3640 GB at any one site as shown in table 10.1. The results are displayed in fig. 10.13.

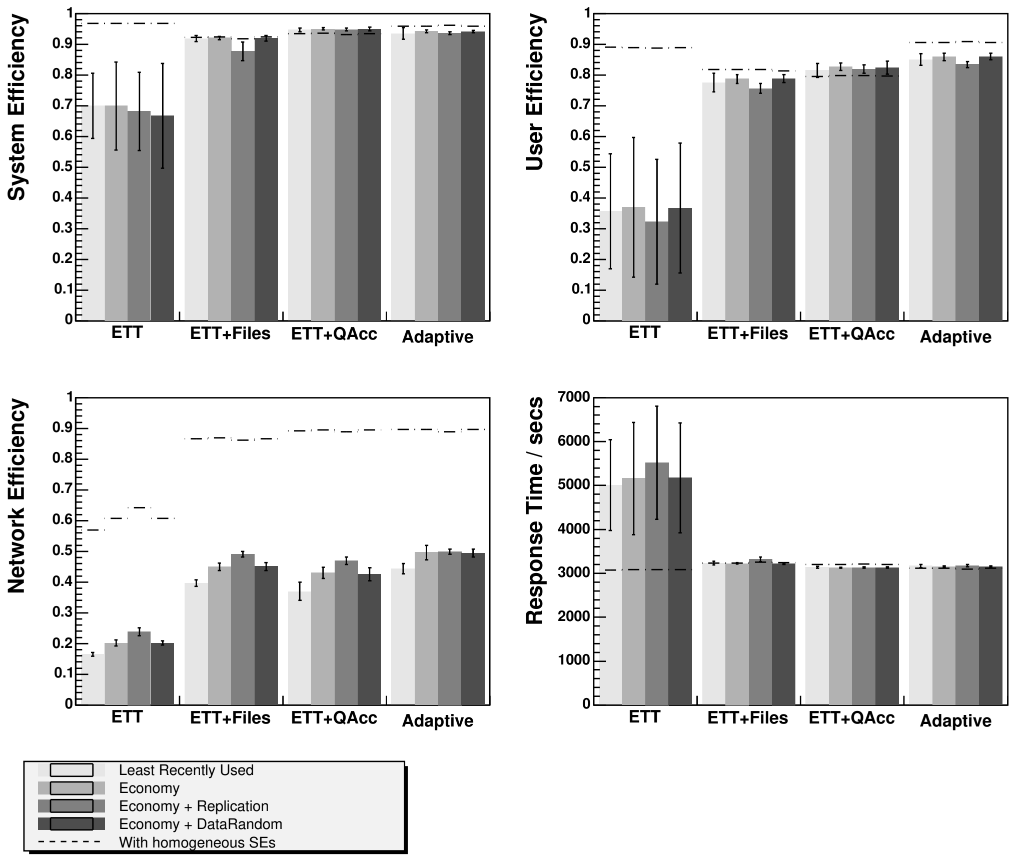

Figure 10.13: Results for Grid with heterogeneous CPU and storage resources, based on GridPP resource distributions. Jobs run for 300s per file, and there are 6000 files available.

The most striking difference to the homogeneous SE case (represented by the dashed histograms) is that $E_{Network}$ has degraded in all cases. The less even distribution of cache space leaves a few sites with large caches, but most with less storage space available, so by necessity more files must be transferred for individual job requests. "ETT" once more suffers the most here, with an approximately equivalent performance by the other algorithms.

This algorithm also fails to schedule jobs effectively, as it cannot take the distribution of storage space into account. Too many jobs are scheduled to sites based on ETT values, which reflect the running times of previous jobs run at a site, rather than considering the data requirements of the new job. Since ETT values are updated at a slower rate than that of job submission, jobs are distributed less effectively between sites, leading to the erratic job performances seen here. The other algorithms take into consideration information relating to data distribution, which differs between jobs, meaning that they do not experience the same problem with IS updates. Their results are comparable with those of the homogeneous storage case, other than the lower $E_{Network}$ values. The "Adaptive" algorithm performs slightly better than the others in terms of E_{User} and $T_{Response}$, although "ETT + Queue Access" produces the highest E_{System}, as it minimises the time spent by jobs waiting for data to arrive.

The "Economy + Replication" algorithm improves $E_{Network}$ by populating all SEs with popular data, but job running metrics are decreased slightly in some cases, as scheduling algorithms that take data location into account have less to choose from, and are affected by the same problems as "ETT" (albeit to a much lesser extent). The "Economy + DataRandom" algorithm has negligible effect, because in the cases of the data-aware scheduling algorithms that could take advantage of it, there are not enough jobs queueing to trigger replication to other sites.

