A Study in Grid Simulation and Scheduling

Mark N.C. Rhodes

M.Sc. Thesis

Department of Computer Science
Durham University

March 2006

Abstract

Grid computing is emerging as an essential tool for large scale analysis and problem solving in scientific and business domains. Whilst the idea of stealing unused processor cycles is as old as the Internet, we are still far from reaching a position where many distributed resources can be seamlessly utilised on demand. One major issue preventing this vision is deciding how to effectively manage the remote resources and how to schedule the tasks amongst these resources. This thesis describes an investigation into Grid computing, specifically the problem of Grid scheduling. This complex problem has many unique features making it particularly difficult to solve and as a result many current Grid systems employ simplistic, inefficient solutions.

This work describes the development of a simulation tool, G-Sim, which can be used to test the effectiveness of potential Grid scheduling algorithms under realistic operating conditions. This tool is used to analyse the effectiveness of a simple, novel scheduling technique in numerous scenarios. The results are positive and show that it could be applied to current procedures to enhance performance and decrease the negative effect of resource failure. Finally a conversion between the Grid scheduling problem and the classic computing problem SAT is provided. Such a conversion allows for the possibility of applying sophisticated SAT solving procedures to Grid scheduling providing potentially effective solutions.

Contents

List of Figures

List of Tables

List of Equations

Acknowledgements

I would like to thank my supervisor Prof. Iain Stewart, for putting up with me and also Andy Lloyd, the lot from Victoria C and the lovely Laura.

Software Availability

The G-Sim toolkit software with source code and example experiments can be downloaded from the website: www.dur.ac.uk/m.n.c.rhodes/.

Statement of Copyright

Chapter 1 – Introduction

1.1 The Problem

It has recently been suggested that by 2015, developments within Computer Science will trigger a scientific revolution on par with the invention of calculus or the telescope [EC2006]. Computing no longer merely helps scientists with their work, its concepts, tools and theorems have become integrated into the fabric of science itself [MI2006]. As we are looking to solve ever more complex problems with increasingly sophisticated techniques, we require ever more computing power to make this possible. From the need to facilitate large-scale problem solving, the Grid paradigm has emerged. Grid computing will drive this revolution by providing seamless access to vast computing power not previously accessible.

Whilst there are numerous distributed computing applications and Grid resource management systems, there remain technical obstacles that must be overcome before Grid computing is both ubiquitous and seamless. One such obstacle, which is the subject of this work, is deciding how to effectively and fairly distribute the tasks amongst the available resources.

Although multiprocessor scheduling is a well understood classic computing problem, scheduling over Grids is a relatively new field of research and one which has proved more complex. The additional complexity stems from the difference between the nature of multiprocessor clusters, which are typically homogenous, housed in a single building and controlled by an individual organisation, and Grids which are unpredictable, heterogeneous, susceptible to individual resource failure, potentially larger scaled and geographically distributed with no centralised control.

Since the subject of Grid scheduling is in its infancy, the majority of the algorithms currently employed are simplistic and ineffective in many scenarios. Although different approaches have been tried, few have yet to be pursued in significant depth

and detail. Work is required to find procedures which are robust and reliable, delivering high-quality performance in a variety of different Grid scenarios.

In order to compare the performance of potential scheduling solutions, one must be able to conduct repeatable, statistically valid experiments. Since Grids are by nature unpredictable, they are not effective as test beds for these algorithms, and so to obtain experimental data we must employ the use of realistic, scalable Grid simulators. Although numerous Grid simulators have been developed for this purpose, few have managed to strike an effective balance between simulation speed and realism. It is also not a case of "one-size-fits-all" since the algorithm may be designed around a specific application and set of resources. This means one must remodel the simulator to effectively capture the scenario; a task that usually requires expert knowledge of its construction and operation.

1.2 Proposed Work

This thesis represents an attempt to contribute to the study of the Grid scheduling problem through the development of a practical algorithm verification tool. This can be utilised to model numerous Grid scenarios without the requirement of further redevelopment on the part of the user. This tool, G-Sim, is developed to facilitate the simulation of large-scale, realistic Grids on a stand-alone PC, a tool which all researchers will have access to.

Furthermore this work looks to add to the understanding of how scheduling algorithms are affected by the properties of the Grid they schedule on. This is accomplished through analysis of experimental results taken from numerous simulated Grid scenarios.

Finally, the work of this thesis endeavours to add to the catalogue of techniques available to Grid scheduling algorithm designers. To this end, both a simple, novel scheduling mode and a method which allows SAT solving algorithms to be applied to Grid scheduling are developed.

1.3 Document Structure

The rest of this thesis is arranged as follows:

Chapter Two is intended to give the reader an overview of the subject of Grid computing and to give more detailed information concerning the issues and techniques employed in both Grid simulation and scheduling.

Chapter Three documents the design and implementation of the Grid simulation tool G-Sim, describing how it operates and models various aspects of real life Grids.

Chapter Four outlines a simple scheduling technique Time-Limit Batch Mode which could be employed to improve the effectiveness of scheduling procedures and to make them more suitable for operation on a Grid. The effectiveness of the technique is verified by thorough testing against other documented procedures in numerous scenarios using G-Sim. Conclusions are drawn from the experimental results and further research possibilities discussed.

Chapter Five describes a method for converting the problem of Grid scheduling to the well known computing problem SAT. This method is included primarily as a theoretical tool to demonstrate how a new approach to Grid scheduling could be made. Unfortunately due to time constraints we were unable to provide experimental results using this approach. A discussion on the limitations of the conversion and its practical application is also provided.

Chapter Six provides a discussion of general conclusions from the work, together with recommendations for further work stemming from it. It also contains a short summary of the work presented in the thesis and a discussion of whether the outcomes presenting in section 1.2 have been met.

Chapter 2 - Background

2.1 Introduction

The last two decades have seen a revolution in computing technology. The Internet has continued to grow at an exponential rate [CO2001] and processing power and network bandwidth is substantially more powerful and widely available. These technologies have led to the possibility of pooling Internet-connected resources to solve large-scale parallelised problems, leading to what is popularly called Grid computing [BU2002]. The term Grid comes from an analogy with the electric power Grid which provides constant, reliable, transparent access to electricity irrespective of source [BU2002]. Instead of electricity, Grid computing will provide secure, transparent, highly efficient access to computational and information resources, irrespective of their physical location. Grid computing is increasingly being viewed as the next phase of distributed computing [GF2005].

Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent [BU2002]. The Grid infrastructure is about bringing together distributed resources such as supercomputers, data sources and specialised measurement systems to enable collaborative working on interdisciplinary projects that could not be otherwise possible. It enables seamless access to resources that simply could not feasibly be installed on the same site.

The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high throughput computing, distributed supercomputing, and service-oriented computing [BU2002]. Computational Grids are characterized by widely distributed computational resources shared by virtual organisations [DE2003]. A virtual organisation can be considered a group of individuals or organisations that share computing resources to attain some common goal.

The Grid is not only a computing infrastructure for solving large scale parallel problems but a structure which will bond and unify all types of distributed resources from metrological sensors to data vaults, from super computers to PDAs, to provide pervasive services to anyone who needs them [BU2002]. The Grid paradigm also has the potential to provide a complementary approach to achieving high-reliability with little additional investment, an infrastructure for large-scale load balancing and a means of exploiting under-utilized resources [ZH2004]. In general the Grid community is more focussed on the sharing of high-end machines such as supercomputers, with the Peer-to-Peer (P2P) community involved in the sharing of low-end machines such as desktop PCs and their contents [BU2002]. (P2P computing is now considered mainstream and a significant social and technological phenomenon, with millions of Internet users involved. P2P systems provide an infrastructure for communities that share CPU cycles (e.g., SETI@Home, Entropia) and/or storage space (e.g., Napster, FreeNet, Gnutella), or that support collaborative environments (Groove) on low level machines [RI2002]. However there are numerous technical and social hurdles to overcome before the vision of seamless collaborative research being accomplished on truly global grids can be realised. Such hurdles involve areas including scheduling, code management, configuration, fault tolerance, security, and payment mechanisms [FO1998].

2.2 Technological Advancement: Enabling Grid Computing

"Lick had this concept of the intergalactic network on which he believed everybody could use computers anywhere and get at data anywhere in the world. He didn't envision the number of computers we have today by any means, but he had the same concept – all of the stuff linked together throughout the world, that you can use a remote computer, get data from a remote computer, or use lots of computers in your job. The vision was really Lick's originally."

Larry Roberts, Principle architect of APRANET, on the vision of the psychologist J.C.R. Licklider ('Lick').

The roots of grid computing can be traced back to the birth of the Internet. The vision J.C.R. Licklider took to ARPA in the 1960's when he initiated the research projects that led to the ARPANET, which in turn became the present day Internet described above, is close to the goals of the UK e-Science initiative set out by John Taylor which looks to develop grid technology to enable scientists to perform 'faster, better and different' research [HY2004]. We briefly discuss the advances in computing, networking and distributed computing that have brought such a vision within touching distance.

Computing power has substantially increased in the last 40 years. The prediction made by Gordon Moore, co-founder of Intel in 1965, commonly known as Moore's law, that the number of transistors on a chip would double about every 2 years and that they would halve in price, has held into the new millennium, fuelling a worldwide technological revolution [IN2005]. This law, based on silicon chips, will inevitably end within a decade as the size of transistors reaches atomic sizes [HE2004], although promising new computing methods such as quantum computing are in development [WE2000]. The continual reduction in the cost of manufacturing computers and the evolution in ever more sophisticated hardware has caused a shift from a few highly expensive mainframes in the 1960's to the millions of high powered low cost desktops and, more recently, portable devices we have today. IBM mainframes took off in the early 1960's; such machines required large dust-free rooms and were so expensive only a handful of large organisations could afford them. A revolution in the size of computers (which has increased their popularity and availability) came in the early 1970's with the invention of minicomputers, which were in turn superseded by the development of Personal Computer (PC's) in the early 1980's. The trend looks set to continue with hand-held Personal Data Assistants (PDA's) introduced in 1994 becoming ever more powerful and compact. The massive increase in processing power has led to the possibility of solving genuinely interesting and potentially useful large scale problems that simply would have been impossible just a few decades ago.

Continued growth of the global Internet is one of the most interesting and exciting phenomena in networking [CO2001]. Twenty years ago, the Internet was a research project that involved a few dozen sites; today it is accessible to millions of people in

every populated country in the world. In America it connects nearly every corporation, school, college, university, military organisation, government office and the majority of private residences [CO2001]. It started as a modest research project during the Cold War of the 1960's, during which time the United States government realised their potential to provide secure, reliable communication in the event of war. The Advanced Research Projects Agency (ARPA) was started in response to the USSR's 1958 launch of the first artificial satellite Sputnik. This agency was responsible for funding the projects which led to the start of the ARPANET in 1969; a network initially consisting of 4 academic sites linked by 56kbps telephone links to pass messages [AH2001]. This network continued to grow and by the mid-1970's linked over 30 academic, military and government contractors and its user base expanded to include the larger computer science research community [BU2002]. The Ethernet came about in 1976 following Bob Metcalfe's PhD Thesis of 1973 and in 1974 a common, reliable connection orientated in TCP was developed and was later split into TCP/IP in 1978 [BU2002]. Responsibility and management of the ARPANET passed to the National Science Foundation (NSF) in 1989; during this time many of the Internet rules and much of the etiquette were developed and steps were taken to privatise the Internet. This allowed the NSF contractors to build the structures for providing commercial Internet services, allowing companies and home users Internet access. The NSF officially terminated their control over the Internet in 1995 [AH2001].

The Internet is commonly associated with the Web, and the two are often confused. The Web was developed in 1989 by Tim Berners-Lee of CERN, Switzerland [BU2002]. It provides a means of presenting and sharing information in a structured, platform-independent manner. Using the HTML language, you are able build a web page of information that links to other web pages, allowing for the development of web sites, consisting of numerous related pages. These sites can in turn be linked together to form a Web of interconnected documents and information. HTML is transported using HTTP protocols and uses the Client/Server paradigm to allow access to these web pages in a distributed seamless manner. The invention of the web has sparked numerous supplementary technologies which look likely to provide the basis for large scale Grid computing, including XML and Web Services. XML is a standard format for information exchange over the Internet, developed in 1998 by the

World Wide Web Consortium (W3C). XML is used by Web services which allow access to remote software and applications through a web browser.

The idea of harnessing unused CPU cycles to process distributed applications was first considered in the 1970's when computers were first connected by networks [BU2002]. The first distributed computing programs were a pair of programs called Creeper and Reaper which made their way through the nodes of the ARPANET; they are also considered to be the first infectious programs ever to have been written [AH2001]. Creeper was a program which copied itself from one node to another and Reaper was responsible for deleting copies of Creeper. Although the programs had no practical significance they provided proof that idle computational power could be remotely harnessed and so distributed computing was born. The first distributed program to have functional use was the XEROX PARC worm invented in 1973 [AH2001]. The worm made use of around 100 computers connected by an Ethernet local area network to perform rendering of computer graphics [AH2001]. Within recent years there has been a shift from developing stand-alone programs to Internet applications; distributed computing is now an integral part of almost all software development. Modern programming languages such as Java and Microsoft's .NET platform have standard built-in tools to support many distributed computing technologies and standards such as Multithreading, Sockets, RMI, CORBA, Web Services, and Soap/XML. Distributed applications have the advantages of being able to be remotely accessible, fault tolerant and efficient, and they allow for distributed resources to be used transparently. Grid computing is about using distributed computing technologies to develop applications which provide a means of accessing remote services and resources, and so enabling large scale multi-institutional problems to be solved efficiently.

Already a wide range of fields are incorporating and embracing Grid technology. It has already been used in genome research [IB2001], cosmological simulations [VU2005], operations research, ecological modelling [BU2002], drug design [OP2004] and to search for signs of extraterrestrial intelligence in satellite data [SE2003].

2.3 Grid Computing Environments and Technologies

Grid computing strives to aggregate diverse, heterogeneous, geographically distributed and multiple-domain-spanning resources to provide a platform for transparent, secure, coordinated, and high-performance resource-sharing and problem solving [FO1998]. There are two players in the Grid paradigm, the producers (also called resource owners) and the consumers (who are the users) [BU2002]. The producers supply access to their resources as a service, typically at a cost to the consumers. The consumers who remotely access the resources have different goals, objectives, strategies, and demand patterns [AB000].

According to Buyya in [BU2002], there are four main aspects which characterise a Grid:

- Multiple Administrative Domains and Autonomy: The resources are geographically distributed, often in different time zones, owned by a number of different organisations and are under the jurisdiction of a number of different administrative domains. The autonomy of resource owners must be honoured and their local resource managements and usage policies adhered to.
- Heterogeneity: A Grid involves a vast range of different resources encompassing a vast range of technologies.
- Scalability: A Grid can grow in size from a few integrated resources to millions. This raises the problem of potential performance degradation. Consequently, Grid applications requiring many resources must be designed to be tolerant to changes in both bandwidth and latency.
- Dynamicity and Adaptability: In a Grid, due to the potentially large number of resources, the chance of an individual resource failure is high, and so should be considered a rule, rather than the exception. Application and resource management systems should behave dynamically to cope with such failure and use the available resources and services efficiently and effectively.

Grid Systems can be divided into different categories according to their target application type and topology [ZH2004], although there are no definite boundaries

between the categories, and real Grids can be constructed from two or more of these types. The types of Grid system are:

- Computational Grids: look to utilise the aggregate computational power of numerous machines to process jobs faster. According to how the computational power is utilized, this category can be further subdivided into distributed supercomputing and high throughput systems [ZH2004]. A high throughput Grid aims to utilise idle computational cycles to process a stream of different tasks, whereas a distributed supercomputing Grid runs parallel applications on multiple processors simultaneously to reduce execution times.
- Data Grids: seek to harness geographically distributed resources for large-scale data intensive problems [RA2002]. They allow large data sets to be managed, accessed and utilised by large numbers of users, such that the physical location of the data is abstracted from them.
- Storage Grids: attempt to aggregate the spare storage resources of numerous machines, and provide users with transparent secure access to the data stored there [ZH004]. Such Grids are usually considered to be of more significance to the P2P community rather than the Grid community.

2.3.1 Grid Taxonomy

The three Grid topology categories relate to the nature of the network they operate over; Figure 2.1 shows how these Grid topologies are related.

- IntraGrids: are typically constructed of machines on a single intranet network. These will usually be owned by a single organisation and operate under one usage policy. Scheduling algorithms are typically simpler to design for an IntraGrid as there is a reduced chance of resources changing and network performance is likely to be both high speed and similar between all machines.
- ExtraGrids: are constructed from multiple IntraGrids and operate over a WAN. They involve multiple administrative domains, possibly making use of a centralised scheduling system impossible. The assumptions about the network which may hold with an IntraGrid may not necessarily apply with an

ExtraGrid. These Grid environments are created for mutual benefit between trusted business partners [ZH2004].

- InterGrids: operate over the Internet and aggregate resources across the globe. There is a need for strict security mechanisms to be employed as sources within an InterGrid should not be considered trusted. Due to the highly distributed nature of such systems, network service quality will potentially be the most significant factor affecting scheduling decisions. Centralised and truly hierarchical scheduling structures will not be viable due to a lack of any centralised control. Such an environment should be employed when there is a need to utilise many resources from many different geographical areas.

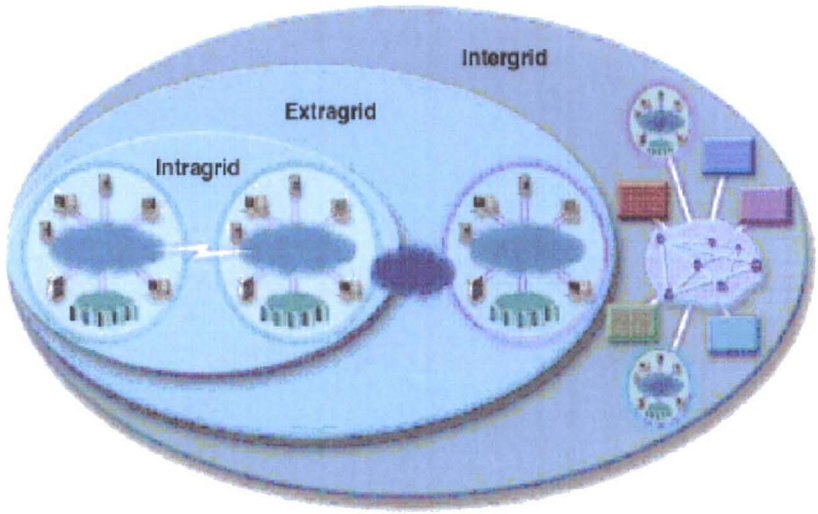

Figure 2.1: The relationship between IntraGrids, ExtraGrids and InterGrids.

2.3.2 Grid Components

There are many components to a Grid environment, with the architecture of a Grid system layered in order to maximise modularity. Figure 2.2 shows a layered Grid architecture and components which make up each layer.

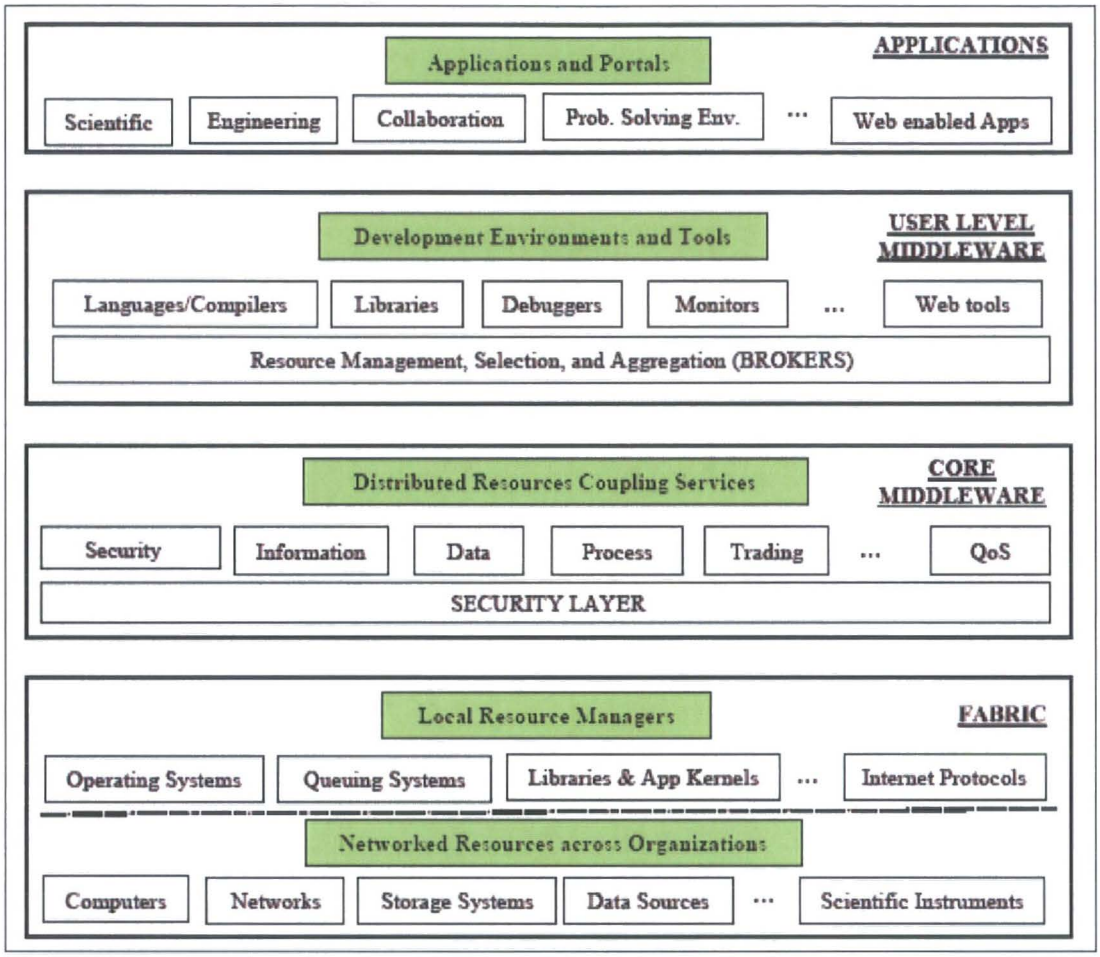

Figure 2.2: A layered Grid architecture and components [BU2002]

The Grid fabric consists of two sections, the physical computational, data or sensor based resources together with the networks which link them and the local resource management systems which operate on the same the local network as the resources and control how they are remotely accessed and harnessed. Examples of local resource management system include Condor, LSF (Load Sharing Facility), PBA (Portable Batch System), and SGE (Sun Grid Engine) [BU2002]. Typically a Grid application will be able to pool resources from a number of different local management systems, to enable a larger scale application execution environment.

The core middleware layer is composed of software and protocols which enable local management systems to communicate with the wider world, i.e., other Grid software operating outside the local area network. It offers the services which are required by all types of Grid applications, such as the ability to make data transactions, stop, pause

or start remote jobs, specify Quality of Service (QoS) requests and contracts and reserve resources.

The user-level middleware uses the underlying core services to implement services of direct use to the user such as developments tools and resource management brokers. The purpose of this layer to provide usable tools for accessing the lower level core services and to ease the process of developing a Grid enabled application. Resource management brokers such as Nimrod-G are the gateway through which the users' Grid application seeks execution. The users are only interested in seeing the output from their applications and wish to know when the output will be produced and how much it will cost them. Resource management systems allow the details about resource selection, QoS contracts, data staging, rescheduling, and results collection to be abstracted from the user.

The top layer of the Grid architecture model is the application level. The application is constructed by the user and is specific to their needs and development methods. Grid applications need to be aware of the nature of the environment they are executed in and so are typically highly parallelised and make use of Grid enabled languages and utilities such as MPI (message passing interface) or the Nimrod specification language [BU2002]. An example application, such as a parameter simulation, would require computational power, access to remote data sets, and may need to interact with scientific instruments [BU2002].

2.3.3 Emerging Standards

The Open Grid Services Architecture (OGSA) [OG2005] and its associated implementation, the Globus toolkit, are becoming the de facto standards for Grid services and Grid environments for application development, respectively [ZU2004]. These technologies were developed with the Global Grid Forum (GGF) which is the community of users, developers, and vendors leading the global standardization effort for grid computing [GF2005].

The definition of the OGSA as a development standard is considered an important step towards creating a seamless Grid infrastructure encapsulating services and resources worldwide [CH2004]. The OGSA provides a framework for combining XML based Web Services and Grid technology by extending the W3C web service standards such as SOAP, Web Services Description Language (WSDL) and WS-inspection to meet Grid specific requirements and concepts [CH2004]. Web Services provide a standard means of calling remote procedures over the Internet; they provide a set of well-defined interfaces for service discovery, dynamic service invocation, lifetime management and notification [FO2002]. Coupling them with Grid technology looks to bring manageability, extensibility and interoperability between loosely coupled services across Grids [CH004]. The OGSA is currently being re-factored into a number of standards, the Web Services Resources Framework (WSRF) and Web Service notification (WS-notification), yet its concepts remain the same [CH2004].

The Globus toolkit [GL2005] based on the OGSA, is currently considered the standard implementation for providing the core middleware services and many user level middleware systems are built upon it including Nimrod-G, Condor/G, AppLeS and the GrADS system [DA2002]. The intention of the Globus project is to provide the infrastructure and low-level mechanisms (in the form of command line utilities and API's) to run a single task on any given system; the task of optimal resource discovery is that of a resource management system [PH2002]. Its major contribution is a PKI-based Grid certificate solution to the Grid security problem, which enables cross-organizational resource access control [LI2003]. It also provides toolkits and mechanisms for job submission and monitoring (GRAM), resource discovery and status monitoring (GRIS and GIIS) and resource reservation and allocation (GARA).

2.3.4 Resource Management Systems

Grid technology is rapidly developing and there are hundreds of projects dedicated to it. Whilst the OSGA and Globus toolkit are setting the standard in providing the core middleware, there are numerous different user-level systems looking to provide solutions to the problems of resource management. Due to the complexity of Grid

resource management, the diversity of such middleware is set to continue. This is because no one system can be considered definitive and capable of catering for the needs of all Grid applications running on all Grids. A comprehensive taxonomy and survey of popular resource management systems is presented in [BU2002]. The following systems are currently used and look set to become standard Grid user-level middleware.

2.3.4.1 Condor

The Condor project has undertaken research into distributed computing for the past 18 years, and the Condor system [LI1988] it has developed is a widely adopted software package firmly established within current Grid computing [CH2004]. Condor is a high-throughput computing environment which aims to hunt down idle workstations and utilise their unused cycles to share job execution. Condor offers check-pointing and job migration which enables it to reschedule tasks to different workstations so that previously accomplished results will not be lost [ZH2004] and they are eventually completed. Although Condor is popularly known for harnessing unused cycles, it can also be configured to share resources and offers powerful and flexible centralised resource management services [BU2002]. Condor is a mature system which offers a number of unique and impressive features making it a good choice for running Grid applications, although these must consist of independent, single processor tasks. One such feature is its remote system calls which allow a job's original execution environment to be simulated on the execution machine, even if the two environments do not contain the same file system or ID scheme [BU2002].

Recently, Condor has been adapted to integrate it with the Globus toolkit; this version is known as Condor/G [FE2002]. By incorporating Globus, Condor/G can pool resources from multiple institutions [ZH2004].

2.3.4.2 Legion

Legion [LE1996] is an object-based Grid operating system developed at the University of Virginia [BU2002]. The philosophy behind Legion is to present a Grid system as a single, coherent, virtual machine [BU2002], abstracting out the

geographically distributed, heterogeneous nature. Legion is an object-oriented system consisting of independent disjoint objects that communicate with one another via method invocation [ZH2004]. Individual objects are responsible for managing a single resource such as a HostObject, which defines attributes of a computation resource such as load, CPU capability and memory size, and VaultObjects which relate to a persistent storage device. Jobs are also considered as Objects, which are in themselves active threads which can be activated, deactivated and stored. As Legion provides an API for object interaction but not a language or communication protocol, users are not restricted in the applications they can develop and run on Legion [BU2002]. Legion also supports advance reservation of resources and rescheduling through the incorporation of job monitoring. Its resource management is in a hierarchical architecture, with a centralised information service and decentralised scheduling policies [BU2002]. Although default system orientated scheduling policies are implemented, Legion allows policy extensibility through the use of resource brokers, and so Nimrod-G and AppLeS brokers can be used in conjunction with Legion to provide user-centric policies [BU2002].

Currently Legion can only be used on single-domain Grids [ZH2004], therefore is unsuitable for large scale Grid applications, or applications which require the use of unique resources such as scientific equipment from outside the local network.

2.3.4.3 Nimrod-G

Nimrod-G [BU2002] was developed by Monash University and looks to provide user-level middleware based on an economic model for resource management and scheduling. This means that Nimrod-G's application-orientated scheduling is done on the basis of users' deadlines, budgets and optimisation strategies. As with Condor/G, the "G" in Nimrod-G refers to it's the fact that it is capable of using the core middleware services supplied by the Globus toolkit. Nimrod-G can also be used in conjunction with other middleware systems too, including Legion, making it suitable for many different Grids. It is based upon the GRACE computation economy services infrastructure [BU2002] allowing for the integration of resource reservation and soft QoS demands [BU2002]. Nimrod/G is designed only for parametric study applications where there is a large set of independent parameters, which are split into

a number of independent tasks. These studies must be implemented through a declarative parametric modelling language or a GUI [BU2002]. Each application specifies a deadline by which it expects to be completed (based on capability estimation performed through heuristics and previous execution times), and a price which the user is willing to pay for its completion. Each computational resource is specified a cost which the consumer should pay in order to use the resource [ZH2004]. As Nimrod-G implements user-centric, economic based scheduling policies, it scheduling organisation is inherently decentralised with each broker instance looking to meet the demands of the user, like in a real market environment.

2.3.4.4 AppLeS

The name AppLeS is derived from Application Level Scheduling, which is further discussed in section 2.4.2.5. The AppLeS project [BE1996] is focussed on scheduling and is designed to use underlying middleware services provided by Legion, Globus or Netsolve [CA1997], it also uses the Network Weather Service (NWS) [WO1999] to find resource information on demand. The idea behind AppLeS is to imbed AppLeS agents into the applications so they become self-schedulable on the Grid [BU2002]. Scheduling is based around the effects of resource selection on the specific application, and is therefore inherently decentralised [ZH2004]. The completion time estimation is based upon predictive heuristic states and online rescheduling is supported [BU2002]. AppLeS has already proven successful in a number of application areas, such as Tomography and Gene Sequence Comparison [BU2002].

2.3.4.5 Sun Grid

The Sun Grid project is a commercial project operated by Sun Microsystems Incorporated. It provides a means of accessing large scale computing power and data storage on a pay as you use basis [SU2005]. It uses Grid technology to allow users to remotely submit jobs to dedicated Sun Grid centres, as an Internet-based utility service [SU2005]. The service provides users with secure, reliable computational power on-demand. The Sun Grid is currently available to certain Sun customers, but some recent large-scale grid deployments have forced Sun to divert systems that were to be used for the public site [INF2005].

2.4 Grid Scheduling

Grid scheduling is concerned with the mapping of tasks to a set of resources. The importance and difficulty of application scheduling for the development and deployment of applications on computational grids has been recognised for some time [FO1998].

In traditional scheduling, scheduling is defined as allocation to resources over time, taking into account some performance measurement criteria and subject to the satisfaction of constraints [PI1995]. With this view, grid scheduling is the problem of allocating tasks to distributed computational resources using system utilisation, sum of completion times [RE2004] or the makespan of the tasks as the performance measurement, whilst satisfying economical system constraints and users' service level agreements [BU2000]. The makespan is defined period between the first job being submitted to the time the last job is completed. The major difference between traditional scheduling problems and grid scheduling appears to the dynamic nature of the resources and the constraints in the grid environment [OP2003].

The resources available to Grid tasks can be anything that is connected to the network they are running on; if this is the Internet then it could be a PDA in China or a super computer in America. Such resources obviously have different processing capabilities, which will affect how quickly they can process the tasks submitted to them. A Grid's resources may use numerous different operating systems which may also affect the running times of different jobs; for instance, a UNIX machine may perform faster on a task involving lots of symbolic computation than a Windows machine, but may perform slower on jobs with a lot of floating point arithmetic [RI2003]. The dynamic nature of the computational resources emanates from the fact that they are outside the control of the scheduler allocating the tasks. They could be switched off, lose connection to the network or be subject to changes in local usage at anytime. Another characteristic of the resources that makes Grid scheduling more difficult is the dynamic and largely unpredictable nature of the environments under which they communicate; wide area large scale networks. There are numerous factors

which affect the performance of WANs including load, bandwidth and latency, making it impossible to accurately predict exactly how long a given transfer will take.

Today's grid applications are also relatively complex and are not built as monolithic entities, rather they are structured as workflows that consist of individual tasks, which consist of various transformations on large data sets [DE2003]. The tasks may have different resource requirements; for example, some may require the use of a specific supercomputer, a Linux-based cluster of processors or the use of an extremely large database generated from a network of satellites [BL2004]. These requirements come in two categories, hard or soft. Hard requirements are those which are rigidly enforced, soft requirements are those which are desirable but not absolutely essential [OP2003]. A common feature of grid applications is that they may require access to large data sets and usually produce another data set which can be shared and replicated once created [DE2003]. This means that it may be necessary to copy the required input files to a resource prior to executing a particular job, a task whose length is difficult to predict and which will ultimately affect the number of resources suitable to run the job and the time to process it.

Finding an optimal schedule in a heterogeneous computing environment has been shown, in general, to be NP-hard (it is a generalised reformulation of problem SS8 from [GA1979]). There is no known way of feasibly finding an optimal schedule without testing every possible one, which gives m^j possibilities, where m is the number of available machines and j is the number of jobs to be scheduled. With the possibility of extremely large numbers of jobs and resources, a complete search on such a space is infeasible. With this in mind, low order polynomial-time heuristics or adaptive algorithms are clearly more suitable. A form of multiple heterogeneous machine scheduling known as resource constrained scheduling, which can be used to schedule jobs on clusters, has been approximated within factors of 2 for makespan and 14.85 for sum of completion times of the optimal solution using a heuristic approach [RE2004]. Although it appears ever more likely that there exists no polynomial-time procedure which will always deliver optimal schedules, the development of better Grid scheduling algorithms can only result in higher throughput of Grid applications, increased resource utilisation, reduced job completion times and

increased efficiency in the use of network bandwidth ensuring the Grid infrastructure is an effective and a realistic paradigm for solving large scale computing problems.

2.4.1 Grid Scheduler Architecture

Grid schedulers are processes that will: (1) discover available resources for a job, (2) select the appropriate system(s), and (3) submit the job [GD2005]. They are complex modular applications consisting of 4 main components: an application model, a resource model, a prediction model and a scheduling policy [ZH2004]. The application model describes the structure and characteristics of the applications submitted to the scheduler. The resource model characterises the nature of the resources available to the scheduler. The prediction model estimates the performance of the applications on the resources; it is usually based on predicting the behaviour of a specific application on a specific machine [ZH2004]. The scheduling policy is an algorithm which decides how to map the jobs to the resources. A common Grid scheduler architecture is given in figure 2.3. This architecture is intended to give a high-level view of the components of a Grid scheduling system; while not all existing Grid scheduling systems have corresponding components, every component seems necessary for any comprehensive Grid scheduling systems [ZH2004].

The various distributed Grid Scheduler components communicate via a WAN such as the Internet or using dedicated private links [ZH2004]. At the base of the architecture are the resources, located at the various distributed sites. Due to the vast number of different types of resource with differing management mechanisms (schedulers, queuing systems, reservation systems, and control interfaces) [GL2005], a Local Resource Manager (LRM) is needed to act as a bridge in making remote requests on applications. LRMs are middleware running at each resource site, their function is to start, monitor, pause and terminate the jobs remotely passed to it, and they are also responsible for updating the information service with information about the current availability and capabilities of the resources that it manages [ZH2004]. The Grid Resource Allocation Management (GRAM) [GL2005] is the Globus implementation of a LRM, using a set of WSDL/OGSI client interfaces it allows job requests written in its Resource Specification Language (RSL) to be executed on a remote resource.

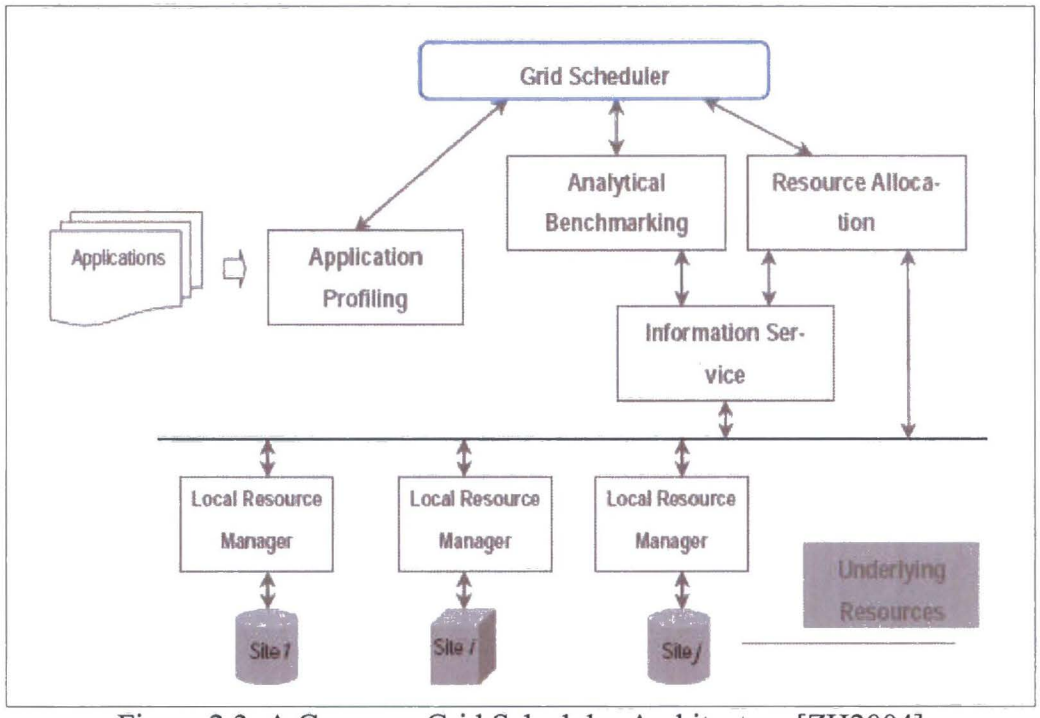

Figure 2.3: A Common Grid Scheduler Architecture [ZH2004]

The Information Service (IS) component is a directory based service which maintains information regarding the current and predicted state of resources. It generally stores static data such as the CPU capacity and architecture, memory size, network bandwidth, and the dynamic properties data such as resource availability, end-to-end TCP/IP performance (bandwidth and latency), CPU percentage load and unused non-paged memory [WO1999]. Two mature IS implementations are the Globus toolkit's Metacomputing Directory Service (MDS) [GL2005] and the Network Weather Service [WO1999] NTS used in the AppLeS [BE1996] scheduling methodology. An alternative approach to the implementation of the IS component is explored in [JI2003], where Grid economic mechanisms provide the basis for a virtual Grid market place where resource brokers and service providers interact to arrange execution of tasks based on QoS needs such as time and cost of completion.

