SNAP-based Grid Resource Broker using the Three-phase Commit Protocol

by

Mohammed Hassan Abdulla Haji

Submitted in accordance with the requirements for the degree of Doctor of Philosophy

University of Leeds
School of Computing

September 2005

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Abstract

Grid computing is diverse and heterogeneous in nature, spanning across multiple domains where resources are not owned or managed by a single administration. This brings about many challenges to Grid resource management and exposes the user to the Grid middleware complexities. Thus this research develops a user-centric resource broker that insulates the users from the Grid complexities, alleviating them from the burden of having to know the various mechanisms of the Grid middleware. The broker is based on the SNAP (Service Negotiation and Acquisition Protocol) framework and focuses on applications that require resources on demand.

It is important for applications that require resources on demand to reserve the necessary resources within the minimum time possible. Thus the work in this thesis has developed a three-phase commit protocol which enhances the traditional two-phase commit protocol. Performance evaluation has been carried out to evaluate the SNAP-based resource broker using the traditional two-phase commit protocol and the newly developed three-phase commit protocol. The evaluation has been conducted on a local Grid test-bed, a distributed Grid infrastructure (the White Rose Grid) and through mathematical modelling and simulation. Throughout the evaluation, the SNAP-based resource broker using the three-phase commit protocol provides a significant performance enhancement, over the use of the traditional two-phase commit protocol, in terms of the time taken between submission (to the broker) of user requirements and the job beginning execution.

Declaration

Some parts of the work presented in this thesis have previously appeared in the following papers:

- **M. Haji**, P. Dew, K. Djemame and I. Gourlay. *A SNAP-based Community Resource Broker using a Three-Phase Commit Protocol*. In Proceeding. of 18th IEEE International Parallel and Distributed Processing Symposium. Santa Fe, NM, USA, April 26–30 2004, pp 56–65, IEEE Computer Society Press.

- K. Djemame, I. Gourlay, **M. Haji**, A. Othman, and J. Padgett. *Resource Management on the White Rose Grid: Current Research Efforts***.** In Proceedings of the 20th UK Performance Engineering Workshop (UKPEW'2004), I.U. Awan (Editor), pp 188-197, Bradford, UK, July 2004.

- I. Gourlay, **M. Haji**, K. Djemame and P. Dew *Performance Evaluation of a SNAP-based Grid Resource Broker*.. In Proceedings of FORTE'2004 Workshops (1st European Performance Engineering Workshop), M. Nunez, A. Maamar, F.L. Pelayo, K. Pousttchi and F. Rubio (Eds.), Toledo, Spain, September 2004, Lecture Notes in Computer Science Vol 3236 pp 220-232, Springer.

- K. Djemame, **M. Haji**, and J. Padgett. *SLA Management in a Service Oriented Architecture*. In Proceedings of International Conference of Computational Science (ICCSA). May 2005. Singapore: Lecture Notes in Computer Science, Vol 3483. pp 1282-1291.

- **M. Haji**, I. Gourlay, K. Djemame and P. Dew. *A SNAP-based Community Resource Broker using a Three-Phase Commit Protocol: a Performance Study.* Computer Journal, Vol. 48, No. 3, 2005, pp.333-346.

Acknowledgement

I would like to take this opportunity to thank a number of people who have provided the support and guidance to complete this degree. I would like to thank Prof. Peter Dew (my supervisor) and Dr. Karim Djemame (my co-supervisor) for their advice, encouragement and guidance on many aspects relating to my research. Special thanks to Dr. Karim Djemame during the final stage of my PhD studies and write up. I would also like to thank Prof. Ken Brodlie and Dr. Iain Gourlay for their input and extensive discussion about my research. Further I would also like to thank Martin Thompson and John Hodrien for their technical help and support in the development of the SNAP-based resource broker and the three-phase commit protocol.

Many thanks to several members of my research group and department especially Abdulla Othman, James Padgett, Duncan Russell, Zukeri Ibrahim, Tran Vu Pham, Sarfraz Nadeem, George Honore, Paul Townend, Erica Yang, Simon Lessels and Bayan Aref Abu Shawar for their useful discussions about my research as well as other useful intellectual topics, and for providing such stimulating friendly environment to work in. Additional I would also like to thank Mosheer Noor, Iyad Hayel, Hamood Kaid and Khalid Alsayar for being good friends and always willing to talk and discuss a great number of topics.

Finally my heartiest gratitude go to my entire family my mother my brother Nageab Haji my sisters Salwa, Amal, Iftakar and Muna, and last but not least my nephew Hamzah Ghaleb for all the wonderful electronic gadgets that he develops and shows me (one day you will become that Professor) and my nieces Naelah Ghaleb and Esra Ghaleb.

Contents

Figures

Tables

Abbreviations

ACT	Agent Capability Table
AppLeS	Application Level Scheduler
B2B	Business-to-Business
BSLA	Binding Service Level Agreement
CBR	Case Based Reasoning
ClassAds	Classified Advertisements
DAME	Distributed Aircraft Maintenance Environment
EDG	European Data Grid
EU	European Union
G_ACT	Global Agent Capability Table
GARA	Globus Architecture for Reservation and Allocation
GASS	Global Access Secondary Storage
GGF	Global Grid Forum
GIIS	Grid Index Information Service
GRAAP	Grid Resource Allocation Agreement Protocol
GRAM	Grid Resource Allocation Manager
GRB	Grid Resource Broker
GridFTP	Grid File Transfer Protocol
GRIS	Grid Resource Information Service
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
gViz	Grid Visualisation
HPC	High Performance Computers
IP	Information Provider
JDBC-ODBC	Java Database Connectivity – Open Database Connectivity
KB	Knowledge Bank
L_ACT	Local Agent Capability Table
LDAP	Lightweight Directory Access Protocol
LRAM	Local Reservation and Allocation Managers