10.6 Longer Job Running Times with a Heterogeneous Grid

In order to find the effect of longer running jobs on these results, the running time per file was increased to 1800 seconds (or half an hour), an approximate upper limit on the running time for a HEP analysis job. In order to maintain a similar level of loading, the interval between job submissions was correspondingly increased to 60 seconds. The results are shown in fig. 10.14, with the results for 300 seconds per file indicated by the dashed histogram. $T_{Response}$ has been normalised to the job running time so that this comparison can be made more easily.

The algorithms that performed well with the shorter jobs continue to do well, with a slight increase in E_{User} corresponding to data transfer times being a proportionally smaller component of the job lifetime. "ETT" is now producing excellent results, as the job submission interval is now larger than the IS update time, allowing perfect load balancing. As in previous cases, this performance comes at a cost of very poor $E_{Network}$ results.

The other algorithms produce very similar values for the job running metrics, but there are some changes for $E_{Network}$. Both "ETT + File Location" and the "Adaptive" algorithm give slightly improved results, as the lower job submission rate means that cache turnover is slower at the SEs. However "ETT + Queue Access" has more trouble as it is even more dominated by ETT values in its scheduling decisions. This leads to better load balancing, but more file transfers as well.

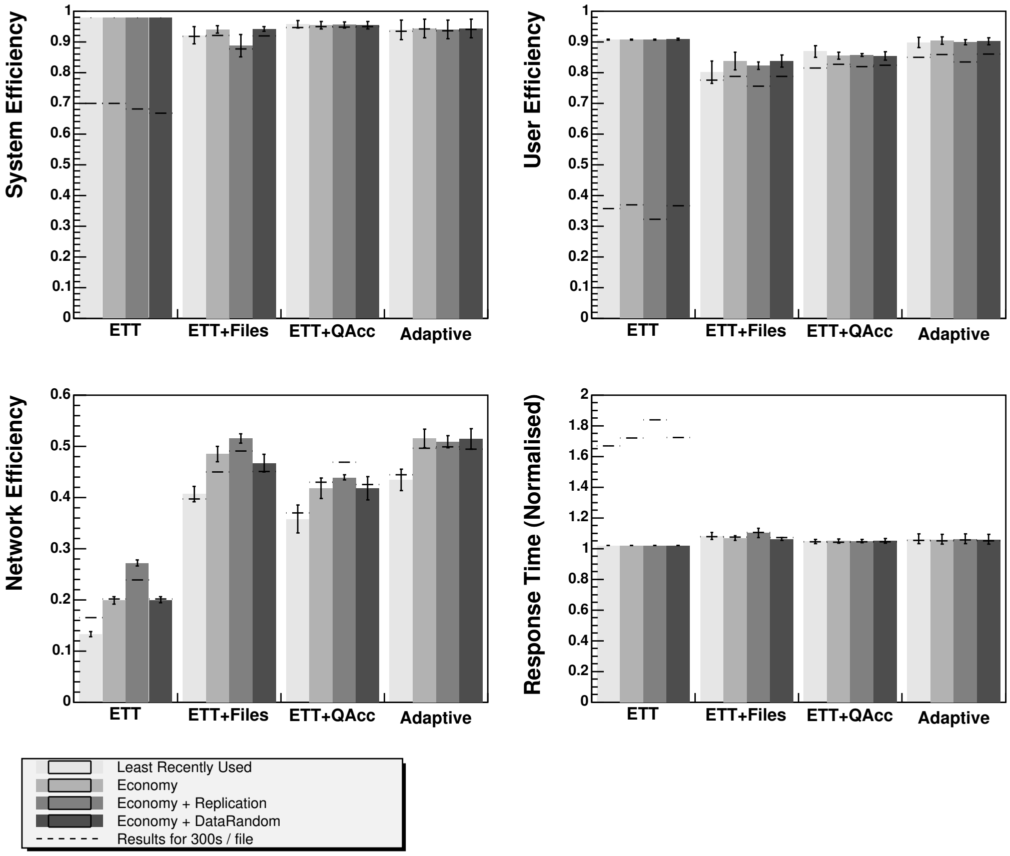

Figure 10.14: Results for heterogeneous Grid with jobs running for 1800s (30 min) per file, submitted every 60 seconds, to maintain a level of loading similar to fig. 10.13.

There is no change in the relative effectiveness of replica management policies, with "Economy" management only giving the best job performance, "Economy + Replication" improving $E_{Network}$, and "Economy + DataRandom" replication having no effect.

10.7 Efficiency with Higher Job Load

The runs in the previous section kept the job load at a steady level. The submission interval for the 300s / file jobs was now halved to five seconds, in order to observe the behaviour of the Grid in an overloaded state. Fig. 10.15 shows the results, with the dashed histogram representing the slower 10s submission interval for comparison.