User applications are fed into the Grid scheduling system through the application profiling interface; this is done to build the application description necessary for accurate performance prediction [ZH2004]. Application descriptions are built in two ways. One way is to force the user to describe their application using a job description language such Condor's ClassAds [LI1988]. The other way is to extract

the applications characteristics at runtime, such as the parallelism type and inter-job dependency [ZH2004] as yet this method has not been applied directly to a Grid resource broker [AF2005].

Capturing information regarding a resource's characteristics is only one aspect of the problem of capturing performance data; it is not clear how the output of the IS (the amount of CPU power and memory of a resource for example) relates to its potential computing power [OP2003]. The Analytical Benchmarking (AB) component is therefore required to sit between the IS and the Grid Scheduler, providing predictions on the performance of a given job on a specific resource, based on the IS readings [ZH2004]. The accuracy of the information it provides is critical to the performance of the scheduling algorithm which will base its schedule on it. Accurately predicting the performance of Grid resources is extremely difficult as it is misguided to believe the data produced from the IS are accurate or will remain unchanged during execution [OP2003]. There are a number of different implementations for this component, generally these estimate job execution time based on a combination of previously observed execution times and heuristic evaluations. Such systems include PACE [NU1999] which has shown to be usually overestimate the completion time of task randomly by 0 and 20% [HE2004] and the AppLeS prediction approach which has roughly an 11% relative error [XI2003]. Perhaps the most accurate of these is the Grid Harvest Service (GHS) [SU2003] based on long-term performance modelling presented in [GO2002], its relative error has been shown to be less than 10% [XI2003]. Experiments presented in [GO2002] on the performance of non-dedicated computational resources have shown that machine utilisation rates have a dominant effect on parallel completion time. Interestingly when utilisation is high performance becomes unpredictable, but when utilisation is relatively low, it has very little effect on the completion time of parallel applications [GO2002]. With this in mind, scheduling algorithm designers must be aware that the information they receive from any AB system is potentially flawed and that if system utilisation is increased during task execution, migrating a job already running may result in decreased completion time.

The Grid Scheduler (GS) is the core component in the architecture [ZH2004], it is where the scheduling algorithm is implemented. The job of the GS is to map the jobs

and communications of the applications onto feasible resources and networks [ZH2004]. The algorithm must take into account QoS arrangements and looks to optimise the mapping in relation to a specific performance model.

The job of implementing the finished schedule onto the resources is that of the Resource Allocation (RA) component. This task may involve data staging and binary code transferring before the job starts execution on the computational resource [ZH2004]. Many Grid applications require access to large data sets, meaning this pre-processing stage may take a considerable length of time; this time span should be taken into consideration when scheduling.

2.4.2 Grid Scheduling Strategies

There are numerous methods and techniques used to schedule Grid applications. Each looks to deal with a particular aspect of the problem to increase the effectiveness of Grid scheduling. [YU2003] and [ZH2004] both present lengthy surveys of these techniques. As yet, there is no clear choice of method which will be the best for Grid scheduling [OP2003]. The following section looks a number of different strategies and their place within Grid scheduling.

2.4.2.1 Scheduling Organisation

Grid scheduling systems are generally considered to be organised in one of three ways, centralised, decentralised or hierarchal.

In centralised scheduling, as shown in figure 2.4, there is one scheduler with knowledge of all jobs and resources. The advantage of centralised schedulers is that due to this global system knowledge, they are able to make effective scheduling decisions. It is typical that a single Grid will incorporate a centralised scheduling system to maximise resource usage, examples of such systems are Legion and Condor. The problem with the centralised approach is scalability and fault-tolerance. Scalability becomes an issue because the scheduler will no doubt use a polynomial time scheduling procedure, making it unable to search the entire search space to find

optimal solutions, as the number of jobs increases, the number of possible schedules gets exponentially larger, yet the scheduler will not be able to search any more possibilities in the set time it has to make a decision, therefore performance is likely to deteriorate. Fault-tolerance is an issue as if the scheduler becomes unavailable due to network failure, system availability and performance will be dramatically affected [ZH2004].

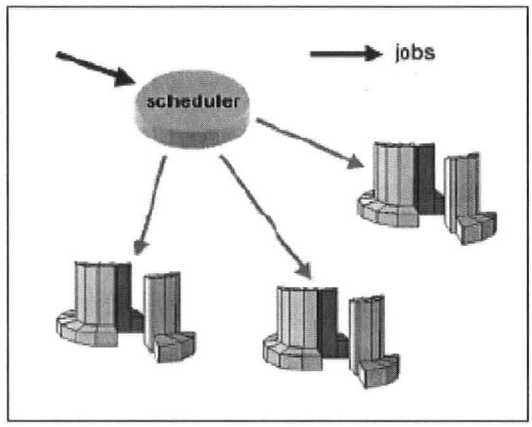

Figure 2.4: Centralised Scheduling [ZH2004]

The decentralised approach such as the one used in Condor/G, is illustrated in figure 2.5. In a decentralised organisation, each site is responsible for scheduling and for accepting local job requests. In this organisation, there is peer-to-peer based communication between sites, which allow for tasks to be executed on non-local resources. The site scheduler, known as a scheduling agent, needs to be able to handle remote requests and make decisions as to whether to decline or accept them. The decentralised approach is highly scalable since scheduling is partitioned into a number of different sections and as each site can operate independently with different scheduling policies, site-autonomy can be achieved easily each can be specialized for the site owner's needs [ZH2004]. Decentralising the scheduling also offers a higher degree of fault-tolerance as the failure of an individual scheduler will not render the entire Grid unusable. The disadvantage of the approach is that due to the lack of global system information, the schedulers generally make poor scheduling decisions. This is because there is no way for any scheduler to know exactly what each of the other schedulers are doing. Experiments are shown that in order to be effective, a decentralised scheduling approach should employ some form of coordination between

the agents produced through inter-agent message passing [BL2004]. Tests given in [WA2004] even indicated that by integrating a number of job migration techniques, implemented using inter-schedule message passing could make a decentralised approach comparable in performance to simple, centralised, non-scalable approaches. Such message passing is however a potentially large overhead since on a large-scale Grid environment where large number of agents each sending each other information could use large quantities of network bandwidth and ultimately lead to performance degradation.

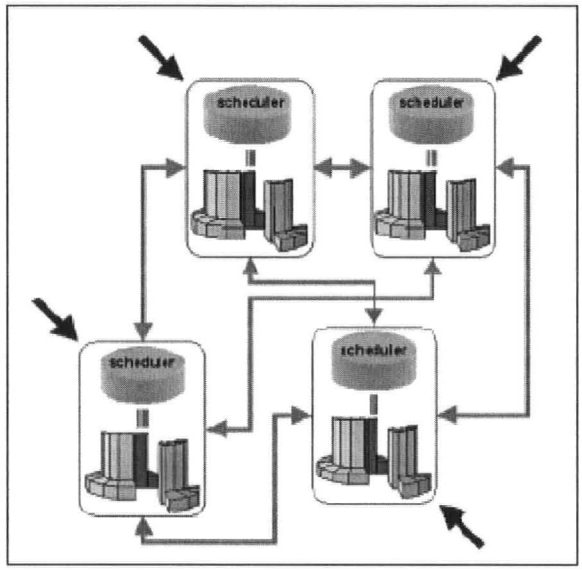

Figure 2.5: Decentralised Scheduling [ZH2004]

Other forms of decentralised scheduling exist, where as opposed to site based agents, each user, or application has its own scheduling agent. Examples of such systems are the Nimrod-G and AppLeS. With such systems, inter-scheduler communication is difficult due to the fact that applications and users, unlike sites are not persistent Grid entities. The way on which such systems hope to provide efficient schedules is through dynamic resource monitoring and rescheduling.

Hierarchical scheduling, depicted in figure 2.6, looks to provide a middle ground between centralised and decentralised approaches, incorporating a global scheduler, through which all jobs are passed, but reducing the scheduling problem by breaking the process into sizeable sections, such that the overhead of inter-scheduler interaction

is reduced. The scheme is based on the use of various scheduling process operating at various levels and controlling different sets of resources. Different algorithms are typically used at each scheduler, the selection of these will depend on the environment it dictates and the level at which it is operating. An example of a hierarchical local Grid scheduler, TITAN, is presented in [SP2003]. The hierarchical approach is effective because it reduces the effects of scheduler failure, inter-scheduler communication is minimised and the meta-scheduler, is capable of distributed jobs effectively as it has knowledge of all applications. The drawback of the approach, as with centralised scheduling is that it may be difficult to produce an effective scheduler whilst keeping site autonomy, as it may be preferable that some schedulers control multiple sites [ZH2004].

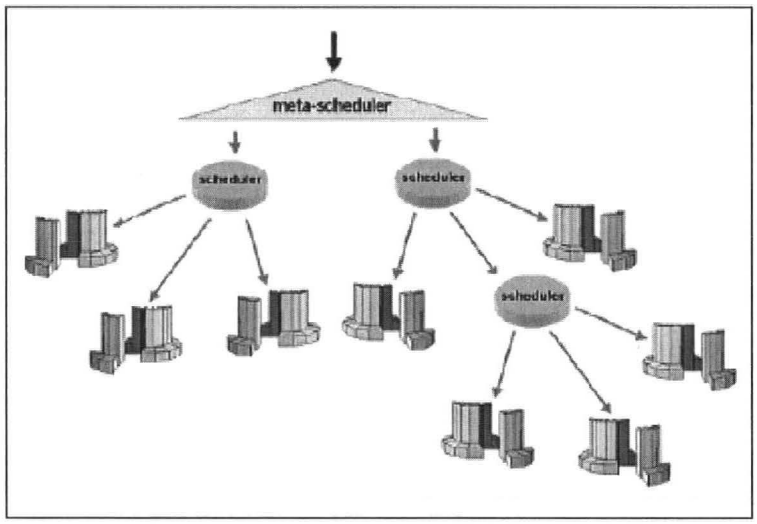

Figure 2.6: Hierarchical Scheduling [ZH2004]

2.4.2.2 Advance Resource Reservation

Advance Reservation (AR) is the process of requesting resources for use at a specific time in the future [BU2003]. The resources which can be reserved or requested are CPU processing time, memory, disk space and network bandwidth [BU2003]. Incorporating AR into a scheduler can ensure QoS agreements are met as jobs are guaranteed (baring any unforeseeable errors) to have access to adequate resources. AR is also useful to consumers who may have deadlines to meet and therefore can not leave the availability of such resources to chance. AR works much like a calendar

system used to reserve conference rooms for meetings [ZH2004]. A reservation maybe in one of several states during its lifetime, these are shown in figure 2.7 below.

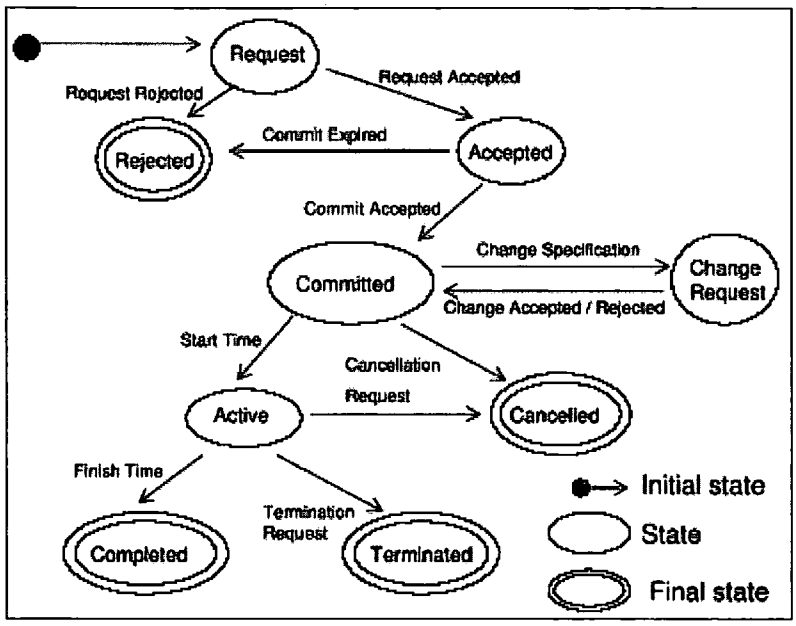

Figure 2.7: The state transition diagram for advance reservation [BU2003]

When the reservation becomes active the jobs already running on the reserved resources must suspended or removed, this is likely to have a negative effect on its service agreements and so alterations to the scheduling procedure to minimise such an effect may need to be incorporated. The difficultly for the consumer when using AR is in fulfilling the requirement that they have the application ready for execution at the start time, to avoid wasting money on reserving resources. Also although there are some systems which support AR such the Globus Architecture for Reservation and Allocation (GARA) [FO1999], most currently deployed local resource management systems do not [ZH2004].

2.4.2.3 Mapping Algorithms

Grid scheduling algorithms which produce a mapping of jobs to resources are still in their infancy. Most attempts make assumptions such as such as uniform job execution times, zero inter-job communication, contention-free communication, and full connectivity of parallel processors which may not hold in Grid environments

[YU2003] allowing for simpler, elegant solutions. The static scheduling model developed in [BR2001], as a means by which to compare the performance of algorithm, for example considers Grid scheduling algorithm to have input as a static matrix of estimated execution times, (*ET*, where the element $ET_{i,j}$ represents the completion time of job *i* on machine *j*), and look to create an effective mapping of jobs to machines based solely upon this. The values within the matrix can also be assumed to include overheads such as the time required to move the executable and data required, and if a job can not be processed on a machine the entry will be set to infinity [RI2003]. With this approach, Grid scheduling only differs from cluster scheduling in that each job could potentially perform differently on each machine, whereas on clusters; each machine is considered to perform equally on any given job.

Dynamic scheduling of independent task on Grids is illustrated in [MA1999], in which heuristic mapping procedures are considered to have two modes of operation; batch and on-line. On-line algorithms distribute jobs one at a time as they arrive at the scheduler and are usually simple to implement. Batch mode heuristics are those which delay the decision of scheduling tasks for a period allowing it to make possibly better schedules as it will have knowledge of the execution times of a larger number of tasks [XI2003]. The difference between the dynamic model and the static one is that the tasks are supplied to the algorithm over time, however batch mode heuristics are essentially the static scheduling heuristics used after a timeout (either a fixed time interval or after some fixed number of jobs have arrived) [RI2003].

Opportunistic Load Balancing (OBL): is a simple on-line heuristic which schedules the presented task on the machine which is predicted to be available next. It does not consider the execution time of the given task on the machine, but looks to maximise machine utilisation.

Minimum Execution Time (MET): this heuristic was originally presented in [MA1999] and is an on-line heuristic which maps task to the machine predicted to complete them quickest. Unlike OLB, it makes schedule based on the predicted execution times, yet, ignores the information on the current load of the machine.

Minimum Completion Time (MCT): is an on-line heuristic which maps the tasks according to the machines that are predicted to complete them first. MCT considers both the effect resource load and the execution time of the job on the machine in making its scheduling decisions. It usually gives dramatically smaller makespans than MET [PH2003].

Min-Min/Max-Min: are two heuristic mapping procedures can be used in both batch and on-line modes. However in on-line mode they are functionally the same as the MCT heuristic. Min-Min and Max-Min are also considered together with Sufferage [MA1999] and XSufferage (a site level equivalent) in [CA2000]. These four algorithms stem from the same base algorithm the General Adaptive Scheduling Algorithm presented in figure 2.8. The algorithms differ only in their definitions of the function f and the definition of "Best".

```
while there are tasks to schedule
  foreach task i to schedule
    foreach host j
      compute CT_{i,j} = CT(task i, host j)
    end foreach
    compute metric_i = f(CT_{i,1}, CT_{i,2}, ...)
  end foreach
  choose "best" metric_{i'}
  compute minimum CT_{i',j'}
  schedule task i' on host j'
end while
```

Figure 2.8: Pseudo code for the General Adaptive Scheduling Algorithm [OB2000]

Max-Min defines f to be the minimum $CT_{i,j}$ and "Best" to be the maximum. The effect of this is a procedure which finds the set of machines which will complete each job first and then schedules the one which will take the longest. The intuition is that by processing the larger jobs as quickly as possible, you leave the smaller jobs use up the slower machines giving a lower makespan. Interestingly [BR2001] has shown that in a static model Max-Min is almost always outperformed by MCT, which could be considered effectively the same algorithm, but defines "Best" to be a random

function (based on the assumption that jobs are supplied to the algorithm in a random order).

Min-Min defines f to be the minimum of all the $CT_{i,j}$ and "Best" defined as the minimum. Effectively it takes the smallest completion time possible from all hosts and tasks. In both [CA2000] and [BR2001], Min-Min has been shown to outperform both Max-Min and MCT.

Suferage has shown to be comparable if not better in performance to Min-Min [MA1999] [RI2003]. The Sufferage heuristic defines f as the difference between the minimum $CT_{i,j}$ and the second minimum $CT_{i,j}$, "Best" is defined as the maximum. The rationale behind Sufferage is that a host should be assigned to the task that would 'suffer' the most if not assigned to that host [YU2003]. It is interesting to note that when the estimated execution time of a task is only related to the number of instructions they need (resulting in a consistent ET matrix, this algorithm will be functionally equivalent to Max-Min).

Genetic Algorithm (GA): is a batch mode evolutionary technique which has been applied to problems in many fields. GAs are based on our understanding of how evolution is performed through genes. In the animal world, genetic information is inherited and changed from one generation to the next, resulting in better more adapted creatures. GAs use the same idea to produce better solutions to large scale search problems from initially poorer ones. The general structure of a GA is given in figure 2.9.

```
t := 0
initialise population(t)
evaluate population(t)
while not termination-condition do
  t := t + 1
  select population(t) from population(t − 1)
  alter population(t)
  evaluate population(t)
end while
```

Figure 2.9: Structure of a Genetic Algorithm [MA2003]

As the above structure shows the problems involves a number of different aspects, a population which must be initialised, a termination condition, a selection process, an alteration process and the ability to evaluate a population. The population consists of a number of strings which are usually binary; these are referred to as chromosomes. Each chromosome corresponds to a complete solution to the problem. In the case of the GA discussed in [BR2001] (which is adapted from [WA1997]), each the chromosomes is a binary matrix representing a mapping for every job to a specific machine. The initial population is usually random generated although some GA based schedulers use a greedy procedure to create a reasonably good starting population such as in [HE2004], and [BR2001] where the min-min solution was used in the initial population. The selection process which determines which chromosomes will be used to create the next generation is usually based on the fitness value of the chromosome in relation to the rest of the population. This process is analogous with the natural process of survival of the fittest. The fitness value is derived from applying the evaluation function to the chromosome. The evaluation function itself simply says how good the solution represented by the chromosome is, with [BR2001] for example, the makespan is the obvious evaluation function, as it is this that the algorithm is looking to minimise. A common technique used in [BR2001] is to use a weighted roulette selection system in which a virtual roulette wheel is spun a selected number of times (relating to the chosen population size) and each chromosome has a chance of being selected which is proportional its fitness against the fitness of the total population. A simpler method also used is to just select the top set of chromosome where the size of the set is the number of chromosomes you want to use in the generation of the next population. Elitism, which is when the highest valued chromosome is always included unchanged in next generation is another technique used in [BR2001], it ensures the best result found will be the one output at the end. The alteration process consists of two separate processes; crossover and mutation. Crossover is how genetic material is inherited, it involves taking sections from both parent's codes to produce a new code. In a GA the crossover operation selects a random pair of chromosomes and chooses a random point or number of points where the genetic code for one will be interleaved with the other. In [BR2001] every chromosome is considered for crossover with a probability of 60%. Mutation is performed after crossover and is a random process which alters the new code by very

occasionally changing the value of individual pieces of code. It is this process which is responsible for much of the inherent variation in a species and has made evolution possible. In the case of the GA discussed in [BR2001], each chromosome has a 40% chance of being mutated, on mutation a randomly selected task will be assigned to a different machine. There are a number of possible termination conditions for a genetic algorithm, such as a chromosome with an acceptably high fitness value has been identified, there is little change from one generation to the next, all the chromosomes are similar, there is no change in the best chromosome for a number of iterations or a maximum number of iterations has been reached. Genetic algorithms have a tendency to get stuck in local minima and often have quick convergence so numerous restarts are usually used.

[BR2001] found that the GA was the best mapping algorithm in terms of the makespan it presented. However this is hardly a surprising result given that the Min-Min solution (always found to give the 2nd best makespan) was included in the initial population and that elitism was used thus making it impossible for the GA to give a poorer mapping than Min-Min. In general the Min-Min mapping was 5-10% worse than that of the GA [BR2001].

There are many other algorithms and techniques which can be applied be the problem if static independent job scheduling on Grids such as Tabu, A*, Simulated Annealing, Ant Colony Optimisation (ACO). Searching has long been a fundamental part of computer science research and many of the techniques available are simply adaptations of algorithms developed to solve other NP-hard problems. There are a number of techniques which can be applied to give better solutions when using these heuristic methods such as:

- Executing a number of different heuristic procedures and picking the most effective such as in the Duplex algorithm [BR2001] which performs Max-Min and Min-Min and picks the best solution.
- Combining effective parts of other algorithms such as in the hybrid GA/ACO algorithm presented in [LE2004] and the Genetic Simulated Annealing heuristic presented in [BR2001].

- Use a hierarchy of procedures such as in the scheduling architecture described [HE2004] where a bin packing style algorithm is used to select which job are distributed to lower level GAs.
- As there is a tendency for heuristic search algorithms to get struck in local optima, the simple process of restarting the algorithm with different random input will result in better results.
- Solutions for many NP-hard problems such as bin packing and cutting stock [LE2003], developed by heuristic methods can often be improved by local searches. Applying an effective local search strategy in static Grid scheduling is demonstrated in [RI2003]. The effect of applying the procedure to the output of the Min-Min and Sufferage heuristics was an algorithm which outperformed the GA considered in [BR2001] and took significantly less time to compute.

As with all searching problems there are trade-offs between time and space (i.e. do you remember all the states you have visited to save re-evaluating them) and more importantly between time and output (i.e. an algorithm which runs for a longer time will be able to search more possibilities and hence potentially give a better result). The best search procedures strike an effective balance between searching for different areas of search space which may contain possibly global minima (branch searching) and searching around smaller areas of search spaces to optimise already encountered solutions (local searching). In the case of the GA, crossover can be considered to be the branch searching component of the search where completely different solutions will arise, and mutation considered the local search component where slight random changes to solutions may result in improving them. In Simulated Annealing, the start of the search where many different solutions are accepted looks to find different areas of search space which appear to have potential, a branch search, and then as the chance of accepting worse solutions lowers, this becomes more of a local search looking to optimise previous solutions.

It is clear that mapping heuristics provide a methods capable of searching exponentially large search spaces and that they can be applied to simplistic Grid scheduling models with good results. However due to the assumptions made by these

approaches, when designing mapping algorithms based on them, careful modifications are necessary according to the characteristics of both Grid applications and Grid resources [YU2003].

2.4.2.4 Economic Scheduling Methods

It has been argued in [BU2001] that a computation economy is required in order to create a real world scalable Grid, because it provides a mechanism for regulating the Grid resources demand and supply, offers an incentive for resource owners to be part of the Grid and encourages users to optimally utilise resources and balance timeframe and access costs. Taking this into account, economic scheduling methods are proposed in [AB2002] and look to apply economic models to Grid scheduling. Economic based scheduling methods have already been applied to cluster scheduling with some success in [SH2004]. The basic components in a market are producer, consumer, and commodities, analogous to resource owner, resource users (the applications), and various computing resources in Grid computing environment [LI2003], so applying an economic model to the scenario is simple.

Economic models have been applied to Grid scheduling for several reasons including:

- Markets and Grids have similar natures; they are both decentralised, dynamic and deal with competitive resources. [LI2003]
- Grids are service orientated and money would provide an incentive for service providers to give up the time and memory of their resources.
- Service users do not want to pay an excessive price for the use of resources; the Grid economy approach means that service providers will be encouraged not to overcharge as they are operating within a competitive market.
- Using price as the single measurement, as in a market, provides a simplified way to express larger parameter spaces in resource scheduling problems [LI2003], and would greatly reduce the communication costs endured by using a decentralised system model.
- Certain micro-economy theories (such as the General Equilibrium Theory) can be applied to solve some optimization problems in Grid scheduling based on

certain restrictions and assumptions [LI2003]. The outcome of the theory is that if the assumptions hold and a state of equilibrium reached, (where supply equals demand for resources), an optimal solution is found.

To put economic scheduling to the test, Nimrod-G, a resource broker which uses the notion of a computational economy as a basis for managing resources and scheduling tasks on large-scale Grids was developed. Its deadline and budget constrained cost-time optimisation algorithm is presented in [BU2005], and has been proven effective.

2.4.2.5 Application Level Scheduling

Applications Level scheduling is about imparting application specific knowledge into the scheduling process in order to improve its performance. Application-level scheduling helps applications adaptively adjust their schedule in two ways:

- Given run-time resource availability information, an application can dynamically decompose its tasks and generate a schedule by itself to get better overall execution performance.
- For iterative or loosely-coupled parallel applications, given previous job execution performance information, an application is able to adjust the schedule for current jobs [LI2003].

Two successful and popular application-level scheduler engineering strategies are to (1) embed scheduling logic into the application or (2) to embed application-specific information into the scheduler [DA2002]. The AppLeS project [BE1996] developed both of these strategies and was successful in designing a number of application specific scheduling mechanisms in collaboration with projects in physics, computational mathematics, and biology [LI2003]. However with both strategies the process of producing such schedulers has proved time-consuming and it has proven difficult to adapt them to other applications or execution environments [DA2002]. To rectify these problems, another effort in the AppLeS project framework is the development of scheduling templates [BU2002], which are targeted towards specific types of application. Currently there are two such templates, the Parameter Sweep

template (APST) [OB2002] and the Master/Worker template (AMWAT) [SH2001]. These templates are reusable agent components that allow programmers to develop applications of the template type without the need to consider scheduling, communications, and fault tolerance.

The difficultly in providing adaptive general purpose application level scheduling in the AppLeS project stem from the coupling of application-specific components and scheduling components [DA2002]. This has been addressed in the more ambitious Grid Application Development (GrADS) project [DA2002] which looks to provide end-to-end Grid application preparation and execution environment [LI2003]. The design of the GrADS decoupled scheduling approach is illustrated in figure 2.10 below.

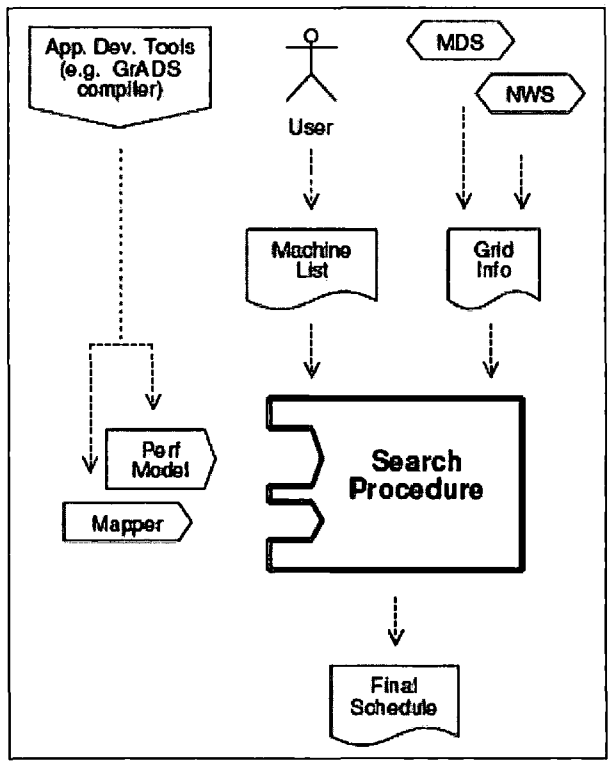

Figure 2.10: The GrADS Decoupled Scheduler Design [DA2002]

To use the GrADs system, the user submits a *Machine list* which contains the list of machines available to execute the application. The scheduler collects data both static and dynamic data through resource information providers such as NWS and MDS

about each machine on the list. The application itself is developed using the GrADS development tools and is supplied to the scheduler together with a *Mapper* and a *Performance model.* The *Mapper* provides the application specific details of how best to schedule it and the logical directives for mapping the required data and/or tasks to the physical resources. The *Performance Model* is an analytical metric for predicting how long the application will take to execute on a given set of resources and a given mapping. The search procedure itself looks to identify subsets within the machine list which have desirable characteristic as execution environments. The GrADS scheduling model also incorporates a number of rescheduling mechanisms to improve execution times which are initiated when load changes on the resources cause a violation of the application's performance contracts. Details of the original algorithm are presented in [DA2002] and details of the new approaches are detailed in [CO2004].

It should be noted that the GrADS project thus far has not yet succeeded in fully implementing the design presented in figure 2.10. The GRaDS compiler is not fully operational and so it is down to the user to provide both the *Performance Model* and *Mapper* for their application which can be a costly task.

Application level scheduling is an adaptive approach to the Grid scheduling problem which has proven successful. It has proven difficult however to create a single scheduler capable of dealing with all applications and execution environments. The AppLeS templates and GrADS project have investigated ways around this lack of scheduler applicability, but, are still some way from making the middleware required to seamlessly execute Grid applications in an effective manner.

2.4.2.6 Job Migration/ Rescheduling

Job migration is the process of rescheduling possibly already in progress jobs, so as to ensure their completion (in the case where the currently allocated resource has crashed, lost power or network access) or simply to speed up execution (in the case of increased load on the executing machine). Job migration has been shown to be an essential mechanism for lowering completion times in non-dedicated computing environments [GO2002]; it provides both flexibility and fault-tolerance to Grid

computing. The Condor project showed it was possible to migrate jobs during execution so that progress was not lost, this was accomplished through the use of continual check pointing. Rescheduling is either done periodically as in the Nimrod-G resource broker or on contact violation as in the AppLeS and GrADS projects.

The GrADS project has looked at two different forms of possible rescheduling, by stop and restart and by processor swapping, both of which have proven effective [CO2004]. Stop and restart is the traditional rescheduling, as used in Condor, where an application is suspended and migrated to another more effective execution environment. This rescheduling strategy is highly flexible, as it can potentially be used to switch applications from any environment to any other compatible one, but, has been shown to be expensive as the process usually involves the transfer of the typically large application files to the new environment. The other disadvantage of this type of rescheduling is that significant application modification maybe required for specialized restart code [CO2004]. To combat these problems, processor swapping rescheduling, originally described in [SI2004], was developed. In processor swapping rescheduling, there are more computation resources available than application level schedulers are aware of. These are split into two sets, inactive and active, where the former are not visible to the scheduler, whilst the later are. When the communication calls to the resources are made, they are hijacked, transforming them to calls to a subset of the entire set of resources [CO2004]. The resources are continually monitored and faster inactive machines are swapping for slower ones in the active set. This approach requires no additional programming, it allows applications to move between different sets of resources, and has been shown to be a useful fix to some performance problems [CO2004]. The downside of the approach is that is in less flexible as the resource set is limited to the original one and data allocation can not be modified.

Job migration does not always mean rescheduling active jobs, it can also be used in a decentralised scheduling system when a local scheduler is presented with an application with more resource requirements than its resource pool can handle, so some of its jobs are migrated to other schedulers. Job migration is the used as the basis for three distributed scheduling algorithms presented in [WA2004], sender-initiated, receiver-initiated and symmetrically-initiated (a combination of the other

two), according to which party requests the migration. The performance of these algorithms has been shown to be comparable to a simple centralised scheduling approach [WA2004].

2.4.2.7 QoS Guided Methods

In [XI2003], it is argued that the two concepts most key to a Grid scheduling model is the consideration of how to handle non-dedicated resources and how to deal with quality of service. The former is considered in most resource management systems through the incorporation of an information service such as NWS, but the later is yet to be fully considered by such systems. Nimrod-G is one of the few that considers both, basing its scheduling decision on the soft QoS constraints of deadline and budget, yet the typical approach to dealing with hard QoS, is simply not to schedule a task with high QoS on a machine which does not offer it. In [XI2003], it is shown that by considering hard QoS as the major scheduling factor, schedules can be significantly improved. They adapt the Min-Min mapping algorithm to schedule high QoS jobs first followed by low QoS jobs. Although the study in [XI2003] is rather limited as it only considers one-dimensional QoS with two distinct levels, it provides a basis for the theory that hard QoS requirements should be central to the scheduling algorithm.

This theory is reinforced by [DO2002] and further by [GOL2002], where more sophisticated QoS model have also been studied in which allows for the presence of any number of QoS dimensions, which can be either hard, soft or unimportant to that task. This is accomplished by associating each QoS dimension is a normalised utility function which defines the benefit that will be perceived by a user with respect to the user's chosen level of QoS values in that QoS dimension. Figure 2.11 shows the utility functions for the timeliness QoS dimension in the three possible cases.

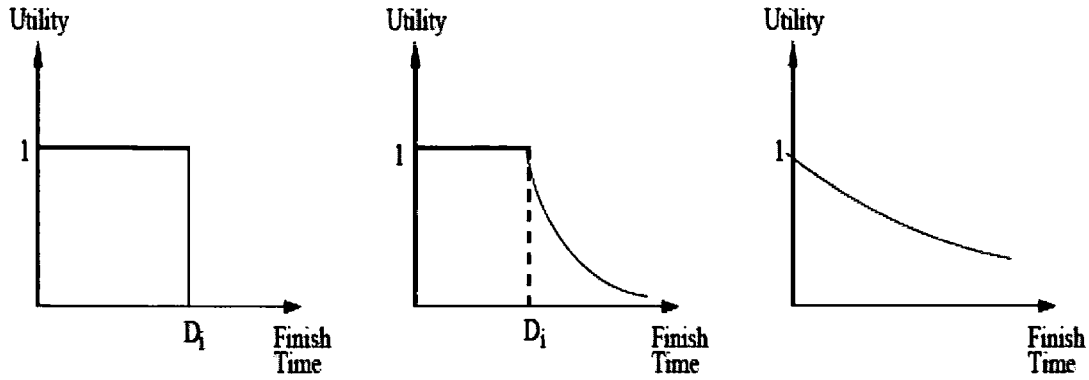

Figure 2.11: Utility functions of the cases where a task is associated with a hard deadline, soft deadline, or no deadline, respectively, for the timeliness dimension where D_i is the deadline of the task. [DO2002]

Scheduling can then be performed on a matrix of values which are not estimated times to compute but rather the benefit gained by placing the given job on the given machine. This is calculated as a weighted sum of the QoS values for that task and machine put through the respective utility functions. Equation 2.1 shows how each value within the matrix is calculated.

$$U_i(q_i) = \sum_{j=1}^{d_i} w_i^j U_i^j(q_i^j)$$

Equation 2.1: The utility function for the task T_i. $0 \leq w_i^j \leq 1$ represents the weight given to that QoS dimension by the owner of task T_i (how important that QoS dimension is to them), q_i^j is the point of T_i within the QoS dimension j (the level of QoS the task owner wants on that dimension) and U_i^j is the utility function for that that QoS dimension (which will depend on the type (e,g hard or soft) the user wants hard or soft QoS on that dimension) [DO2002]

Results in [GOL2002] suggest that the best algorithms for scheduling using such a method are those which have been custom made to suit the idea of QoS scheduling rather than a generic matching procedure such as those given in section 2.4.2.3.

2.5 Grid Simulation

Experiments on real grid environments are extremely difficult, to carry out. Nevertheless experimentation on these environments is essential to effectively evaluate their performance. Any experiment should be repeatable and controlled for any valid scientific conclusions to be taken from the results. The difficulty with Grid environments is that they are large, dynamic, distributed and heterogeneous [SU2004] and real Grid test beds are difficult to setup, change and are expensive. A local real grid test bed will not be totally representative of a real Grid environment where the computers are managed by different organisations with different usage policies [SU2004] and where data transfers may take considerably longer. Experimenting using simulated Grid environment is therefore the best way to produce scientifically valid results.

Simulation is the technique of a building a model of a real or proposed system so that the behaviour of the system under specific conditions may be studied. One of the key powers of simulation is the ability to model the behaviour of a system as time progresses [BA1996]. With Grid simulation you are modelling the grid resources (machines, data vaults, and specialist equipment), the network under which they communicate, the users using the system, the tasks which they are submitting and the scheduling system which describes how the jobs are distributed. A good simulator allows researchers to explore more alternatives and give accurate, statistically valid results [PU2003]. Applying simulation to model problems is a non-trivial task. Good simulation models are difficult to design and maintain and their development is sometimes comparable to the costs of building the actual systems [SU2004]. In addition, the users of the simulation tool may not be experts in simulation, thus may have problems creating simulation models successfully, which could lead to them having unfounded belief in their systems. Therefore, there is a need to have effective simulation tools that enable easy and fast creation of accurate simulation models [SU2004].