LS	Local Scheduler
LSF	Load Sharing Facility
MDS	Monitoring and Discovery System
MoSeS	Modelling and Simulation for e-social Science
MPI	Message Passing Interface
MPL	Mentat Programming Language
NBQS	Network Batch Queueing Systems
NWS	Network Weather Service
OGSA	Open Grid Service Architecture
PBS	Portable Batch System
RSL	Resource Specification Language
RSLA	Resource Service Level Agreement
SGE	Sun Grid Engine
SLA	Service Level Agreement
SM	Scheduling Manager
SNAP	Service Negotiation and Acquisition Protocol
SQL	Structured Query Language
SRM	Storage Resource Manager
TCP	Transmission Internet Protocol
TSLA	Task Service Level Agreement
TTL	Time-To-Live
UDP	User Datagram Protocol
WASS	Wide Area Scheduling Systems
WRG	White Rose Grid
XTO	eXtract Tracked Orders

Chapter 1

Introduction

1.1 Research Motivation

Grid computing is diverse and heterogeneous in nature, spanning across multiple domains where resources are not owned or managed by a single administration. This brings about many challenges to Grid resource management [1] such as site autonomy, heterogeneous substrate, and policy extensibility. The Globus [2, 3] middleware toolkit addresses these issues by providing services to assist users in the utilisation of Grid resources. However users are still exposed to the Grid middleware complexities and there is a substantial burden for the users to have extensive knowledge of the various Grid middleware components in order to be able to utilise Grid resources. This ranges from querying the information providers, selecting suitable resources for the user's job, forming the appropriate RSL (Resource Specification Language), submitting the user's job to the resources and initiating the execution.

A central component in Grid computing is resource brokering, insulating the user from the Grid middleware complexities, by performing the task of mapping the user's job requirements to resources that can meet these requirements. This can include searching multiple administrative domains to use a single resource or scheduling a single job to use multiple resources at a single site or multiple sites. As discussed in [4] resource brokering can be classified in two categories: *system-centric* and *user-centric*. A system-centric broker allocates resources based on parameters that enhance system utilisation and throughput. Conversely a user-centric broker such as Nimrod-G [5] adheres to the user requirements, and utility of computation is enhanced compared with system utilisation. A key goal of Grid computing is to deliver utility of computation as defined by the users' requirements, thus Grid resource brokers are focused on providing user-centric services.

The Grid also needs to cater for multiple users from different sites, who may potentially be interested in the same resource, without each other's knowledge of their existence or interests [6-8]. These users could potentially be competing for the same resources which could possibly affect the execution start time of a job, as some resources that have been previously selected for use could be taken by others. Thus it is important to secure resources prior to run time especially for applications that are time critical and require resources on demand such as those used by DAME [9] (Distributed Aircraft Maintenance Environment), where it is essential that jobs start execution with minimal delay.

1.2 Research Context

A user-centric resource broker needs to cater for the user's requirements, ensuring appropriate resources are selected that have the capability to meet the user's requirements. A framework that fits well for this form of brokering is SNAP [10] (Service Negotiation and Acquisition Protocol) as it is inspired by the requirement, in a Grid environment, to reconcile the needs of the users with those of the resource providers. The user's job requirements are examined and resource providers that can support such requirements are identified to cater for the job execution. A Service Level Agreement (SLA) is established to ensure that the user's job will be performed under the specified resource requirements. The use of an SLA ensures that the user knows what the resources can be expected to deliver without requiring any detailed knowledge of local resource provider policies, which the resource provider may not be willing to share.

A user's job may require multiple resources, owned by different providers and using a single SLA across multiple sites may not be possible. SNAP addresses this problem by co-ordinating resource management through the use of a three-tiered framework each encapsulating an SLA. In the first layer, namely Task Service Level Agreement (TSLA), the user's resource requirements are ascertained. This forms the basis of the job's requirement and propagates through to the next two layers. The second is the Resource Service Level Agreement (RSLA) which discovers, selects and nominates the appropriate resources for the job as well as ensures the user has the credentials to use the resources. The third is the Binding

Service Level Agreement (BSLA) which associates the job to the resources for execution.

Thus the SNAP framework is used in this research for the development of a user-centric resource broker that handles jobs which require resources on demand. The broker also uses a three-phase commit protocol, which provides an enhancement over the traditional two-phase commit protocol in terms of the time taken between submission (to the broker) of user requirements and the job beginning execution.

This thesis will evaluate the three-phase commit protocol against the conventional two-phase commit protocol. This will be evaluated through the use of a Grid test-bed, the White Rose Grid (WRG) [11] and through mathematical modelling and simulation.

1.3 Aims and Objectives

The aim of this research is to study resource brokering in the context of computational Grids. This is to insulate the users from the middleware complexities, alleviating them from the burden of having to know the various mechanisms of the Grid middleware. The research applies the DAME XTO (eXtract Tracked Orders) application as an exemplar for the need to secure resources on demand. The purpose of the DAME project is to design and implement a prototype system to facilitate the diagnoses and maintenance of aircraft engines through Grid computing. This is motivated by the need to reduce the cost of unexpected and unplanned maintenance of aircraft engines by processing and diagnosing the problems as they occur, which is why there is a need to secure resources on demand.