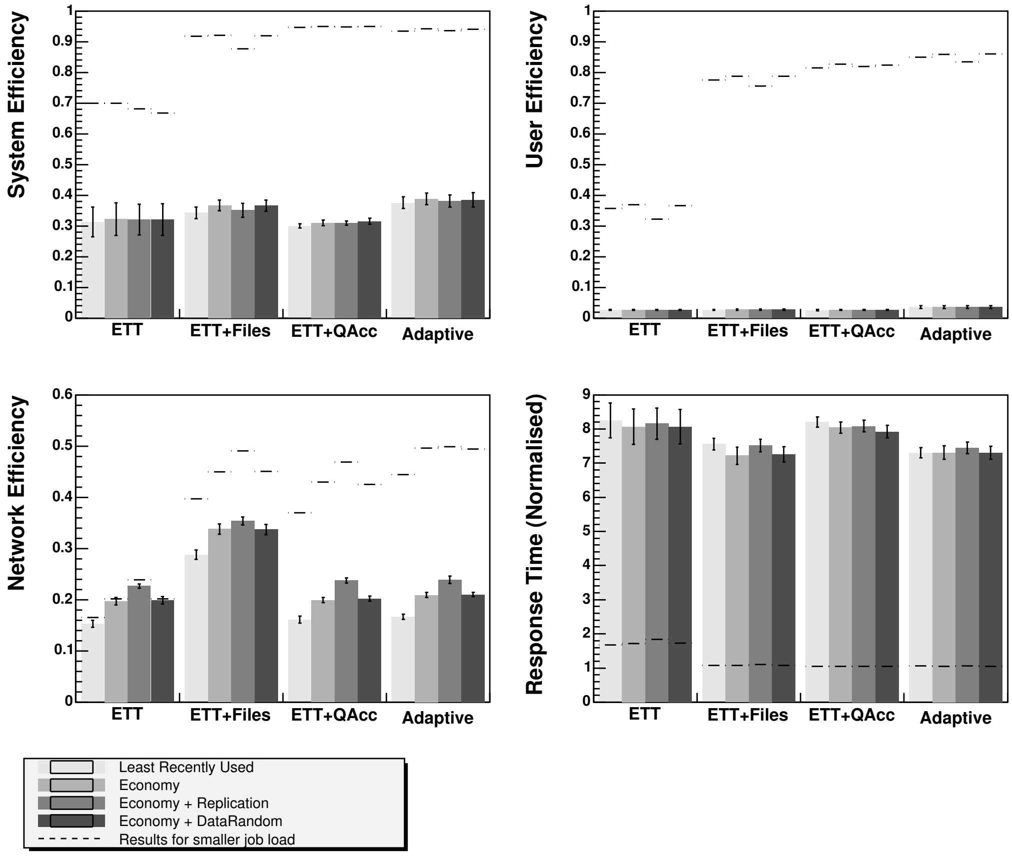

Figure 10.15: Results for a heterogeneous Grid as in fig. 10.13, but with jobs (running for 300s / file) submitted twice as rapidly.

The performance of all algorithms has degraded, as might be expected.

"ETT" now gives comparable E_{System} values to the other scheduling algorithms, as the increase in submission rate has caused them all to experience the load balancing problems caused by slower IS update rates, as described in section 10.5. E_{User} is now negligibly small in all cases, as the heavy load causes significant queueing at the CEs. The normalised $T_{Response}$ plot gives a clearer picture of job running, and it can be seen that the "Adaptive" algorithm is running jobs the most quickly, although "ETT + File Location" is competitive. "ETT + Queue Access" does no better than "ETT", because of the time between a job being scheduled and assigned to a CPU. The network status data used to minimise file transfer times will be out of date by the time the job is able to begin requesting those transfers.

All algorithms give lower $E_{Network}$ values except "ETT", which already gave poor results here as it has no knowledge of data dependency. However the "ETT + Queue Access" and "Adaptive" algorithms are now doing no better, the former for the reasons of old network information given above, and the latter because it reverts purely to load balancing when the Grid is heavily loaded. The simplistic approach of "ETT + File Location" is more effective, because it is still able to respond to distributions of data which change slowly enough for this approach to work relatively well.

For the first time the "Economy + DataRandom" replication algorithm makes a difference, reducing $T_{Response}$ slightly in comparison to "Economy" replica management alone. "Economy + Replication" again improves $E_{Network}$, but reduces system and job performance slightly.

10.8 Summary

The initial runs in this chapter took the testbed used by OptorSim as their starting point, with simple homogeneous resources, and very short job running times per data file. The OptorSim findings that "ETT + Queue Access" performed well were borne out here - E_{System} was more than 100% higher than with the other algorithms, job running measures improved by almost as much, and $E_{Network}$ was also significantly higher (a 20% or more increase). All schedulers suffered with more restricted cache space, but "ETT + Queue Access" was degraded less than the others. "Economy" management improved performance according to all metrics by 5-10% in most cases, although preemptive replication seemed to be less effective.

The IS update interval was increased, as was the time taken for scheduling decisions, but this had little effect on the results, since the more complex situation, as compared to the runs in chapter 8, resulted in less instances of the same resource being repeatedly chosen.