The process of constructing and using a simulation program consists of the following stages:

1. Starting with a real system and understanding its characteristics
2. Building a model from the real system in which aspects relevant to simulation are retained and irrelevant aspects are discarded
3. Constructing a simulation of the model that can be executed on a computer
4. Analysing simulation outputs to understand and predict the behaviour of the real system [MI1986].

In the context of Grid simulation, stage one refers to understanding the nature of Grid environments including, how the resources and Grid components operate and communicate, how Grid users make use of the system, the possible structures of the jobs they submit, the likelihood of resource failure, and the effects of outside load on the resources and networks.

Stage two is about removing the irrelevant details of the real Grid environment which are of no consequence to the area the researcher is considering. This is a key stage in simulating such environments as they are extremely complex and therefore including a complete model would result in inefficient run-times, excess memory usage and make the simulator's use overly complex. It is at this stage that the appropriate level of abstraction is applied, allowing the base elements in the simulation to be described in terms of what they do, rather than how they do it. A commonly used model in Grid simulation is to consider machines as black boxes capable of processing jobs at a certain rate, relating to the number and speed of processors, the OS architecture and the job requirements type. A more realistic model would be to consider the individual hardware components of each system, yet such details would be difficult to ascertain and have such a minimal impact on the results of any scheduling experiments that such details are always omitted. Often Grid simulations are overly simplified to allow for simpler experiments, quicker execution times and to eliminate factors affecting the results. For example in most Grid simulations machines and networks are considered to be always available and 100% reliable, but, resource failure and limited availability are inevitably on real Grid environments and are guaranteed to affect the system's performance. Working with assumptions such as these will lead to the simulated model outperforming the real system, yet this is not necessarily a problem if such factors are considered to affect each experiment equally. Therefore the results output

from a scheduling experiment on a simulated environment may not give an accurate indication of the performance of the applied algorithms on a real Grid but should provide an accurate indication its performance relative to that of another algorithm simulated using the same model. The outcome of this is that considering the simulated performance of an algorithm is often meaningless without comparison to the results of at least one other.

The third stage of the simulation process; implementation of the model can be done using a variety of techniques. The implementation of a Grid simulation does not appear to differ greatly from implementing any other form of simulation; however this does not mean that the task is a simple one. There are numerous ways to represent the entities, the passing of simulated time and the output, and there are a number of tradeoffs which must be considered before selecting which to use, a discussion of these is presented in [BA1996]. The implementation language and structure of the simulation will also have a large effect on the simulation software's usefulness and execution speed, as there are often tradeoffs between the various options available so it unlikely that any single simulation software package will be suitable for all researchers and execution environments. For instance a parallel structured simulation [MI1986] will run faster when executed by more than one processor but will usually be outperformed by a sequential based simulation on a standard single processor PC due to the overhead of context switching. The tradeoffs in selecting implementation languages are between simulation execution speed and a combination of readability, reusability and operating system dependability. Simulation libraries written in C and C++ are typically much faster than their Java based equivalents [PU2003] but many consider Java code to be easier to read and understand and it has the advantage of being able to execute on numerous different OSs without alteration.

The final stage of using a simulation, like any scientific experiment, is the analysis of the results, in this case the simulation output, in order to understand the processes which have given rise to them. The difference between a normal scientific experiment and one conducted on a simulated environment is that the results on the former are taken directly from the object of study so give a direct indication of its behaviour, yet, those conducted on a simulated environment can only give an indication the behaviour of the real system, given the assumptions made when

constructing the model. Therefore to understand the simulation results and make valid conclusions form them, it must be clear how the simulator is generating the data; a simulator should not be delivered as a black-box on which to conduct experiments. With Grid simulation the output can be relating to a number of areas, depending on the interest of the researcher, for example the speed at which the implemented scheduling algorithm operates, the makespan of a number of tasks on a given set of resources, the effective computing power of a Grid under differing levels of local loads, the effect of resource reservation, the utilisation of a specific resource on a Grid or the effect of data replication across a number of Grid nodes. Due to the large number of possible experiments which can make use of Grid simulation, Grid simulation software should provide a means of easily configuring the output to suit the needs of the researcher.

2.5.1 Discrete Event Simulation

Discrete event simulation is a well understood, commonly used technique for modelling real world systems over time. The idea is to model the possible events which can occur then observe the effect they have upon the system over a simulated period of time. The alternative to considering using discrete events is to consider time as a continual variable which is controlled by a series of differential equations which are integrated during simulation [BA1996]. The continual simulation approach makes it difficult to construct complex simulations, especially for non-expert users as it is not always clear as to how to express the behaviour in such as formal form, although combining both events and differential equations in some simulations has proven to be a powerful tool [BA1996]. The events considered in a discrete event simulation system occur at specific points in time, alter the state of the system and can give rise to other events. Although simulators differ greatly in the way they operate and represent the simulation time, the events, the entities and the relationships between them, they all share a common structure which is presented in figure 2.12.

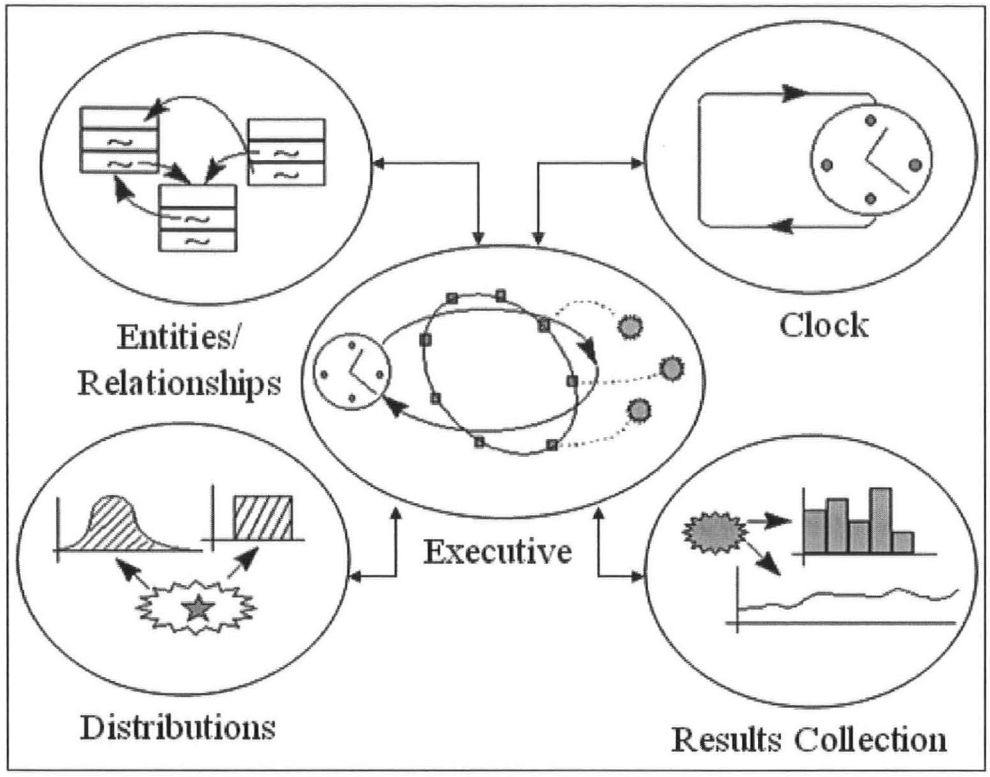

Figure 2.12: The structure of a simulation system [BA1996]

When simulating a system it is critical to correctly identify the resources and the relationships between them in order to obtain a successful model. The entities are the tangible elements found in the real world which make up the system being modelled e.g. in a manufacturing line simulation these maybe the machines or parts which pass through them [BA1996]. The concepts of temporary and permanent are usually associated with these entities, the permanent entities are those which remain throughout the simulation [BA1996]. Within the context of Grid simulation the networks and resources are usually considered permanent, the users and jobs they process, temporary. Selecting an appropriate level of abstraction for the entities plays a key role in the success of the simulator. With Grid simulation one could for example model each part of the computational resources in great detail, down to the individual transistors in each CPU, while this may provide accurate results, it would be difficult to construct and operate so slowly and require so much memory that it would be impractical to use. There is usually a trade-off between the processing time and memory usage performance of a simulator and the accuracy of the results it provides. The entities are linked by logical relationships [BA1996], e.g. a computational Grid resource entity will process a Grid job entity. The ability to

express these relationships provides the user with a powerful means of constructing complex models with non-trivial behaviour.

The central executive and the clock are essential components of a simulation system although they have simple behaviour and are usually easy to implement [BA1996]. The clock is responsible for keeping track of the simulation time and is usually accessible by all the entities in the simulation. The executive is responsible for time advance of the clock and executing the actions caused by the events it encounters. It is essential to providing the dynamic time based behaviour of the model [BA1996].

Discrete event simulations use different distributions to model when the events occur and the values of some entity properties. Some events may simply occur at exact intervals such as the earth completing a revolution of the sun but many are subject to random variations, such as when customers walk into a shop or the length of time between machine breakdowns. This stochastic behaviour is typical of the real world and simulators need to provide random number generators to model the distributions which govern such behaviour, such as the normal or exponential distributions [BA1996]. Within Grid simulation, random number generation is often used to model the variation in the number and size of the tasks submitted by the users, the time between users submitting tasks and the local load on the resources and network links.

The results collection component provides the output of the simulation. It is responsible for collecting data on the performance of the system during simulation and presenting it in an acceptable format. The output of this component can come in numerous forms. Some simulations tools such as SimJava [SI2002] can provide the user a with 2D graphical representation of the simulated system in action, allowing the user to visualise where potential problems could arise, yet, many tools simply provide raw tabulated data which can then be analysed and studied, often with the use of graphing software. The results collection components of some tools provide different output options allowing the user to focus on different aspects of the system's operation, with Grid simulation a scheduling algorithm designer may be interested in the run times of the algorithm under different loads, the mapping of the jobs to the machines it provided, the time taken to process each job or the utilisation of a particular set of machines over the course of the simulation.

2.5.2 Object Orientated Approach to Simulation

The power of the Object Orientated (OO) approach to programming is that code is modularised in a logical and intuitive manner making it possible to reuse and modify the existing code. Programmers can build new models from existing functionality with only knowledge of an objects' interface; its' inner-workings are abstracted out. OO languages have the power of inheritance; a language feature which allows new objects to be developed from extending ones. The advantage of using inheritance is that code can be reused without the need to adapt it for new classes. OO languages have made building discrete event simulation simpler [BA1996]. The entities of the simulation can be modelled as objects in the program and the relationships between the entities can be expressed through the use of fields (in the case of a "has a" relationship) and inheritance (the case of a "is a" relationship). The methods supplied by an object provide a means of manipulating it state.

The functionality provided by OO languages allow for the production of simulation libraries which provide base functionality which programmers can extend and manipulate to create their own complex models as quickly as possible. Providing simulation modelling functions through such libraries is often preferable to developing a simulation language as it does not require the development of a complex parser or compiler, it is often far more flexible as it is based upon a general purpose programming language, it is likely the user will have previous knowledge of the language the library is written in so the learning curve will be shallower and modern programming language have many tools such as powerful Integrated Development Environments (IDEs) which will aid the user in developing a simulation.

2.5.3 Grid Simulation Systems

There are numerous Grid simulators available. An extensive taxonomy covering the different possible design simulation strategies available and their associated implementations is presented in [SU2004].

GridSim [MU2002] is a highly flexible, Java based, open source simulation library for modelling Grids. GridSim was developed by the Grid computing and Distributed Systems (GRIDS) Laboratory at The University of Melbourne, Australia. It has been proven to be a useful simulation tool [BU2000] and supports the modelling of realistic Grid environments with heterogeneous resources and differing scheduling policies, multiple schedulers, users and application types. GridSim has recently been expanded to include a realistic network model [GO2004] and support advanced reservation of resources [BU2004]. GridSim is built on the general purpose discrete-event simulation package SimJava2 [SI2002]. The result of this is that all simulated resources and users run in their own separate threads with unique names, which all run in parallel. This restricts GridSim to modelling medium sized Grid environments; it is not scalable because simulation size is limited by the relatively small number of threads the Java Virtual Machine can handle [PH2003]. In addition to this, the thread management in Java creates a very high overhead which results in very slow run time [PU2003]. Another weakness of GridSim is the amount and complexity of the coding required to construct a simulation; the basic examples provided with the toolkit are several hundred lines long and all events representing communication between the entities are initiated by the user.

The OptorSim grid simulator was built and used to test the effects of several optimisation strategies and replica optimisation strategies on the UK Grid for Particle Physics [CA2003]. OptorSim allows numerous evaluation metrics to be taken from a simulation, which allows the algorithm designer to consider not just the performance of the job scheduling but the performance for all the Grid's resources. The simulator allows the user to specify a grid with computing or storage resources and link them via a network topology defined as a graph, with the resources or routers as nodes and dedicated network cables as the links, where the weight of the links is representative of the bandwidth of the connection. To speed up simulations, OptorSim does not consider an accurate network model, has no way of modelling local loads on any resource and considers job processing to be done in constant time; employing these abstractions allows for larger scale simulations. OptorSim is not publicly available at this time.

HyperSim is a simulation library implemented in C++. It is developed to be a general purpose, extensible, configurable and high-speed library [PU2003]. HyperSim is scalable and fast, and has been shown to be accurate, 1000 time faster than GridSim and 10 times faster than SimGrid [PU2003]. HyperSim follows the event graph model presented in [SC1983]. Using this approach allows for fast simulation but it requires the user to draw up the event graph model prior to designing the simulator [PU2003]. This could prove a difficult, error prone task and requires learning the technique as well as expert knowledge of the events and possible states of the simulator. The simulator uses the approach that you build the scheduling heuristic separately from the simulation. This has the advantage that implementing a scheduler simply requires implementing the given interface and you can easily reuse any previous simulation setup with different schedulers without the need to rewrite it [PU2003]. The simulator currently only models time shared machines and does not have inbuilt realistic network simulation support.

Bricks [AI2000] is a Java based discrete event simulator built from the ground up. It follows the client-server architecture to maximise modularity and has been applied to both scheduling heuristics and data migration procedures. It offers realistic load modelling which is based upon real world system traces. Bricks is designed around a centralised server acting as a scheduler; this is restrictive as it does not allow for easy modelling of a decentralised architecture with many competing schedulers. At this time Bricks is not publicly available.

MicroGrid [SO2000] is a Grid Emulator based on Globus developed at the University of California at San Diego (UCSD). It is designed to realistically emulate a grid of resources on a real resource, to increase the test bed grid size using limited resources [PU2003]. It is used to run real Globus Grid applications on a real, controllable environment [MU2002]. This means however that runtime is not reduced and that the need to develop a real Grid application on which to test a scheduling algorithm is a significant overhead. For these reasons it not a viable tool on which to test grid scheduling algorithms but can be used as a complementary tool for verifying simulation results with real applications due to the fine accuracy of the run times it produces [MU2002].

SimGrid [CA2001] developed at UCSD is a powerful C-based discrete event scheduling simulation library. It provides an accurate network model and has been used for modelling data-intensive applications. It is also capable of accurately modeling resources according to standard machine capability, machine load from real traces and constants and jobs according to their execution times. It uses user level threads to model resources so is restricted in speed by the thread-switching capability of the system, although is generally faster than Java based simulators [PU2003]. The modelling limitations of SimGrid are that it only allows for one scheduling entity and can only model time shared machines, meaning modelling real large scale Grids is impossible without significantly extending the tool. Real large-scale Grids generally contain many schedulers which are communicating, competing and scheduling on different sets of resources, and large resources with many processors are often space-shared so they too need to be simulated [MU2002].

GridNet [LA2002] is a realistic Data Grid simulator which provides a modular simulation framework. GridNet is written in C++ and is built on top of the network simulator ns [NS2005]. It uses the underlying ns structure of links, nodes and TCP protocols as a basis for implementing a number of Grid specific simulated application level services [LA2003]. GridNet was designed to test a number of dynamic data strategies on a relatively small hierarchical Data Grid, although the approach appears to give accurate results it is unlikely that the model will be scalable to large-scale Grid environments and adaptable to evaluate different resource allocation issues.

There are other Grid simulators including as ChicSim [RA2002], EDGSim [ED2005], both of which look at modelling data grids for large scale physics experiments.

2.5.4 Network Simulation

Traditionally there has been an intellectual disconnect between the networking and scheduling communities; whilst the former is considered with designing network protocols, the latter is concerned with the performance of an application on a network that is considered a black box. For this reason scheduling researchers tend to use simpler network models which fall down when trying to model large scale networks

[CA2005]. One of the key differences between Grid computing and traditional cluster computing is the nature of the networks over which the machines are connected. Whilst in a cluster environment, network latency is likely to be negligible and equal between each machine, this will not necessarily be the case in a Grid. Likewise the data transfer times within a cluster will be much lower, than in a highly distributed Grid. Cassanova, in [CA2005] looks at a problems introduced by the underlying network on Grid application scheduling, he argues that analytical models can not fully represent realistic network properties and that comprehensive network simulation is required to built and test effective Grid scheduling algorithms. The following factors are considered to be of significance:

- Network Latency: This is defined as the time it takes for a bit of data to physically travel down the network cables to the destination. This is affected by the length of the distance the bit will travel.
- Queuing Delay: Is defined as the time packets are queued for at each hop of their transfer, waiting to be sent to the next. This factor is dependant on the load as each hop.
- Computational delay: Although this is not an effect within of the network itself, it is a consequence of using wide area networks to distribute tasks. This overhead is the time for remote processes to be initialised which are required to run a Grid task on a remote resource. These processes includes authentication, process creation and resource acquisition, it has been shown to taken up to 25 seconds per task using the Globus toolkit [CA2005].
- Bandwidth Sharing: With possibly many processes each utilising a single link within a network and each link having a limited bandwidth (the amount of data which can be sent down the link in a fixed time), the processes must share the bandwidth. The maximum useable bandwidth for each process is related to the lowest bandwidth it receives on its transfer, i.e. where the smallest bottleneck for that process is.
- Packet Loss/Corruption: Real networks are not perfect; there is always a chance some packets will not be received due to misdirection or network failure. Also some packets maybe subject to corruption and may not be

received exactly as they were sent. These packets will need to be resent in order to recover the original message.

- Protocol: It is important to understand the operation of the transfer protocols being used to transmit the data, as this will affect the performance. For example TCP/IP, used on the Internet is a connection-orientated protocol, and as such requires a number of packets be sent in order to initiate the connection, also for each packet sent an acknowledgement is sent back, which will use network resources and obviously slows down communication.

Network simulation has a long history and has made many advances. Spurred by the Internets rapid growth, researches look for new algorithms and protocols to meet changing operational requirements, and to simulation as an effective means of providing performance analysis [BR2000]. In an effort to appreciate the current state-of-the-art in network simulation, the approaches of four network simulators are outlined:

Flowsim [AH1996] was the first hybrid simulator possessing the ability to speedup simulation performance by an order of magnitude or more by aggregating individual packets into larger groups, known as packet trains [AS2000]. The effect of going up a level of abstraction is that it the number of events the simulator had to process was greatly reduced. To accomplish this move Flowsim operates under assumptions as to the arrival and dispatch times of the packets from each node, since it considers packets within a train to be evenly distributed throughout it [AS2000]. Despite the fact the each of these assumptions inevitably introduces simulation error, Flowsim has proven to be remarkably accurate, with predicting FTP transfers within 90% of the base simulator (with no approximating assumptions), and packet loss to within 85% [AS2000].

NS [NS2005] is the most widely-used network simulator today [GA2002]. It is a discrete event simulator written in C++, developed for use in network research. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks [NS2005]. It typically uses a packet level simulation granularity which is both extensive and highly accurate, but, suffers from slow execution times and large memory requirements, making it

unsuitable when modelling large-scale networks with many thousands of nodes [GA2002]. It help improve simulation speed, NS incorporates the Flowsim model for allowing different abstraction levels, trading off performance with accuracy. NS employs a split programming model allowing users to develop network topologies rapidly using a high-level scripting language or even a GUI interface and algorithmic behaviour in lower level compiled C++ [AS2000]. NS is a useful tool to network protocol designers and has been significantly enhanced by the introduction of packages to randomly create topologies as a means of testing and visualisation tools such as the Network Animator (NAM) [NA2002] allowing the user to see the simulation in action, or view it afterwards [AS2000].

The GUTS network simulator [GA2002] is an attempt to simulate large scale networks using a transfer-level abstraction. Using this approach has been shown to consistently produce runtimes of 2 magnitudes faster than ns and ns-simple, and require significantly less memory. As with ns the simulated network is built on a graph model, with the nodes representing routers and machines, and the links representing the cables linking them. The transfer time is calculated at the start of a transfer and is never changed, regardless of later network traffic. To model the bandwidth available to a given transfer the simulator uses the notion of over and under committing links in an attempt to balance out the margin of error. The simulator also accounts for network latency and queuing delays. The GUTS approach introduces a level of inaccuracy which is designed to be the same for all simulated services, therefore preserving an accurate estimate of their relative performances. The simulator was found to produce two sources of error in modelling transfers on simple topologies which were in bandwidth reservation and packet-level abstraction [AS2000]. However GUTS has been shown to be comparable to ideal TCP and to model realistic bandwidth and latency similarly to ns in larger scale experiments [GA2002].

SSFNET [CO1999] is a collection of models for simulating Internet protocols and networks. SSFNET is built on the Scalable Simulation Framework (SSF), which is an open source framework for discrete-event simulation of large, complex systems. Built on five primitive classes, users can extend them to build more significant devices [AS2000]. SSFNET is a packet level simulator and is freely available in both C++

and Java languages. SSFNET models are uniquely *self-configuring*; each class within it can automatically configure itself by querying a parameter database. This approach allows the designed network model to be easily verified for large models. It has been shown to be both accurate and able to model large scale networks up to 100,000 nodes, using a parallel execution on a cluster of processors [CO19999]. Although, SSFNET is more scalable than ns, it has not enjoyed such widespread developments so lacks the features and support.

2.6 Summary

It is not a case of if, but rather when a real world scale Grid becomes a reality and the paradigm is integrated into the fabric of our computing landscape. The technology is rapidly developing and a large number of different approaches are being researched. There is still plenty of work to be done, including perfecting and incorporating ubiquitous core Grid middleware services, aiding Grid application development, creating truly collaborative environments and developing effective scheduling mechanisms.

There are many different plausible scheduling architectures and methods. A number of these have been incorporating into user-level middleware systems and have shown to be useable. We have investigated a number of these systems and techniques, together with a look at Grid environments and attempts to simulate them.

Chapter 3 - G-Sim: A Grid Simulation Toolkit

3.1 Introduction

Carrying out performance testing on real Grid environments is difficult. Real Grid test beds are expensive and time consuming to construct. The dynamic nature of Grids makes them difficult to conduct repeatable and controllable experiments on. Never the less, experiments on Grids are necessary to verify the effectiveness of possible Grid scheduling procedures, which decide which tasks are distributed to which resources. To solve this problem, Grid simulation has emerged as an important tool. Grid simulations, discussed in the section 2.5, provide a means of making "virtual Grids" on which scheduling procedures can be evaluated. This chapter presents the design and implementation of G-Sim, a Grid simulation toolkit. This tool was developed as a means of testing various scheduling algorithms presented in the following chapters. Although numerous other Grid simulators have been developed, and are discussed in section 2.5.3, it was felt that none combined simplicity of use, speed of simulation, extendibility and numerous features, to make the process of creating and using virtual Grids easy.

3.2 G-Sim Overview

G-Sim is a discrete event Grid simulator written in Java. It can be used to model centralised, hierarchical and decentralised scheduling approaches, scheduling jobs for users on any number of different heterogeneous resources. G-Sim uses a simple, extendable application model which allows for the simulation of processor intensive tasks, where each may have any number of associated input or output files. Jobs can be made independent or dependant on the output of other jobs, allowing for the simulation of hierarchical applications models, as experimented with in [BL2004].

The use of Java as the implementation language, with its platform independence, allows the simulator to be run on multiple different execution environments. It is a

modern, object orientated (OO) language and popular with commercial developers, students and researchers alike. It also means that any user developed G-Sim extensions and experiments are also portable, allowing the possibility for scheduling algorithm designers to share code to improve the tool and provide a means of algorithm comparison. The fact that Java is an OO language means that the G-Sim can be easily extended to provide the functionality required by an individual user.

3.2.1 Features

The G-Sim tool provides the ability to model the following Grid entities:

Resources: G-Sim allows any number of heterogeneous resources to be modelled. These can operate under a time-shared or space-shared scheduling approach and have any number of homogeneous processors. These resources are also modelled with the ability to store files, which jobs may require as input, or output. Functions to manage the files on a resource are provided and the storage capacity can be finite or unlimited and allows for the modelling of scenarios involving data intense application. G-Sim offers the ability to model local processor usage on the resources which is based on given usage levels. These usage levels can change over simulation time, and be related to the time of day, the time zone and the holiday period of the resource.

Applications: The users of a Grid system submit applications to a scheduler to provide execution. These applications are typically constructed on many individual tasks. In G-Sim tasks are specified in relation to their size, given in millions of instructions (MI) and can require any number of specific input files and produce any output files. These files can be specified in terms of their size in KB. A job can not be executed on a resource without that resource having access to the required input files. When a job is completed the output files it produces can then be used as input for other jobs allowing the modelling of dependant DAGs of tasks as used in a number of real Grid applications.

Users: In G-Sim users create and submit jobs to a scheduler. They are capable of submitting any number of heterogeneous tasks at anytime and can be given a host

machine which output files from their application can be sent to. G-Sim offers functions for users to randomly generate jobs with various sizes within presented parameters and allows a group of users to submit their applications to their respective schedulers at random time intervals, based on an exponential distributed.

Network: G-Sim offers a comprehensive network model, however if the effects of the underlying network are of no significance to the simulation, instant transaction of data between the resources can be assumed. The G-Sim network model in based on the GUTS network simulator [GA2002], in this model, resources are connected in a network topology, represented by a graph where links represent the network cables, which have a given latency and bandwidth, and the nodes are the resources or the routers in the network. The GUTS, transfer level abstraction allows for realistic and fast simulation, and a simple way of exactly predicting the execution time of a given transfer in the model. This model is designer for users who are specifically interested in the effects of the underlying network on their algorithm, such as data Grid algorithm designers.

Schedulers: The purpose of G-Sim is to provide a test bed for Grid scheduling algorithms, so it is up to the user to implement the scheduler component of the experiment. G-Sim allows the modelling of multiple competitive schedulers, hierarchical schedulers and single centralised schedulers, which can operate in batch or online mode. In G-Sim, schedulers can access the underlying resources to find static properties such as processor numbers and speeds, scheduling policy and dynamic properties such as job, network and memory loads.

G-Sim also offers the ability to record experimental data in output files and can be configured to produce a range of different information according to the needs of the user. Although it is easy to output any data from the experiment, functions are supplied to output the makespan of the experiment, the time taken to complete the simulation, the activities of all the jobs processed and the actual time taken to run the scheduling algorithms, together with the number of jobs processed. The output files can then be used in conjunction with a spreadsheet package such as Microsoft Excel, to produce graphs from the experiment, allowing for easy analysis of the experiment.

3.3 System Architecture

A simulation built using G-Sim has a layered architecture as described in figure 3.1. At the base is the Java Runtime Environment which is built directly on the operating system and in turn the underlying hardware. This software provides a consistent, platform independent execution environment for the program, together with access to a variety of useful libraries. Built on this is the G-Sim toolkit which provides a structure for creating Grid simulations with various options together with classes for modelling Grid resources with various local loads models, the network over which the machines communicate and the users of the Grid environment together with their applications. Due to the varied nature of the Grid scheduling it is likely that each researcher will be focusing on a different aspect of the problem and as such will have different requirements of the system. To meet such requirements, the third layer incorporates users' own extensions of the G-Sim classes which adapt the tool to simulate experiments within their specific area of interest. To implement this layer, users are expected to be proficient in Java programming and have a fair understanding of the working of the tool itself. Such requirements of the user are reasonable since Grid scheduling algorithm designers are likely to be able programmers who would expect to have knowledge of the simulation toolkit they are using to be able to build their required experiments and to gauge the validity of their results. At the highest level is the experiment class which is also developed by the user and contains a main method allowing it to be directly run by the Java Virtual Machine. This class utilises the lower levels to build a virtual Grid environment with schedulers, resources, users and jobs, and is responsible for controlling output and initialising the simulation.

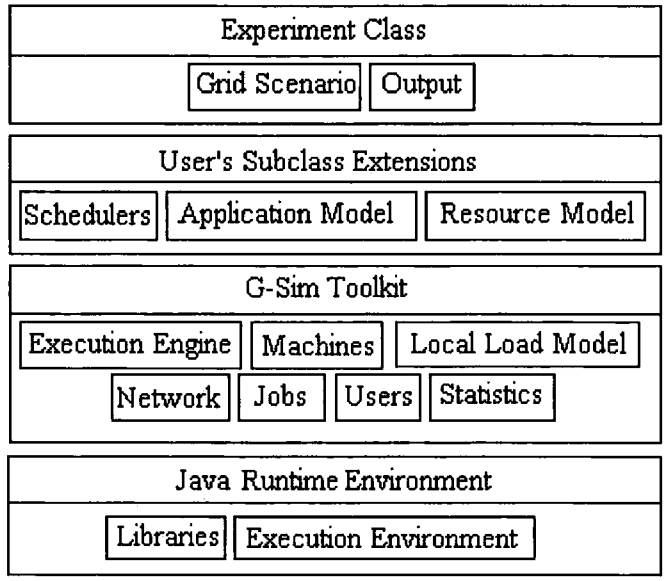

Figure 3.1: A layered architecture for G-Sim simulations

3.3.1 Execution Engine

The execution engine, or executive, defines how the simulator processes events and controls time progression. G-Sim, unlike GridSim, is built from the ground up, rather than building upon existing simulation infrastructure [MU2002]. Doing this allows G-Sims execution engine to be both simple to use and efficient.

To accomplish this G-Sim uses a serial execution engine, consisting of only a single thread of execution. This means that although it will operate slower than a parallel based simulator on a cluster of processors, it should operate faster on a single processor machine, which every user is likely to have access to, because the overhead of context switching (time to move from one thread of execution to another), is minimised. The use of a serial approach to simulation also means that the problems associated with parallel applications, namely deadlock, mutual execution and starvation can be ignored making development for users simpler. The other advantage of using a single processor approach is that the maximum number of threads that a machine can handle is usually the limiting factor in the number of entities that can be modelled [PU2003], meaning that G-Sim will be able to handle larger scale simulations on most machines than a parallel execution simulator such as GridSim. Psuedo code for the main loop which runs the simulator is given in figure 3.2 below.

```
1. while true
2.   if there are no events left
3.     end simulation
4.   end if
5.   currentEvent = get next event
6.   if currentEvent time >= time limit
                or currentEvent time – last job activity > no job activity limit
                or max number of jobs reached
7.     end simulation
8.   end if
9.   perform currentEvent
10.  remove currentEvent
11. end while
```

Figure 3.2: Pseudo code for G-Sim's thread of execution

The events themselves are objects which perform a specific function when activated by the code on line 9. The events are stored in order of the time they occur and so simulated time progresses as each event is activated. The events are stored in a binary tree set data structure, allowing them to be removed, added and checked in guaranteed $O(\log n)$ time [JA2005]. Simulation can be ended in a number of ways. If there are no events left, in line 3, if the maximum job or time limit is reached or if no jobs have been processed for a given time, in line 7 or if an event is processed which causes the simulation to end. The code given in figure 3.2 is contained within the Executor class. This class is responsible for keeping track of the events, executing their code and ensuring data is output at the end of the simulation. To reduce the amount of object passing in the system, the functions and data structure within the Executor class are declared static. This has the effect that the public (visible) methods are given unrestricted access throughout the program and that there can effectively be only one loop running at once.

3.3.2 Entity Classes

G-Sim has a number of base classes which are used to represent the entities described in section 3.2.1. The idea is that these classes provide methods and data structures to make them both usable and flexible. Flexibility is required for the classes to be extended and adapted for many different experiments and usability is required so that they can be easily included in experiments straight away without the need for additional coding. Figure 3.3 shows the relationships between the base classes.

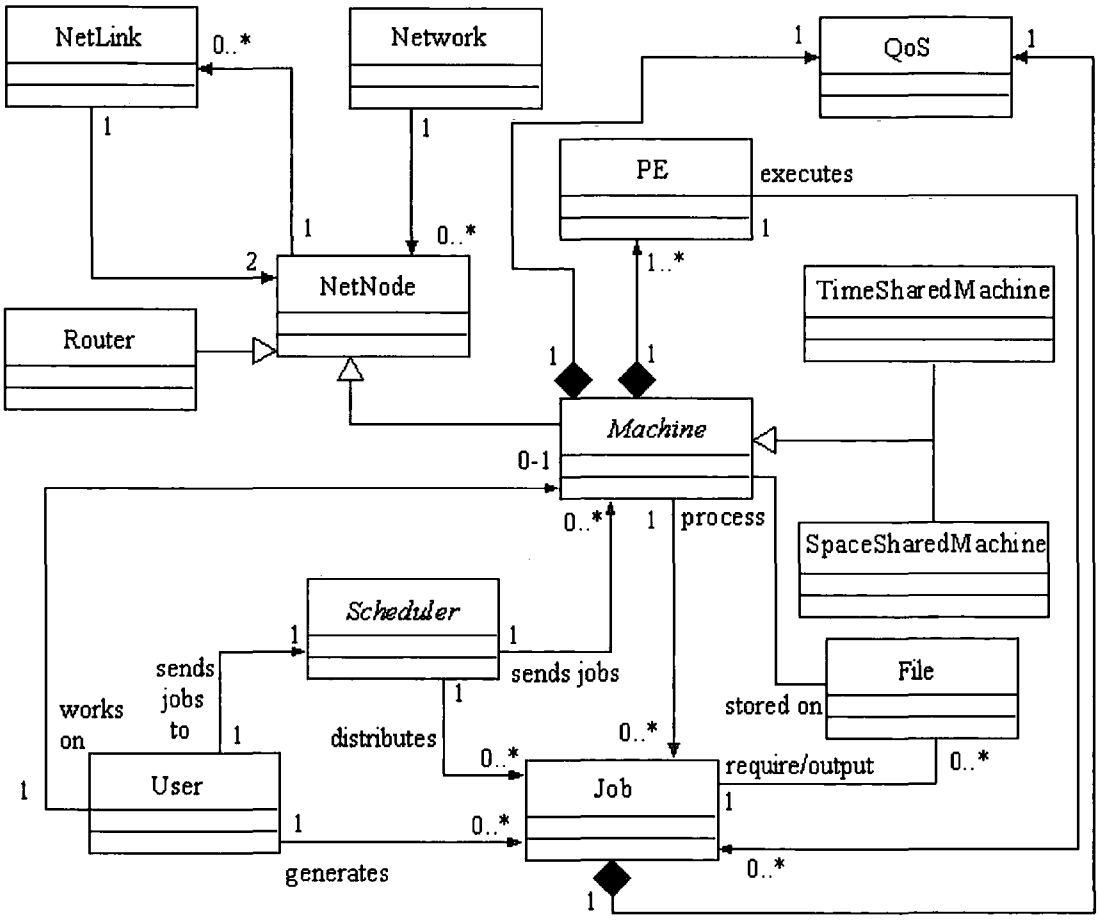

Figure 3.3: UML class diagram for the G-Sim base classes

As there is no obvious default implementation, the Machine class is declared abstract, although two concrete implementations are included to represent machines with single or multiple processors operating under share-shared or time-shared scheduling

procedures. Further details of these implementations are given in section 3.4. Each machine has a number of processor elements (PEs), which run at a given speed; these processors execute users' jobs when given to them by a scheduler. A machine can be associated with a user, allowing it to be used to host input or output files for their jobs. These Jobs may require Files as input or create them as output. Machines are capable of storing Files, and can be configured to have either an infinite memory capacity or a specified limited one. To support the simulation of DAGs of tasks, each file has a unique number which identifies it, a Job may produce a file as output, which anther job can use as input. A Job will only be accepted by a machine if that machine has enough free space to store the Jobs input and output files and has equal or greater Quality of Service (QoS). The default implementation of the QoS is to simply say all machines are capable of processing all jobs; it is up to the user to extend the class if they require a more specific QoS model. In G-Sim, the Machine objects, like Routers, are also NetNode objects. The NetNode class represents nodes in a network topology which can be connected via links (represented by NetLinks objects). The design of the NetLinks and the controlling Network class is based on the GUTS network simulation model and are discussed further in section 3.6.

The function of the scheduler component is to act as a gateway through which users' jobs are processed on the machines. There is no limit to how many schedulers a simulation can have and any number of users can use the same scheduler. The scheduler class is declared abstract and the user is responsible for creating a concrete version. As opposed to on a real Grid where the scheduler runs as a network service on a machine connected to the same network as the users and resources (usually the Internet), schedulers in G-Sim are not considered to execute on an actual machine, but, rather are considered to be an abstract components which users and machines can access instantly. This eliminates the effects of network traffic and local load on executing machine and helps to simplify the task to creating the scheduler.

3.4 Simulating Multitasking and Multiprocessing

On a computational Grid resource, there are often many processes running concurrently, these are allocated resources including processing time, according to the

scheduling algorithm provided by the operating system it is running, a process known as multitasking. Some resources may have more than one processor, and so the scheduling algorithm must allocate jobs between processors to ensure effective execution, the process of multiprocessing. Due to the fast switching between processes (many times a second), the large number of possible operating system algorithms and the number of factors affecting them, simulating these two processes exactly would be both difficult and result in slow simulator execution. Hence, in G-Sim, these two processes are simplified by using discrete events to signal the ending or beginning or a process on a processor. An event is continually scheduled to trigger when the smallest job on each PE is due to finish. Each PE is considered to operate on a round robin algorithm with each process running on it given equal processor time, equivalent to the PE's speed divided by the number of processes. The algorithm used to select the next job to finish and update the progress of all the jobs, called when a job is started or complete or when the speed of the processor is changed to model local loads, implemented within the PE class, is given in figure 3.3.