With this in mind the objectives of this thesis are:

- The design of a user-centric resource broker architecture to insulate the user from the Grid middleware complexities and to support job submission that requires resources on demand. The current brokering systems have been tailored to adapt to the Grid as they were originally designed to serve other infrastructures, thus suffering from legacy problems. Alternatively they provide limited insulation to the Grid

middleware and require the user to have considerable knowledge of the Grid.

- The development of a three-phase commit protocol to reserve resources on demand. This will be evaluated by comparing it to the traditional two-phase approach through the use of a Grid test-bed to investigate its performance enhancement. Then through a distributed Grid infrastructure, the White Rose Grid (WRG), to validate the performance, verifying it still maintains its enhancement over the traditional approach in a distributed environment. Finally through the use of mathematical modelling and simulation to allow for a larger parameter space and traffic conditions other than those studied through the Grid test-bed and the WRG.

1.4 Major Contributions

The major contributions of this thesis include:

- The design and development of a three-phase commit protocol to submit jobs that require resources on demand. Traditional advanced reservation such as that supported by MAUI [12] cannot cater for such application requirements due to the prior time that it requires to schedule the reservation. The three-phase commit protocol uses probes that provide rapid updates of the resource's status, allowing decision making services to adjust swiftly to the changes. This also provides an enhancement over the conventional approach of gathering resource information and reducing the likelihood of an oscillation situation occurring between a broker and the information providers.

- The design and development of a user-centric resource broker architecture that uses the SNAP framework. This provides a modular layered structure to handle user requirements, and find, select and submit jobs. The broker insulates the user from the Grid middleware complexities and ensures jobs are submitted to the appropriate resources for execution. This is done within the SNAP framework by ascertaining

the user's requirements through a user interface, dynamically gathering resource information, co-allocating a user's job, securing the resources and submitting the job on the user's behalf.

- The proposal of the use of a Knowledge Bank (KB), a data repository that stores static information of resources as attributes which provide a description of their characteristics. This helps in many ways such as supporting automated resource discovery. It facilitates brokers to filter out all resources that can handle a job's needs prior to contacting their information provider avoiding unnecessary processing of resource contacts. Also it alleviates the user from the burden of having to keep a log file of the resources with their associated descriptions. An analogy of the KB is a telephone directory where the information stored directs to a particular service that can cater for a user's needs.

- A performance evaluation of the three-phase commit protocol compared to the traditional two-phase commit protocol. This study is undertaken through the use of a Grid test-bed.

- The deployment of a SNAP-base resource broker using the three-phase commit protocol onto a distributed Grid infrastructure (WRG). This subsequently allowed for the validation of the performance study carried out on the Grid test-bed.

- The performance evaluation through the use of mathematical modelling and simulation, complementing the previous study undertaken through the Grid test-bed and the WRG. This allowed for a wider parameter space and traffic conditions to be considered.

1.5 Thesis Overview

The previous sections have given an introduction to the work presented in this thesis, below summarises how it is organised.

Chapter 2 surveys the background and related work on all major research issues covered in this thesis. First, it describes the challenges in Grid resource management, why it differs from distributed systems and the *de facto* middleware technology used in the Grid, namely Globus. It then outlines major projects that utilise the Grid which leads to the discussion of DAME and its requirements. This is then followed by a survey of Grid resource brokers with their architectures. A review of Grid information providers is provided and finally an overview of the major Grid approaches used to securing resources is presented.

Chapter 3 begins by describing the process of submitting a job directly to a local scheduler and then through the Grid. This is to highlight the complex process a user would endure when submitting a job through the Grid and the need for a Grid resource broker to insulate a user from such problems. Then before the discussion of the SNAP-based (Service Negotiation and Acquisition Protocol) Grid resource brokers architecture, an overview of Grid Service Level Agreements (SLAs) will be presented as well as SNAP. This is followed by a scenario that highlights the necessity to secure resources on the Grid, motivating the need for the three-phase commit protocol.

Chapter 4 begins by providing an overview of the objectives and the experiments carried out on both the Grid test-bed and the WRG. The chapter then describes the Grid test-bed with the experimental results and discussion. This is followed with the WRG description and its experimental results and discussion. The chapter then provides an overall evaluation of both the Grid test-bed and the WRG experiments, which shows that the three-phase commit SNAP broker outperforms the simple SNAP broker (a version that uses the traditional two-phase protocol).

Chapter 5 further evaluates the simple SNAP broker compared to the three-phase commit SNAP broker, through mathematical modelling and simulation. This approach allows for wider traffic conditions to be used than that used in chapter 4. The experiments also show that the three-phase commit SNAP broker still outperforms the simple SNAP broker.

Chapter 6 concludes the thesis and outlines some future work.

Chapter 2

Background and Related Work

This research focuses on computational Grid resource management and resource brokering to insulate the user from the Grid middleware complexities. It also looks into reserving resources prior to submitting application jobs that require resources on demand. This chapter provides the background and research efforts in this field. Section 2.1 discusses Grid resource management, the challenges and the technologies used in the Grid. Section 2.2 outlines various Grid projects as well describing the Distributed Aircraft Maintenance Environment (DAME) project [9]. Section 2.3 reviews existing resource brokers. An overview of resource information providers is given in section 2.4. Section 2.5 reviews resource reservation and finally Section 2.6 summarises the chapter.