A significant change was observed when the job running times were increased. When running time was increased from 1 to 5 seconds per file, the advantage that "ETT + Queue Access" had over the other algorithms disappeared, because file transfer times were now a proportionally smaller component of a job's lifetime. All algorithms now produced roughly equal E_{System}. The "ETT + File Location" algorithm resulted in the shortest $T_{Response}$, and gave the most consistent running times. It also gave the best $E_{Network}$ overall. "ETT" performed less well than the others, particularly in terms of $E_{Network}$, as it has no awareness of data transfers.

These effects were more pronounced with jobs running for 10s per file (in this case the Grid was now also slightly overloaded). "ETT + File Location" gave 10% better E_{System}, 50% better $T_{Response}$ and 80% better $E_{Network}$ than the other algorithms. "Economy + Replication" now made a noticeable difference, improving $E_{Network}$ by a further 5% beyond that of "Economy" alone, and giving a slight improvement in other metrics.

The introduction of heterogeneity in CPU resources levelled out the results considerably - the only major difference between algorithms was the poor $E_{Network}$ results given by "ETT", around 30% less than the others. When storage resources were made heterogeneous as well, $E_{Network}$ was degraded in all cases, but the "Adaptive" algorithm gave the most consistent values, as it also did with $T_{Response}$.

When the running time was increased (and the rate of job submission was reduced to compensate), "ETT" actually produced the best E_{System} and $T_{Response}$ because submission rates were slower than the IS updates. However it still gave very poor $E_{Network}$ results. Again the "Adaptive" algorithm performed much better here, and it was also competitive for the other metrics.

Increasing the submission rate instead to produce an overloaded Grid inevitably led to a degradation in performance in all cases, but the "ETT + File Location" and "Adaptive" algorithms gave the best E_{System} and $T_{Response}$ values. However the "ETT + File Location" values for $E_{Network}$ were the better of the two by more than 50%.

It is more difficult to pick out a scheduling algorithm that is definitively the most efficient for a Data Grid than it is for the simple CPU-bound Grid described in chapter 8. They have different strengths and weaknesses, and there will be a certain degree of subjectivity involved in deciding their relative importance. However it is likely that some minimisation of network costs will be desirable, ruling out a pure "Estimated Traversal Time" scheduler despite its strengths from a load balancing perspective.

Of the rest, "ETT + Queue Access" is the least effective, because its strengths do not usually apply in a HEP analysis context. Its impressive ability to minimise transfer times is only really effective when the running time of the job is relatively short. "ETT + File Location" is a very simple algorithm, but it produces good results, particularly in a very loaded Grid. However the "Adaptive" algorithm is the most consistent, resulting in high efficiencies and short response times in a variety of scenarios.

The "Economy" replica management algorithm consistently outperforms the "Least Recently Used" algorithm because it is able to learn from file access patterns. However the preemptive replication policies give more mixed results. "Economy + DataRandom" seems to have very little effect at all, because it is rare for a large enough imbalance in loading to occur, and thus trigger replication of data to another site. "Economy + Replication" has more obvious results though, as it is often able to improve $E_{Network}$ results. It can also lead to a decrease in other metrics, although this is more because of the scheduler being affected by the imperfect Information Services, rather than a problem with the "Economy + Replication" algorithm itself.

Chapter 11

Conclusions

11.1 Summary of Results

A simulation, named EDGSim, has been created with an object oriented structure to represent the middleware components of the European Data Grid. It concentrates on events representing job running, file transfer and interaction between the components.

In the first part of this work EDGSim was calibrated with real Grid logging information. This data was gathered by monitoring the behaviour of an active Resource Broker, as well as member sites that contribute their computational resources to the Grid. Parameters were extracted from this information relating to the timings of middleware functions, and job submission patterns. A simulated Grid was constructed in EDGSim using these parameters, and jobs with similar properties were submitted to both the real and simulated Grids. Metrics were devised to measure the efficiency of job scheduling from different perspectives - System Efficiency (E_{System}) and User Efficiency (E_{User}).

From comparison of these runs, it was determined that:

- Inefficiencies in the real Grid were dominated by delays incurred in the job scheduling mechanism of the RB and the Job Submission Service. When these delays were of the same order as the running time of the jobs, both E_{System} and E_{User} could be degraded by 10-40%, depending on the length of the delay.

- The EDG's "Estimated Traversal Time" scheduling algorithm could be improved by randomising the choice of resource in the case of a tie when ranking their suitability. This produced an increase in both metrics of 60%, decreasing when the system was heavily loaded (No. of jobs $\geq$ 2 * no. of CPUs).

- In the case of simple data-independent jobs it is possible to satisfy the requirements of users and resource owners simultaneously, i.e. optimising E_{System} also optimises E_{User} and vice versa.