The size of jobs in G-Sim is given in the number of millions of instructions (MI) they require. The speed of processors is defined by the number of millions of instructions they can perform in one second (MIPS). The algorithm in figure 3.3 finds the jobs with the smallest amount left to process in lines 10 to 17 and updates the amount of instructions they have left to process in line 12. This amount is calculated in line 8, in which it assumes each job has received equal processor time for the length of time between the current simulation time and the time the algorithm was last run.

```
1. begin void setNextJobToFinish()
2.    time = Executor.getCurrentTime()
3.    if time == timeLastUpdated
4.        return
5.    end if
6.    if there are jobs running
7.        currentRatePerJob = current processing speed / number of jobs running
8.        MISinceLastEvent = currentRatePerJob * (time-timeLastUpdated);
9.        smallestMILeft = first job in list
10.       while not all jobs have been considered
11.           jobGoing = the next job in the list
12.           MILeft = jobGoing.getMILeft() - MISinceLastEvent;
13.           jobGoing.setMILeft(MILeft)
14.           if smallestMILeft.getMILeft() > MILeft
15.               smallestMILeft = jobGoing
16.           end if
17.       end while
18.       nextJobToFinish = smallestMILeft;
19.   end if
20.   else
21.       nextJobToFinish = null;
22.   end else
23.   timeLastUpdated = time;
24. end setNextJobToFinish
```

Figure 3.4: Algorithm used to update the progress of the jobs running on a PE and find the next job to finish

G-Sim uses the Machine class to represent resources connected to the Grid. These resources can have any number of processing elements (PEs) which execute users' jobs depending on their given speeds. Shared memory clusters of machines can be modelled as machines with multiple PEs and standard PCs as machines with one PE. The way in which the jobs are allocated across the processors is dependent on the allocation policy defined by the implementations of the Machine class's abstract methods provided within a concrete subclass. If the user requires a different allocating policy to the two provided, they must create a subclass which overrides these methods. To simplify the task of creating a new allocation policy, the user has only to override two methods, *jobStarted* and *jobCompleted*. The user does not need to deal with the creation and deletion of the events used in simulating the processors' behaviour as this is abstracted away from these methods, making them simple to implement. Appendix figure A.1 shows how these abstract methods are invoked,

together with how the entities interact when called by events to schedule, start and complete users' jobs.

The *jobStarted* method takes a job and starts processing it. By the time this method is called, the job has been accepted by the machine and the required input files are loaded onto it. The function of the *jobCompleted* method is to decide what to do when a processor has finished executing a job. Three other empty, non-abstract methods *haltJob*, *restartJob* and *cancelJob* are included in the Machine class to allow users' to temporarily halt, restart or cancel jobs whilst in execution respectively. These methods are not declared abstract so as not to force them to be implemented if they are not required. If these methods are called on an instance of a subclass without them being implemented, an error message is reported and the simulation ended. As the simulations carried out in this project did not require this functionality, the provided implementations do not override these methods, but, could be altered to do so.

The two resource models provided in G-Sim are time-shared and space-shared. These models are based on those used in the popular GridSim toolkit [BY2002]. They assume the processors are homogeneous in speed; this is because this is the case with real clusters and trivially in any single processor machine. The Machine class does not specifically limit resources to being homogeneous, so creating a machine class which deals with heterogeneous processors is possible within G-Sim. Alternatively if network issues are not important to the simulation, heterogeneous machines can be modelled as a collection of single processor machines, using the same Grid scheduler.

To illustrate the operation of the space-shared and time-shared operation, let us consider a simple scenario. There is only one Grid resource, a machine with two processors, each with a MIPS value of 1. Four jobs are presented to this machine to be processed; their MI requirements are 14, 5, 10, and 8. These jobs arrive at the machine after 0, 2, 5 and 10 seconds respectively. The way in which the machine processes the jobs using a space-shared allocation policy is shown in figure 3.6 and using a time-shared policy is shown in figure 3.8. The statistics for this scenario are presented in table 3.1.

3.4.1 Space-Shared Resource Model

In a space-shared resource, processors only ever run one job at a time. Typically large cluster machines will use a space-shared scheduling policy. In the G-Sim model, space-shared resources operate on a first-come-first-served basis, if there are no free processors to run jobs, they are placed in queue. The *jobStarted* and *jobCompleted* methods which provide this behaviour, implemented in the SpaceSharedMachine class are given in figure 3.5. Figure 3.6 shows the how the space-shared resource model handles the given scenario.

```
1.  begin void jobStarted(Job job)
2.      PE freeProcessor = get a free processor
3.      if freeProcessor != null
4.          add job to the Queue
5.      end if
6.      else
7.          start job on freeProcessor
8.      end else
9.  end jobStarted

1.  begin void jobCompleted(PE processor)
2.      if job Queue isn't empty
3.          Job job = get first job in queue
4.          start job on processor
5.      end if
6.  end jobCompleted
```

Figure 3.5: Pseudo code of the implementation of the abstract Machine class methods provided within the SpaceSharedMachine class

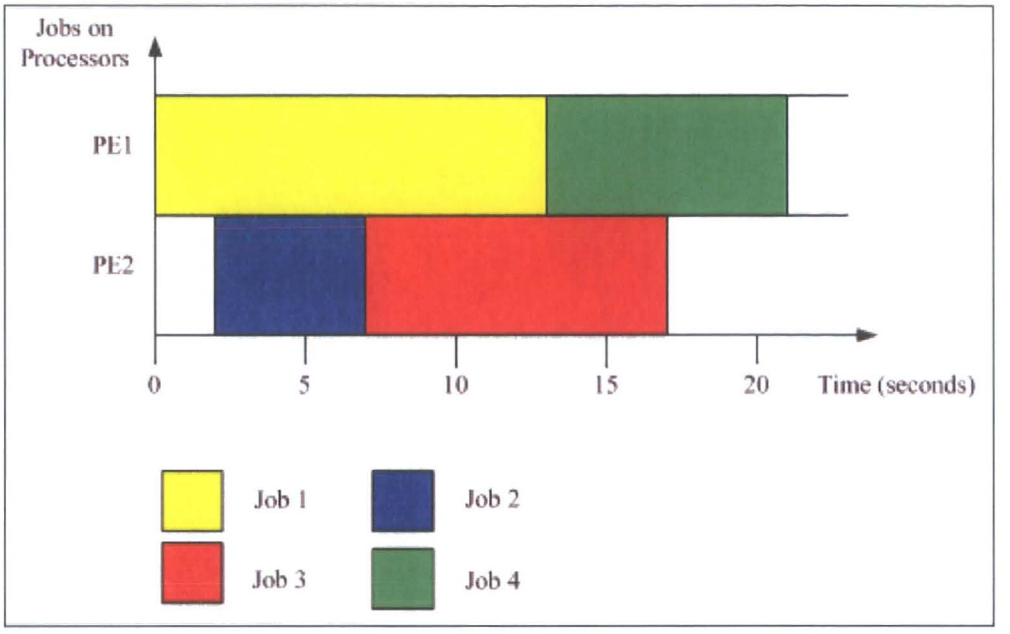

Figure 3.6: View of the scenario, with a space-shared allocation policy

As figure 3.6 shows, each processor only ever runs one job on each processor at a time. When jobs 1 and 2 arrive, there is a free processor so they begin execution immediately, however when jobs 3 and 4 arrive (at times 5 and 10 seconds respectively) both processors are busy and so they are placed in a queue until the other jobs have finished (at times 7 and 14 seconds), when they can begin execution. When job 1 arrives an event is scheduled to trigger at time 13s to signal when it will finish execution, and invoke the *jobCompleted* method. This event is removed before it gets a chance to be activated when the second job arrives at time 2s as, having less instructions left to process, will finish first. In its place another event is scheduled at time 7s to signal the end of job 2, which does get activated. When this occurs, the *jobCompleted* method is invoked causing job 3 to begin execution, and it is removed and eventually deleted by Java's garbage collector thread. Another event the same as the first is then rescheduled to be activated after 13s, to signal the end of job 1. The process of creating events to signal the end of a job then either activating or deleting them if they become obsolete, due to other smaller jobs beginning execution, continues until all the jobs are completed.

3.4.2 Time-Shared Resource Model

A time-shared resource can execute many jobs on each of its processors concurrently. Unlike space-shared resource time-shared ones begin executing jobs as soon as they arrive at the machine regardless of load. Single processor machines and lower-end multiprocessor machines usually use a time-shared allocation policy. The time-shared resource model in G-Sim treats all jobs equally and schedules them as they arrive on a processor which is currently running the least number of jobs. When a job finishes, if another PE is running two jobs more than the processor on which the job finished, one of its jobs are swapped onto the other machine, balancing the load. The code to provide this behaviour is presented in figure 3.7.

```
1.  begin void jobStarted(Job job)
2.      lowestNumOfJobs = floor(number of jobs running / number of PEs)
3.      while not all processors have been considered
4.              PE processor = get next processor
5.              if processor is running lowestNumOfJobs jobs
6.                      start job on processor
7.                      return
8.              end if
9.      end while
10. end jobStarted

1.  begin void jobCompleted(PE processor)
2.      runningJobs = number of jobs processor is running
3.      while not all processors have been considered
4.              PE p = get next processor
5.              if runningJobs + 1 < number of jobs p is running
6.                      swap a job from p to processor
7.              end if
8.      end while
9.  end jobCompleted
```

Figure 3.7: Pseudo code of the implementation of the abstract Machine class methods provided within the TimeSharedMachine class

The methods in figure 3.7 look to balance the load of the jobs across all the processors and works on the basis that processors are homogeneous in execution speed. Line 2 of the *jobStarted* method is used to calculate the minimum number of jobs running on any of the processors. The first processor running this number of jobs is then used to

run the given job. The *jobCompleted* method performs load balancing by finding the number of jobs currently in execution on the given processor, which has just finishing running a job, then attempting to locate a processor executing at least two jobs more, if such as processor is found then one of its jobs is selected to be transferred onto the given processor. The selection process used to decide which job to swap is simple to maximise simulation speed. The first job to begin execution is swapped unless it is due to finish next, in which case the second job is used. This is the quickest selection process since the first job can be found fastest and removing the next job to finish on a processor will result in having to remove the *jobFinishedEvent* object associated with it and schedule another one. Figure 3.8 shows how the methods given in figure 3.7 operate on the described scenario.

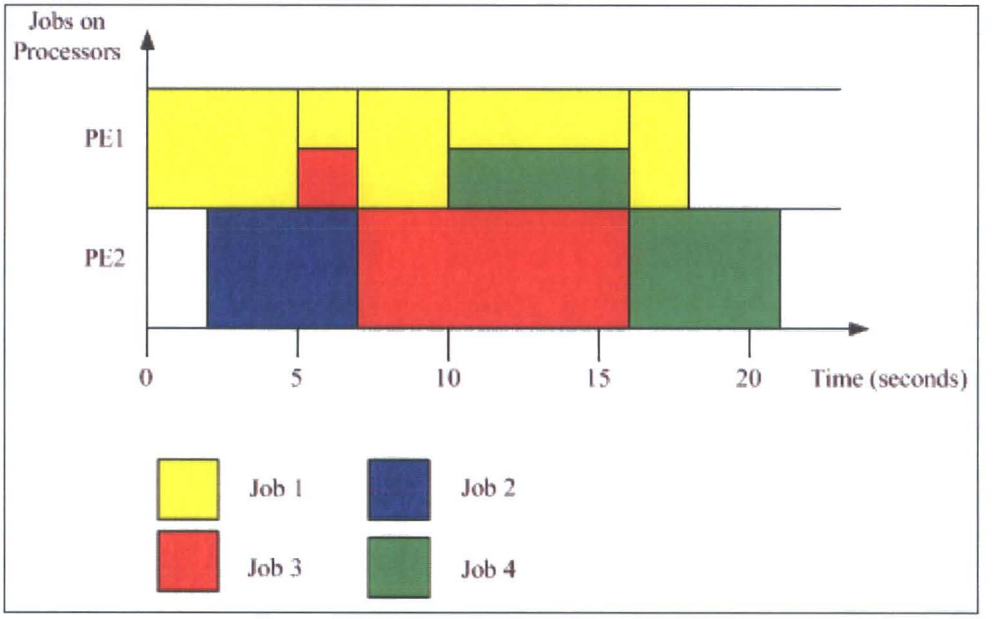

Figure 3.8: View of the scenario under a time-shared allocation policy

As figure 3.8 shows, the time-shared machine always allows jobs to execute as soon as they arrive regardless of load. To achieve this, it is sometimes necessary to execute more than one job at a time on a processor (such as on PE1 between 5 and 7 seconds and between 10 and 16 seconds). When PE1 begins executing job one, at time 0, an event is scheduled for time 14, when the job is expected to be complete. However when job 3 comes along, at time 5, this event must be removed and replaced as the next job is expected to complete after 23 seconds, (as the smallest job it has left has 9

MI left and the rate per job drops to 0.5 MIPS). This event is also removed and replaced before being activated as, once PE2 has finished processing job 2, after 7 seconds, it is running two less jobs than PE1 and therefore balances the load by accepting one of them. This has the effect that PE1 is now only processing jobs 1 and therefore can provide it with the full 1 MIPS rate, enabling it to be completed within 8 seconds. The process of removing and rescheduling events occurs frequently in time-shared machines, making their simulation slightly slower than space-shared resources.

			Space Shared Resource			Time Shared Resource		
Job	Length (MI)	Arrival Time	Start Time	Finish Time	Elapsed Time	Start Time	Finish Time	Elapsed Time
1	14	0	0	13	13	0	17	17
2	5	2	2	7	5	2	7	5
3	10	5	7	17	12	5	16	11
4	8	10	13	21	11	10	21	11
Total	37				41			44

Table 3.1: Statistics for the scheduling scenario

3.5 Local Load Modelling

On real Grids, resources are usually non-dedicated; they will not only process the jobs presented to them by a Grid scheduler, but, also local jobs. In some models such as that considered in [SU2003], Grid and local jobs are considered to have equal priority. Within G-Sim, this could be modelled by setting up another Grid scheduler which would present jobs generated by a different set of users to those being analysed, to the machine on which you wished model the local load. However this method is not suitable for modelling local loads on Grid models which consider local jobs to have priority over remote ones. Due to the need for site autonomy and the high cost of purchasing and running resources, this is nearly always the case within real Grids.

Local usage values are subject to random change and maybe dependant on the type of machine, the time of day and the time of year. For instance you might expect networked PCs in a teaching laboratory to be busy during times when practical lessons are conducted and be inactive or at low loads at night time or in the holidays. Cluster machines are likely to have different usages patterns as often they are put to work on calculations over night so as to be ready for the next day morning. Taking

these factors into account, the G-Sim local usage model was produced, the class diagram for which is presented in figure 3.9.

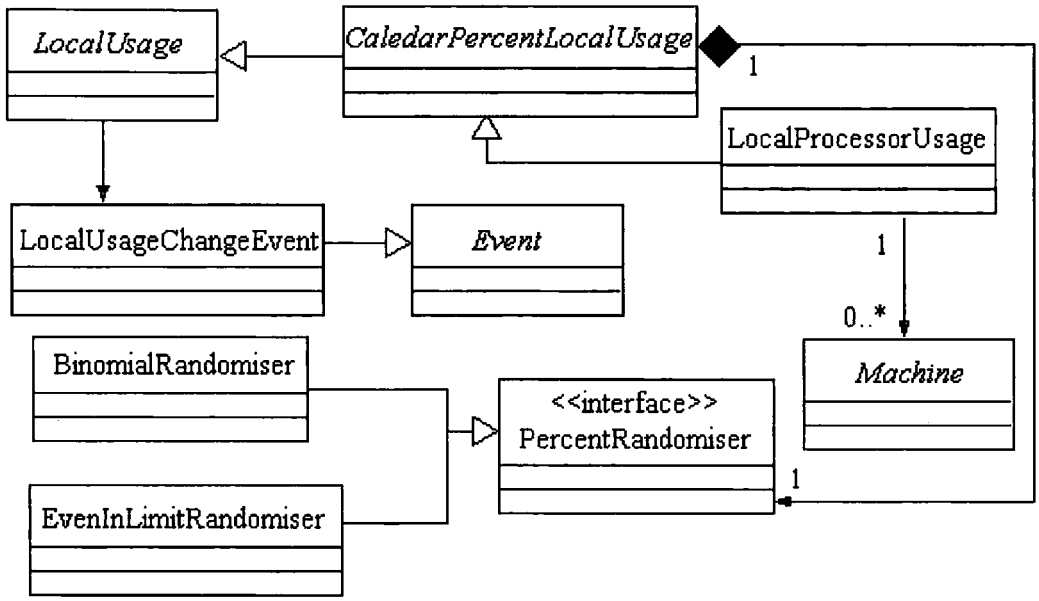

Figure 3.9: UML class diagram for the local usage model in G-Sim

G-Sim's local usage model has several levels. The lowest is the LocalUsage class, which simply describes when a change in the local usage value can occur. This class allow the details of scheduling events to activate the change in local usage value to be abstracted from any subclasses, yet does not restrict the value itself to any specific concrete form or the design of the algorithm which generates it.

The next level, built as a subclass of LocalUsage is the CalendarPercentLocalUsage (CPLU) class. This supplies a structure for creating local usage values represented as percentages which can be configured to change according to a random generator and time of day, week and year, but, does not restrict the resource on which it operates to any specific type. To fluctuate according to time of day, week and year, the class requires four inputs which are as follows:

- byte [] weekdays
- byte [] weekends
- byte [] holidays
- boolean [] holidaysInYear

These arrays contain average, percentage local load values for a single day, inside their respective periods of time. Weekdays are considered to be 12 a.m. on Mondays to 12 a.m. Saturdays, weekends to be 12 a.m. Saturdays to 12 a.m. Mondays, while not in the holiday period. The holiday period is defined by the holidaysInYear arguement, which is required to be 52 elements in length; each representing a week in the year (as defined in the java.util.GregorianCalendar class definition presented in [JA2005]). If no holiday period is required, the holidays and holidaysInYear arguments can be left undefined. The number of arguments within the byte arrays defines how the long each element is considered to give the average value and each element is considered to cover an equal amount of time. For instance, presenting a 24 element byte array would make each represent the average value for an hour, and the transition between elements would take place on the hour, with a single element array the value would represent the average value for the entire day. Figure 3.11 gives example arrays and shows the corresponding value/time functions they relate to.

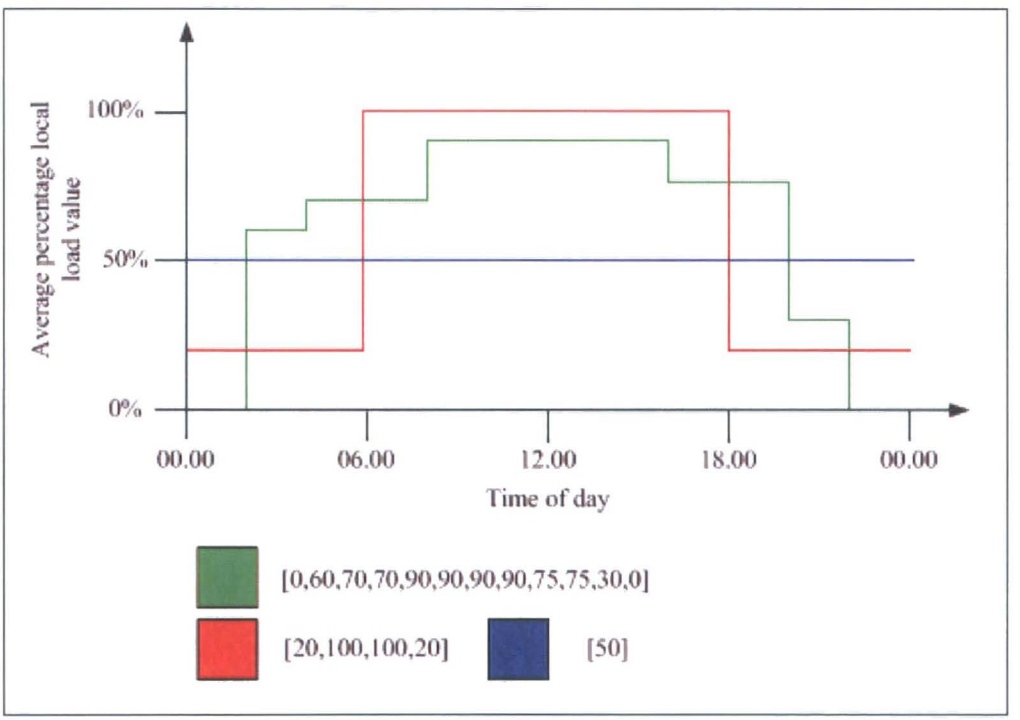

Figure 3.11: View of example average percentage load values

The object representing the simulation time and date (an instance of java.util.GregorianCalendar) is held by the Executor, along with a value representing a standard time-zone. These are accessed by the CPLU class through a number of static methods. Using this information together with details of the time-zone in which the resource is located (given as an offset from GMT), the CPLU class is able to calculate the time of day and year at each resource, which it uses to find the corresponding byte array and elements storing the required average value. These values are considered to be "average" since they can be altered using a randomising function, which is a class realising the PercentageRandomiser interface. This interface has only one method; *randomise*. This method takes a given average percentage seed value and outputs another percentage value. Two implementations of this class are provided, the EvenInLimitRandomiser (EILR) and the BinomialRandomiser (BR). The EILR class is designed to allow the modelling of random local load functions which are evenly distributed within a given range. It performs a randomising operation which takes a value for the maximum error of the average value and returns a value evenly distributed between the average minus the maximum error value and the average plus the maximum error value. This is simple to accomplish with the exception of when the range of possible output values goes below 0% or above 100%, the outputs are then squashed into a limited range. This causes the average output value to move away from the intersecting boundary, and hence the given average, if the output continues to be evenly distributed. To rectify this, the algorithm has been designed to increase the probability of getting a boundary value (either 0% or 100%) to ensure the average value output by the algorithm continues to be the average value input. This change in probability is calculated using the mean average function (equation 3.1) together with the fact that the output values are evenly distributed between the possible range of values. With this information, we can adapt equation 3.1 to equation 3.2 for when the range goes below 0%, and to equation 3.4 for when the range goes over 100%. These can then be made easy to calculate using a computer and given in terms of X, resulting in equations 3.3 and 3.5.

$$\frac{\sum x}{N} = AV$$

Equation 3.1: The mean average function, where x's are the values, N is the number of values and AV is the mean average.

$$\frac{\sum_{i=1}^{AV+MAXOUT} i}{X + AV + MAXOUT} = AV$$

Equation 3.2: Adapted mean average equation, for calculating the share of values required at 0% when the range of output value goes below it. *AV* is the average value, *MAXOUT* is the maximum error of the function and *X* is the share of values given at 0% to the 1 share of the other possible outputs.

$$\frac{(AV + MAXOUT) * \left(\frac{AV + MAXOUT + 1}{2} \right)}{AV} - AV - MAXOUT = X$$

Equation 3.3: Easily calculated function for *X*, derived from equation 3.2. *AV* is the average value, *MAXOUT* is the maximum error of the function and *X* is the share of values given at 0% to the 1 share of the other possible outputs.

$$\frac{100X + \sum_{i=AV-MAXOUT}^{99} i}{X + 99 - AV + MAXOUT + 1} = AV$$

Equation 3.4: Adapted mean average equation, for calculating the share of values required at 100% when the range of output value goes above it. *AV* is the average value, *MAXOUT* is the maximum error of the function and *X* is the share of values given at 100% to the 1 share of the other possible outputs.

$$\frac{-AV \times (100 - AV + MAXOUT) + (100 - AV + MAXOUT) \times \left(99 - \left(\frac{99 - AV + MAXOUT}{2} \right) \right)}{AV - 100} = X$$

Equation 3.5: Easily calculated function for *X*, derived from equation 3.4. *AV* is the average value, *MAXOUT* is the maximum error of the function and *X* is the share of values given at 100% to the 1 share of the other possible outputs.

Pseudo code for the randomising algorithm, given in the EILR class, which uses equations 3.3 and 3.5. to randomise the average local usage percentage values is given in appendix figure A2.

The BR class is used to model random local load functions where the values follow a binomially distribution. The equation for the discrete probability distribution function given by the binomial distribution is given in equation 3.6.

$$P_p(n \mid N) = \binom{N}{n} p^n q^{N-n} = \frac{N!}{n!(N-n)!} p^n (1-p)^{N-n}$$

Equation 3.6: The binomial distribution gives the discrete probability distribution, P_p ($n \mid N$), of obtaining exactly n successes out of N Bernoulli trials (where the result of each Bernoulli trial is true with probability p and false with probability $q = 1 - p$) [WE1999].

As percentages in G-Sim are treated as discrete values, the binomial function can be applied in randomising the local load values by treating each percent as an independent event with the average divided by 100 as the probability of it being true using, then using 100 trials. This allows for a probability density function based on the average value to be built and used with evenly distributed random variables, between 0 and 1, to output binomially distributed random variables. However this function would be difficult to calculate due to the sheer size of the numbers involved (i.e. 100! is far larger than the maximum number any Java primitive can represent). Even if the probability values could be pre-calculated and stored, it would be an inefficient use of memory as the number of possible average values, together with their respective probability density functions gives 10,201 values that would need to be stored; equivalent to almost 80 megabytes of memory when using Java's standard double precision floating point representation. To combat these problems, the BR class considers each "trial" to represent the chance of 5 percent being true, greatly reducing the number of possible combinations of outputs and inputs to 441 (21 x 21). These are pre-calculated and stored, allowing the algorithm, whose pseudo code is

given in figure 3.12, to work quickly. The outcome of using this simplification is that input average values are rounded to the nearest 5%, and the probabilities are calculated from the mid-point of each 5% trial e.g. when the average load value is 0-2.5%, the chance of each trial being true is treated as 0.125%. This makes little difference to the realism of the model since the rapid fluctuations and unpredictability of local load values for real computing resources such as network bandwidth, memory usage and processor usage make them extremely difficult to calculate to the degree of accuracy of a single percent.

```
1.  begin byte randomise(byte averageValue)
2.      if averageValue > 100
3.          return 100
4.      end if
5.      if averageValue < 0
6.          return 0
7.      end if
8.      index = round(averageValue/5)
9.      cumulative [] = cumulativeBinomialData[index]
10.     random = randomNumberGenerator();
11.     for i = 0..21
12.         if random < cumulative[i]
13.             return 5*i
14.         end if
15.     end for
16.     return 100
17. end randomise
```

Figure 3.12: Pseudo code for the randomise method of the BR class.

The 2D array *cumulativeBinomialData*, accessed in line 9 of the code given in figure 3.12, stores the probabilities of each number of trials being true for all the different probabilities of a single trial being true. As its name suggests, the probabilities are stored cumulatively, so each 1D array returned on line 9, has elements with values representing the probability of getting at least its index number of trials right. This allows the simple loop in lines 11 to 15 to locate the number of trials, and hence the output percentage using the evenly distributed random variable created in line 10.

In order to test the randomise methods provided by the BR and EILR classes, their output was recorded after 10,000 runs with different inputs, the results of these

experiments are presented in figure 3.13. The average of all the values output in each test was no more than 0.01% from the input average value, the BIRL class produced results mimicking the binomial distribution and the EIRL class gave roughly evenly distributed values, suggesting the algorithms are correctly implemented.

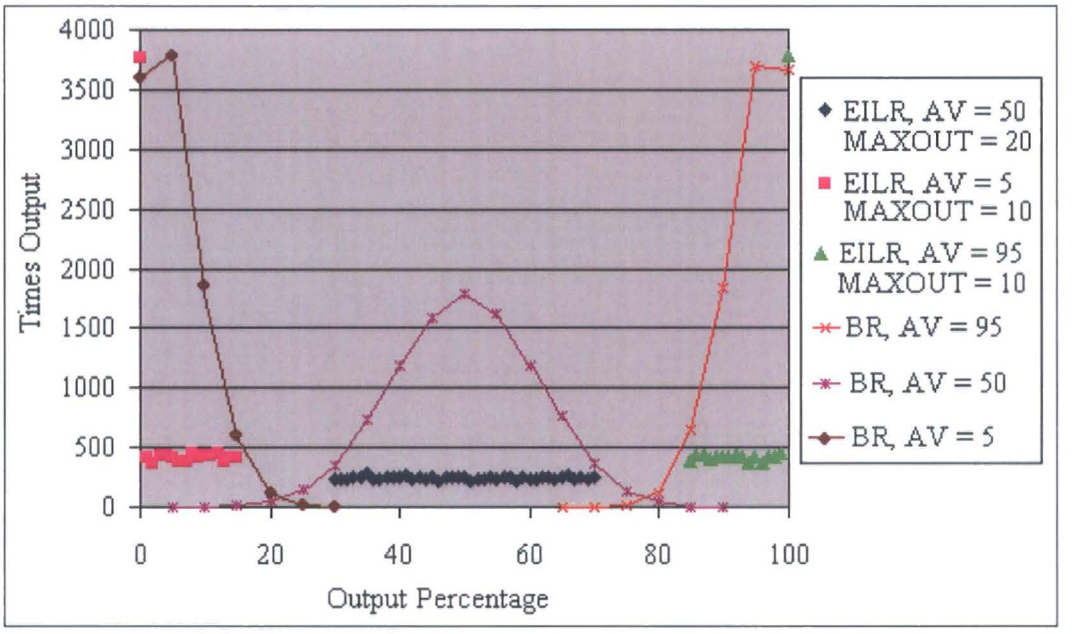

Figure 3.13: Number of times output percentages appeared after 10,000 calls to the randomise methods of the BR and EILR classes.

At the highest level of the local load model in G-Sim are the concrete subclasses of the CPLU class. These specify how the percentage values are applied to the resource to affect its load. The only such class supplied in the G-Sim toolkit is the LocalProcessorUsage class which operates on the resource of processor speed. This supplies a simple implementation of the single abstract method, *updateLocalUsage*, in which it decreases its associated machines' processors' speed by the given average value or the value run through a randomising method depending on whether a randomiser is set. The local load on other resources such as network bandwidth and memory could also be modelled by implementing other subclasses of the CPLU class although no default implementation is required.

The local load model was selected due to its flexibility; in that virtually any local usage model can be constructed using it, its ease of use; since only a few values are required to create the function and its applicability; most resource loads can be modelled in percentages and their values are likely to change in relation to weekends, weekdays and holidays periods due to the structure of working weeks.

3.6 Network Model

As previously mentioned, G-Sim's network model is based on the GUTS network simulator [GA2002]. This model is a transfer level simulation of a network in which the time to finish a particular transfer is calculated based on the current load of the network and remains unchanged despite any future changes which may occur. The idea behind GUTS is that by creating a balance between over-committing and under-committing links in the network, an accurate, course-grained and fast network simulation can be created. As with most network simulators, GUTS considers the network to be a graph, where the routers and hosts are represented as the nodes and the cables linking them to be the edges [GA2002]. Each link has two static properties representing its maximum bandwidth capacity C_l and propagation delay D_l and two dynamic properties, updated at the beginning and end of each transfer, the allocated bandwidth A_l and the number of transfers utilising it T_l. The algorithm used to calculate the length of time to complete transfers of size s between two nodes at time t is given in figure 3.14.

1. **Find the set P of all links on the path from source to sink, and label the link l_b which has the least *available bandwidth* B_{l_b} as the *bottleneck* link.**
2. **For each link $l \in P$, "reserve" bandwidth by incrementing A_l by B_{l_b}.**
3. **Increment the T_l for each link $l \in P$.**
4. **Calculate the total duration d of the transfer.**
5. **Schedule an event to occur at time $t + d$ that, for each $l \in P$, releases bandwidth B_{l_b} and decrements T_l.**

Figure 3.14: Main algorithm of the GUTS network model [GA2002].

The available bandwidth for each link B_l is defined as the largest of either the amount of unallocated bandwidth or the capacity divided by the number of transfers utilising the link plus one for the new transfer, this is described formally in equation 3.7, where the function *MAX* returns the largest of its arguments.

$$B_l = MAX\left(C_l - A_l, \frac{C_l}{T_l + 1}\right)$$

Equation 3.7: Definition of available bandwidth in GUTS [GA2002]

The path P between the source and sink nodes is found using a form of Dijkstra algorithm for finding the shortest path in a graph, the pseudo code for which is given in figure 3.15.

The idea of Dijkstra's algorithm is that when starting at the source node you always follow the shortest length path, each node you get to is considered to be settled meaning that no other path to that node will be shorter. You continue following the shortest possible path until the sink node becomes settled. The output path is then created by looking at the predecessor of each node on the shortest path from the source, starting with the sink node. The implementation of Dijkstra's algorithm used in G-Sim is adapted from [WA2005] and uses the TreeMap priority queue [JA2005] in which to store unsettled nodes, resulting in a reduction in the algorithm's runtime from $O(n^2)$, where n is the number of nodes, to $O(n \log m)$, where m is the number of unsettled nodes on each iteration and $m < n$.

```
1. begin List shortestPath(Node source, Node sink)
2.    add source node to unsettledNodes
3.    set shortest path to source to be 0 and infinite to others
4.    while unsettled nodes isn't empty
5.       Node node = node with smallest path to it in unsettledNodes
6.       remove node from unsettledNodes
7.       if node == sink
8.          break
9.       end if
10.      add node to settledNodes
11.      List adjacent = get list of nodes linked to node
12.      for all nodes in adjacent
13.         Node adjacentNode = next node in adjacent
14.         if adjacentNode isn't in settled
15.            pathLength = distance to node + weight of link between adjacentNode and node
16.            if pathLength < smallest path found to adjacent
17.                set shortest distance to adjacentNode to be pathLength
18.                set adjacentNode's predecessor to be node
19.                add adjacentNode to unsettledNodes
20.            end if
21.         end if
22.      end for
23.   end while
24.   List path = a new empty list
25.   Node node = get sinks predecessor
26.   add link between sink and node
27.   while node != source
28.      Node pre = get node's predecessor
29.      add link between pre and node to path
30.      node = pre
31.   end while
32.   return path
33. end shortestPath
```

Figure 3.15: Pseudo code for Dijkstra's shortest path algorithm used to find shortest path between two nodes in the GUTS network model.

The total duration of the transfer d is calculated as the sum of the propagation delays on each link in the path P, plus a queuing delay q related to the number of transfers on each link at the transfer start time, multiplied by three to account for the two SYNs (packets sent to request to synchronize sequence numbers, used when opening a TCP connection), plus, the actual on-link transfer time given as the size of the file being transferred divided by the bandwidth available at the bottleneck link B_{lb} [GA2002].

The queuing delay q is calculated on the assumption that every transfer in progress on each link in the path P has at least one packet of maximum size MTU on the link's queue ready to send and one of those is currently on the link, halfway through on average [GA2002]. The start of the new transfer on this link will be delayed by these packets before it gets a chance to use the link therefore q is defined by equation 3.8.

$$q = \sum_{l \in P} \frac{MTU}{B_l}(T_l - 0.5), \text{ for } T_l > 0$$

Equaton 3.8: Definition of the queuing delay in the GUTS network simulator

The method given in figure 3.14 is implemented within G-Sim's Network class, together with an additional method to calculate the duration of a transfer without actually changing the dynamic properties of the links. The Network class also maintains a list of all the NetNodes in the network, whilst they themselves contain a list of the NetLink objects which directly connect it to any other NetNodes. The two possible types of NetNode in a network are Routers and Machines. Routers are simply NetNodes which link to other NetNodes, whilst Machines are capable of hosting, storing and downloading files from other Machines. Storing files on a Machine is either a prerequisite for a Job object starting execution on it or as a result of its execution. As file transfers maybe required prior to initiating execution, there needs to be a means for locating hosts storing particular files within the network. In G-Sim this is accomplished by declaring Machines to be file hosts; a file host allows other Machines to download the files stored on it. These hosts can then be located through calls to the File class's static methods and data structures which act as a central repository, mapping files to the hosts on which they are stored. If a requested file is located on at least one host, a download is initiated between the Machine that made the request and the host with the most available bandwidth. The method for selecting the host from which to download the file can be easily altered by extending the File class and overriding the *selectBestHost* method. If there is no host for a particular requested file (which will be the case if it is produced as the output of a job yet to finish execution), and the request is not cancelled, it is recorded and processed as soon as a host is available. It is important for the user to ensure that files which are

required by jobs to begin execution at the start of simulation are hosted on at least one machine, or they will not be able to run.

3.7 User and Application Model

The users of a Grid environment utilise the resources offered by the Grid for some purpose. This purpose is usually to calculate an answer to the question their application is asking. These applications will typically be complex, large-scale, data intensive entities which may involve several stage of execution. Grid applications are normally considered to be partitioned into smaller parts often referred to as jobs. This is done so as to parallelise its execution, allowing many resources to compute it simultaneously. The performance of Grid applications is difficult to judge since the major factors in determining it will differ from application to application and the presence of large number of factors involved including the speed of the processors running the jobs, the operating systems controlling them, the availability of resources and the transfer time of the files required to the resources involved. These applications are heterogeneous in nature and each is likely to differ greatly in requirements, scale and structure. The users themselves often have different requirements of the Grid. Some may require their results within a given deadline whilst some may be more interested in computing the answer for the minimum financial cost.

Clearly the complex and varied nature of Grid applications means no single model will be suitable for all applications. In response to this, G-Sim's application model provides only primitives for creating an application, namely jobs and files, but, is designed so as to let the user define the application model of their choice. However simple utility methods are implemented to create applications consisting of numbers of independent jobs, which neither require nor output any files, of randomly generated sizes within a given range, a model which is constant with the parameter sweep applications as experimented with in [BU2002]. In G-Sim user's requirements can be modelled by giving each user their own personalised scheduler and configuring it to suit. Unlike GridSim however, G-Sims resource model would need to be extended to implement user's economic requirements as it is not designed towards scheduling

according to an economic model. Application level schedulers can be modelled by setting each user to have their own personalised scheduler which is designed specifically for the application they submit to it.