2.1 Grid Resource Management

Grid resource management differs from the traditional distributed systems [4, 13, 14] where resources on these systems are centralised, follow a common fabric management policy and are usually homogeneous, serving under a single administration domain. Distributed systems are also designed under the assumption that their administrators have complete control over the resources. Further the administrator can implement the mechanisms and policies necessary for effective use of the resources in order to maximise their utilisation and throughput [4]. However computational Grids, by nature, are a collection of global resources which are decentralised and loosely coupled [14, 15]. They span across multiple administrative domains and geographical boundaries with no absolute central full control of the resources. This introduces several challenges [1, 16] that underlay the construction of computational Grids, which are listed below:

- *Site autonomy*, this refers to the fact that resources are heterogeneous, typically owned and managed by different organisations in different

administration domains. Further, each site is likely to have a different infrastructure, operating system, local resource management system (local scheduler) and its own user policies. Note that local resource management systems and local schedulers are used interchangeably to mean the same thing.

- *Heterogeneous substrate*, this is related to site autonomy, referring to the fact that different sites may use different local scheduling systems such as LSF (Load Sharing Facility) [17], PBS (Portable Batch System) [18] and SGE (Sun Grid Engine) [19]. However even when the same system is used at different sites, local configurations made by the administrator to suit their domain needs often lead to a significant difference in their functionality.

- *Policy extensibility*, as resources on the Grid are drawn from a wide range of domains, each with its own requirements and configurations. A Grid resource management system should support regular developments of new domain-specific management structures. For example it should not restrain sites to a particular scheduler or installation settings.

- *Co-allocation*, many Grid applications have resource requirements such that a single site may not be able to cater for their needs or the application can only be satisfied by using resources simultaneously at several sites. With site autonomy and resources decentralised on the Grid, there is a need for a mechanism that is able to gather information from candidate sites. Then must decide where to submit application jobs (to multiple resources if necessary) and possibly secure the resources before the submission.

There are also other issues relating to Grid resource management. For example a user would possibly have to compete against other candidates for the resources, with limited or stale information as explained by the scenario in Section 3.4.1. Further the Grid is focused on the user's requirements [4] which is perhaps the most important difference to distributed systems where they strive to maximise utilisation and throughput.

2.1.1 Distributed Resource Management Approaches

Conventional distributed resource management systems can be broadly classified into two categories:

- **Network Batch Queueing Systems (NBQS).** These systems focus on managing a set of network resources, such as local clusters or high performance computers.
- **Wide Area Scheduling Systems (WASS).** These systems handle resource management over several sites. They also serve as a module for mapping application jobs to resources and scheduling their execution at different local resource domains.

Both NBQS and WASS systems lack the full necessities to facilitate global resource management such as that needed in the Grid as will be discussed in the following two sub-sections.

2.1.1.1 Network Batch Queueing Systems

There are many NBQS such as PBS, LSF, SGE and LoadLeveler [20], which generally handle user submitted jobs by assigning resources from a network pool of computers. A user can either explicitly characterise his/her job, through the use of a descriptive control language, supplied by the NBQS or implicitly by selecting a queue to which a request is submitted for processing.

These systems are typically designed for a single administrative domain making them autonomous to a single site. They also suffer from heterogeneous substrate problems due to the fact that they assume they are the only resource management system in operation. Further co-allocation is not usually provided in these systems.

Overall NBQS alone do not provide a complete solution to Grid resource management. However these systems will necessarily be part of a local resource management solution and a Grid resource management architecture should be able to interface to these systems.

2.1.1.2 Wide Area Scheduling Systems

WASS systems usually cross over several sites and are diverse in the methods they adopt to handle resource management compared to NBQS. In order to provide a prospective overview of how they operate, three popular systems are discussed namely Gallop [21], Legion [22, 23] and Condor [24-26].

Gallop [21] supports parallel applications. It allocates and schedules jobs defined by a static task graph [27] onto a network of computational resources. Resource allocation is based on two main components, a Scheduling Manager (SM) and a Local Scheduler (LS), which both run at each domain site, as shown in Figure 2.1. The SM co-ordinates and runs wide area scheduling algorithms which interface to other SMs at different sites and to their own LS. The LS is responsible for managing the local site resources transparently to enable the SM to perform its scheduling. Further the LS is made of two parts, one for interfacing to the SM and the other to a site specific managing system such as the ones mentioned in section 2.1.1.1 e.g. PBS, LSF, etc.

In Gallop resource selection is performed by attempting to minimise the execution time of the task graph as predicted by a performance model for the application and the prospective resources. However due to the minimisation procedure and the cost model being fixed there is no support for policy extensibility. Further the system is designed primarily for parallel applications, which is a limitation as the Grid is designed to support a wide range of applications.

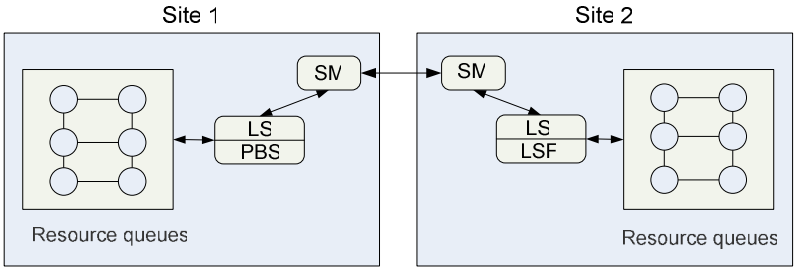

Figure 2.1: The Gallop scheduling components.

Legion [22, 23] is an object-based system designed to harness hosts across multiple sites which are tied together through a high-speed link. Resource management is performed through two specialised objects, an application-specific *Scheduler* and a resource-specific *Enactor* that negotiate with one another to make

allocation decisions. Further it supports a range of applications such as parallel applications and parameter sweep studies.