The simulation was then developed further to simulate data dependent jobs, based on the type of job used to analyse particle physics data. Scheduling algorithms from different sources were introduced in order to determine the most effective way of managing such a workload: the "Estimated Traversal Time" algorithm again; "ETT + File Location" and "ETT + Queue Access", adapted from other Grid projects; and an "Adaptive" algorithm created for this work. Several strategies for the management of data with varying levels of complexity, based on other Grid simulation work, were tested alongside the scheduling policies, investigating how they complement each other. New metrics were introduced to quantify the efficiency of network use ($E_{Network}$) and job lifetimes (Response Time), as well as appropriate adjustments to existing metrics.

Runs were conducted initially on a simplified, homogeneous Grid, and then with more heterogeneous Grids with realistic resource distributions. Different job running times and loading levels were investigated to find out which job and data management policies were the most efficient, and the most stable under different conditions. It was found that:

- Any algorithm with even a basic awareness of data file distributions resulted in a better performance (measured by all metrics) than the randomised "ETT" algorithm used in data-independent runs.

- When file transfer times were larger than job running times, the "ETT + Queue Access" algorithm produced up to 300% better E_{System} than other data-aware algorithms, jobs completed 25% more quickly, and improvements of 20-40% in $E_{Network}$ were observed.

- For a heterogeneous Grid system with longer running jobs, the "Adaptive" algorithm gave a better performance than the others, with a 5% improvement in E_{System} and Response Time, and 10% better $E_{Network}$.

- In an overloaded Grid (no. of jobs $\geq$ 2 x no. of CPUs) the performance of all algorithms was degraded, but "ETT + File Location" still resulted in 100% better $E_{Network}$ than the other algorithms, and was competitive using other metrics.

- "Economy" data management typically produced a 10% increase in $E_{Network}$ compared to a basic "Least Recently Used" policy, and sometimes more for less network efficient schedulers. It also improved E_{System} and Response Time by around 5% in many cases.

- "Economy + Replication" produced an additional increase of up to 5% in $E_{Network}$ compared to "Economy" alone, but in some cases produced a 5% penalty in E_{System} and Response Time.

11.2 Future Directions

Trials of Data Grid technology so far have only employed rudimentary scheduling, as testing the functionality of middleware has been the initial priority. The appropriate policy to use will depend on the characteristics of the Grid (such as the number of resources and their distribution), and those of the jobs submitted to it (file access patterns, CPU cycles required, submission patterns, etc.).

While Grid resources and the volumes of data that will be generated by experiments can be estimated with some confidence, the manner in which experimental physicists will use them is less certain. Usage patterns are likely to differ from those in earlier experiments because the datasets involved will be so much larger, as will the pool of resources available to them. How physicists will choose to manage their workload remains to be seen, as does the means of regulating access to ensure that resources are shared fairly between users. A scheduling policy for a Data Grid will have to be flexible in order to respond appropriately to user requirements. Metrics such as those used in this thesis can be applied to Grid logging data in order to monitor the effectiveness of job and resource management. Efficiency can be increased by responding to this feedback, and adjusting management policies accordingly.

The performance of the Grid is described comprehensively with the following metrics:

- Response Time
- E_{System}
- $E_{Network}$

Response Time describes job performance more effectively than E_{User}, because it shows the range of job running times more clearly, as the results in the previous chapter indicate. It is difficult to determine an ideal value for these running times, due to ambiguities in a job's CPU requirements, and the power of the CPUs available in a Grid. However with appropriate normalisation it is possible to make useful comparative measurements with this metric.

The other job performance metric, Last Completion Time, can be a misleading measure, because it is not sensitive to the processing of individual jobs. It also has the problem that it requires a job batch of fixed size in order to make a measurement, and so it can only be meaningfully applied in a comparison of identical batches of jobs submitted under different conditions. For monitoring in an ongoing production-scale Grid, this metric will be less useful.

E_{System} and $E_{Network}$ measure the usage efficiency for the Grid's CPUs and network, which comprehensively describes resource usage. While job running and data management are coupled to some extent, they can be optimised individually. This makes it useful to measure these quantities separately, rather than creating a combined metric to measure resource usage in general.

The success of the adaptive scheduling algorithm introduced in this work can certainly be improved upon. Its branching structure came from consideration of the relative importance of load sharing and access to data as the overall workload increases. However the parameters chosen to determine when the scheduler should change its approach have not been optimised, suggesting that the balance between loading and data considerations could be improved upon. For instance, the algorithm that gave equal weighting to loading and data access resulted in relatively high network efficiency for a heavy workload, while the adaptive algorithm performed poorly in this respect. The latter policy's structure could be modified to behave more like the former under such conditions. This study indicates that an improved scheduler would:

- Cluster jobs around data in low loading conditions
- Give weighting to both load balancing and data distribution in high loading conditions

Further work could shed light on the behaviour of a Grid system under different conditions, indicating further modifications to this algorithm.