As previously mentioned jobs differ in size, given as the number of instructions needed to process them, making them CPU intensive, and in the files they require as input or output. As files are uniquely identifiable, it is possible to build applications based structured as DAGs of tasks, such as those investigated in [BL2004] where the output files from some jobs are required to run others. This is common in real Grid applications where the initial tasks within an application may apply a variety of filters a large data set before it is analysed by subsequent tasks. As a job's size is measured solely on the number of instructions and machines rate of execution is based on the number of instructions its processors can process per second, the G-Sim model greatly simplifies how application performance is calculated. This is done to make execution faster and to make it simple to predict exactly how long the applications will take to complete with a given resource model. Being able to predict the length of each job's execution is useful because it allows scheduling algorithm designers to gauge the effectiveness of their procedures under different levels of performance prediction accuracy (knowing the exact value means you can then introduce a controlled level of inaccuracy to see the effect it has on the algorithm's performance), a technique used in experiments presented in [XI2003]. Real Grid schedulers have to employ often complex predicting algorithms of which the best have been shown to have an inaccuracy of around 10% [XI2003], G-Sim's simplistic model is also advantageous in that is that it removes the need for the user to implement any such procedures to get their algorithms to work effectively. In relation to the base Grid simulation model described in [BR2001], G-Sim's schedulers can be considered to schedule on a number of consistent 'expected time to compute' matrices, between jobs and resources, over the course of an experiment.

To simulate the varying requirements of applications, tasks and resources (represented by the Job and Machine classes respectively) in G-Sim can be associated with a QoS object. These objects represent either the quality of service requirements of the job and that offered by the machine. As quality of service is an abstract concept, the QoS

class is declared as a simple interface, the concrete implementation of this object is left to the user.

In a Grid, users can submit their applications to Grid schedulers at anytime. G-Sim offers a utility class, the GroupSchedulingEvent (GSE) class, to model this behaviour. This class uses a similar technique to the simulator developed in [XI2003] and controls when users submit their jobs to schedulers based on a Poisson distribution with a given mean rate per second. This is implemented using an exponential distribution function to generate the job inter-arrival times. The exponential function is the only memory less random distribution and describes the waiting times between successive changes in a given Poisson distribution [WEI1999]; its probability distribution function is given in equation 3.9. This function is adapted to give the function *f* given in equation 3.10 so as to produce exponentially distributed random variables from evenly distributed random variables.

$$P(x) = D'(x) = \lambda e^{-\lambda x}$$

Equation 3.9: Probability distribution function of the exponential distribution, λ is the rate of change in the Poisson distribution [WEI1999]

$$f = \frac{-\ln(1-r)}{\lambda}$$

Equation 3.10: Function for creating exponentially distributed random variables from evenly distributed random variables, where λ is the rate of change in the original Poisson distribution and *r* is the evenly distributed random variable

The function given in equation 3.10 is implemented in G-Sim's Utilities class, its output it demonstrated in figure 3.16.

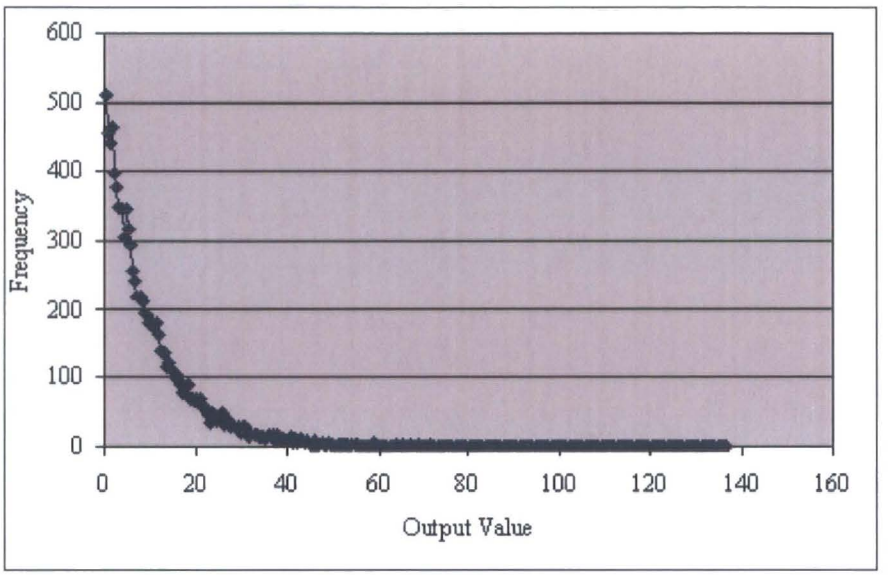

Figure 3.16: The output of the function implementing equation 3.10 over 10000 runs, with the rate of change given as 10. The frequency represents the number of times the output was recorded in the range between the graphed value +/- 0.25

Although the job model presented in G-Sim is relatively simple, it is envisaged that all the models given in G-Sim could be extended to create different behaviour, allowing for all types of application model to be experimented with. For instance I/O intensive applications could be modelled by extending the Machine class to have a field describing the number of I/O operations they can make per second and define Jobs with a number of I/O operation required as opposed to CPU cycles. Similar methods to those seen in the PE class could then be implemented to control how the machine's operations are split between the I/O intensive jobs it is running. It should be remembered that the toolkit is simply a base on which to build scheduling experiments; such experiments may involve manipulating the tool in ways unforeseen by the toolkit designer.

3.8 Summary

In a dynamic Grid environment it is hard, if not impossible to perform scheduling experiments that are repeatable and controllable. Grid scheduling algorithm designers require such experiments to be run in order to verify the effectiveness of their procedures. To overcome this problem the G-Sim toolkit has been created; the design

and implementation is the subject of this chapter. This software provides the primitives for creating virtual Grids with any number of heterogeneous Grid resources running under either time or space shared allocation policies with various local load models, users and their applications and the network under which the distributed Grid entities communicate.

The suitability of the G-Sim toolkit is demonstrated in chapter 4, where it is used to evaluate the performance of newly developed procedures against known ones under numerous different scenarios.

Chapter 4 – Time-Limit Batch Mode Scheduling

4.1 Introduction

Grid scheduling is the mapping of tasks to distributed heterogeneous networked resources. The optimal mapping, in anything but the most trivial cases, is impossible to find due to the vast number of possible alternatives which may need to be searched. Grid scheduling algorithm designers look to produce heuristic based procedures which will produce effective mapping which operate within a reasonable time frame. Such procedures operate in one of two ways; in online or batch mode. Online procedures are those which map jobs to resources as they arrive at the scheduler, batch mode procedures wait until a number of seconds has passed or a number of jobs have arrived then schedules them together. Generally waiting a given number of seconds is more effective since job flow is unpredictable, it prevents the possibility of jobs waiting for long periods after being sent to the scheduler to start execution. This chapter presents and a novel mode of scheduling, time-limit batch mode, which looks to provide better foresight and reactivity than batch mode whilst minimising the effects resource error has on application performance.

4.2 Problems with Current Schedulers

It has been noted from the birth of the Grid paradigm that resource failure is a rule not the exception [FO1998]. As Grids have many distributed resources, the chance of total system failure is extremely minimal, one of the great advantages of the paradigm, yet, the chance of a single resource failing is high. Rescheduling has emerged as an essential tool for not only lowering the makespan of schedules, but also ensuring the eventual completion of tasks which are sent to resources which fail or become inactive [GO2002]. Rescheduling features have already been incorporated into current Grid resource management systems such as Condor-G.

Rescheduling on a Grid tasks takes time. The often large input files required have to be transferred to a new resource and previously calculated results may be lost. The problem with batch mode schedulers is that they are not reactive since they schedule many jobs at the same time and are unable to predict which resources will incur increased local loads or fail whilst processing these. Batch mode's inability to react to future changes means that the effect of resource failure or increased workload is maximised.

4.3 Time-Limit Batch Mode

Time-limit batch mode scheduling is a technique which can be applied to batch mode schedulers to improve their performance. The idea is simple; don't give resources more jobs than are necessary to keep them running. By putting off scheduling decisions until later, and so limiting the number of jobs which are actually run on hosts at any one time, the number of reschedules per resource failure is minimised and the makespan of the schedule can be improved. To ensure the effective utilisation of resources, the scheduler must map at least enough tasks to ensure every resource in its pool is utilised until it is due to schedule again, if possible. Whereas in online or batch mode scheduling each resource may have many tasks sent to it that are guaranteed not to be completed for a long period, so in the event of a failure many jobs will have to be rescheduling, requiring more downloads and ultimately resulting in inefficient execution, time-limit batch mode ensures this is kept to a minimum.

The remainder of this chapter looks at the effects of incorporating time-limit batch mode into schedulers and how this affected by the state of the underlying Grid.

4.4 Developed Algorithms

To test the performance of time-limit scheduling, new algorithms which use it need to be developed and tested. The algorithms chosen to be adapted to incorporate time-limit scheduling in this project are the well known Min-Min heuristic and its QoS guided variant (QGMM). Min-Min was selected as it is a well documented algorithm and so any difference between it and its time-limit equivalent can be gauged in

relation to the other algorithms it completes against in works such as [BR2001]. QGMM was also included as it allows for similar experiments to be conducted as those published in [XI2003], verifying their results and providing a basis of comparison between the newly developed algorithms and their documented counterparts.

The QGMM heuristic was developed in [XI2003] to show that Grid scheduling could be made more effective if QoS was made a major factor in determining the mapping between jobs and resources. The algorithm is essentially the same as the Min-Min heuristic, discussed in section 2.4.2.3, where high level QoS jobs are scheduled before low ones. The QoS model considered in [XI2003] is hard with only one dimensional, differentiated into just two distinct levels; high or low. The pseudo code for the QGMM heuristic is given in figure 4.1.

(1) for all tasks t_i in meta-task M_v (in an arbitrary order)
(2) for all hosts m_j (in a fixed arbitrary order)
(3) $CT_{ij} = ET_{ij} + d_j$
(4) do until all tasks with high QoS request in M_v are mapped
(5) for each task with high QoS in M_v, find a host in the QoS qualified host set that obtains the earliest completion time
(6) find the task t_k with the minimum earliest completion time
(7) assign task t_k to the host m_l that gives it the earliest completion time
(8) delete task t_k from M_v
(9) update d_l
(10) update CT_{il} for all i
(11)end do
(12) do until all tasks with low QoS request in M_v are mapped
(13) for each task in M_v find the earliest completion time and the corresponding host
(14) find the task t_k with the minimum earliest completion time
(15) assign task t_k to the host m_l that gives it the earliest completion time
(16) delete task t_k from M_v
(17) update d_l
(18) update CT_{il} for all i
(19)end do

Figure 4.1: The QoS Guided Min-Min Heuristic [XI2003]. CT_{ij} refers to the estimated completion time of task t_i on machine m_j, ET_{ij} to the estimated execution time, d is the time delay to complete the tasks already allocated

The adaptation of the QGMM and Min-Min algorithm to use time-limit batch mode requires only minor changes. A check is added to cease scheduling once all resources have been given enough tasks to ensure they are all utilised until the scheduler runs

again. The pseudo code for the adapted algorithms is given in figures 4.2 and 4.3. It should be noted that implementation of the TLQMM algorithm, is contrary to what one might expect. The first do loop, starting on line 11, does not exit until all resources have a delay time of greater than the scheduling wait time or all the high QoS jobs have been mapped. Doing this will no doubt decrease its ability to cope with resource failure since many high QoS jobs will be mapped to resources before the low QoS machines see any increase in delay time. However, the decision to use this implementation over the alternative, where this first do loop can exit when just the high QoS machines are occupied until the next run, was made because preliminary experiments showed it suffered from greatly increased makespans. This was the effect of low QoS jobs being scheduled on the high QoS machines after the first loop had exited. When the algorithm ran again, the low QoS jobs already scheduled on these resources prevented the high QoS jobs accessing them, decreasing the performance.

The other slight alteration made between the QGMM and TLQMM heuristics was to set CTij to inifinity if the host can not match the QoS requests of the task. This was simply done to relieve the need for a later check within the first loop and makes it easier to adapt it to Min-Min; it has no impact on the operation of the algorithm.

```
1. for all tasks i in meta-task M
2.         for all hosts j
3.                 if j can service i's QoS request
4.                         CTij = ETij + dj
5.                 end if
6.                 else
7.                         CTij = infinity
8.                 end else
9.         end for
10. end for
11. do until all tasks in M are mapped
12.         find minimum delay time d
13.         if d > schedulingWaitTime
14.                 return
15.         end if
16.         find task t with the minimum CTij
17.         map task t to machine l which gives it the earliest completion time
18.         delete t from M
19.         update dl
20.         update CTil for all i
21. end do
```

Figure 4.2: The Time-Limit Min-Min (TLMM) Algorithm

```
1. for all tasks i in meta-task M
2.       for all hosts j
3.                if j can service i's QoS request
4.                         CTij = ETij + dj
5.                end if
6.                else
7.                         CTij = infinity
8.                end else
9.       end for
10. end for
11. do until all tasks with high QoS requests in M are mapped
12.       find minimum delay time d
13.       if d > schedulingWaitTime
14.                return
15.       end if
16.       find high QoS task t with the minimum CTij
17.       map task t to machine l which gives it the earliest completion time
18.       delete t from M
19.       update dl
20.       update CTil for all i
21. end do
22. do until all tasks with low QoS requests in M are mapped
23.       find minimum delay time d
24.       if d > schedulingWaitTime
25.                return
26.       end if
27.       find low QoS task t with the minimum CTij
28.       map task t to machine l which gives it the earliest completion time
29.       delete t from M
30.       update dl.
31.       update CTil for all i
32. end do
```

Figure 4.3: The Time-Limit QoS Guided Min-Min (TQGMM) Algorithm

4.5 Experiments

To see how effective time-limit batch mode is, if at all, scientific experiments must be conducted on it. The section describes, and presents the results of a number of experiments which test the performance of the newly developed time-limit variant algorithms against known ones under varying conditions. The G-Sim toolkit is used to simulate the test bed Grid environments and scheduling scenarios.

The TLMM and TLQGMM procedures were tested against the original Min-Min and QGMM as well as the MCT heuristic discussed in section 2.4.2.3. This allows us to see the improvement made by introducing time-limit batch mode and also to judge the

new algorithm's performance against a well known online procedure, acting as a bench mark. The results cover numerous different Grid scenarios in an attempt to discover which factors affect Time-Limit batch mode and hence establish when the method will prove most effective and whether it is sometimes unsuitable.

The factors used to evaluate the change in performance provided by time-limit batch mode were the makespan, to evaluate the quality of the mappings produced, and the average number of jobs on resources at any time, used to judge the effectiveness of the procedures to cope with resource failure. The later metric was chosen as it represents the average number of reschedules which would be required in the event of a single host failing. It is calculated on each resource by recording the length of time each number of jobs appeared on it, multiplying it by that number and then dividing this by the length of time between the resource first receiving a task and all the tasks finishing. The final value given is a mean average of these values across all the resources. Where it is not clear, the significance of the difference between the algorithm's performances is demonstrated using Student's t-test, presented in equation 4.1. This metric is used in conjunction with the table giving the percentage points of the t-distribution presented in [MI1983] to give the probability of the difference between the two algorithms' performances being significant. If the probability of the difference between the two sets being caused by random error is less than or equal to the default alpha value of 0.05, the performance between two algorithms are considered to be significantly different.

$$t = \frac{\bar{x}_t - \bar{x}_c}{\sqrt{\frac{\mathrm{var}_t}{n_t} + \frac{\mathrm{var}_c}{n_c}}}$$

Equation 4.1: Formula for the t-test [MI1983], $\bar{x}_t$ and $\bar{x}_c$ refers to the mean values in the sets, var_t and var_c are the variances and n_t and n_c are the number of elements in the sets.

The variance of the sets is calculated using the equation 4.2 given below. The mean value is calculated using equation 3.1.

$$S^2 = \frac{\sum (X - \bar{X})^2}{n}$$

Equation 4.2: Formula for calculating the variance of a set of numbers [MI1983]. $\overline{X}$ refers to the mean value, X is the individual value and n is the number of elements in the set.

4.5.1 Grid and Application Model

The applications considered in these experiments are based on a parameter sweep model where the required input file size is considered negligible. These applications consist of a given number of independent jobs which differ only in the number of instructions required to complete them and the level of QoS they require. This QoS model is presented in [XI2003] and can be either high or low and must be equalled by a resource in order to execute it. The QoS is defined simply as an abstract term but could be used to represent constraints on any factor such as the required bandwidth, operating system, average local usage statistics, memory capacity or presence of specific hardware. This model was used as it is simple, comparable to some real Grid applications and has been used for other documented experiments.

The test-bed virtual Grids considered in these experiments are composed of a number of G-Sim machines which differ in whether they were dedicated or non-dedicated, the number and speed of the processors they have and whether they have a time or space shared allocation policy. These hosts are considered to have no other remote jobs to process at the start of each experiment. The local load on the resources is subject to change over simulated time and random variation. Since the Grid applications are not considered to have input files, a network topology is not defined and resources are considered to communicate over a network with infinite bandwidth and no propagation delay.

In each experiment there is one Grid scheduler, operating using the given algorithm, to which a specified number of users each submit a single application of the fore

mentioned nature. Each scheduler waits for a number of seconds between when it activates, sending the remote tasks presented to it to the test-bed Grid, this period is called the scheduling wait time. G-Sim's *GroupSchedulingEvent* class, as described in section 3.7, is used to calculate the times between the users' submitting their applications and is calculated based upon a Poisson distribution with a given average number per time period. As with all Grid simulation, the remote, memory limited and possibly non-dedicated nature of the machine running this scheduler is abstracted out of the experiment as it is unlikely to be a significant factor and simplifies the model. The scheduler is considered as an abstract entity with infinite memory and execution speed, possessing the ability to communicate instantly with both hosts and users.

4.5.2 Initial Tests

In the initial tests, a single scheduling entity was used to map jobs to a Grid consisting of a number of heterogeneous dedicated single-processor space-shared resources. This particular Grid model was chosen as it allows the schedulers to be tested when the estimated completion times were 100% accurate. These estimated completion times are calculated using equation 4.3, where the time to complete the currently allocated tasks on resource j, $DELAY_j$, is initiated to 0 (as it is considered to be idle at the start of simulation), and is continually updated when jobs are allocated to it and completed by it.

$$CT_{ij} = \frac{MIi}{MIPSj} + DELAY_j$$

Equation 4.3: Calculation of estimated completion times in initial experiments

Although this Grid model may not be wholly representative of many real Grids it provides a controllable environment from which to investigate the effect various factors alone have on the performance of time-limit Grid scheduling.

In each set of tests, each value presented represents a mean average taken over 100 runs. This number was chosen as it represents a middle ground between the factors of execution time and the significance of the results (i.e. each experiment takes time to

compute but the more results, the less chance any documented change in performance being down to chance). To make the experiments fair, each algorithm was tested using the same sets of resources and applications.

4.5.2.1 Effect of QoS Requests

QoS requests have a considerable effect on the performance of task scheduling [XI2003]. To identify how differing QoS requests affect the performance of the algorithms, experiments were performed where the percentage of jobs with high QoS is altered.

In these experiments a small Grid of 10 machines was used, a similar number to the experiments described in [MU2002] and [XI2003], where half offered high quality of service. The speed of the processor of each machine was randomly set to between 100 and 1000 MIPS. The schedulers wait times were each set to 10 seconds and 100 users submitted their application to the scheduler on average every 10 seconds based upon a Poisson distribution. Each application consisted of 10 independent tasks and the size of each was randomly generated between 100000 and 200000 MI. The QoS requirements for each application were randomly generated with each having the given percentage chance of being high. As these algorithms had not previously been investigated, all these values were simply made-up, based knowledge of real applications and Grids, since there is no definite basis on how best to select values for these parameters. Figure 4.4 gives the results, where the makespan is given in seconds.

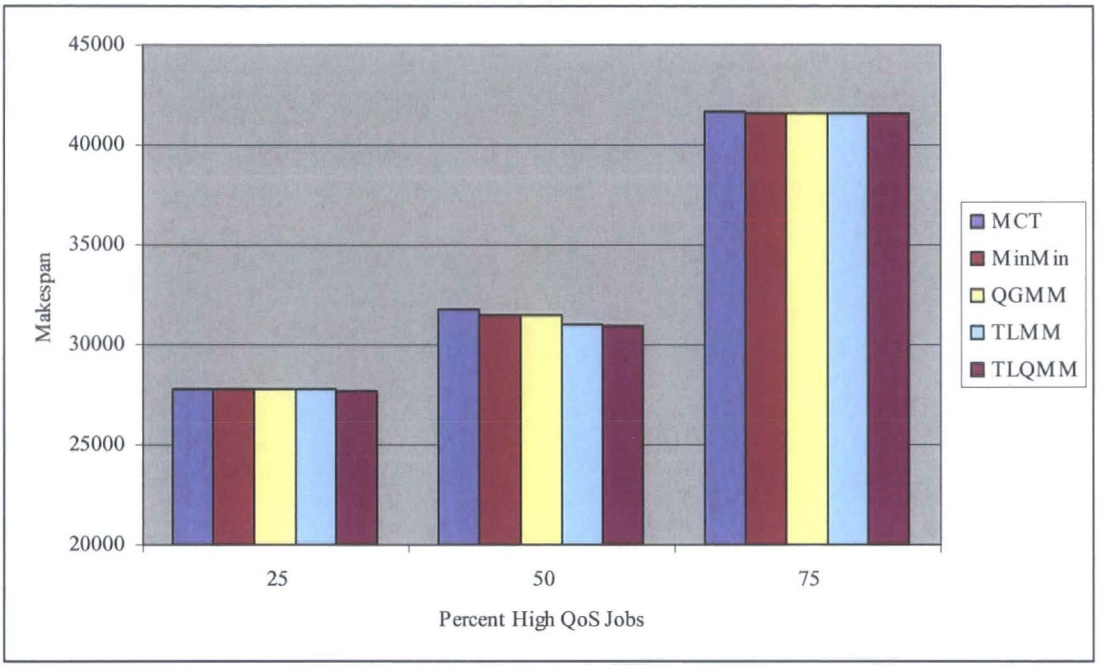

Figure 4.4: Makespan of the algorithms with a varying percentage of high QoS jobs

The results show that raising the percentage of high QoS jobs greatly increases the total makespan. This is due to the high QoS machines being utilised for longer as they are the only ones capable of processing the high QoS jobs. This trend was also found with the results given in [XI2003]. However it must be noted that there appears to be little difference between the makespans of each of the algorithms, the only visible difference occurring when 50% of the jobs have high QoS requirements. To highlight this variation in the makspans of the algorithms, figure 4.5 shows the percentage decrease in makspan over the MCT that the other algorithm had. The minimal variation in the makspans could be down to the selected values for the parameters discussed above; under other scenarios it is likely that the difference in the performance of the algorithms would be more significant.

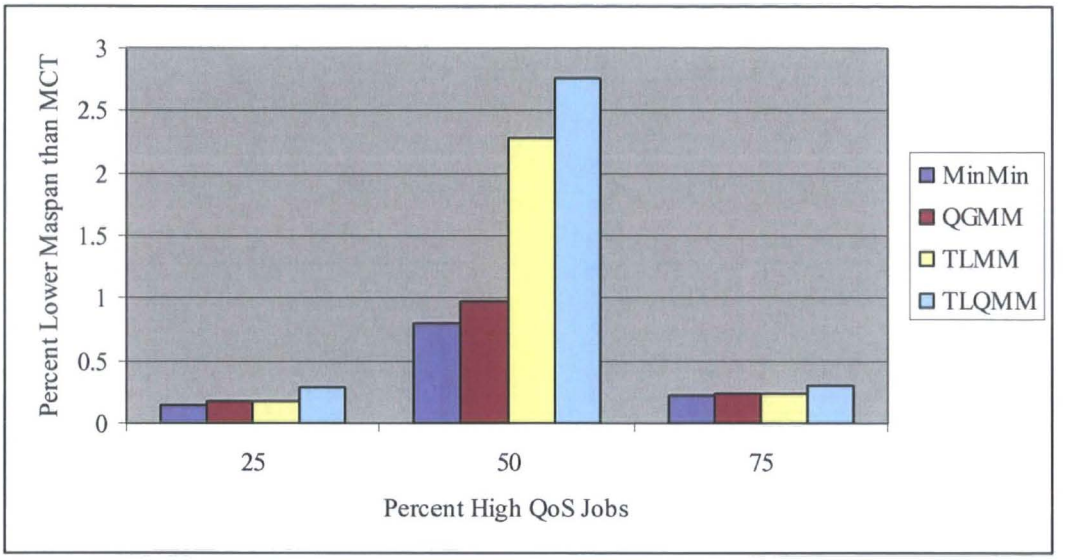

Figure 4.5: The percentage of high QoS jobs against the decrease in makspan over the MCT algorithm

Figure 4.5 shows that the increase in performance of the other algorithms over MCT is most prominent when half of the jobs require high QoS. Indeed the t-test results given in table 4.1 verify that the only significant differences between the algorithms performance appear in this scenario. It also shows that by incorporating time-limit batch mode, the makespan of both the Min-Min and QGMM algorithms was lowered, by up to 1.8%; a performance increase shown to be significant in table 4.1. The results concur with [BR2001] that Min-Min outperforms MCT, although the t-value between the two algorithm's makespans indicate there is a slightly greater than 5% chance that the observed difference is the effect of random variation.

Although the results do show the biggest performance increase between Min-Min and QGMM occurring when half the jobs require high QoS, as documented in [XI2003], they do not agree on the level of performance increase, which [XI2003] finds to be as much as a 11.41% reduction in makepsans, these experiments show it to be only at most 0.17%, an insignificant amount according to the t-test results given in table 4.1. The differences in result are likely to be the result of changes to the values used for the various factors and show that in some cases incorporating a QoS guided approach to scheduling will not improve on performance.

	25%			50%			75%		
	t value against MCT	t value against Min-Min	t value against QGMM	t value against MCT	t value against Min-Min	t value against QGMM	t value against MCT	t value against Min-Min	t value against QGMM
Min-Min	0.334			1.620			0.400		
QGMM	0.388	0.054		**1.968**	0.349		0.444	0.044	
TLMM	0.386	0.054		**4.577**	**2.966**		0.445	0.047	
TLQMM	0.642		0.254	**5.540**		**3.574**	0.560		0.117

Table 4.1: *t* values for algorithm's makspans. The bold values indicate that they are greater than 1.96 and hence considered statistically significant assuming an infinite number of samples and an alpha value of 0.05

4.5.2.2 Effect of Scheduling Frequency

The scheduling frequency affects all the performance of batch mode Grid schedulers and must be carefully chosen when applying such a procedure to a Grid environment. Setting it close to 0 typically decreases its effectiveness as it becomes more like an online scheduler with minimal look-ahead, as the parameter gets larger, the more jobs the scheduler has knowledge of, so potentially the better the mapping it can create, but, this must be traded off against a possible decrease in resource utilisation and the fact that more jobs will be given to resources at one time reducing its reactivity to errors. Such a decrease will occur if the time between scheduler runs is larger than the runtime of the tasks given to each resource. This parameter could be configured dynamically, on an event-based scheme, dependant on job flow to the scheduler or the state of the Grid or simply set to a constant, static value. How best to decide the value of this variable is still a subject of research [XI2003].

To investigate the effects of the scheduling frequency on the performance on time-limit batch mode scheduling experiments were carried out using differing values for the parameter. To keep consistency between the experiments, the same Grid scenario as in section 4.5.2.1 was used where the percentage of high QoS jobs was set to 50%. Figure 4.6 shows the results.

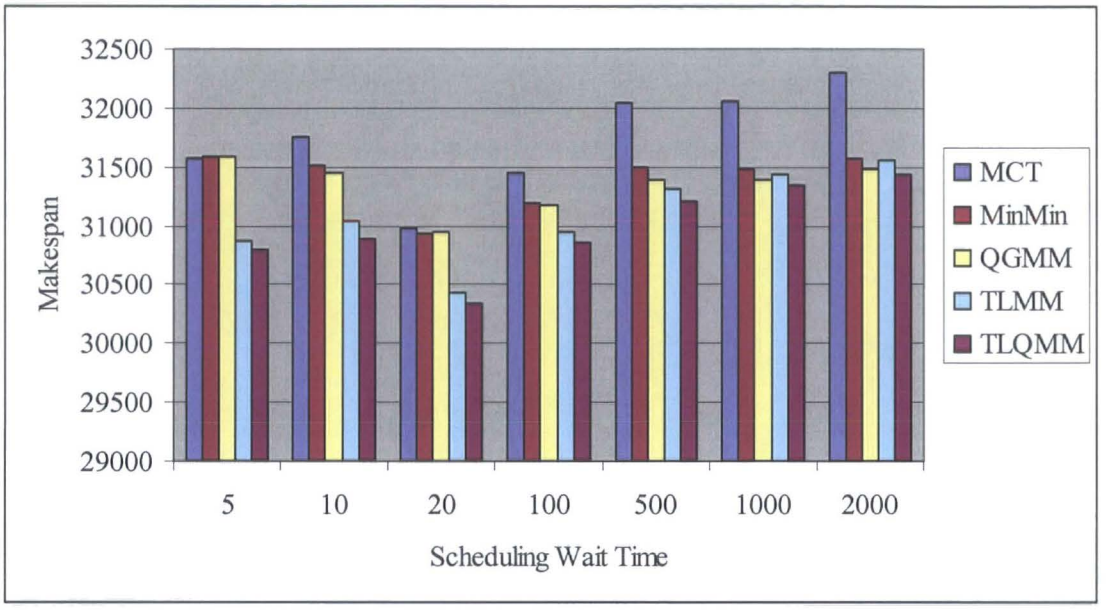

Figure 4.6: Makespan of the algorithms with a varying scheduling wait time

The decrease in the makepsan shown by the MCT between when the scheduling wait time is 10 to 20, a reduction of 783.27s, demonstrates the potent effect of the random variation on the parameters which define whether applications require high QoS, the time between users submitting their jobs, the size of the individual jobs and the speed of the machines have on the measured performance. The difference must be down to such random variation since MCT does not utilise any ability to look ahead, so increasing the time between scheduling should have no positive impact on its performance. Taking average results over far more than 100 runs is likely to alleviate the effect of this variation although this must be traded-off against the increase in execution time it would incur. Figure 4.7 shows the percentage decrease in execution time between other algorithms and MCT, a measure which is easier to interpret as it effected less by the random variation.

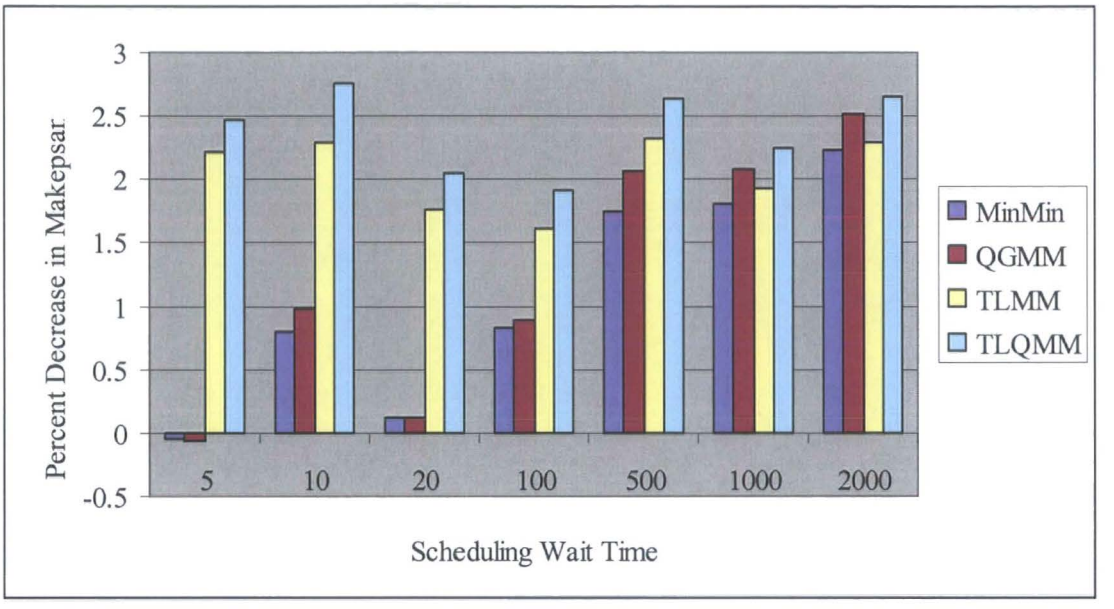

Figure 4.7: Percent decrease in makespan over the MCT algorithm on different scheduling wait times

The results show that the time-limit batch mode schedulers are less affected by the scheduling frequency than traditional batch mode schedulers, and that the rise in performance they provide decreases with the scheduling frequency. T-test values from the experiments, presented in table 4.2, verify that when the scheduling frequency is high, with only a wait time of 5 seconds between runs, TLMM and TLQMM significantly outperform the other algorithms, but, when the scheduling frequency is low, with a 2000 seconds wait, they only outperform MCT significantly.

	5 seconds			2000 seconds		
	t value against MCT	t against Min-Min	t value against QGMM	t value against MCT	t against Min-Min	t value against QGMM
Min-Min	0.075806			**3.865047**		
QGMM	0.107361	0.031555		**4.356207**	0.49116	
TLMM	**3.738025**	**3.813831**		**3.968068**	0.103021	
TLQMM	**4.174165**		**4.281526**	**4.579394**		0.223186

Table 4.2: *t* values for algorithm's makspans. The bold values indicate that they are greater than 1.96 and hence considered statistically significant assuming an infinite number of samples and an alpha value of 0.05

This decrease in performance enhancement is likely the cause of Min-Min and QGMM algorithms improving as the scheduling wait time increases. This has been shown to be the case with batch mode heuristics in a number of studies such as [XI2003] and [MA1999] due to the fact they have more information on the nature of the jobs they are required to schedule. It is access to this information which makes time-limit batch mode heuristics advantageous whatever the scheduling wait time, when the flow of tasks is sufficient to maintain utilisation across the resources. If this is the case, which it is in the presented experiments, the amount of task information they have access to increases since they hold back tasks from the hosts, while more jobs arrive. As the scheduling wait time increases, they become more like traditional batch mode schedulers as they are required to present more jobs to each resource in order to keep them utilised until the scheduler is run again, leaving them with fewer to hold back. The relationship between the increase in performance offered by incorporating time-limit mode scheduling and the scheduling wait time is shown in figure 4.8. This graph shows roughly linear performance degradation when the scheduling wait time scale is logarithmic, suggesting it is related to the log of the wait time and as such is only a minor factor in determining the performance difference.

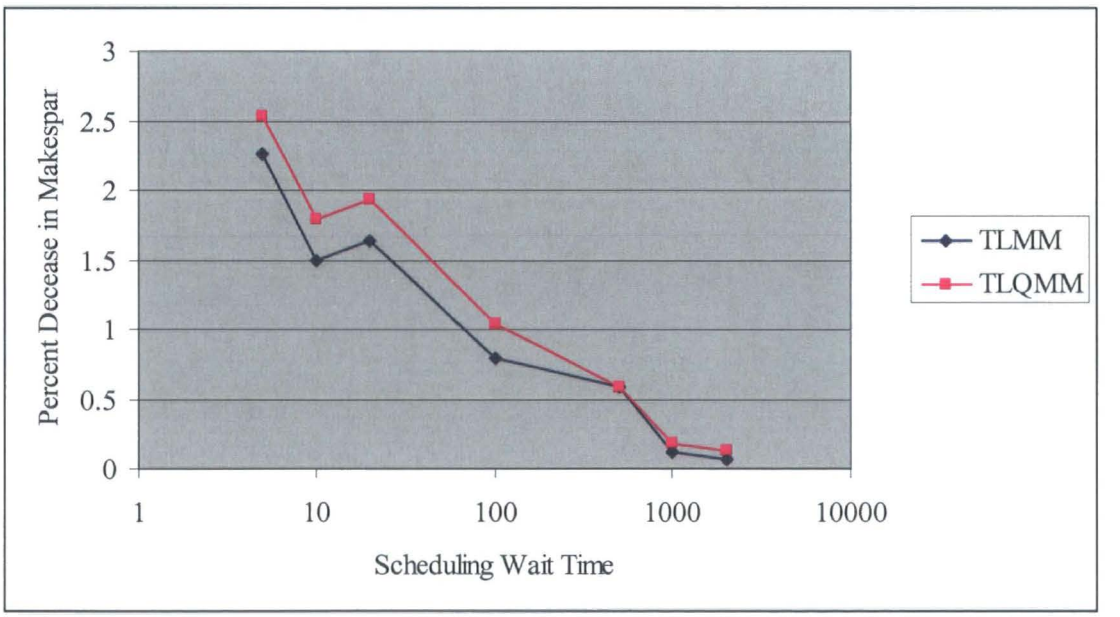

Figure 4.8: Percent decrease in makespan of time-limit batch mode schedulers over their batch mode equivalents using different scheduling wait times on a logarithmic scale

The scheduling wait time also impacts on the difference time-limit mode scheduling makes on the effect of resource failure. This is because the longer the wait time the more jobs that need to be scheduled on each machine in order to keep it occupied until the next iteration. The effect of scheduling wait time on the average numbers of jobs on each machine is demonstrated in figures 4.9 and 4.10. Figure 4.9 presents results using wait times up to 2000 seconds, as in the previous experiments. They show that MCT, Min-Min and QGMM all give similar recordings since they all schedule every job presented to them each time they run. Slight variations begin to occur between the results of MCT and the other two as the wait time increases; this is most likely the result of the improved schedules they provide. The results also show that TLQMM reduces the average number of jobs at each host by 42.02% over its batch mode equivalent and TLMM by 92.07% on average over the 7 different scheduling wait times. The difference between these two time-limit batch mode schedulers is down to the way in which they operate. TLMM simply gives out jobs until all the hosts' delay times are greater than the scheduling wait time, meaning that the average number of jobs at each resource is likely to be slightly larger than the number which it can process in the period until the scheduler runs again, depending on the heterogeneity of the hosts and job size. TLQMM on the other hand is likely to hand out many if not all high QoS jobs each time it schedules, since its first do loop does not end until all machines have a delay time of greater than the scheduling wait time or all high QoS jobs are mapped, even though it deals only with high QoS resources. The reason for this is discussed in section 4.4.