Legion ensures that local policies of participating sites are respected, by allowing the final authority over the use of a resource to be in place with the resource itself. However it assumes that network resources and protocols currently in use will not change. Legion also does not support co-allocation. Further it is written in Mentat Programming Language (MPL), thus it is necessary to have MPL on each platform before Legion can be installed.

Condor [24-26] is designed to support high-throughput computation by taking advantage of idle compute resources and allocating those resources to an application job. A predominant feature of Condor is the utilisation of resources that otherwise would be idle. However resources are de-allocated and released once their owner begins to use them, which is supported by checkpointing that is incorporated into Condor.

Condor provides an extensible resource description language called Classified Advertisements (ClassAds), which allows for both the application job to be described in terms of its requirements and for a resource to describe its capabilities. The matching of application jobs to resources is performed by a *matchmaker* that uses the ClassAds to facilitate how resource management takes place. ClassAds are further discussed in Section 2.4.2. However Condor does not interface with existing local resource managers and does not provide support for co-allocation [1].

The above review of the current WASS systems shows a wide range of resource management approaches, with each system targeted to serve a particular purpose. Gallop is designed to facilitate parallel applications. Legion supports a range of application but suffers from policy extensibility. Condor utilises idle resources but assumes it is the only resource manager and does not interface with local resource management systems. Overall there is not a single system that provides a solution to all the challenges outlined in Section 2.1.

2.1.2 Hierarchical Components of Grid Resource Management

Before discussing the technologies of Grid resource management in the next section (Section 2.1.3), it is important to describe the components that integrate the Grid, which also serve to aid resource management. The components are constructed through hierarchical layers [13, 28, 29] described below and show in Figure 2.2:

- **Grid Fabric:** This layer comprises of two levels, a network of physical resources and local resource management mechanisms. The network of physical resources consists of all resources geographically distributed across the globe, with shared access mediated by the Grid. The resources could be computers such as PCs, clusters, High Performance Computers (HPC) or storage devices and even scientific instruments such as telescopes. The local resource management mechanism encapsulates operating systems (such as UNIX or Windows), system libraries and application kernels. This also includes resource management systems such as SGE, LSF and PBS and internet protocols such as TCP (Transmission Internet Protocol) and UDP (User Datagram Protocol). The latter level is used by its above preceding layers as a gateway to the physical resources. The resource management systems are the predominant central point of access in the fabric level as they usually govern the resources.

- **Core Grid Middleware:** This is a layer of software that masks heterogeneity and provides transparency from the underlying details such as communication protocols. It is concerned with providing a building block for the construction of software components that can work with one another in a distributed Grid environment. This layer contains the core services that facilitate the operability of the Grid and helps to couple the (remote) resources to the Grid users. The main services include:

  Security service [30-32], to authenticate and authorise users. This is to ensure interaction between the user and the resources takes place

without hindering the integrity of individual systems or the environment as a whole.

Information service [33, 34] (which is further discussed in section 2.3), is a vital part to enable resource discovery due to the dynamic nature of the Grid, where resource status frequently changes. This service facilitates the planning and decision making of where to submit jobs. It provides the mechanisms for registering and obtaining information about the resources and their status.

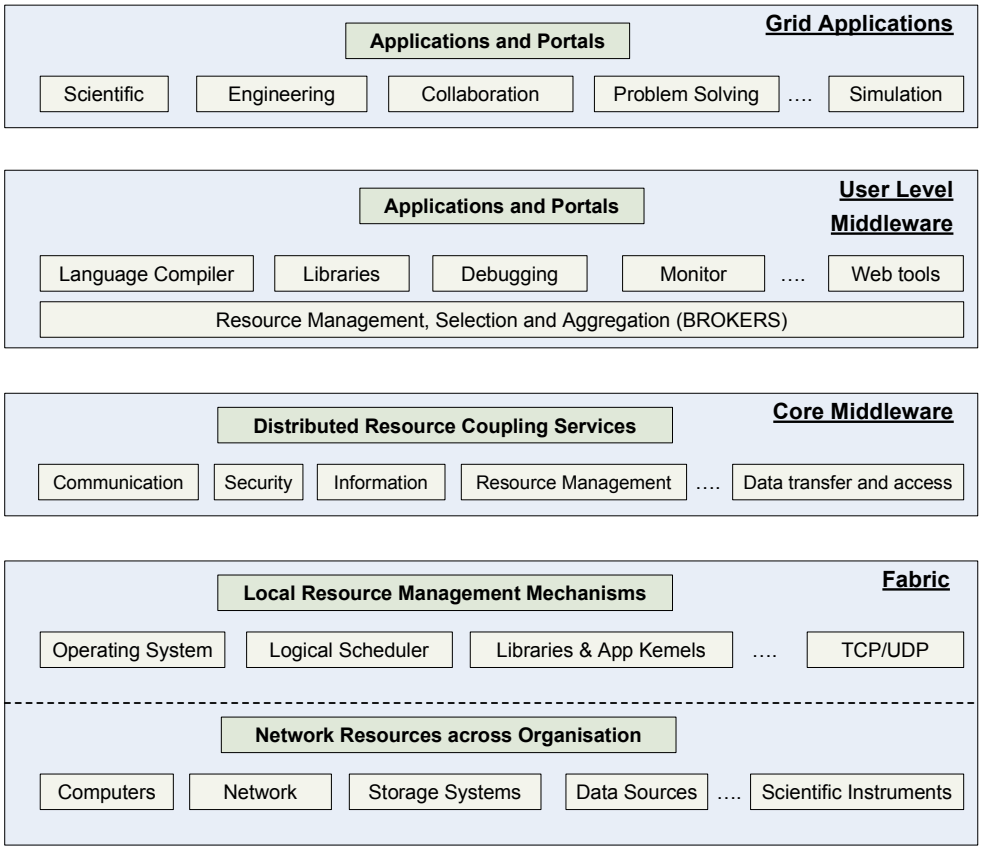

Figure 2.2: The Grid Hierarchy layer (Adapted from [29]).