For analysis jobs with a running time significantly larger than file transfer times, job performance will not be improved significantly by preemptive data replication. However if it is implemented effectively, as in the case of the "Economy" model used here, it can significantly reduce network traffic. Again the efficiency of this policy can be improved, by optimising the profit margins used to determine whether replication takes place, or adjusting the interval between executions of the algorithm.

When devising any job or data management policy, it is important to consider the limitations imposed by inefficiencies in the functioning of Grid middleware. A policy that is strongly dependent on resource properties that change rapidly, such as the Queue Access function that monitors network activity, may suffer if there is an interval between decision making, and the effect of that decision. An algorithm that takes no account of the frequency of resource status updates may also develop problems, as with the "Estimated Traversal Time" scheduling algorithm for an underloaded Grid. Inefficiencies of the kind seen in the GridPP Resource Broker can cause serious problems - delays at this crucial stage were seen to cause large performance penalties, and in a highly loaded production Grid it will be very important to have a reliable, robust job scheduler.

A system such as a worldwide Data Grid is too complex to optimise for all resource and user efficiency considerations. In this case the objective is serving the needs of the HEP community, which means that the users' needs are the priority, and the Grid must be administrated with this in mind. CPU, storage and network resources must be coordinated effectively in order to handle the unprecedented demand placed upon them by the investigation of new physics.

The results presented in Chapter 8 were published in the IEEE journal *Transactions in Nuclear Science* [59].

Appendix A

Glossary of Grid Terminology

Term	**Definition**
Actor	Component of a Ptolemy II application
API	Application Programming Interface
CE	Compute Element
ClassAd	Condor Classified Advert
E_{Crude}	Crude Efficiency
$\mathrm{E}_{Network}$	Network Efficiency
E_{System}	System Efficiency
E_{User}	User Efficiency
EDG	European Data Grid
ETT	Estimated Traversal Time
GIIS	Grid Information Index Server
GridPP	UK Particle Physics Grid
GRIP	Grid Resource Information Protocol
GRIS	Grid Resource Information Server

GSI	Grid Security Infrastructure
IS	Information Service
II	Information Index (RB's cache for IS data)
JDL	Job Description Language
JSS	Job Submission Service
L_{CPU}	CPU Load
LB	Logging and Bookkeeping Server
LCG	LHC Computing Grid
LCT	Last Completion Time
LFN	Logical File Name
LRU	The Least Recently Used file at an SE
MDS	Metadata Directory Service
NM	Network Monitor
PFN	Physical File Name
PSA	Parameter Sweep Application
PII	Ptolemy II Modelling Package
QoS	Quality of Service
RB	Resource Broker
RC	Replica Catalog
RM	Replica Manager
R-GMA	Relational Grid Monitoring Architecture
SE	Storage Element
SDK	Software Development Kit
$T_{Response}$	Response Time
UI	User Interface
U_{CPU}	CPU Usage

Vergil	The Ptolemy II GUI
VO	Virtual Organisation
WN	Worker Node
WP	European Data Grid Work Package

Appendix B

The Logging and Bookkeeping Database

B.1 Database Schema

The columns in the tables belonging to the LB database schema are shown in tables B.1 to B.5. Each user has an entry in the **users** table; each job has an entry in the **jobs** table.

Job events will be entered in the **events** table, and some cases an event will receive multiple entries, as the same event is registered by more than one middleware entity (differentiated by the prog and host columns). In addition, the events will be registered in either the **short_fields** table or the **long_fields** table. The table used is determined by the event type, as some have a single word in the value column, and others have a long character string, such as the Condor ClassAd submitted with the job (see section 3.2.2). The structure of these two tables is identical.

The final table, **weeklystats**, has not been used in this work, as it appears to be unreliably updated. It contains an entry for each week of Grid running, with columns representing the number of each job lifetime event registered in that week. However many of these entries remain blank, making it an unreliable guide to job activity, leading to the reconstruction of job lifetimes described in chapter 5. The meaning of the event codes is described in the following section.

B.2 Event Codes in Database

The event codes registered in the Logging and Bookkeeping database are shown in table B.6, along with the stage in the job life cycle that they represent.

Not all of the codes are used, and in many cases there is overlap between them, or ambiguity in their meaning. For instance the JobAbort and JobFail codes fulfil similar functions, but the latter is not always a fatal error, and the job may be returned to the RB to be matched again. JobFail can refer to failure at the JSS stage, or later at the site level, and if the job's arrival at the chosen resource is not registered, the point at which the job failed can be unclear.

events	
Column of Table	Comments
jobid	Job ID
event	No. of event in job lifetime
code	Event code
prog	Middleware entity registering event
host	Middleware host
time_stamp	Event time stamp
userid	User ID of entity registering event
usec	Millisecond component of timestamp
level	Unused

Table B.1: Columns in the "events" table of the Logging and Bookkeeping database schema.

jobs	
jobid	Job ID
dg_jobid	Unique EDG job ID
userid	User ID of job owner

Table B.2: Columns in the "jobs" table of the Logging and Bookkeeping database schema.

users	
userid	User ID of job owner
cert_subj	User's certificate of authenticity

Table B.3: Columns in the "users" table of the Logging and Bookkeeping database schema.

short_fields / long_fields	
userid	User ID of job owner
event	No. of event in job lifetime
name	Name of event type
value	Other information associated with event

Table B.4: Columns in the "short_fields" and "long_fields" tables of the Logging and Bookkeeping database schema.

weeklystats	
rbsubmit	...
rbaccept	...
transfer	...
goodmatch	...
jssaccept	...
jsssubmit	...
run	...
done	...
clear	...
lastupdate	...