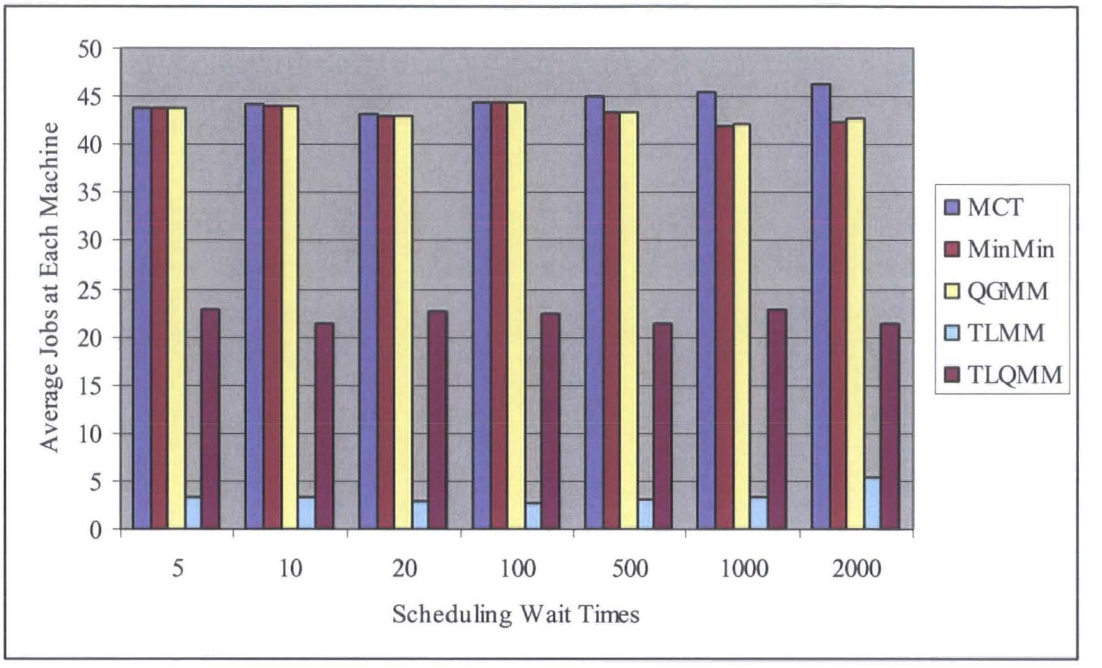

Figure 4.9: The average number of jobs at each resource under differing scheduling wait times

The result of scheduling wait times up to 2000 seconds, given in figure 4.9, show little variation from one wait time to another, and as such do not exhibit the anticipated pattern of increasing numbers of jobs per resource at each time with increasing wait time. To show this is the case more experiments with larger scheduling wait time values were carried out, the results of which are displayed in figure 4.10.

These results show that the scheduling wait time does indeed affect the number of jobs at each resource although the point at which the change becomes significant is large enough to make the effect of no practical significance in this scenario. This is because with a scheduling wait time of greater than around 100 seconds the makespans of the algorithms begin to increase, as shown in figure 4.6, so it is unlikely that a scheduling wait time of several thousand seconds, in which the effect on the number of jobs at each resource would be felt, would be selected.

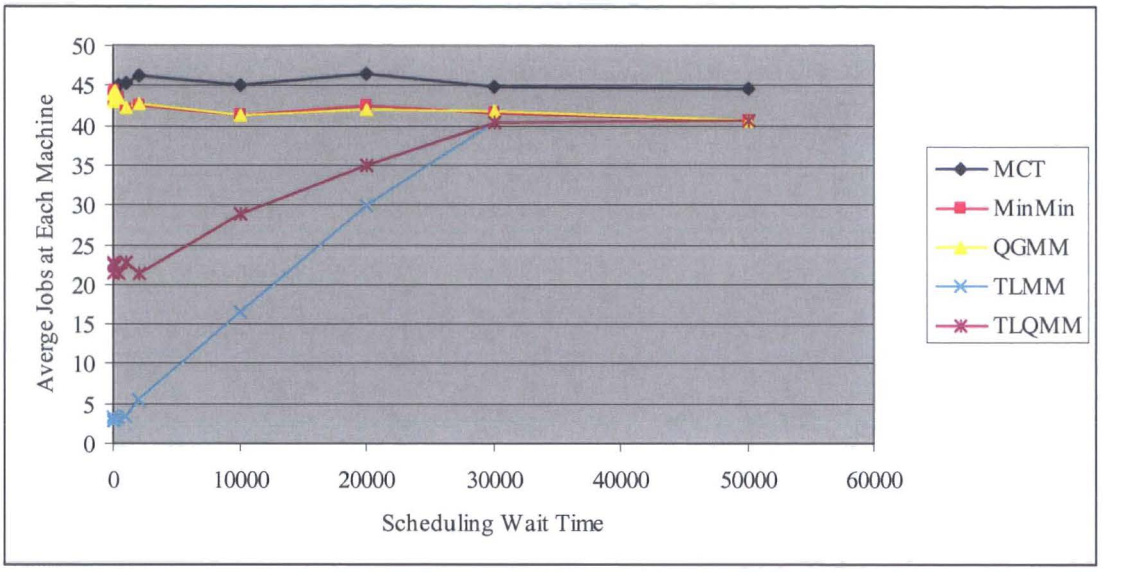

Figure 4.10: The average number of jobs at each resource under differing scheduling wait times

Besides this, the results in figure 4.10 are interesting as they show the traditional batch mode algorithms quickly diverting away from MCT, probably due to their improved mappings, but failing to make further reductions afterwards. This is because the when the scheduling wait time goes beyond 1000 seconds (the average length of time for all users to submit their applications in this scenario), the first time the schedulers run they will have the maximum possible look ahead, and so any additional time wait for jobs to arrive will not improve their performance. The results also show that both TLMM and TLQMM see a linear increase in the number of jobs at each host until they are close to intersecting the amount seen by their batch mode equivalents, which occurs just after 30000 seconds scheduling wait time, when they level out. This point is related to average makespan of the experiments minus the initial wait period (leaving the length of time the jobs are actually on the resources), as it is at this point that any increase in the scheduling wait time can not increase the average number of jobs at each resource since the time-limit schedulers are simply handing out all the tasks at once in the same way the other algorithms are. The makespan of the experiments, given in figure 4.6, indicate that this point is indeed around this area.

4.5.2.3 Effect of Grid Size

The size of the Grid on which the scheduler operates affects the overall makespan, since the more machines the more the jobs can be spread between them. It is also likely to affect the improvement made by using time-limit batch mode since more resources per job means fewer jobs per machine at any time and hence a deduction in the look-a-head the technique will bring. To investigate this, the algorithms were tested whilst operating of Grid on different sizes and under the same conditions as previously used, with the scheduling wait time set to 10 seconds. The results of these experiments are given in figure 4.11.

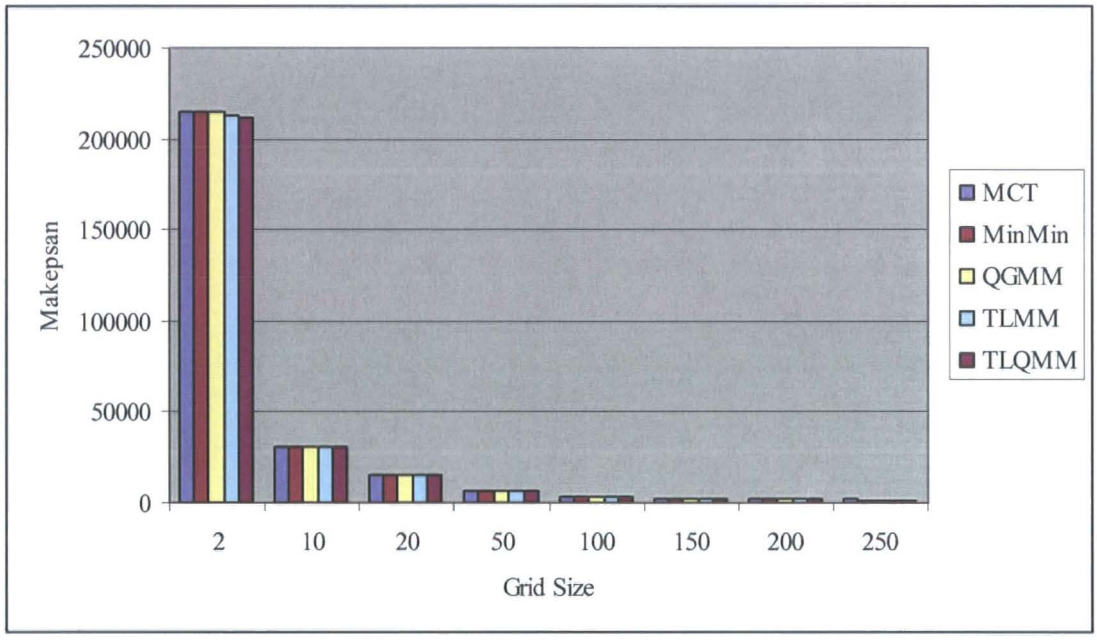

Figure 4.11: Makespan of the algorithms over increasing Grid size

As expected the makespans of the algorithms decreases greatly as the number of machines increases. Generally, the makespan can be considered inversely proportional to the total number of resources, a statement which is truer the more homogeneous these resources are. Figure 4.11 shows that no single algorithm offers a vastly numerically lower makespan over the other in any of the tested situations. On this scale the only visible differences occur when the Grid size set to 2 and 250. To give a better indication of the differences in the algorithms' performances, the

percentage decrease in makespan over MCT seen by the other algorithms is graphed in figure 4.12, and the t-test values between the sets of results given in table 4.3.

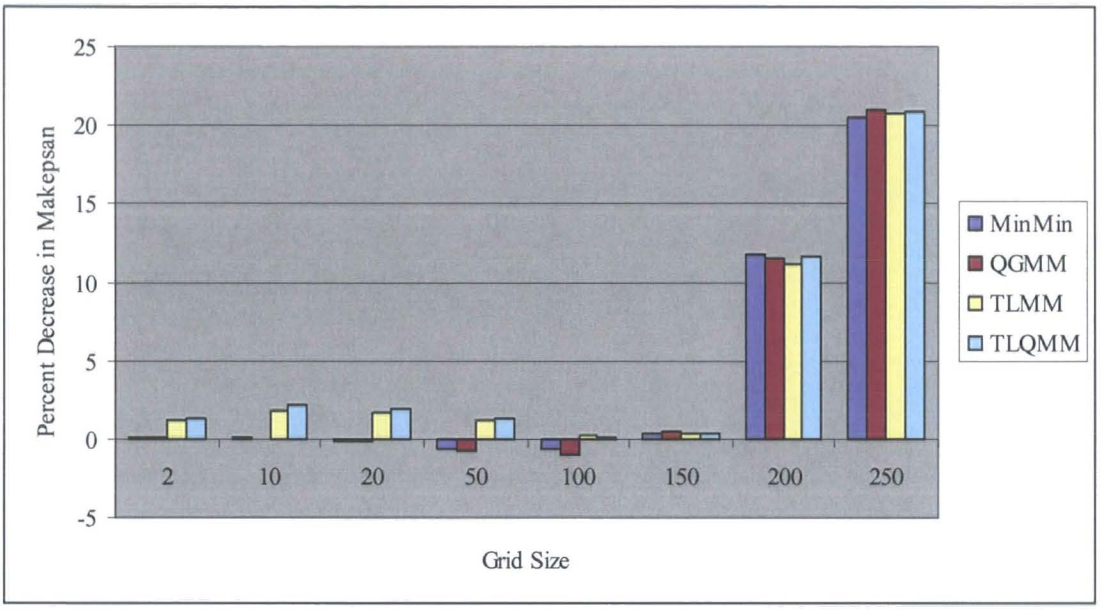

Figure 4.12: The percentage decrease in makespan of the other algorithms over MCT, against the number of resources in the test bed Grid

The percentage differences show that the improvement made by the time-limit batch mode algorithms is only apparent when the Grid size is low. The maximum improvement seen for TLMM over min-min, a 1.86% reduction in the makespan, occurred when the Grid size was set to 20 and TLQMM reduced the makepsan delivered by QGMM most with a Grid size of 10; by 2.09%. The decrease in performance change between the scheduling modes as the Grid size increase is expected since the more resources available, the more jobs the time-limit schedulers give out on each run and hence the less difference between them. Table 4.3, which shows the t-test values for the makespan results, verifies that when the Grid size is 100 or 250 machines, no significant differences between the two sets of algorithms was recorded, however it is apparent with only 10 machines. This decrease in the difference between the algorithm's performances as the Grid size increases maybe explained by the fact that the look-ahead gained by incorporating time-limit batch mode scheduling becomes less significant the more machines available, since each test considered an equal number of tasks. Figure 4.12 also shows a sudden and rapid increase in the performance of the other algorithms over MCT with large Grid sizes of

200 and 250 machines. Since only 1000 jobs are considered in the experiment, these results, are likely the cause of only minor changes in the mappings provided by the other algorithms over MCT, the effect of which are amplified by considering by percentage decrease in makespan metric.

	10 Machines			100 Machine			250 Machines		
	t value vs MCT	t vs Min-Min	t value vs QGMM	t value vs MCT	t vs Min-Min	t value vs QGMM	t value vs MCT	t vs Min-Min	t value vs QGMM
Min-Min	0.16			0.56			**10.23**		
QGMM	0.23	0.11		0.57			**10.76**	0.65	
TLMM	**3.20**	**2.94**		0.25	0.39		**10.33**	0.08	
TLQMM	**3.85**		**3.74**	0.08		0.66	**10.51**		0.20

Table 4.3: *t* values for algorithm's makspans under different Grid sizes. The bold values indicate that they are greater than 1.96 and hence considered statistically significant assuming an infinite number of samples and an alpha value of 0.05

As the Grid size increases, with only a limited number of tasks, the number allocated to each resource decreases. This means that the increased ability to react to a resource crash, introduced by time-limit batch mode scheduling, is reduced. Figure 4.13 shows the average number of jobs at each resource recorded for each algorithm under differing Grid sizes. These results are as one would expect given that, as with the makepsan, the average number of jobs on each resource can be considered inversely proportional to the total number of resources under the batch mode algorithms and therefore the improvement seen by time-limit batch mode is far greater the larger the ratio of jobs to machines.

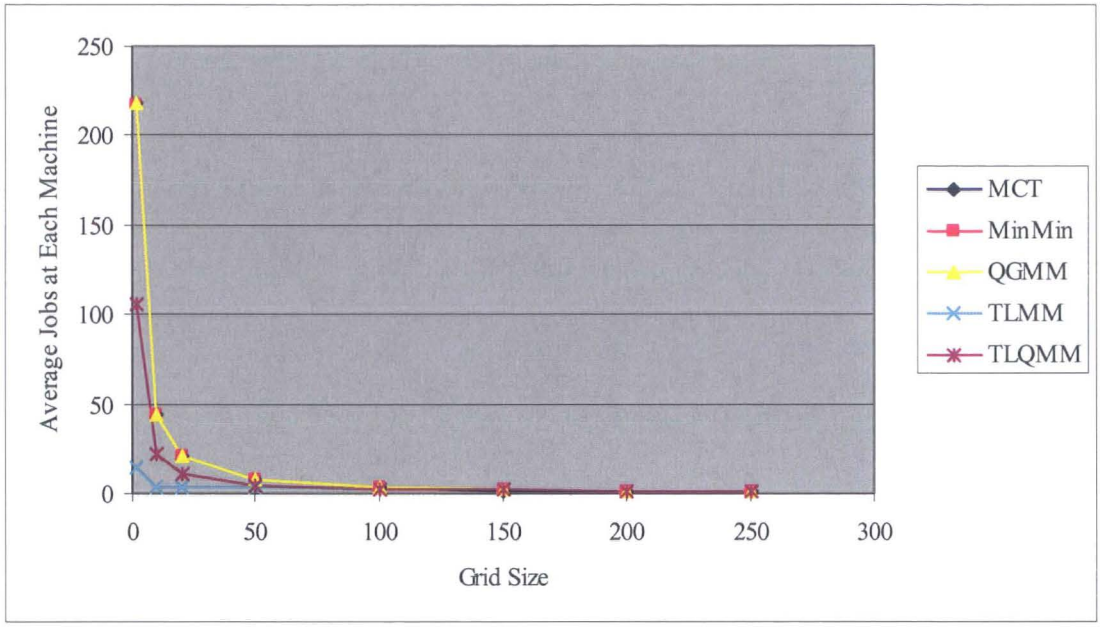

Figure 4.13: The average number of jobs at each resource against the number of resources

4.5.2.4 Effect of Grid Heterogeneity

The Grid heterogeneity in the context of these experiments refers to the differences in the processing speed of the resources. This factor, affects time-limit scheduling since the larger the difference in speed between the slowest and fastest machines, the more jobs will be scheduled on each run, as more jobs will be given to the faster machines before the slow ones have enough jobs to keep them utilised, and so the more it becomes like traditional batch mode scheduling. It should be noted that it is the ratio between the fastest and slowest machines rather than the actual difference in processor speed that is important. This is because this ratio defines roughly how many tasks the fastest machine will do in the time it takes the slowest to do one, and hence the larger this number the larger the average number of jobs on each machine at any time under a time-limit batch mode scheduler.

To investigate it affect on time-limit batch mode scheduling, the algorithms were first tested operating on a Grid where each host's processor rating varied randomly between 100 and a given maximum number of MIPS, up to 10000. The same scenario as described in section 4.5.2.1 was used, where 50% of applications required

high QoS. The results are given in figures 4.14 and 4.15. These experiments consider some extreme cases since the fastest machines in a real Grid are unlikely to be 100 times faster than the slowest; however the speed variation in future Grids are likely to be far higher. This is because the computers of the future are likely to be exponentially faster than those currently available, if Moore's Law [IN2005] continues to be true, yet it is possible that the currently available resources will still be available. Indeed P2P computing already deals with high variation in resource capability since home PC's used to run load sharing programs such as Kazaa [KA2005] and SETI@Home [SE2003] are compatible with low end machines with processors varying from 486DX's running at 66 MHz with a MIPS rating of 54 [WI2005] to a 3.6 GHz Intel Pentium 4 which has a MIPS rating of 10,224 [TH2005].

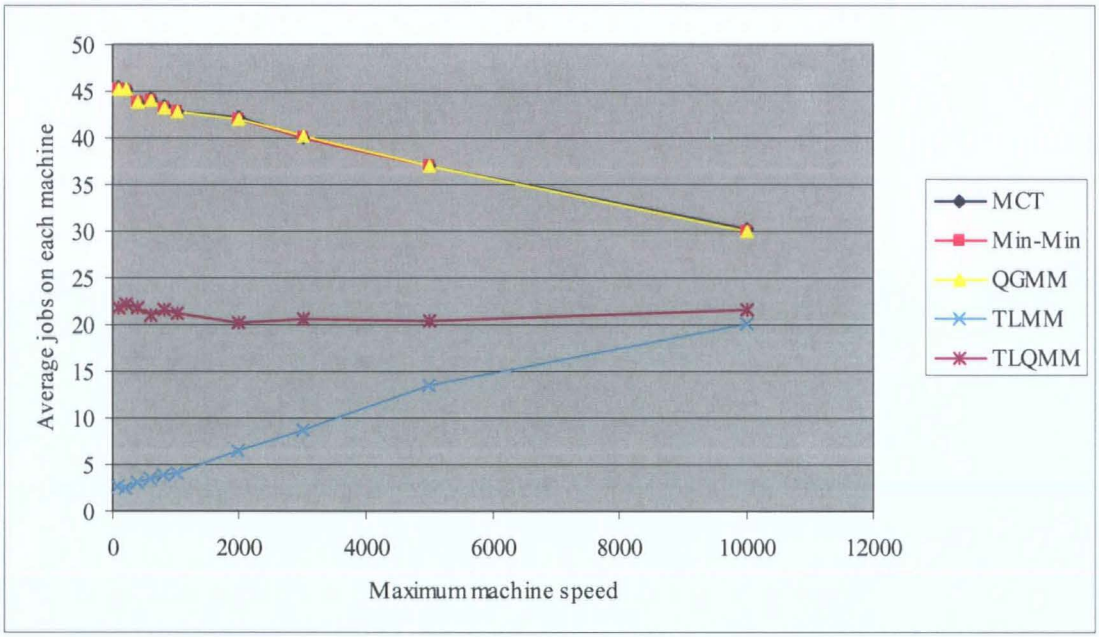

Figure 4.14: The average number of tasks on each host under different degrees of Grid heterogeneity. The maximum resource speed is given in MIPS.

We first explore the affect of Grid heterogeneity on the average number of jobs allocated to each machine at any time, the relevant result are given above in figure 4.14. As expected, the three non time-limit batch mode heuristics always produced similar results as they all operate by releasing all the jobs sent to them each time they schedule. However, one might assume that these algorithms would be unaffected by an increase in resource speed variation, this is not the case. These algorithms show a

constant decrease in the average number of jobs at each resource as the maximum speed increases. This decrease is the effect of fewer, much faster resources taking a larger proportion of the jobs, and completing them before a large number of slower machines with just a few or one task, as this is the case for a period the average number of tasks time recorded is lower as the difference between machine speeds increases. The results show that in this situation, the heterogeneity of the Grid has little effect on the average number of jobs on each machine under the TLQMM heuristic. This procedure outperformed the batch mode heuristics by at least 50.08%, in a homogeneous environment, but fails to maintain this increase in performance as the Grid more away from this state. A different pattern appears to emerge from the TLMM algorithm, which saw a steady increase in the average number of jobs on a machine as the Grid become more heterogeneous. It gave at least 93.65% few jobs on each machine than the traditional batch mode algorithms under a homogeneous Grid, this different was reduced to just 34.02% in the most heterogeneous situation tested. The decrease in performance seen by the TLMM algorithm can be explained by the fact that this algorithm is affected differently by an increase in Grid heterogeneity; as differences between the hosts appear, it presents the faster ones with more tasks, and the same number as before to the slower ones, causing an increase in the average. It is likely that the TLQMM algorithm stays relatively unchanged since the two factors affecting the decrease in performance of the TLMM procedure and the increase of the other batch-mode heuristics balance each other out.

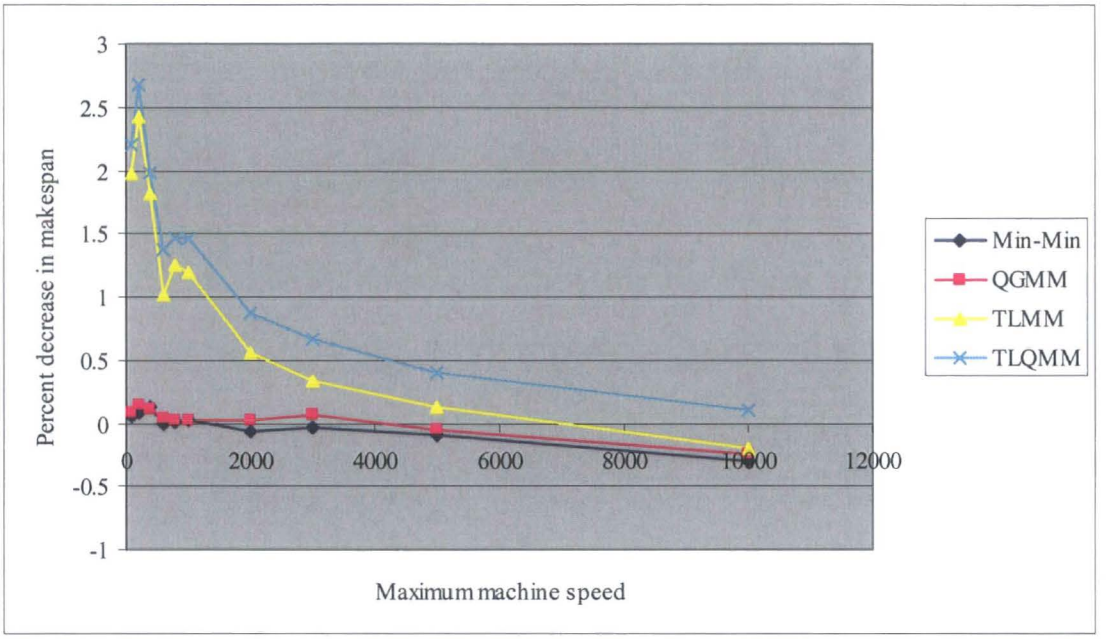

Figure 4.15: The percentage decrease in makespan over MCT under differing degrees of Grid heterogeneity

The heterogeneity of the Grid also affects the performance of the algorithms in terms of the makespans of the schedules they produce. Figure 4.15 shows the decrease in makespan over MCT that the other algorithms enjoyed against the maximum possible machine speed in the Grid. The graph indicates that in general greater the difference in the speed of the resources in a Grid the less the improvement in performance over MCT the other algorithms have. Contrary to this is the fact that the most significant improvements for all the algorithms over MCT occurred when the maximum machine speeds were set to 200 MIPS, rather than when the Grid was totally homogeneous. However the t-test values between the results of each algorithm in both scenarios was at most 1.29 (with TLQMM) so it is likely that difference between the 2 scenarios is down to random variation. The t-test values given in table 4.4 show that the difference in the performance between any of the procedures is statistically insignificant when the maximum machine speed is 10000 MIPS. In fact even at 1000 MIPS only TLQMM offered significantly different results to MCT, although TLMM was close to this. The low t-values from the experiments with high maximum machine speeds is expected due to the large increase in the variance in the results stemming from the fact that the throughput of the simulated Grid can be vastly different from one experiment to the next. Throughout the tests the time limit batch

mode procedures provided the lowest makespans, suggesting it is a suitable for use in many different scenarios.

	Max Rate 100 MIPS			Max Rate 1000 MIPS			Max Rate 10000 MIPS		
	t value vs MCT	t vs Min-Min	t value vs QGMM	t value vs MCT	t vs Min-Min	t value vs QGMM	t value vs MCT	t vs Min-Min	t value vs QGMM
Min-Min	1.42			0.22			-0.11		
QGMM	1.66	0.38		0.23	0.02		-0.08	0.01	
TLMM	**4.58**	**3.70**		1.85	1.77		-0.07	0.01	
TLQMM	**4.93**		**3.91**	**2.10**		1.81	0.04		0.20

Table 4.4: *t* values for algorithm's makspans under differing levels of Grid Heterogeneity. The bold values indicate that they are greater than 1.96 and hence considered statistically significant assuming an infinite number of samples and an alpha value of 0.05

4.5.2.5 Effect of Quality of Information

The quality of information refers to the accuracy of the estimated completion times available to the scheduler. One of the main complications in Grid scheduling that sets it apart from other scheduling problems is the fact that the resources are non-dedicated and distributed. Having resources of this nature makes it increasingly difficult to predict the run times of the tasks on the hosts, because the state of the network and resources can never be accurately calculated for the duration of the tasks execution. For instance, whilst the input files for a task are being transferred to a host, another network user may log on and run a bandwidth intensive application, causing the transfers to take far longer than anticipated. This estimated completion time information, however is crucial since it is the only data on which the scheduler bases its decisions. To study the effect of inaccuracy in the estimated completion time the algorithms were tested on the same scenario as stated in section 4.5.2.4, where 50% of the tasks require high QoS, whilst the estimated prediction times were subjected to a random error of at most the given percentage. This error was introduced to the 100% accurate execution times by calculating equation 4.3 using values for the job size put

through the randomising method presented in figure 4.16. This method allows the execution times to be both under and over estimated with equal probability and gives a roughly linear distribution of values, however is limited to considering scenarios in which the maximum possible error is < 100%. Experimental research indicates that although such situations are possible, the maximum error of real problem runtimes is unlikely to be greater than 10% [SU2003] and therefore using this error introducing function allows us to model both expected and extreme conditions. The results of these experiments are graphed in figure 4.17.

```
1. amount = randomNumberGenerator() * maxErrorPercent
2. random = randomNumberGenerator()
3. if random > 0.5
4.         executionTime = executionTime * (100+amount) / 100
5. end if
6. else
7.         executionTime = executionTime * (100-amount) / 100
8. end else
```

Figure 4.16: Method used to randomise the estimated completion times. The random number generator function refers to the Math.random() function as given in [JA2005] and the variable *maxErrorPercent* to the given error percentage.

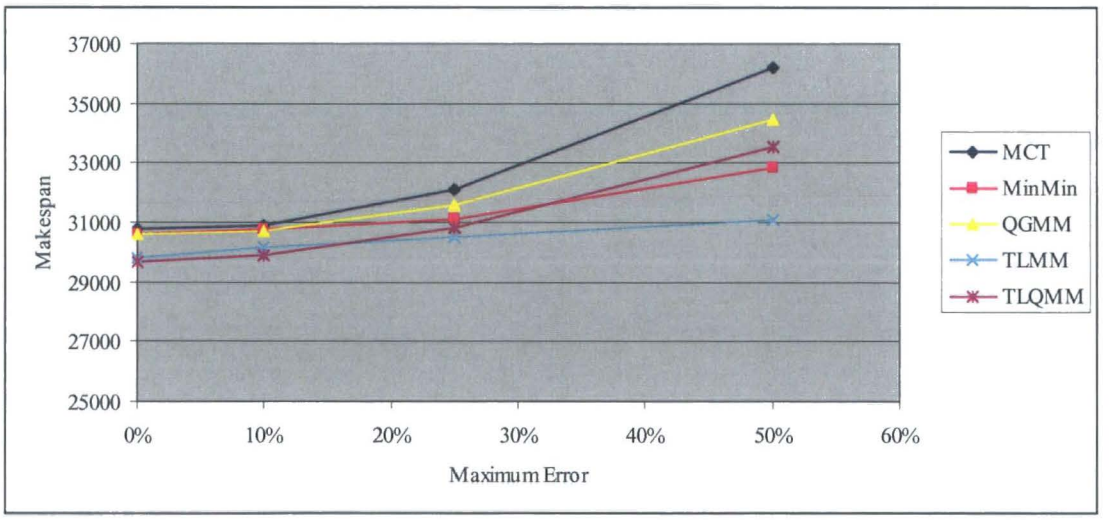

Figure 4.17: Makespan against the maximum percentage error in execution time estimation

The results given in figure 4.17 clearly show that information quality is an important factor in determining the makespan and that the choice of algorithm becomes more

significant the greater the error in the estimate of the execution time. The results appear to disagree with [XI2003] which suggest that QGMM continues to outperform Min-Min as the estimated completion time error increases, these results show that although QGMM marginally outperforms Min-Min when the error is minimal, Min-Min provides ever lower makespans over QGMM as the information quality degrades. This pattern is also continued with time-limit batch mode equivalents. However, under normal operating conditions when the error is likely to be around 10% [SU2003], the algorithms appear to give similar results to when there is no error at all, except that the makespans are slightly larger. In both these cases inclusion of QoS guided scheduling improves the Min-Min algorithm's performance. Perhaps the most interesting result is that of TLMM when the maximum error is set to 50%, this suggests that time-limit batch mode can reduce the effect inaccurate information quality; it is difficult to be certain this is the case as the variance between the results representing each point greatly increases as the information quality decreases. However since the improvement in performance between the time-limit batch mode algorithms and their batch mode equivalents remains significant in all tests we can conclude that the inclusion of time-limit batch mode does not reduce their ability to cope with poor information quality.

4.5.3 Further Experiments

Although experimenting on dedicated Grids with space-shared, single-processor resources allows us to identify how factors affect the performance of time-limit batch mode algorithms, the results do not accurately represent how the method is likely to fare on real Grid environments.

To produce more realistic simulations, experiments were conducted on simulated non-dedicated Grid environments consisting of resources based on actual computers in the World Wide Grid (WWG) [BUY2001], as used in experiments given in [MU2002]. Table 4.5 gives further details of these.

Simulated resource characteristics: vendor, resource type, node OS, No. of PEs	Equivalent resource in WWG (hostname, location)	A PE SPEC/MIPS rating	Resource manager type
Compaq, AlphaServer, CPU, OSF1, 4	grendel.vpac.org, VPAC, Melbourne, Australia	515	Time-shared
Sun, Ultra, Solaris, 4	hpc420.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared
Sun, Ultra, Solaris, 4	hpc420-1.hpcc.jp AIST, Tokyo, Japan	377	Time-shared
Sun, Ultra, Solaris, 2	hpc420-2.hpcc.jp AIST, Tokyo, Japan	377	Time-shared
Intel, Pentium/VC820, Linux, 2	barbera.cnuce.cnr.it, CNR, Pisa, Italy	380	Time-shared
SGI, Origin 3200, IRIX, 6	onyx1.zib.de, ZIB, Berlin, Germany	410	Time-shared
SGI, Origin 3200 IRIX, 16	Onyx3.zib.de, ZIB, Berlin, Germany	410	Time-shared
SGI, Origin 3200, IRIX, 6	mat.ruk.cuni.cz, Charles U., Prague, Czech Republic	410	Space-shared
Intel, Pentium/VC820, Linux, 2	merge.csm.port.ac.uk, Portsmouth, U.K.	380	Time-shared
SGI, Origin 3200, IRIX, 4 (accessible)	green.cfs.ac.uk, Manchester, U.K.	410	Time-shared
Sun, Ultra, Solaris, 8	pitcairn.mcs.anl.gov, ANL, Chicago, U.S.A.	377	Time-shared

Table 4.5: WWG test-bed resources simulating using G-Sim [MU2002]

The non-dedicated nature of these resources is simulated by applying different local CPU usage models. In using a non-dedicated Grid test bed, difficulties arise in calculating the time for the machines to process the tasks. This is because CPU utilisation affects the task's run time and is subject to rapid, violent and unpredictable change. In these experiments, a simple short-term completion time estimation functions are used to predict the completion time of each task on each resource. These are designed to be fast to compute and as accurate as possible although will inevitably introduce a level of error, the vast majority of which is determined by the change in the resources state after allocation. The functions, given in equation 4.4 and 4.5, are accurate in predicting the completion time of a task on their respective resource types in the case where the state of the resource does not change during execution and all the jobs running already allocated have an equal number of instructions still to process when the task arrives. They are derived from the operation of the resource's *scheduleJob* methods presented in figures 3.4.1 and 3.4.2 and the definition of the local CPU usage models described in section 3.5. The time-shared completion time function operates by dividing the number of instructions the job

requires by the speed the machine can currently process them at, whilst the space-shared version adds the estimated time until a processor is available to the time required to process the given job. The time until a processor is available is calculated by taking the average time to complete a single job in the queue and multiplying it by the minimum number of these job a processor will compute, plus one for the one it is currently running. These functions are comparable to the short-term completion time estimation schemes used in NWS [WO1999] and AppLeS [CA2000], although they does not take into consideration previous application runtimes since the size of the task and speed of the machine can be exactly determined. These algorithms can be considered to give a fair representation of the operation of a non-dedicated time-shared and space-shared resource scheduling algorithm where local tasks have higher priority to remote ones since having under a heavy local load the machines will be unable to process any remote tasks. Such a situation is commonplace in real Grid environments.

$$CT_{ij} = \frac{MI_i \times \left(\left\lfloor \frac{JOBS_j}{PES_j} \right\rfloor + 1 \right)}{MIPS_j \times (100 - LOCAL_j)}$$

Equation 4.4: Estimated completion time function for task i on time-shared machine j. CT_{ij} is the estimated completion time, MI_i the number of millions of instructions to complete task i, $JOBS_j$ the number of jobs currently j, $MIPS_j$ the speed of one of j's processors, PES_j the number of processors j has and $LOCAL_j$ is the percentage of processing power allocated to local tasks as defined by the local usage function.

$$CT_{ij} = \left(\frac{\left(\frac{QMI_j}{QJOBS_j} \right)}{MIPS_j} \right) \times \left(1 + \left\lfloor \frac{QJOBS_j}{\text{nint}\left(PES_j \times (100 - LOCAL_j) \right)} \right\rfloor \right) + \frac{MI_i}{MIPS_j}$$

Equation 4.5: Estimated completion time function for space-shared machines. The terminology used is the same as in equation 4.4 with the exception that nint refers to

the nearest integer operation, $QJOBS_j$ the number of jobs in j's job queue and QMI_j the total number of instructions of the jobs in this queue.

The simple parameter-sweep application model, where each application consists of a number of independent jobs of differing sizes, as used in the initial tests was reused since it enables us to capture differences in the algorithm's performance without having to consider the effects of the underlying network which are not the subject of study. Since a number of other factors such as computational latency and resource failure are also abstracted out in these experiments, the results are not intended to give a fair representation of how these procedures would operate on a real Grid environment but allow us to gauge their effectiveness relative to one another.

4.5.3.1 Effect of Local Usage

Dealing with the non-dedicated nature of Grid resource is a challenge of algorithm designers. This difficult arises from the fact the scheduler has no control over the future load of the resource and hence it is difficult to predict the completion time of tasks, unlike in traditional scheduling problems. To see how changes to the local usage rate of the resources affect the performance of time-limit batch model scheduling algorithms, experiments were conducted using a variety of different local usage models. The experiments used the test-bed Grid given in table 4.5 was used together with the completion-time estimation algorithms given in equations 4.4 and 4.5. The experiments are based on the cases used in [MU2002], they consider a scenario where 20 users each submit an application consisting of 100 independent tasks each with size evenly distributed between 100000 and 110000 MIPS. Each application has equal chance of requiring either high or low QoS. All users submitted their applications one after the other with an average 10 second gap between each to a single scheduler operating under the given algorithm. This situation can be considered to be one in which there is a sudden and dramatic increase to the load on the Grid. Each result given below represents the mean average of 100 independent runs.

The local usage experiments look to investigate the effect of variation in local usage. The scenario described above was used in conjunction with a local usage model in

which the average local usage was 50% but the actual value was randomly distributed between this amount and plus and minus a given maximum percentage. The value of the local usage given to each machine was changed every 5 minutes of simulated time. The results are presented in figure 4.18.