Data transfer service [35, 36], most Grid applications would need to transfer data to the remote site(s). This is a more predominant factor for applications such as high-energy physics and bioinformatics where they require intensive transfer of and access to large amounts of data. This requirement is supported by Grid File Transfer Protocol (GridFTP) [37] to accommodate for their need.

Resource management service [38, 39] (which is further discussed in section 2.1.3), is an important and integral part of this layer and the Grid infrastructure as a whole. It provides transparency and assists the interaction and communication between a wide range of local resource managements systems, which are the gateway to the Grid resources. This service also supports submission, execution and monitoring of job progress.

- **User Level Middleware:** These are high level services that include application development environments and programming tools. These tools are used to develop procedures and resource brokers that insulate the users from the Grid middleware complexities, alleviating the user from the burden of having to know how to operate the various Grid middleware components. This layer also holds the responsibility to manage and schedule computation across global resources.

- **Grid Application:** These are applications that use the Grid, which range from scientific, engineering, and astrophysics just to name a few.

Having described the hierarchical layers of the Grid infrastructure, it is also important to define basic terminologies that have a broad meaning [40] in the Grid paradigm:

- **Resources:** A resource is an entity that is utilised to help solve problems and can be used over a period of time either explicitly by an individual or inclusively by a number of users. Resources are diverse as stated earlier, range from CPU, storage, memory to scientific instruments and are usually governed by a local resource management system. However for the purpose of this research, local resource management queues that are associated with individual CPUs would be used as core resources. Further the CPU(s) should be mutually exclusive to a queue and not be used in conjunction with other queues. This is to ensure that application jobs execute securely at the resources without exposing their computational process to other third party users. This is due to the fact that applications such as Distributed Aircraft Maintenance Environment

(DAME) [9] hold sensitive data which require their jobs to run on resources without others using them simultaneously.

- **Scheduling:** This is an extensive topic [41, 42] which centres on mapping jobs to resources over time. There are many different scheduling algorithms with different characteristic properties. These properties are affiliated to performance criteria [43] that include:

  CPU utilisation: This is the percentage of time that the processor is busy.

  Throughput: Maximising the number of processes completed per unit of time.

  Turnaround time: This is the interval time between the submission of a job and its completion including the actual execution time, plus the time spent waiting for the resource.

  Waiting time: The total time a job spends waiting in the pending queue.

Well-known scheduling algorithms are First-Come First-Serve, Round-Robin and Shortest-Job-First just to name a few [44]. However some critical application jobs such as those used in DAME are not tolerant to delay. Further it is difficult to anticipate when resources will be needed as these applications require resources on demand. The scheduling method adopted for this research is to locate resources (queues) that are available at the current time of request. Ideally the resources located should not have jobs waiting in the pending queue, if they have then they should not be considered. This is to allow the submitted job to begin executing on arrival. Once the job begins execution on a resource it should be able to put a lock on to it until it has completed.

2.1.3 Grid Resource Management Technology

The Globus toolkit [2, 3] has been developed by the Argonne National Laboratory [45] to support computational Grids. It has become the *de facto* toolkit for Grid resource management and is being used by many major global Grid development

teams including UK e-Science projects [46]. Since its first release it has undergone many evolutionary refinements, for example in Globus 2 it did not support Internet technologies. Globus 3 supported "Grid Services", which are an extension of Web Services [47], using XML (Extensible Markup Language) and SOAP (Simple Object Access Protocol) to provide greater flexibility in the use of the Grid over the Internet. However Grid services tackled the limitation of stateless and non-transient problems that Web services did not address by introducing factories and instance models [48]. In Globus 4 the term Grid services became deprecated as Web services introduced the WSRF (Web Service Resource Framework). This introduction aligned with the functionality of the Open Grid Service Architecture (OGSA) principles [49] and addressed the Web services limitations outlined above. It is also important to state all Globus versions are based on essential core services which are listed in Table 2.1, with version 2.4 being the most stable release to date. The latter is the one used by this research and production Grids such as the White Rose Grid (WRG) [11].

Globus provides a software infrastructure that enables applications to handle distributed heterogeneous computational resources as a single virtual machine, despite the geographical distribution of both resources and users. It is necessary for computational Grids to support a wide variety of applications and programming paradigms. Consequently rather than providing a uniform programming model such as object-oriented which is used in Legion, Globus provides well-defined interfaces (APIs) to allow developers of specific tools or applications to select and use services to meet their needs. Globus is constructed within a layered architecture in which high-level services are built upon essential low-level local services. The architecture is shown in Figure 2.3, which is that of Figure 2.2 but with the Globus services in place (headed core middleware).

The most predominant service in relation to this research and Grid resource management as a whole is the Grid Resource Allocation Manager (GRAM) [3]. It resides on top of local resource manager systems, proving the mechanisms to communicate between these systems and external Grid entities such as applications or brokers. This is enabled through the use of standard APIs that provide the transparency to allow for the expression of resource allocation to a wide range of local resource management systems. This alleviates individual sites from being constrained in their choice of the local resource management systems.