Table B.5: Columns in the "weeklystats" table of the Logging and Bookkeeping database schema.

Code	Name	Comments
0	Undefined	Unused
1	JobTransfer	Job transfer between middleware components
2	JobAccept	Job is accepted by middleware component
3	JobRefuse	Job is refused by middleware component
4	JobAbort	Job aborted at RB
5	JobFail	Job not successfully submitted by JSS
6	JobScheduled	Job scheduled to site (possibly queueing)
7	JobRun	Job has started running
8	JobChkpt	Unused
9	JobDone	Job has completed running
10	JobClear	Job results collected by owner
11	JobPending	Job ready to run
12	JobMatch	Job matched to resource by RB
13	JobStatus	Unused
14	JobCancel	Job cancelled by owner
15	SysCmpStat	Unused
16	SysClStat	Unused

Table B.6: The event codes in the Logging and Bookkeeping database.

Bibliography

[1] I. Foster, C. Kesselman, *The Grid; Blueprint for a New Computing Infrastructure*, Morgan Kauffmann, 1999.

[2] I. Foster, C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, Int'l J. Supercomputer Applications **11(2)** (1997) pg. 115–128.

[3] The OpenSSL Project, `http://www.openssl.org/`.

[4] A. Grimshaw, A. Ferrari, F. Knabe, M. Humphrey, *Legion: An Operating System for Wide Area Computing*, IEEE Computer **32(5)** (1999) pg. 29–37.

[5] S. Agrawal, J. Dongarra, K. Seymour, S. Vadhiyar, *Netsolve: Past, Present and Future - A Look at a Grid Enabled Server*, in: F. Berman, G. Fox, T. Hey (Eds.), Making the Global Infrastructure a Reality, Wiley Publishing, 2003.

[6] R. Buyya, D. Abramson, J. Giddy, *Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid*, in: HPC Asia, IEEE CS Press, 2000.

[7] H. Casanova, G. Obertelli, F. Berman, R. Wolski, *The AppLeS Param-*

eter Sweep Template: User-Level Middleware for the Grid, in: Proc. Super Computing Conference, 2000.

[8] Network Weather Service, `http://nws.cs.ucsb.edu/`.

[9] P. Nevski, T. Wenaus, A. Vaniachine, *Bilevel Architecture for High-Throughput Computing*, in: Proc. Computing in High Energy and Nuclear Physics, 2001.

[10] S. Larson, C. Snow, M. Shirts, V. Pande, *Folding@Home and Genome@Home: Using distributed computing to solve previously intractable problems in computational biology*, in: R. Grant (Ed.), Computational Genomics, Horizon Press, 2002.

[11] S. Brunett, D. Davis, T. Gottschalk, P. Messina, C. Kesselman, *Implementing Distributed Synthetic Forces Simulation in Metacomputing Environments*, in: Proc. Heterogeneous Computing Workshop, 1998.

[12] C. Mechoso, C. Ma, J. Ferrara, J. Spahr, R. Moore, *Parallelization and distribution of a coupled atmosphere-ocean general circulation model*, Monthly Weather Review **121** (1993) pg. 2062–2076.

[13] M. Perry, D. Agarwal, *Collaborative Editing within the Pervasive Collaborative Computing Environment*, in: Proc. Fifth Int. Workshop on Collaborative Editing, 2003.

[14] U. Joshi, CMS Virtual Control Room, `http://www.agsrhichome.bnl.gov/RemOp/docs/joshi_Shelter_Island_Slides.pdf`.

[15] G. von Laszewski, I. Foster, J. A. Insley, J. Bresnahan, C. Kesselman, M. Su, M. Thiebaux, M. L. Rivers, I. McNulty, B. Tieman, S. Wang,

Real-Time Analysis, Visualisation and Steering of Microtomography Experiments at Photon Sources, in: Proc. Ninth Conf. on Parallel Processing for Scientific Computing, 1999.

[16] CosmoGrid, http://www.escience.cam.ac.uk/projects/cosmogrid/.

[17] *ATLAS Detector and Physics Performance Technical Design Report*, Tech. rep., The ATLAS Collaboration (1999).

[18] The Collider Detector at Fermilab (CDF), http://www-cdf.fnal.gov/.

[19] B. Huffman, R. McNulty, T. Shears, R. St. Denis, D. Waters, *The CDF/D0 UK GridPP Project*, CDF Note 5858 (2002).