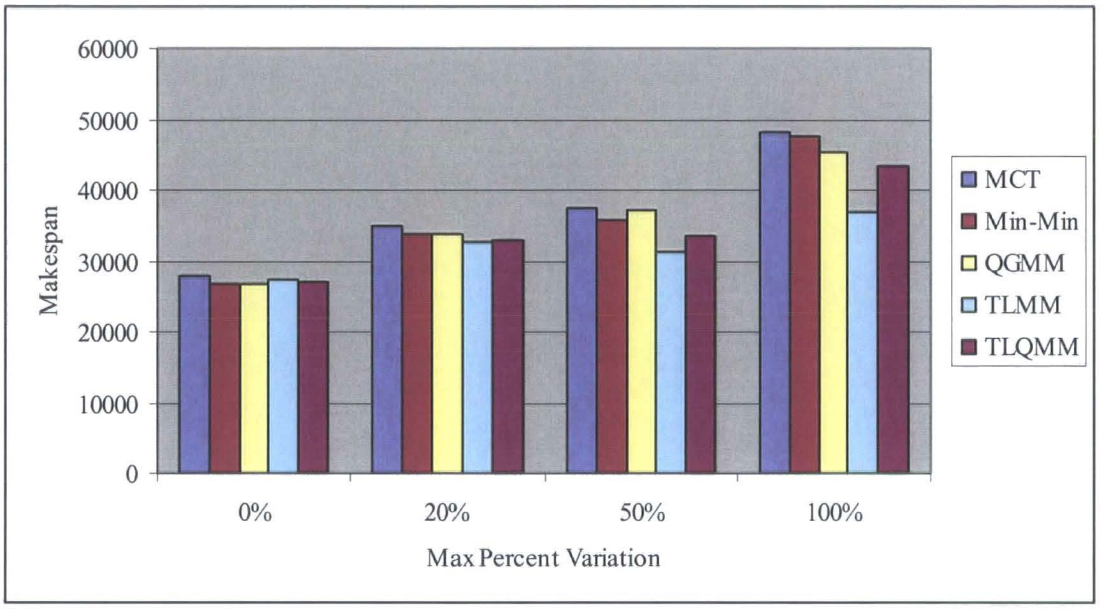

Figure 4.18: The maximum variation in local CPU percentage use against the makespan of the experiment. The makespan is given in seconds.

These results show that increase in the variation increases the makespan for all algorithms. This fits with what one would expect since the higher the variance, the higher the likelihood of at least one machine which has far higher than average local usage rate and so would complete its allocated tasks far slower than anticipated. The results also indicate that time-limit batch mode reduces the negative effect of variation in local usage, this is more clearly seen in figure 4.19 which shows the decrease in makespan the other algorithms gave over MCT. Although the traditional batch mode procedures always improved on the MCT algorithm's allocations the difference remained roughly the same as the variation increases. The time-limit batch mode procedures on the other hand saw a greater improvement over MCT as the variation increases. Such results are not surprising as time-limit batch mode holds back the decision of which mapping to make, it is more likely to put tasks on machines which are ahead rather than behind its predicted schedule and so has greater chance of producing effective mappings. In these tests the TLMM algorithm was by far the

most successful at dealing with the increase in local load variation. Although it gave the second worse mappings when the variation was 0%, it provided by far the best in all other tests. This result is due to the fact that it allocates far fewer jobs at one time than the other algorithms, as demonstrated in figure 4.20, and so can react better and allocate more tasks to machines which are ahead of schedule rather than saturate those which have had a higher local load levels.

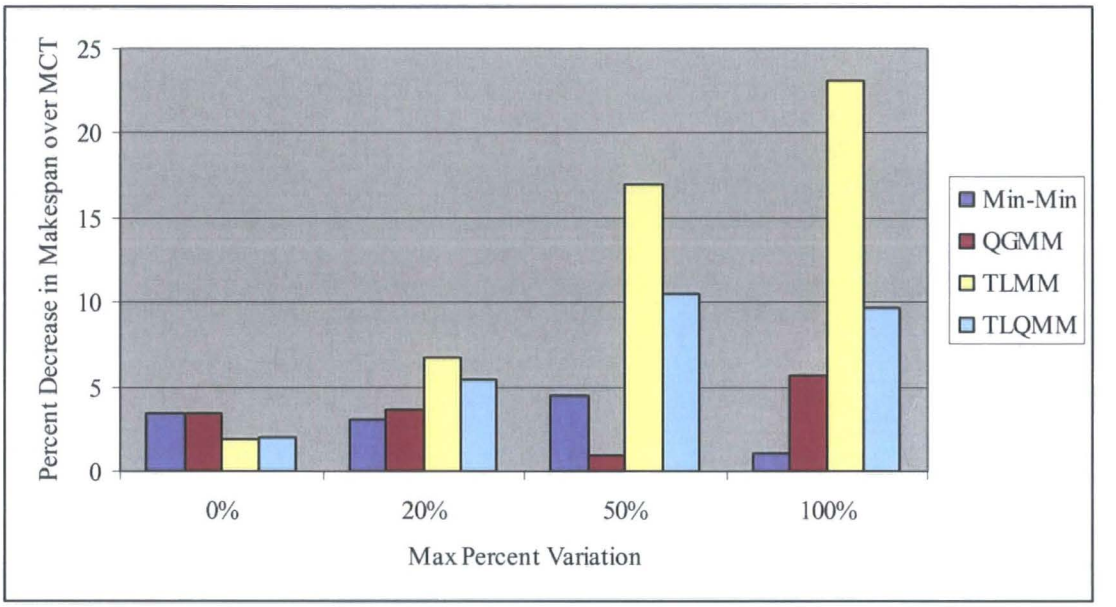

Figure 4.19: The percentage decrease in makepan over the MCT algorithm under different levels of variation in the percentage of local CPU usage

The average number of tasks on each host seems to be affected by the level of variation in local load in much the same way as the heterogeneity of the Grid, making figure 4.20 similar to figure 4.14. This is expected since from the view of the scheduler the higher the variation in local load the higher the heterogeneity of the Grid. The traditional batch mode schedulers see a decrease in the average number of jobs at each resource as the variation increases whilst this performance metric remains relatively unchanged for the TLQMM algorithm and it increases for the TLMM algorithm. It is likely that it lowers for the batch mode schedulers as there becomes an increasing larger period of time where a small number of machines, which have seen an unusually large local load are running whilst the other have stopped. The opposite effect occurs with TLMM which sees an increase in the average number of jobs on each machine with increasing local load variation, this effect can be explained

by the fact it is more likely to release more jobs at one time the greater variance between the resources capacities. The fact that the metric is unaffected for the TLQMM algorithm could be related to the factors which increase and decrease it balancing out.

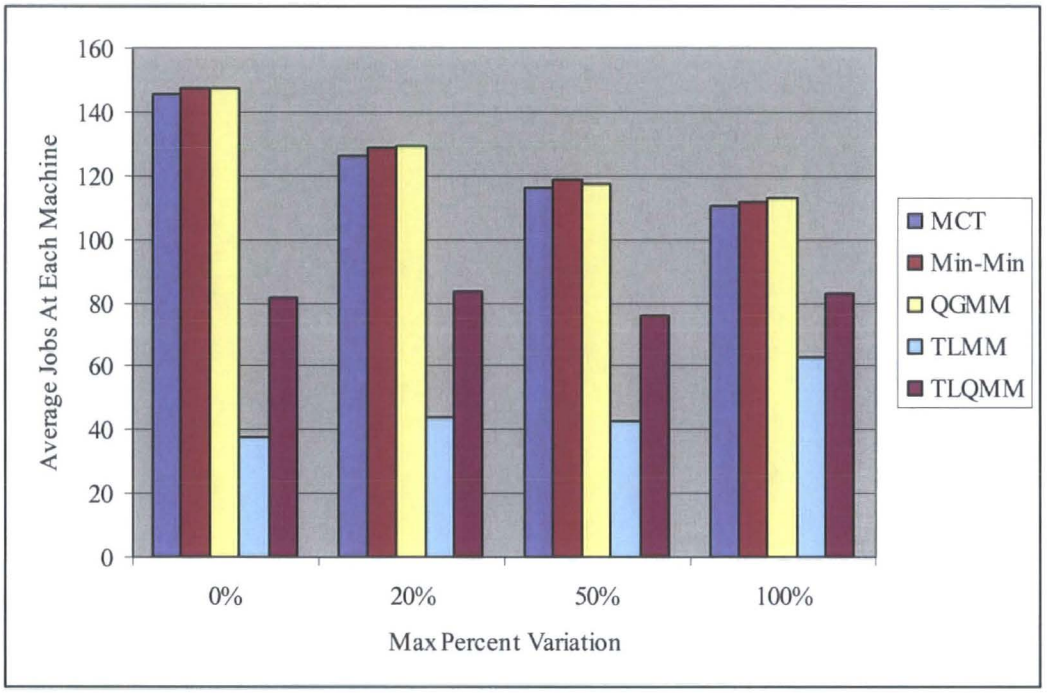

Figure 4.20: Average number of tasks on each host against over different levels of variation in the percentage of local CPU usage

4.6 Conclusions and Further Work

The results presented throughout this chapter show that time-limit batch mode has the potential to improve Grid scheduling algorithms. It has been shown to consistently decrease the makspan of both heuristics it was applied to and, in most circumstances, significantly reduce the effect of resource failure. As the adaptation from a batch mode algorithm to a time-limit one can be made with only minor changes, it provides a simple and highly effective improvement with little effort on the part of the scheduling algorithm designer. The results also show that some algorithms are more suited to be converted to time-limit batch mode as although both procedures were successful, the TLMM algorithm was particularly impressive in lowering both the average number of tasks at each resource and the makespan.

From the results we can conclude that time-limit batch mode is most likely to be beneficial over batch mode when:

1. Scheduling wait time is relatively small
2. When the Grid size is relatively small in comparison to the number of tasks
3. The resources in the Grid operate at roughly the same speed
4. There is potentially a large differentiation between the predicated and actual execution times of tasks
5. The amount of local load varies highly

The further the state of the Grid moves away from such a state the less difference time-limit batch mode will make to the makespan of the schedule or the number of jobs it holds back. Since there are no set of experiments in which the batch mode algorithms proved significantly outperformed their time-limit counter parts it is not clear whether its use in some scenarios will prove detrimental.

One interesting outcome of the tests is that they show QoS guided methods, although generally slightly more effective, appear to be less effective than the original mapping procedures from which they were derived, when faced with a large errors in the predicted runtimes. This is not always the case however and appears to be dependant on the simulation model and the scenario since results presented in [XI2003] conclude their algorithm copes with error just as well as Min-Min. Another point of interest is that the decrease in makespans between the QoS guided approaches and non-QoS guided equivalents are often so small as to make them insignificant. In no set of results did either of the QoS guided algorithms outperform the others by anything greater than a few percent, this is contrary to the finding in [XI2003] which suggest around an 11% improvement in performance is typical. This difference suggests that minor differences within the experiments set-up can produce a large impact on the result. If true this certainly has ramifications for the validity of results gathered from Grid simulation since it is likely that one can find a scenario and model in which their presented algorithm was the most effective, even over a number of trials. This demonstrates a need for Grid scheduling algorithm designers to adopt a set of standardised benchmark tests against which their procedures should be judged, much

like those used employed by the SAT solving community (see [SI2005]). This would require significant further study and collaboration on the part of the Grid scheduling community since currently most scheduling algorithm designers are developing solutions to different variants of the problem, based around their individual scenarios; therefore it would be difficult to produce tests which were inclusive enough. Perhaps the best comparison of algorithms so far is presented within [BR2001], although this work only considers static mapping procedures and therefore the results can not be considered to reflect how the procedures will fare when operating on dynamic Grid environment over a period of time.

The results concur with [BR2001] that Min-Min consistently outperforms MCT and that the level of improvement is highly dependant on the scenario being tested. In their results, Min-Min gave a similar performance improvement over MCT to the majority of these experiments (around 1%), when the execution time matrix was inconsistent and by only a slightly larger margin when the matrix was partially consistent. Min-Min did however outperform MCT by around 11-20% when the matrix was consistent. Whilst this difference appears to be a far more impressive improvement than these results indicate, even though G-Sim can be considered to use a consistent execution time matrix in these experiments, the work by Braun et al. does not consider how the algorithms operate over a period of numerous runs in succession as these experiments do, which may explain this disparity.

It should be taken into consideration that this work is not intended to show how this method should be implemented on a real Grid which would require careful consideration and further research, rather it is to verify whether the strategy could be effective. One practical issue, that is trivial in simulation, and which should be taken into consideration when employing this technique on a real Grid, is how to determine when to schedule again. The problem being that to ensure maximum efficiency the machines should be kept constantly busy, so as not to waste processing time, however if the predicted runtimes are higher than the actual runtimes, some machines will complete their allocated tasks faster than anticipated and the process of scheduling again should then by put forward to compensate for this. Likewise, machines which are taking longer than expected to complete should not be presented with the same

number of tasks they would have been if the predicted execution times had been accurate.

It is possible that time-limit batch mode will not be ubiquitously suitable for all types of Grid application and it will certainly require further work to consider how to apply it to data-intensive tasks. This is because these types of application typically require large data sets to be uploaded to the resource prior to execution and therefore effective makespans can only be found if one ensures that each resource is not only given enough processing tasks to keep it busy until the next scheduling period, but, also ensures that their bandwidth capacity is utilised for transferring files during this period. As traditional batch mode schedulers potentially present many tasks to each resource at once, the resources can prepare to run future tasks by downloading their required files whilst others are being processed (since processor and bandwidth intensive task are normally considered able to be run simultaneously with little effect to the performance of either task). In time-limit batch mode, such preparation can not be performed as holding tasks back could lead to avoidable situations in which the resources are unaware of the requirements of the next jobs they will process.

Chapter 5 – Converting Grid Scheduling to SAT

5.1 Introduction

Grid scheduling is a relatively new subject and as such the algorithms used to generate the mappings between jobs and resources are often unsophisticated in comparison to those used to solve other, better known problems. In particular the first problem to be proven NP-Complete (SAT by Cook in 1971 [CO1971]), has been the subject of years of theoretical and pragmatic research culminating in the development of some extremely efficient solvers [SI2001]. This chapter demonstrates how the Grid Scheduling problem can be modelled as SAT such that the resulting instance requires only a polynomial amount of time to solve in the size of the original instance in the event of finding a polynomial algorithm for solving SAT. Such a conversion allows SAT solvers to be applied to the Grid scheduling problem, potentially providing more effective algorithms than traditional approaches. The idea of applying SAT solvers and general Constraint Satisfaction Problem Solvers (the SAT problem where variables can take a number of values ≥ 2) is a well used technique which has proved successful on a vast number of problems, most notably planning [KA1996] [BU2003], which has been extensively studied due to its applications in AI. The author is not aware of any other attempt to solve scheduling problems using SAT solvers.

5.2 Problem Definitions and Characteristics

The Grid Scheduling problem can be defined in a number of ways. Since there are so many factors which one may take into consideration when designing a Grid scheduler, scheduler algorithm designers will typically work on a version of the problem which abstracts as out as many of these factors deemed less significant in their given scenario as possible. For example the designer of an algorithm to schedule a parameter sweep application may not be interested in modelling bandwidth limitation factors, since such applications are typically processor rather than bandwidth

intensive. Likewise someone using a largely homogeneous Grid may consider the time required to process a particular task to be independent of the processing speed of the host machine. We will take a general form of the problem derived from [BR2001], where a base-line definition is given in where the schedulers must base their decisions on a matrix of 'estimated times to compute'. The definitions of this Grid Scheduling problem (GSP) and the SAT problem are given below.

Grid Scheduling (O)

Input:

- A set of tasks T of size n, where n > 1.
- A set of hosts H of size m, where $m > 1$.
- A matrix E of size $n*m$ where each element $Eij \varepsilon N$ and gives the estimated execution time of the ith task in T when run on the jth host in H.

Output:

- A mapping of tasks to machines such that each task $t_i \varepsilon T$ is mapped to a single host $h_j \varepsilon M$ such that the maximum combined size of all jobs mapped to any h_j is minimal.

SAT (D)

Input:

- A set of n Boolean variables $V, x_1,\ldots,x_n$.
- A set of m clauses C where each $c_i \varepsilon C$ is a Boolean function in conjunctive normal form on a number of variables in V.

Output

- The Boolean value 0 in the case that there is no assignment that will allow every $c_i \varepsilon$ C to be satisfied and the Boolean value 1 otherwise.
-

Since the SAT problem is a decision problem outputting only a Boolean value, we must adapt our definition of GSP to be not one which produces a mapping of jobs to machines but an equivalent decision problem for the conversion to be possible. To do this we can ask the question of whether there exists a mapping which is of size at most some value k. We will then answer this decision problem a number of times changing both the value of k then the instance itself until we have an answer to the

original decision problem. In section 5.5 we discuss more thoroughly the method of obtaining the solution to GSP mapping from successive solutions of its decision problem equivalent. The Grid Scheduling decision problem (GSPD) is defined below.

Grid Scheduling (D)

Input:

- A set of tasks T of size n, where n > 1.
- A set of hosts H of size m, where $m > 1$.
- A matrix E of size $n*m$ where each element E_{ij} gives the estimated execution time of the ith task in T when run on the jth host in H.
- A number $k \varepsilon$ N.

Output:

- The Boolean value 0 in the case that it is impossible to allocate all elements of T to the machines H such that the maximum combined size of all jobs mapped to any $h_j \varepsilon M$ is no more than k and the Boolean value 1 otherwise.

5.3 Problem Conversion

We now demonstrate how to convert instances of GSDP to instances of SAT. We then show that the size of the resulting SAT instance is polynomial in the size of the original instance and takes only a polynomial amount of time to construct.

We start by defining variables whose value determines whether or not a particular task is allocated to a particular host. These Boolean variables are denoted as X_{ij} and setting it to true indicates that the task $i \varepsilon T$ is allocated to host $j \varepsilon H$, therefore the opposite literal $- X_{ij}$ indicates that i is not allocated to j. Since we know that each task must be allocated to exactly one machine we add clauses to reflect this. This situation can be described for task i by the set of clauses given below (note that the character V denotes the logical OR operator and N the logical AND operator).

$$(X_{i1} \text{ V } X_{i2} \text{ V } X_{i3} \text{ V } \ldots \text{V } X_{im}) \text{ N}$$

$$(-X_{i1} \text{ V } -X_{i2}) \text{ N } (-X_{i1} \text{ V } -X_{i3}) \text{ N } \ldots \text{ N } (-X_{i1} \text{ V } -X_{im}) \text{ N}$$
$$(-X_{i2} \text{ V } -X_{i3}) \text{ N } (-X_{i2} \text{ V } -X_{i4}) \text{ N} \ldots \text{ N } (-X_{i2} \text{ V } -X_{im}) \text{ N}$$
$$(-X_{i3} \text{ V } -X_{i4}) \text{ N} \ldots \text{ N } (-X_{i3} \text{ V } -X_{im}) \text{ N}$$
$$\ldots \text{ N}$$
$$(-X_{i(m-1)} \text{ V } -X_{im})$$

The first clause ensures that at least one of the variables $X_{i1} \ldots X_{im}$ must be true, this is because setting them all to false will make it impossible to satisfy it. The later clauses prevent any two variables both being set to true since in each clause if one is true then the other must be false otherwise the instance cannot be satisfied. Since we require a clause for every combination of 2 hosts we have mC2 +1 clauses for each task and therefore $n(m\text{C2}+1)$ clauses to do this for all tasks. This is still polynomial in the size of the GSP instance size mC2 is $(m^2 + m)$ /2 making the total number $(nm^2 + nm)/2 + n$, which is clearly $O(nm^2)$.

The more challenging part of this conversion is in defining clauses to describe how the size of the jobs allocated to a particular machine is added to that machine's total allocation and in determining whether a given allocation exceeds the limit k. The answer to these problems lies in the operation of a core circuit in computer science, the full-adder. We can represent the estimated execution time of a job on a particular machine simply as a binary number using single literal clauses, one for each bit required to store the number. For instance the decimal number 19 is equivalent to the binary digit 10011, if this were the size E_{ij} we would add the clauses X_{ijs5} N $-X_{ijs4}$ N - X_{ijs3} N X_{ijs2} N X_{ijs1} to the SAT instance, where the variable X_{ijsb} represents whether the bth binary digit is set to 1 or 0 in the binary number representing the size of E_{ij}. We then add the clauses $(-X_{ij} \text{ V } X_{ijk} \text{ V } -X_{ijsk}) \text{ N } (-X_{ij} \text{ V } -X_{ijk} \text{ V } X_{ijsk})$ for all tasks $i \varepsilon T$, all $j \varepsilon H$, and bits k. We introduce the variable X_{ijk} as representing whether the kth bit of the number E_{ij} will be significant when calculating the total size of the tasks allocated to host j. These clauses ensure that if the job i is allocated to machine j, then the variable X_{ijk} will have the same value as the variable X_{isk} otherwise one of these will not be able to be satisfied, however they do not restrict the value X_{ijk} can take in the event that X_{ij} is set to false. This means that X_{ijk} can be set to false unless i is allocated to j and the kth bit of the size of i is positive. This behaviour is crucial as it allows us to

use the variable X_{ijk} in constructing an "adder" for determining the size of all jobs allocated to j.

The behaviour of the adder is constructed by adding variables to describe the value of each bit of the number representing the size of the jobs allocated to a machine after each job has been added, together with additional variables to allow us to carry overflowing bits from one to the next. We will denote the variable which describes the value of the kth bit of the total size of the jobs allocated to machine $j \varepsilon H$ after i tasks have been added as X_{jsik} and the value of the carried overflow from the kth to the $(k+1)$th bit after adding i jobs to the total size of jobs allocated to j as X_{jsikc}.

We proceed as if we were adding the size of all jobs to the total for each machine except that instead of "adding" the actual kth bit of the size of E_{ij} (X_{ijsk}) to a machine j, we "add" the variable X_{ijk}. This "adding" is performed by including clauses to ensure the variables representing the output of the addition and any carried values must be true when the truth tables 5.1 and 5.2 specify them as being so in relation to the possible values the variables representing the input can take, if the instance is to be satisfied.

X_1	X_2	X_o	X_{co}
F	F	F	F
F	T	T	F
T	F	T	F
T	T	F	T

Table 5.1: The truth-table for the adding of 2 binary bits, X_1, X_2. The output variables X_o and X_{co} describe the value that bit takes and the value carried to the next bit respectively. Note that the Boolean values true and false are represented in tables as T and F respectively.

X_1	X_2	X_{ci}	X_o	X_{co}
F	F	F	F	F
F	F	T	T	F
F	T	F	T	F
F	T	T	F	T
T	F	F	T	F
T	F	T	F	T
T	T	F	F	T
T	T	T	T	T

Table 5.2: The truth-table for the adding of 3 binary bits, X_1, X_2 and X_{ci} (the carried input from the previous bit). The output variables X_o and X_{co} describe the value that bit takes and the value carried to the next bit respectively.

The value of the variable X_{jsil} is dependant on the value of the previous total $X_{js(i-1)l}$ and value of the least significant bit of the next job to be added represented by the variable X_{ijl}. Since the value of the variable X_{js11} is equivalent to the value of the variable X_{1jl} we do not consider it. We control the behaviour of the variables X_{jsil} and X_{jsilc} by adding the clauses given below.

$$(X_{jsil} \text{ V } X_{js(i-1)l} \text{ V } -X_{ijl}) \text{ N } (X_{jsil} \text{ V } -X_{js(i-1)l} \text{ V } X_{ijl}) \text{ N } (X_{jsilc} \text{ V } -X_{js(i-1)l} \text{ V } -X_{ijl})$$

These clauses give us the required behaviour of table 5.1 since the carry variable must be true if both inputs are true and the output must be true when the inputs have opposite values.

In dealing with other bits (those which are not the least significant) we need consider the carry variable from the previous bit as an additional input and so we must implement the behaviour of table 5.2. This is accomplished with the following clauses.

$$(X_{jsik} \text{ V } -X_{js(i-1)k} \text{ V } -X_{ijk} \text{ V } -X_{j(i-1)(k-1)c}) \text{ N } (X_{jsik} \text{ V } -X_{js(i-1)k} \text{ V } X_{ijk} \text{ V } X_{j(i-1)(k-1)c}) \text{ N}$$
$$(X_{jsik} \text{ V } X_{js(i-1)k} \text{ V } -X_{ijk} \text{ V } X_{j(i-1)(k-1)c}) \text{ N } (X_{jsik} \text{ V } X_{js(i-1)k} \text{ V } X_{ijk} \text{ V } -X_{j(i-1)(k-1)c}) \text{ N}$$

$$(X_{jsikc} \text{ V } -X_{js(i-1)k} \text{ V } -X_{ijk} \text{ V } -X_{j(i-1)(k-1)c}) \text{ N } (X_{jsikc} \text{ V } -X_{js(i-1)k} \text{ V } -X_{ijk} \text{ V } X_{j(i-1)(k-1)c}) \text{ N}$$
$$(X_{jsikc} \text{ V } -X_{js(i-1)k} \text{ V } X_{ijk} \text{ V } -X_{j(i-1)(k-1)c}) \text{ N } (X_{jsikc} \text{ V } X_{js(i-1)k} \text{ V } -X_{ijk} \text{ V } -X_{j(i-1)(k-1)c})$$

The first 4 clauses ensure that the output variable X_{jsik} is true when one or all of the inputs are true whilst the later 4 ensure the carry variable X_{jsikc} is true when two or all the inputs are true, as with in table 5.2.

The number of clauses and variables required to create this behaviour is dependant on the number of bits required to store the size of each E_{ij} and the total assignment to each machine. Since k is the limit which can not be exceeded by the combined total of the execution times of the jobs allocated to any host we can take the maximum number of bits required to represent such a number to be $\lfloor \log k \rfloor$ since this is the number of bits required to represent any number in binary. However we add an additional bit which simplifies the task of testing whether the limit k has been exceeded by the size of the jobs allocated to a particular machine and hence how we will determine whether or not the instance can be solved.

To test whether the size has been exceeded we first represent the binary number for k using clauses as before using $\lfloor \log k + 1 \rfloor$ single literal clauses, the variables in which we denote X_{kb}, where b refers to the bit number such that when $b = 1$ it refers to the least significant bit of the binary number representing k. Since we are using one bit more than is required the variable representing the most significant digit ($X_{k\lfloor \log k+1 \rfloor}$) must always be false. We then check whether this number is larger or smaller than the size of the jobs allocated to machine $j \varepsilon H$ by adding clauses to mimic the behaviour of the simple algorithm given in figure 5.1.

```
boolean greaterThan(boolean [] k,boolean [] s){
        return greaterThan(k,s,0);
}
boolean greaterThan(boolean [] k,boolean [] s,int position){
        if(k[position] && !s[position]){
                return true;
        }
        if(!k[position] && s[position]){
                return false;
        }
        postion++;
        if(position == k.length){
                return false;
        }
        return greaterThan(k,s,position);
}
```

Figure 5.1: A simple algorithm, written in Java, which returns true if the binary number represented by k is greater than the one represented by s.

This procedure works by checking the value of the bits starting with the most significant. If this bit is true for k and not for s we can conclude that k is bigger than s, if the opposite is true then k is smaller than s. In the case that the bits have equal value (they are either both true or both false), the next most significant bit (MSB) is checked and the same procedure applied. If the procedure goes past the last bit we know that the values s and k are equal and hence we can return false.

This behaviour can be created using just a polynomial number of clauses. To do this we first add the single literal clause $-X_{jsn\lfloor \log k+1 \rfloor}$ to control the behaviour of the MSB. This clause makes it impossible for the MSB representing the total size of the of jobs assigned to host j to be true, since we know that any number with this bit set to true is larger than k. When performing the addition we must ensure that if this bit is ever set to be true then it must always be true (i.e. if for any i any $X_{jsi\lfloor \log k+1 \rfloor}$) must be true then so must the variable $X_{jsn\lfloor \log k+1 \rfloor}$). To do this we add the single literal clause $-X_{jsi\lfloor \log k+1 \rfloor}$ to the instance for all machines j and all jobs i, since then it will be

impossible for this bit to ever be set to true without rendering it impossible to solve the instance. The next MSB could be controlled with the clause ($X_{k\lfloor \log k \rfloor}$ V $-X_{jsn\lfloor \log k \rfloor}$), since the inclusion of this means this bit could not be set to true for the total size of the jobs on machine j if it were not also true in k. However this clause is rendered unnecessary since we already know that $X_{k\lfloor \log k \rfloor}$ must be set to true anyway. The clauses required for the third MSB are given below:

$$(-X_{k\lfloor \log k \rfloor} \text{ V } -X_{jsi\lfloor \log k \rfloor} \text{ V } X_{k\lfloor \log k-1 \rfloor} \text{ V } -X_{jsi\lfloor \log k-1 \rfloor} \text{ N}$$
$$(X_{k\lfloor \log k \rfloor} \text{ V } X_{jsi\lfloor \log k \rfloor} \text{ V } X_{k\lfloor \log k-1 \rfloor} \text{ V } -X_{jsi\lfloor \log k-1 \rfloor})$$

These clauses can only not be satisfied if the second MSBs in the numbers representing for the total for j and the number k have equal value and the third MSB for k is false and the third MSB for the total for j is true, thus mimicking the behaviour of the algorithm given in figure 5.2 on its third recursive call. On each subsequent bit one must check to see if each of the proceeding bits has the same value in both the numbers for k and the total for j, since there are then two situations where this can occur for each bit we require 2^{b-2} clauses, where b is the bit number and b equals 1 when referring to the MSB, for each one. These sets of clauses are constructed by simply having every combination of both true and both false values for each proceeding bit. To demonstrate this, the clauses required for the fourth MSB are given below.

$$(-X_{k\lfloor \log k \rfloor} \text{ V } -X_{jsi\lfloor \log k \rfloor} \text{ V } -X_{k\lfloor \log k-1 \rfloor} \text{ V}$$
$$-X_{jsi\lfloor \log k-1 \rfloor} \text{ V } X_{k\lfloor \log k-2 \rfloor} \text{ V } -X_{jsi\lfloor \log k-2 \rfloor})$$

$$(-X_{k\lfloor \log k \rfloor} \text{ V } -X_{jsi\lfloor \log k \rfloor} \text{ V } X_{\lfloor \log k-1 \rfloor} \text{ V}$$
$$X_{jsi\lfloor \log k-1 \rfloor} \text{ V } X_{k\lfloor \log k-2 \rfloor} \text{ V } -X_{jsi\lfloor \log k-2 \rfloor}) \text{ N}$$

$$(X_{k\lfloor \log k \rfloor} \text{ V } X_{jsi\lfloor \log k \rfloor} \text{ V } -X_{k\lfloor \log k-1 \rfloor} \text{ V}$$
$$-X_{jsi\lfloor \log k-1 \rfloor} \text{ V } X_{k\lfloor \log k-2 \rfloor} \text{ V } -X_{jsi\lfloor \log k-2 \rfloor}) \text{ N}$$

$$(X_{k\lfloor \log k \rfloor} \text{ V } X_{jsi\lfloor \log k \rfloor} \text{ V } X_{k\lfloor \log k-1 \rfloor} \text{ V}$$

$$X_{jsi\lfloor \log k-1 \rfloor} \text{ V } X_{k\lfloor \log k-2 \rfloor} \text{ V } \text{-}X_{jsi\lfloor \log k-2 \rfloor})$$

Fortunately since only $\lfloor \log k+1 \rfloor$ bits are required to represent each number we only require a polynomial number of clauses to create this behaviour this in the size of the instance. Checking the least significant bits of k and the total for j will require $2^{\lfloor \log k-1 \rfloor}$ clauses, which is no more than $k/2$ and since, starting with the MSB, the next smallest bit always requires twice as many as the last we know that the total clauses required for all these bits will only be at worst k-2 clauses.

Now that we have specified clauses allowing us to recreate the GSP instance as a SAT instance we must now assure ourselves that this conversion requires only a polynomial number of clauses (and hence variables) in the size of the original GSP instance. We have already concluded that the number of clauses required to specify that a job must be allocated to exactly one machine to be number $(nm^2 + nm)/2 + n$ clauses. We also require $\lfloor \log k+1 \rfloor$ clauses to represent the size of each element of E so therefore require $nm\lfloor \log k+1 \rfloor$ clauses to represent the size of execution time. There are an additional $2mn\lfloor \log k+1 \rfloor$ clauses required to make the clauses ($\text{-}X_{ij}$ V X_{ijk} V $\text{-}X_{ijsk}$) N ($\text{-}X_{ij}$ V $\text{-}X_{ijk}$ V X_{ijsk}) for all tasks $i \varepsilon T$, all $j \varepsilon H$, and bits k. To build the "adder" for calculating the size of the jobs allocated to a particular machine host we require the clauses 3 clauses for each time the least significant bit is incremented and 8 for each time any other bit is incremented (which is done n-1 times for each machine), this leaves us with $3(n\text{-}1) + 8(n\text{-}1)\ \lfloor \log k \rfloor$ for each machine and therefore a total of $m(3(n\text{-}1) + 8(n\text{-}1)\ \lfloor \log k \rfloor)$ for all the adders, this work out to be $3mn - 3m + 8nm\lfloor \log k \rfloor - 8m\lfloor \log k \rfloor$ clauses, which is still just polynomial in the size of the GSP instance. The representation of the number k requires another $\lfloor \log k+1 \rfloor$ clauses and to check whether this number is exceeded by the combined size of all the jobs allocated to any all $j \varepsilon H$ requires $m(k\text{-}2)+m(n\text{-}1)$ clauses. This amount is calculated from the fact that (n-1) additions are made for each of the m hosts and for each of these iterations a single clause is added to ensure the clause representing the MSB can not be set to true, the rest of the clauses come from the need to implement the algorithm covered in figure 5.1 for each host, the number of clauses required for this is previously discussed. This makes the total number of clauses required for the

resulting SAT instance equal to $(nm^2 + nm)/2 + n + 3mn\lfloor \log k + 1 \rfloor + 3mn\text{-}3m + 8nm\lfloor \log k \rfloor - 8m\lfloor \log k \rfloor + \lfloor \log k + 1 \rfloor + m(k\text{-}2) + m(n\text{-}1)$. Since each of these factors (of which there are a constant number) is only polynomial in the size of the Grid scheduling instance we can conclude that the resulting SAT instance will only be polynomially larger than the original instance.

The time required to create this instance is also polynomial in the size of the GSP instance since we perform no calculation on the instance other than finding the binary representation of each of the elements of E and the value k a task which is $O(mn)$. The time required is therefore bounded only by the number of clauses required, which we have shown to be polynomial. We do not consider the size of each clause as a factor dominating the time required to create the instance since no clause will have more than a polynomial number of variables, if this were not the case it could be ignored anyway since it could not render the instance unable to be satisfied.

5.4 Correctness of the Conversion

Having created the SAT instance to represent our GSP instance we must be able demonstrate that this newly created instance is in fact an accurate representation of the original. To do this we show that the following properties hold:

1. The SAT instance can not be solved if the original instance can not be solved.
2. The SAT instance can be solved if the original instance can be solved.

We will demonstrate these properties by presenting logical arguments that when followed reveal that the resulting SAT instance will have the same false/true properties as the original GSP instance.

We will first consider property 1. If it is not possible for the original instance to be solved then no matter how we set the variables in our SAT instance we should be unable to satisfy all the clauses. From our work earlier we know that each job must be assigned to exactly one machine otherwise the instance can not be satisfied, this means that exactly one clause X_{ij} for each i must be set to true for the instance to be

satisfied. From the clauses $(-X_{ij} \text{ V } X_{ijk} \text{ V } -X_{ijsk})$ N $(-X_{ij} \text{ V } -X_{ijk} \text{ V } X_{ijsk})$ for all tasks $i \varepsilon T$, all $j \varepsilon H$, and bits k we can conclude that the instance can then only be solved if the variables X_{ijk} for all k are set to the value of the variable X_{ijsk} when the variable X_{ij} must be true. The variable X_{ijsk} in turn can only be set to the same value as it appears in the binary representation of the size of the job i, this is ensured by the inclusion of the single literal clause X_{ijsk} or $-X_{ijsk}$. This means that the adder for each host will take at least the combined total size of all the jobs allocated to it as input. The adder can not set the variables representing its output to a binary number less than the size of the numbers given to it. This is because if any of the other variables which can be true or false given to it are set to true it only increases the size of the output variables, if they are all set to false then the output will be equivalent to the binary number for the total size of all the jobs. If this output is at any point this number requires more than k bits to store then the assignment does not fit the criteria and returns false since the single literal clause $-X_{jsi\lfloor \log k+1 \rfloor}$ is included for all i and j meaning that such a state will make it impossible to satisfy the instance. It should be noted that we ignore any instances of GSP in which there is a job with size bigger than k since it can trivially not be solved. If the variable $X_{jsn\lfloor \log k+1 \rfloor}$ can be set to false without rendering the instance unsolvable for each host j, then it may still be the case that the original instance can not be solved, since k maybe exceeded at some point. If this is the case then the SAT instance is guaranteed to be unsatisfiable because the clauses which implement the algorithm in figure 5.1 ensure this case stops the instance being solvable as previously documented. Since we always render the SAT instance unsolvable if any machine has total job size assigned to it which exceed the limit k, we always assign every job to a machine and if the original instance was unsolvable there would be no assignment which would give all machines a load of less than or equal k we can conclude that property 1 is true.

Property 2 is also true because if we assigned the tasks to the hosts such that the load assigned to each host was $\leq k$ then set all other variables which did not explicitly have to be set to true to false we could solve the SAT instance. This is because the adder for each machine would then only output the binary representation of the actual load associated with that machine (which would be $< k$) then all the variables $X_{js\lfloor \log k+1 \rfloor}$ could be set to false and the clauses implementing the algorithm in figure

5.1 would also be satisfied. The result would be that no clause would render the instance unsolvable and hence property 2 is true.