The principal components of GRAM are the Gatekeeper and the Job Manager, which are shown in Figure 2.4. The Gatekeeper is the first point of contact with the local resources. It is responsible for authenticating and authorising a Grid user through the use of the Grid Security Infrastructure (GSI) [30], which handles the issue of site autonomy. The authentication is based on the user's Grid credential and an access control list contained in a configuration file called the grid-mapfile. This file (which is located at the local site) is also used to map the user's Grid identity to a local account, translating the user's Grid credential into a local credential. Once the authentication process is complete, the Gatekeeper starts up the Job Manager which handles the resource request.

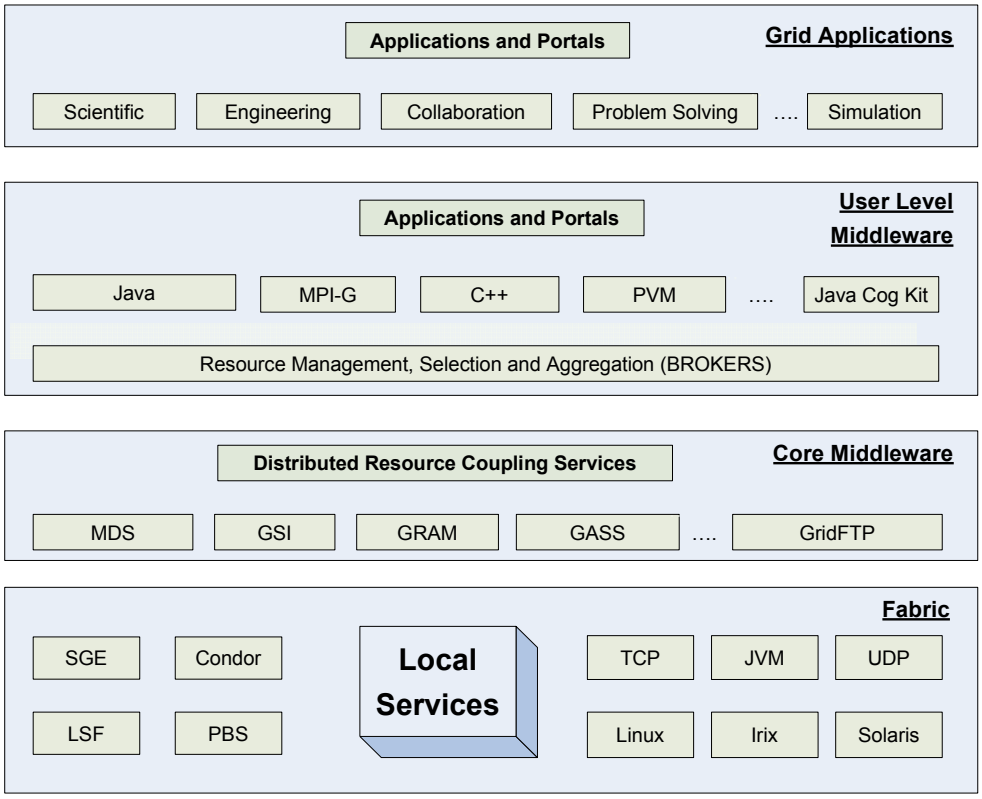

Figure 2.3: The Grid hierarchy with Globus services (Adapted from [29]).

The Job Manager is responsible for creating the actual process requested by the user in order to use the resources. The user's request is composed in a Resource Specification Language (RSL). The RSL is parsed by the Job Manager and translated into a syntax understood by the local resource management system before a process for resource allocation is submitted to it.

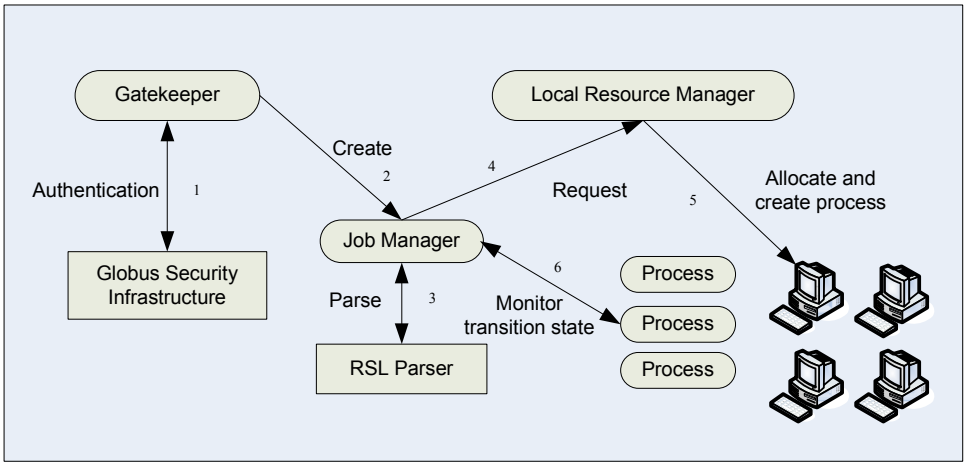

Figure 2.4: The GRAM Components.

Once this procedure is complete the Job Manager is also responsible for monitoring and notifying the user of the transition state for the created process ranging from pending, active to done. It also executes the control operation such as process termination if requested by the user. The Job Manager terminates, once the job for which it is responsible has ended.

GRAM also allows for co-allocation as it is only responsible for its set of resources and is therefore independent from other GRAMs. This paves the way for multiple resources from different sites to be co-ordinated and used simultaneously.