[20] The MONARC Architecture Group, http://monarc.web.cern.ch/MONARC/.

[21] Particle Physics Data Grid, http://www.ppdg.net/.

[22] iVDGL (International Virtual Data Grid Laboratory, http://www.idvgl.org/.

[23] GriPhyN (Grid Physics Network), http://www.griphyn.org/.

[24] NEESgrid, http://www.neesgrid.org.

[25] Earth System Grid, https://www.earthsystemgrid.org/.

[26] NASA Information Power Grid, http://www.ipg.nasa.gov.

[27] The European DataGrid Project, http://web.datagrid.cnr.it/LearnMore.jsp.

[28] The LHC Computing Grid Project, `http://lcg.web.cern.ch/LCG/`.

[29] CrossGrid, `http://www.crossgrid.org`.

[30] GRIDSTART, `http://www.gridstart.org`.

[31] Openlab, `http://proj-openlab-datagrid-public.web.cern.ch/proj-openlab-datagrid-public/`.

[32] DOE Science Grid, `http://www-itg.lbl.gov/Grid`.

[33] Astrophysical Virtual Observatory (AVO), `http://www.eso.org/projects/avo`.

[34] DataTAG, `http://www.datatag.org`.

[35] Global Grid Forum, `http://www.gridforum.org`.

[36] EDG Work Package 1, `http://server11.infn.it/workload-grid/`.

[37] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, S. Tuecke, *Condor G: A Computation Management Agent for Multi-Institutional Grids*, in: Proc. Tenth IEEE Symp. on High Performance Distributed Computing, 2001.

[38] *Information and Monitoring Services Architecture*, Tech. rep., EDG Work Package 3 (2004).

[39] The MONARC Architecture Group, `http://monarc.web.cern.ch/MONARC/`.

[40] R. Buyya, M. Murshed, *GridSim: A Toolkit for the Modelling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, The Journal of Concurrency and Computation: Practice and Experience (CCPE) **Vol. 14**.

[41] A. Legrand, J. Lerouge, *MetaSimGrid : Towards realistic scheduling simulation of distributed applications*, ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2002/RR2002-28.ps.gz (2002).

[42] Bricks, `http://www.is.ocha.ac.jp/~takefusa/bricks/`.

[43] MicroGrid, `http://www-csag.ucsd.edu/projects/grid/microgrid.html`.

[44] ChicSim, `http://people.cs.uchicago.edu/~krangana/ChicSim.html`.

[45] OptorSim, `http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html`.

[46] Ptolemy II, `http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm`.

[47] ROOT, `http://root.cern.ch/`.

[48] D. Colling, *Quality and Performance Indicators for DataGrid*, EDG Draft Document, http://edms.cern.ch/document/386039 (2003).

[49] S. Orlando, P. Palmerini, R. Perego, F. Silvestri, *Scheduling High Performance Data Mining Tasks on a Data Grid Environment*, Proc. Int. Conf. Euro-Par, 2002.

[50] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, A. Keren, *An Opportunity Cost Approach for Job Assignment and Reassignment in a Scaleable Computing Cluster*, IEEE Trans. on Parallel and Distributed Systems **11**.

[51] A. Gantman, P.-N. Guo, J. Lewis, F. Rashid, *Scheduling Real-Time Tasks in Distributed Systems: A Survey*, http://www.cs.ucsd.edu/classes/fa98/cse221/OSSurveyF98/p8.pdf (1998).

[52] D. Colling, Private conversation (2003).

[53] D. Cameron, R. Carvajal-Schiaffino, A. Millar, C. Nicholson, K. Stockinger, F. Zini, *Evaluating Scheduling and Replica Optimisation Strategies in OptorSim*, in: 4th Int. Workshop on Grid Computing, 2003.

[54] K. Ranganathan, I. Foster, *Computation and Data Scheduling for Large-Scale Distributed Computing*, `http://www.cs.uchicago.edu/~krangana/papers/Sched_extended7.pdf` (2002).

[55] A. Takefusa, O. Tatebe, S. Matsuoka, Y. Morita, *Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications*, in: Proc. 12th IEEE Int. Symposium on High Performance Distributed Computing, 2003.

[56] LCG Resource Planning, `http://lcg.web.cern.ch/LCG/peb/rc_resources/`.

[57] T. G. Collaboration, *The GridPP2 Proposal*, Tech. rep., http://www.gridpp.ac.uk/docs/gridpp2/ (2003).

[58] D. Cameron, R. Carvajal-Schiaffino, A. Millar, C. Nicholson, K. Stockinger, F.Zini, *UK Grid Simulation with OptorSim*, in: eScience UK All-Hands Meeting, 2003.

[59] P. Crosby, D. Waters, D. Colling, Efficiency of resource brokering in grids for high energy physics computing, IEEE Trans. Nuclear Science **51(3)**.