5.5 Solving the Grid Scheduling Optimisation Problem

Although we have shown that GSDP can be converted into SAT and solved this is of little significance to people wanting to know where to distribute their tasks to get the optimal performance. However it is easy to construct a solver for the optimisation variant that calls our conversion just a polynomial number of times. We will denote the procedure which converts the given GSDP instance to a SAT instance and then returns true or false depending on whether the instance can be solved as Q.

The algorithm to solve the optimisation problem has two stages, the first is to locate the minimum value for k which allows the resulting SAT instance to be satisfied, and the second part is to determine where each job should be sent.

Locating the minimum value for k requires only in the worst possible case $\lceil \log nE_{ijMAX} \rceil$ calls to Q, where E_{ijMAX} refers to the largest element in E, a value which can be determined by a simple search method in $O(mn)$ time. We choose the number nE_{ijMAX} as it is clear that the makespan would never be bigger than this value, even if all the tasks are allocated to a single machine, a more practical solution would be to run a simple algorithm such as MCT then use the makespan of the schedule it produces divided by two as the initial value for k. We take this initial value for k and then perform a binary search [COR2001] for the minimum value for k. This involves halving the value for k given to Q each time until false is returned. We then continually search between the points we know the value lies between select the value in the middle of this point to be k. After just $\lceil \log nE_{ijMAX} \rceil$ calls to Q you find the value for k such that Q returns true for this value and not for k-1 this value is the minimum value [COR2001].

Using this minimum value we then run Q again and replace the variable X_{11} with the identity TRUE, if Q then returns true we know that job 1 can be assigned to machine 1 and still give us the minimum makespan, otherwise we know that it can not. We

then run Q again and remove from the SAT instance any clause with X_{11} in it if we found it can be true and taking away $-X_{11}$ from any clauses with it in, if however we found that X_{11} can not be true we do the opposite. We then move on and do the same with variable X_{21} in the case that X_{11} can be true, else, we move on to the variable X_{12}. We continually look for where each job must be assigned until we find a position and move onto the next job. In the worst case we will check all jobs in all positions making a total of mn calls to Q.

Since we only require at most $\lceil \log nE_{ijMAX} \rceil + mn$ calls to Q to find an assignment which gives the minimum makespan we can conclude that the GSP problem is only polynomially harder than GSDP.

5.6 Limitations of the Conversion

The conversion makes use of an execution time matrix as a basis for solving Grid scheduling; the idea of using such a given as the benchmark input for scheduling algorithms in [BR2001]. The execution time matrix (E), can be used to model resources which are heterogeneous in processing speed, local load and operating system and tasks which vary in size since, depending on the method used to derive the values within it were calculated. We can also model task QoS such as investigated in [XI2003] since we can ensure that tasks requiring high QoS cannot be assigned to low QoS machines. More complex QoS based scheduling problems such as the one described in [GOL2004], which allows for multiple QoS dimensions that are both hard and soft, could be modelling by restricting which machines the jobs can be allocated to and by multiplying E by utility functions such as those given in [GOL2004].

However there are more complex Grid scheduling problems such as when tasks are considered to be dependant, which can not be modelled using this conversion. This is because the estimated execution time for each task can potentially be altered by the decision of which machine each other job is allocated to. It is unlikely that such a can be modelled effectively through any conversion to SAT since the instance would be exponentially large.

5.7 Conclusions and Further Work

This chapter demonstrates how GSPD can be modelled as a SAT instance and hence how the optimisation variant can also be solved using SAT solvers. This conversion is certainly not optimal, especially as it contains a number of one literal clauses which could be removed along with all clauses with those literals in and all the opposite literals.

Careful consideration should be taken when selecting which solver to use as it must be ensured that the runtime of the entire procedure is acceptably low so as not to waste too much processing time before the tasks are allocated. Another consideration is that one must realise that the SAT algorithms will only provide a "well informed guess" as to whether the instance can be solved or not, since there is no known algorithm to solve SAT in polynomial time, therefore when performing the first stage of the algorithm (to determine the minimum possible makespan), one should be willing to accept a value which is close to the minimum rather than the true minimum. Further work is required to calculate which value should be accepted and how this affects the operation of the second stage of the algorithm. For example it could be that some algorithms will be able to determine that the minimum makespan is at most some value, but, the second stage of the algorithm is unable to determine an allocation which provides this makespan.

Chapter 6 – Conclusions and Further Work

6.1 Introduction

This thesis is concerned with the subjects of Grid Scheduling and Grid Simulation.

The ultimate goal of Grid computing is to allow all kinds of distributed resources to be seamlessly utilised on a world-wide scale, providing access to computing power and specialised resources that would otherwise be impossible. One problem preventing this vision is Grid scheduling; how to best distribute the tasks amongst the resources. Even in its most basic form, it is thought to be practically unsolvable. The geographically distributed nature of the resources and the fact that they are owned by numerous organisations, all of whom will employ different usage policies make Grid scheduling problem more complex than traditional scheduling, which has been the subject of much research. As a result, many of the procedures currently employed are insufficient, and there is a need to develop more robust and more efficient solutions. This work includes the development of a simple, novel scheduling technique which could be used to improve Grid scheduling algorithms in numerous scenarios. In addition, this work documents a conversion between the Grid scheduling problem and the classic computing problem SAT. Such a conversion allows for the possibility of applying sophisticated SAT solving procedures to Grid scheduling providing potentially effective solutions.

The process of developing a Grid scheduling algorithm involves testing to verify its effectiveness. The nature of real Grids make experiments on them extremely difficult to carry out and experimenting using simulated Grid environment is the only viable way to produce scientifically valid results. Simulating a Grid involves modelling the grid resources (machines, data vaults, and specialist equipment), the network under which they communicate, the users using the system, the tasks which they are submitting and the scheduling system which describes how the jobs are distributed. The simulated Grid must accurately describe the real Grid, be able to be simulated within a reasonable time frame and be easy to set up and use. This thesis describes

the development of a simulation tool, G-Sim, which can be used to test the effectiveness of potential Grid scheduling algorithms under realistic operating conditions.

The rest of this chapter looks at whether the work described in this thesis concurs with the work proposed in section 1.2 and presents conclusions about the work. Finally there is a discussion for further work.

6.2 Evaluation of Proposed Work

Section 1.2 presents a list of proposed work, this section evaluates the work within this thesis against this list.

- **Develop a simulation tool allowing simulation of large-scale, realistic Grids on a stand-alone PC**

To this end the G-Sim toolkit was developed. Its design and implementation are documented in chapter 3. This software provides the primitives for creating virtual Grids with any number of heterogeneous Grid resources running under either time or space shared allocation policies with various local load models, users and their applications and the network under which the distributed Grid entities communicate. The experiments documented in chapter 4, some of which employ relatively large scale Grids, were all run on a Pentium 4 running at 1800MHz and each experiment took no more than 20 minutes.

G-Sim's classes are to be fully extendable to ensure all types of scenario can be modelled, this means researches are able to model different resource types, users, tasks, local resource loads and communication networks. The downside of this is that in order to make use of the program, the user must have knowledge of both Java programming and specifically the operation of the toolkit.

- **Add to the understanding of how scheduling algorithms are affected by the properties of the Grid they schedule on through analysis of experimental results**

Within chapter 4, a comprehensive analysis of five scheduling algorithms is presented; Minimum Completion Time, Min-Min, QoS Guided Min-Min, Time Limit Min-Min and Time-Limit QoS Guided Min-Min. The performance of the algorithms is recorded over numerous Grid environments so as we can see the effect each individual property has upon the performance. The results show many trends that would be trivially expected, such as the makespan increases as the number of tasks increases and the makespan is inversely proportional to the number of processors. However they also show more subtle patterns such as, that making an algorithm QoS guided, only significantly improves performance when half the tasks require high QoS and when the level of error in estimation is relatively low. Chapter 4 includes a more in depth analysis of the results. In general, Time-Limit batch mode offered the greatest improvement in performance under the following circumstances:

1. Scheduling wait time was relatively small
2. The Grid size was relatively small in comparison to the number of tasks
3. The resources in the Grid operated at roughly the same speed
4. There was potentially a large discrepancy between the predicated and actual execution times of tasks
5. The amount of local load varied wildly

It was clear from the results and the difference from those published in [XI2003] that even minor changes to properties of the Grid selected can dramatically affect the relative performances of the algorithms. This demonstrates a need for Grid scheduling algorithm designers to adopt a set of standardised benchmark tests against which their procedures should be judged, much like those employed by the SAT solving community (see [SI2005]).

Having shown how changes in the properties of the Grid scenario can affect these five algorithms, it appears that each is affected slightly differently depending on the nature of the property being changed. Given this, it is likely that it will be difficult if not

impossible to determine, given a set of algorithms, which will be the most effective for a particular scenario without experimentation.

- **Add to the catalogue of techniques available to Grid scheduling algorithm designers**

To satisfy this proposal a simple novel scheduling mode, Time-Limit batch mode, was developed and tested in chapter 4 and a conversion to the well known problem SAT was developed in chapter 5. Time-Limit batch mode is briefly discussed in the previous section and is likely to improve the performance of scheduling algorithms in many scenarios. However, implementing it for a real application is a non-trivial task as the specifics of the implementation will be dependant on how data intensive the application and the type and operating policy of the resources available.

The conversion to SAT allows for the possibility of solving numerous forms of the Grid scheduling problem using SAT solving algorithms. Such algorithms have recently seen significant advancement [SI2001] and have been shown to solve hard instances with around a thousand variables. Although this has not yet been tested such an approach has been successful in solving other NP-Hard problems most notably planning [KA1996] [BU2003]. The conversion could be used to solve instances involving resources which are heterogeneous in processing speed, local load and operating system and tasks which vary in size and QoS constraints. However there are more complex Grid scheduling problems such as when tasks are considered to be dependant, which can not be modelled using this conversion.

6.3 Further Work

A number of issues requiring further research have presented themselves from the work in this thesis. Firstly, the need to develop the idea of Time-Limit Batch Mode to enable it to be deployed on real Grid applications. This would involve investigating how the implementation of the method should be adapted to different algorithms, application types and Grid scenarios. Furthermore, additional research to identify

which situations are likely to benefit from the inclusion of the method over traditional batch and online scheduling is required if it is to be ultimately useful.

Research into the effectiveness and applicability of the conversion to SAT presented in chapter 5 is also required if it is to be used for scheduling real Grid applications. This would involve finding which SAT solvers are the most effective on the generated instances and the number of iterations of the first stage of the algorithm (to determine the minimum possible makespan), in order to ensure the execution time is acceptable.

This work also demonstrated a need for the development of a set of standardised benchmark tests against which their procedures should be judged. This would require significant further study and collaboration on the part of the Grid scheduling community to make them inclusive enough to model all types of Grid scenario whilst providing a useful means of comparison between the algorithms.

References

[AB2000] An Economy Driven Resource Management Architecture for Global Computational Power Grids. R. Buyya, D. Abramson and J. Giddy. Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, June 26-29, 2000, Las Vegas, USA, IEEE CS Press, USA, 2000.

[AB2002] Economic models for resource management and scheduling in Grid computing. Rajkumar Buyya, David Abramson, Jonathan Giddy and Heinz Stockinger. Concurrency and Computation: Practice and Experience, 2002, number 14, pp. 1507-1542.

[AF2005] Grid Resource Broker using Application Benchmarking. Enis Afgan, Vijay Velusamy, Purushotham and V. Bangalore. Proceedings of the European Grid Conference, February 14 - 16, 2005.

[AH1996] Speedup and accuracy versus timing granularity. Jong-Suk Ahn and Peter B. Danzig. IEEE/ACM Transactions on Networking, Volume 4, number 5, October 1996, pp. 742-757.

[AH2001] A Brief History of the Internet. Natalie Ahn, Julie Black, Jonathan Effrat. Research project for the class of The Intellectual Excitement of Computer Science, Stanford University Computer Science Department. http://cse.stanford.edu/class/sophomore-college/projects-01/distributed-computing/html/history.html accessed on 15th May 2005.

[AI2000] Performance Evaluation for Scheduling in a Global Computing System. Kento Aida, Atsuko Takefusa, Hidemoto Nakada, Satoshi Matsuoka, Satoshi Sekiguchi and Umpei Nagashima. International Journal of High Performance Computing Applications, Vol. 14, No. 3, 2000, pp. 268-279.

[AS2000] Analysis of the GUTS Network Model for Distributed Systems Applications. Omar M. Asad and Kyle A. Farlow. 14th December 2000.

Unpublished. www.duke.edu/~kaf3/works/guts.ps accessed on 24th May 2005.

[BA1996] An Introduction to Discrete Event Simulation. Peter Ball, Strathclyde University. 2nd DYCOMANS workshop on "Management and Control : Tools in Action" in the Algarve, Portugal. 15th - 17th May 1996, pp. 367-376.

[BE1996] Application level scheduling on distributed heterogeneous networks. Berman, F., Wolski, R., Figueria, S., Schopf, J., and Shao, G. In Proceedings of Supercomputing, November 1996.

[BL2004] Coordinating Workflows in Shared Grid Environments. Jim Blythe, Yolanda Gil and Ewa Deelman. Workshop on Planning and Scheduling for Web and Grid Services held in conjunction with The 14th International Conference on Automated Planning and Scheduling , (ICAPS 2004), Whistler, British Columbia, Canada, June 3-7 2004

[BR2000] Advances in Network Simulation. Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John S. Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu and Haobo Yu. IEEE Compute, Volume 33, 2000, pp.59-67.

[BR2001] A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Tracy D. Braun, Howard Jay Siegel, Noah Beck, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao and Richard F. Freund. Journal of Parallel and Distributed Computing, volume 61, 2001, pp. 810-837.

[BU2000] Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. Buyya, R., Abramson, D. and Giddy J. HPC Asia 2000, May 14-17, 2000, pp. 283-289, Beijing, China.

[BU2001] A case for economic Grid architecture for service-orientated Grid computing. Buyya R, Abramson D, Giddy J. 10th IEEE International Heterogeneous

Computing Workshop (HCW 2001), in conjunction with IPDPS 2001, San Francisco, California, USA, April 2001.

[BUY2001]World Wide Grid testbed. Buyya R. June 2001. http://www.buyya.com/ecogrid/wwg/ accessed on 9th March 2006.

[BU2002] Economic-based Distributed Resource Management and Scheduling for Grid Computing. Rajkumar Buyya PhD Thesis, School of Computer Science and Software Engineering, Monash University, Melbourne, Australia. April 12 2002.

[BU2003] Applying Constraint Satisfaction Problems to AI Planning Problems. Daniel Buettner. MSc. Thesis, supervised by Professor Berthe Choueiry, The Graduate College at the University of Nebraska. December, 2003.

[BU2004] A Grid Simulation Infrastructure Supporting Advance Reservation. Anthony Sulistio and Rajkumar Buyya. Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems, November 9-11, 2004, MIT, Cambridge, USA, pp. 1-7.

[BU2005] Scheduling Parameter Sweep Applications on Global Grids: A deadline and Budget Constrained Cost-Time Optimisation Algorithm. Rajkumar Buyya, Manzur Murshed, David Abramson and Srikumar Venugopal. Software—Practice & Experience, Volume 35, Issue 5, April, 2005, pp. 491 – 512.

[BY2002] GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing. Rajkumar Buyya and Manzur Murshed. Concurrency and computation: practice and experience, volume 14, 2002, pp. 1175–1220.

[CA1997] NetSolve: A Network Server for Solving Computational Science Problem. H. Cassanova and J. Dongarra. International Journal of Supercomputing Applications and High Performance Computing, Volume 11, Number 3, 1997, pp. 212-223.

[CA2000] Heuristics for scheduling parameter sweep applications in Grid

environments. Henri Cassanova, Arnaud Legrand, Dmitrii Zagorodnov and Francone Berman. Proceedings of the 9th Heterogeneous Computing Workshop, Cancun, Mexico, 2000, pp. 349-363.

[CA2001] SimGrid: A toolkit for the Simulation of Application Scheduling. H. Cassanova. The 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, Brisbane, Australia, 2000.

[CA2003] UK Grid Simulation with OptorSim. David G. Cameron, Ru'ben Carvajal-Schiaffino, A. Paul Millar, Kurt Stockinger, Caitriana Nicholson, Floriano Zini. UK e-Science All Hands Meeting 2003.

[CA2005] Network Modelling Issues for Grid Application Scheduling. Henri Casanova. International Journal of Foundations of Computer Science (IJFCS), Volume 16, Number 2, April 2005, pp. 145-162.

[CH2004] Condor services for the Global Grid: Interoperability between Condor and OGSA. C. Chapman, P. Wilson, T. Tannenbaum, M. Farrellee, M. Livny, J. Brodholt, and W. Emmerich. Proceedings of the UK e-Science All Hands Meeting 2004, pp 870-877.

[CO1971] The Complexity of Theorem Proving Procedures. Stephen Cook. Proceedings of the third annual ACM symposium on Theory of computing, p. 151–158, 1971.

[CO1999] Modelling 100,000 Nodes and Beyond: Self-Validating Design. James H. Cowie, David M. Nichol and Andy T. Ogielski. DARPA/NIST Workshop on Validation of Large Scale Network Simulation Models, May 25-29, 1999 Reston, VA.

[CO2001] Computer Networks and Internets with Internet Applications, Third Edition. Douglas E. Comer. Prentice Hall, 2001.

[CO2004] New Grid Scheduling and Rescheduling Methods in the GrADS Project. K. Cooper, A. Dasgupata, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina,

J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan and J. Dongarra. NSF Next Generation Software Workshop, International Parallel and Distributed Processing Symposium, Santa Fe, April 2004.

[DA2002] A decoupled scheduling approach for the GrADS program development environment. Holly Dail and Henri Casanova and Fran Berman. Proceedings of the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland, IEEE Computer Society Press, 2002, pp. 1-14.

[DE2003] Mapping Abstract Workflows onto Grid Environments. Ewa Deelman, Jim Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Journal of Grid Computing, Vol. 1, No. 1, 2003.

[DO2002] On QoS-Based Scheduling of a Meta-Task with Multiple QoS Demands in Heterogeneous Computing. Atakan Dogan and Füsun Özgüner. Proceedings of the 16th International Parallel and Distributed Processing Symposium, IEEE Computer Society Washington, DC, USA, pp. 227, 2002

[EC2006] The Scientific Method, Computing the Future. The Economist. March 23rd 2006.

[ED2005] EDGSim: A simulation of the European DataGrid. www.hep.ucl.ac.uk/~pac/EDGSim accessed on 5th July 2005.

[FE2002] James Frey, Todd Tannenbaum, et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Journal of Cluster Computing, volume 5, pp. 237-246, 2002.

[FO1998] The Grid: A Blueprint for a New Computing Infrastructure. Ian Foster and Carl Kesselman. Morgan Kaufmann, September 1998, pp 17 -51.

[FO1999] A Distributed Resource Management Architecture that Supports Advance Reservation. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt and A. Roy. International Workshop on Quality of Service, London, U.K., 1999, pp. 268-279.

[FO2002] The Physiology of the Grid: A Open Grid Services Architecture for Distributed Systems Integration. Foster, I., Kesselman, C., Nick, J., and Tuecke, S. Globus project, 2002. http://www.globus.org/research/papers/ogsa.pdf accessed on 2nd June 2005.

[GA1979] Computers and Intractability: a Guide to the Theory of NP-completeness. M. R. Gary and D. S. Johnson. W.H. Freeman, New York, 1979, pp.65, pp. 238-240.

[GA2002] Coarse-Grained Network Simulation for Wide-Area Distributed Systems. Syam Gadde, Jeff Chase and Amin Vahdat. Proceedings of Communication Networks and Distributed Systems Modeling and Simulation, January 2002.

[GD2005] Grid Scheduling Dictionary of Terms and Keywords. Wolfgang Ziegler and Philip Wieder. Global Grid Forum, Scheduling and Resource Management Area. http://www.fz-juelich.de/zam/RD/coop/ggf/sched-sd.html accessed on 29th May 2005.

[GF2005] Global Grid Forum. http://www.gridforum.org accessed on 12th May 2005.

[GL2005] The Globus Alliance. http://www.globus.org/ accessed May 20th 2005.

[GO2002] Performance Modelling and Prediction of Nondedicated Network Computing. Linguo Gong, Xian-He Sun and Edward F. Watson. Transactions on Computers, Volume 51, No.9, September 2002.

[GO2004] Constructing A Grid Simulation with Differentiated Network Service Using GridSim. Anthony Sulistio, Gokul Poduvaly, Rajkumar Buyya, and Chen-Khong Tham. Technical Report, GRIDS-TR-2004-13, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Dec 9, 2004.

[GOL2004] A Comparison of Static QoS-based Scheduling Heuristics for a Meta-Task with Multiple QoS dimensions in Heterogeneous Computing. Kavotha S. Golconda, Füsun Özgüner and Atakan Doğan. 18th International Parallel & Distributed Processing Symposium, April 26-30th 2004.

[HE2004] Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04) - Volume 00, 2004, pp. 402 – 409.

[HY2004] Welcome and Introduction. Tony Hey. Proceedings of the UK e-Science All Hands Meeting. Sept 2004 pp 4-6.

[IB2001] IBM joins on life sciences "grid". IBM News – United States 2001-11-14 Nonprofit Group. http://www.ibm.com/news/us/2001/11/14.html accessed on 20th May 2005.

[IN2005] Moore's Law, The Future – Technology and Research at Intel. Intel Corporation 2005. http://www.intel.com/technology/silicon/mooreslaw/ accessed on 12th May 2005.

[INF2005] Sun Delays Grid Rollout. Robert McMillan, IDG News Service. May 9th 2005. InfoWorld News, 2005. http://www.infoworld.com/article/05/05/09/19NNsungrid_1.html accessed on 20th May 2005.

[JA2005] Java 2 Platform Standard Edition 5.0 API Specification http://java.sun.com/j2se/1.5.0/docs/api/ accessed on 15th July 2005.

[JI2003] A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services. Jia Yu, Srikumar Venugopal, and Rajkumar Buyya. Technical Report, GRIDS-TR-2003-0, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia, January, 2003.

[KA1996] Pushing the envelope: Planning Propositional Logic and Stochastic Search. Kautz, H. and Selman, B. In Proceedings of National Conference on Artificial Intelligence. 1194-11201. AAAI Press, 1996.

[KA2005] Kazaa. Sharman Networks, 2002-2005. http://www.kazaa.com/us/index.htm accessed on 12th October 2005.

[LA2002] Dynamic Replication Strategies in Grid Environments. H. Lamehamedi, B. K. Szymanskim Z. Shentu and E. Deelman. Proceeding f the ICAP '03, Beijing, China, October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.

[LA2003] Simulation of Dynamic Data Replication Strategies in Data Grids. H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman. Proc. 12th Heterogeneous Computing Workshop (HCW2003) Nice, France, April 2003, IEEE Computer Science Press, Los Alamitos, CA, 2003.

[LE1996] M. J. Lewis and A. Grimshaw. The core legion object model. In Proceedings of The Fifth IEEE International Symposium on High Performance Distributed Computing. IEEE Computer Society Press, August 1996.

[LE2003] Ant colony optimisation and local search for bin packing and cutting stock problems. J. Levine and F. Ducatelle. Journal of the Operational Research Society, Special Issue on Local Search, Volume 55, Number 7, July 2004, pp.705-716.

[LI1988] Condor - A Hunter of Idle Workstations. Litzkow, M., M. Livny, and M. Mutka. In Proceedings of 8th International Conference of Distributed Computing Systems, June 1988, pp. 104-11.

[LI2003] Grid Scheduling. Yan Lui. Department of Computer Science, University of Iowa, PhD. Qualifying paper, 2003.

[MA1999] Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel,

Debra Hensgen, and Richard F. Freund. Journal of Parallel and Distributed Computing, Special Issue on Software Support for Distributed Computing, 1999.

[MA2003] Problems in Artificial Intelligence, lecture notes. Florent Madelaine. 3rd Semester teaching, Durham University, 2003. http://www.dur.ac.uk/f.r.madelaine/local/pai.html accessed on 20th Jume 2005.

[MI1983] Statistics for Advanced Level. J.C. Miller. Cambridge University Press, 1983.

[MI1986] Distributed Discrete Event Simulation. Jayadev Misra. ACM Computing Surveys (CSUR), Volume 18, Issue 1, March 1986, pp. 39-65.

[MI2006] Towards 2020 Science. Report from Towards 2020 Science workshop, Venice, 30 June-1 July 2005. http://research.microsoft.com/towards2020science/downloads/T2020S_ReportA4.pdf accessed 28th March 2006.

[MU2002] GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing. Rajkumar Buyya and Manzur Murshed. Concurrency and Computation: Practice and Experience, volume 14, 2002 pp.1175–1220.

[NA2002] NAM: Network Animator. http://www.isi.edu/nsnam/nam/ accessed on 6th July 2005.

[NS2005] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns/accessed on 28th May 2005.

[NU1999] PACE: A Toolkit for the Performance Prediction of Parralleland Distributed Systems. G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, J. S. Harper, S. C. Perry and D. V. Wilcox. International Journal of High Perfomance Computing, 1999.

[OB2000] The AppLeS parameter sweep template: User-level middleware for the Grid. Henri Cassanova, Graziano Obertelli, Francine Berman and Rich Wolski. In Proceeding of the Super Computing Conference (SC'2000), Dallas, Texas, Nov. 2000.

[OG2005] The Open Grid Services Architecture Version 1.0. Foster et al. 29th January 2005. Global Grid Forum documents GFD.30. http://www.ggf.org/documents/GWD-I-E/GFD-I.030.pdf accessed on 20th May 2005.

[OP2003] Open Issues in Grid Scheduling. Report of the workshop held at the e-Science Institute, October 21st -22nd 2003. http://www.nesc.ac.uk/technical_papers/UKeS-2004-03.pdf accessed on 29th May 2005.

[OP2004] OpenMolGRID, a GRID based system for solving large-scale drug design problems. Darvas F., Papp Á., Bágyi I., Ambrus-Aikelin G., Urge L., edited by Dikaiakos. Lecture Notes in Computers Sciences: Grid Computing, vol. 3165, pp69-76, 2004.

[PH2002] The Architecture and Implementation of a Grid Resource Manager. Sugree Phatanapherom, Putchong Uthayopas and Somsak Sriprayoonsakul. Proceeding of ANSCSE 6th, NakhonSiThammarat, Walailuk University, Thailand, 3-5 April, 2002.

[PI1995] Scheduling theory, algorithms and systems, first edition. Prentice Hall. 1995.

[PU2003] Fast Simulation Model For Grid Scheduling Using HyperSim. Sugree Phatanapherom and Voratas Kachitvichyanukul. Proceeding of the 2003 Winter Simulation Conference.

[RA1998] PARSEC: A Parallel Simulation Environment for Complex Systems. Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu an Chen, Xiang Zeng, Jay Martin and Ha Yoon Song. IEEE Computer, Volume 31, October 1998, pp. 77-85.

[RA2002] Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. Kavitha Ranganathan and Ian Foster. 11th IEEE International Symposium on High Performance Distributed Computing, 2002, pp. 352.

[RE2004] Resource constrained scheduling on multiple machines. Jan Remy. Information Processing Letters 91, August 2004, pp. 177-182.

[RI2002] Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. Matei Ripeanu and Ian Foster and Adriana Iamnitchi, edited by Li Gong. IEEE Internet Computing Journal, Volume 6, number 1, August 2002.

[RI2003] A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. Graham Ritchie and John Levine. Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG). 2003, pp 178-183.

[SE2003] SETI@Home: Search for Extraterrestrial Intelligence. Space Sciences Laboratory, University of California Berkley, SETI@Home 2003. http://setiathome.ssl.berkeley.edu/ accessed on 20th May 2005.

[SH1983] Simulation Modelling with Event Graphs. L. Schrubben. Communications of the ACM, volume 26, pp. 957-963.

[SH2001] Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources. Gary Shao. Ph.D. thesis, University of California, San Diego, May 2001.

[SH2004] Libra: a computational economy-based job scheduling system for clusters. Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayta and Rajkumar Buyya. Software - Practice and Experience, Volume 34, issue 6, May 2004, pp. 573-590.

[SI2001] The SAT-Ex Site: The experimentation web site around the satisfiability problem. Laurent Simon, 2000-2001. http://www.lri.fr/~simon/satex/satex.php3 accessed on 30th March 2006.

[SI2002] Making SimJava Count. Costas Simatos. MSc. Project report, University of Edinburgh, Sept. 12th, 2002.

[SI2004] A simple MPI process swapping architecture for iterative applications. O. Sievert and H. Cassanova. The International Journal of High Performance Computing Applications, 2004, to appear.

[SO2000] The MicroGrid: a scientific tool for modelling computational grids. H. J. Song et al. Conference on High Performance Networking and Computing archive, Proceedings of the 2000 ACM/IEEE conference on Supercomputing, Dallas, Texas, United States, article 53, 2000.

[SP2003] Local Grid Scheduling Techniques using Performance Prediction. D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini and G. R. Nudd. IEEE Proc. Comp. Digit. Tech., Nice, France, Volume 150, Number 2, April 2003, pp. 87-96.

[SU1994] Java Bean 1.01 Specification. Sun Microsystems, 1994. http://java.sun.com/products/javabeans/docs/spec.html accessed on 22nd Auguest 2005.

[SU2003] GHS: A performance prediction and task scheduling system for Grid computing. Sun Xian-He and Wu Ming. Proceedings of the 2003 IEEE International Parallel and Distributed Processing Symposium, Session 6, pp.73-235.

[SU2004] A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. Anthony Sulistio, Chee Shin Yeo and Rajkumar Buyya. Software – Practice and Experience, Volume 34, Issue 7, June 2004, pp. 653 – 673.

[SU2005] Sun Grid Overview. Sun Microsystems Inc., 2005.

http://www.sun.com/service/sungrid/overview.jsp accessed on 20th May 2005.

[TH2005] Tom's Hardware Guide Peripherals & Consumer Electronics Creative's X-Fi: A New Age in Sound Card Power? TG Publishing AG 1996 - 2005. http://www.tomshardware.com/consumer/20050516/ accessed on 10th October 2005.

[VU2005] An informal talk on VirtU: Virtual Universe. Dr. Adrian Jenkins. University of Durham, e-Science Institute, 1st February 2005.

[WA1997] Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Journal of Parallel Distributed Computing, volume 47, issue 1, Nov. 1997, pp. 1-15.

[WA2004] Job Scheduling in a Heterogeneous Grid Environment. Hongzhang Shan, Warren Smith, Leonid Oliker and Rupak Biswas. eScholarship Repository, University of California. http://repositories.cdlib.org/lbnl/LBNL-54906 accessed on 29th March 2006.

[WA2005] Dijkstra's Shortest Path Algorithm in Java. Renaud Waldura. http://renaud.waldura.com/doc/java/dijkstra/ accessed 1st August 2005.

[WE1999] Binomial Distribution. Eric W. Weisstein. From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/BinomialDistribution.html accessed on 29th July 2005.

[WEI1999] Exponential Distribtion. Eric W. Weisstein. From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/ExponentialDistribution.html accessed on 19th August 2005.

[WE2000] An Introduction to Quantum Computing. Jacob West. Academic Division of Computer Science, Californian Institute of Technology, 28th April 2000. http://www.cs.caltech.edu/~westside/quantum-intro.html accessed on 12th May 2005.

[WI2005] Million Instructions per Second – Wikipedia, the free encyclopaedia. Wikipedia. http://en.wikipedia.org/wiki/Million_instructions_per_second accessed on 12 October 2005.

[WO1999] The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Rich Wolski, Neil Spring, and Jim Hayes. Journal of Future Generation Computing Systems,Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

[XI2003] QoS Guided Min-Min Heuristic for Grid Task Scheduling. HE Xiaoshan, Xian-He Sun and Gregor von Laszewski. Journal of Computer Science and Technology, volume 18 , issue 4, July 2003, pp. 442 – 451.

[YU2003] A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Service. Jia Yu, Srikumar Venugopal, and Rajkumar Buyya. Technical Report, GRIDS-TR-2003-0, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia. January 2003.

[ZH2004] A Survey on Grid Scheduling Systems. Yanmin Zhu. PhD. Qualifying Paper, Department of Computer Science Hong Kong University of Science and Technology, 2004.

Appendix

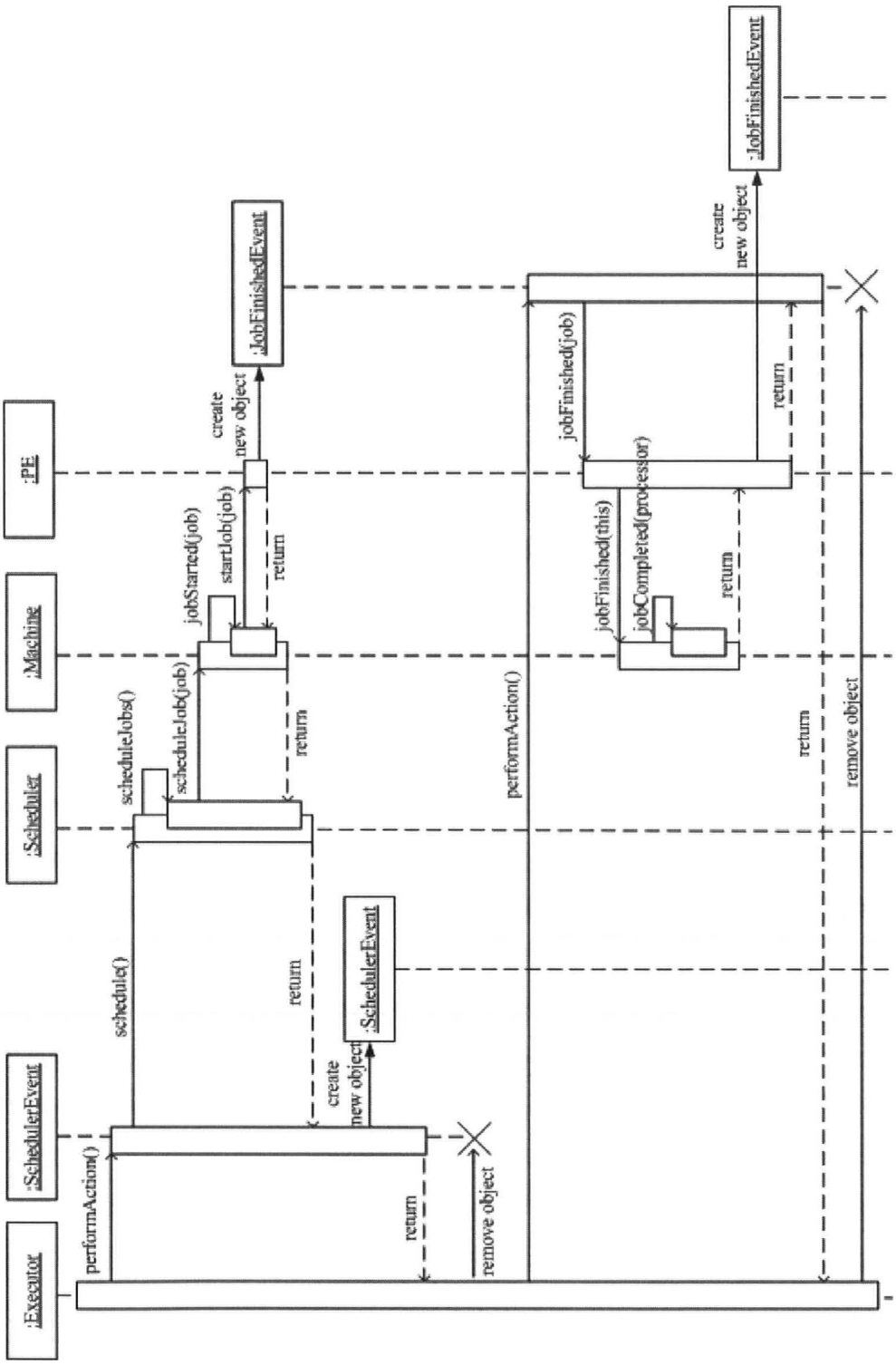

Figure A1: Sequence diagram showing the context of the invocation of the Machine class's abstract methods

```
1.  begin byte randomise(byte average)
2.          if maxAmountOut > 50
3.                  indicate invalid use of method
4.          end if
5.          if average == 0
6.                  return 0
7.          end if
8.          if average == 100
9.                  return 100
10.         end if
11.         if average - maxAmountOut < 0
12.                 x = calcLowXValue(average,average+maxAmountOut)
13.                 levels = 0
14.                 if x is a whole number
15.                         levels = average+maxAmountOut+x
16                  end if
17.                 else
18.                         x = x*2
19.                         levels = 2*(average+maxAmountOut)+xx
20.                 end if
21.                 rand = Math.random()
22.                 if rand <= x/levels
23.                         return 0
24.                 end if
25.                 else
26.                         rand = Math.random()
27.                         increase = Math.round(rand * (maxAmountOut+average))
28.                         if increase == maxAmountOut+average
29.                                 return 1
30.                         end if
31.                         return increase + 1
32.                 end else
33.         end if
34.         else if average + maxAmountOut > 100
35.                 x = calcBigXValue(average,average-maxAmountOut)
36.                 levels = 0
37.                 if x is a roundNumber
38.                         levels = 100-(average-maxAmountOut)+x
39.                 end if
40.                 else
41.                         x = x*2
42.                          levels = 2*(100-(average-maxAmountOut))+x
43.                 end else
44.                 rand = Math.random()
45.                 if rand <= x/levels
46.                         return 100
47.                 end if
48.                 else
49.                         rand = Math.random()
50.                         increase = Math.round(rand * (100-(average-maxAmountOut)))
51.                         if increase + (average-maxAmountOut) == 100
52.                                 return average-maxAmountOut
53.                         end if
54.                         return increase + (average-maxAmountOut)
55.                 end else
56.         end else
57.         rand = Math.random()
58.         increase = Math.round(rand *(2*maxAmountOut+1))
59.         if average-maxAmountOut + increase > average+maxAmountOut
60.                         return average-maxAmountOut
61.         end if
62.                 return average-maxAmountOut + increase
63.         end if
64. end randomise
```

Figure A2: Pseudo code for the randomising algorithm used in G-Sim's EILR class