Overall the Globus toolkit addresses the underlying Grid constraints listed in section 2.1. Site autonomy is handled by the GSI which authenticates the user to ensure he/she is entitled to use the resources in a particular domain. Heterogeneous substrates, policy extensibility and co-allocation are supported by GRAM as it is independent from the underlying local resource management systems. This allows for multiple simultaneous job submissions while at the same time providing consistent transparency to these systems.

Service	**Name**
Security	Globus Security Infrastructure (GSI) [30]
Resource management	Grid Resource Allocation Manager (GRAM) [3]
Information provider	Monitoring and Discovery System (MDS) [33, 34, 50]
Data transfer	Grid File Transfer Protocol (GFTP) [37]

Table 2.1: The core services in the Globus toolkit

2.2 Grid Projects

Grid technology is being used in many areas of research and there are currently numerous projects that are using the technology. An overview of some projects is provided below based on geographical regions, which can be broadly categorised into International, European, U.S.A Asia-Pacific and UK based:

International Grid projects

- *International virtual data Grid Laboratory (iVDGL*) [51] is a global Data Grid that will aid experiments in physics and astronomy. Its computing, storage and networking resources in the US, Europe, Asia and South America provide a unique laboratory that will test and validate Grid technology at the international and global scale. Sits in Europe and the US will be linked by a multi-gigabit per second transatlantic link funded by the European Data Trans-Atlantic Grid (DataTAG) [52],

- *IBM IntraGrid* [53] is aimed to bring together IBM R&D (Research and Development) projects from around the world. The organisational aim is to provide the flexibility to integrate the company from end-to-end to respond with speed to any customer demand, market opportunities or external threats. Through the development and management of the IntraGrid, IBM's goal has been to gain valuable, first –hand experience in building and operating a true commercial computational Grid.

European Grid projects

- *European Union (EU) DataGrid* [54], aims to create and apply an operational Grid for applications in high energy physics, environmental science and bioinformatics.

- *European Grid Solar Observations (EGSO)* [55], this is a project that will lay the foundation of a virtual solar observatory. The EGSO addresses the problems of combining heterogeneous data from scattered archives of space and ground-base observation into a single virtual dataset. The project will also create catalogues of solar features and observation data to enable

innovative searching and provide visualisation tools for user-friendly data browsing.

- *CrossGrid* [56], this project will develop techniques for large scale Grid enabling simulation and visualisations that require response in real time.

US Grid Projects

- ***Gri**d **Phy**sics **N**etwork (GriPhyN)* [57], this is a project that involves physicists and informational technology researchers who plan to implement the first Petabyte-scale computational environments for data intensive science. The project is driven by unprecedented requirements for geographically dispersed extractions of complex scientific information from very large collections of measured data.

- *Earth System Grid (ESG)* [58], the goal of this project is to address the formidable challenges associated with enabling analysis of, and knowledge development from global earth system models. This will be achieved through a combination of Grid technologies and emerging community technologies, providing a seamless and powerful environment that enables the next generation of climate research.

- *Fusion Collaboratory* [59], the goal of the project is to advance scientific understanding and innovation in magnetic fusion research. This will be achieved by enabling more efficient use of existing experimental facilities and more effective integration of experiments, theory and modelling.

- *Information Power Grid (IPG)* [60], this is a high-performance computing and data Grid built primarily for use by NASA (National Aeronautics and Space Administration) scientists and engineers.

Asia Pasific Grid projects

- *Asia Pasific Bioinformatics Network (APBioNet)* [61], the project focuses on the promotion of bioinformatics in the Asia Pasific. Its mission has been to pioneer the growth and development of bioinformatics awareness, training and education.

- *Grid Datafarm* [62], this project is a petascale data-intensive computational project initiated in Japan. The project is a collaboration amongst the University of Tokyo and Tokyo Institute of technology as well as other industrial partners.

UK Grid projects

- *AstroGrid* [63] aims to build a Grid infrastructure that will allow a Virtual Observatory, unifying interfaces to astronomy databases and providing remote access, as well as assimilation of data.

- *CombeChem* [64] supports combinational synthesis of new compounds by combining structure and property data sources into Grid-based information and knowledge sharing environments.

- *Reality Grid* [65] enables the realistic modelling of complex condensed matter systems at the molecular levels.

- *Distributed Aircraft Maintenance Environment (DAME)* [9], was a Grid-based demonstrator that built distributed diagnostics system for aircraft engines.

The above list shows the diversity for which the Grid technology is being used geographically. At the University of Leeds the DAME project is being developed as is described in the next section (Section 2.2.1).

2.2.1 Distributed Aircraft Maintenance Environment (DAME) Project

DAME is a project funded by the UK e-Science [46] program for research and development towards Grid technologies. It is a joint project between the Universities of Leeds, York, Sheffield and Oxford, and Rolls Royce and Data Systems & Solutions as industrial partners. Its purpose is to design and implement a system to facilitate the diagnosis and maintenance of aircraft engines through the use

of Grid computing. This is motivated by the need to reduce costs of unexpected and unplanned maintenance of aircraft engines.

One of the key components of this project is the vibration data analysis application eXtract Tracked Order (XTO), others include pattern matching and CBR (Case Based Reasoning). The XTO studies at Leeds analyse the engine data recorded by an on-engine QUICK system [66] during a flight. This recorded data is stored in a file system and consists of a single control file and a number of binary files that are all in the Rolls-Royce proprietary ZMOD format [67]. The binary files contain the recorded data for periods of 101 seconds and the number of these files depends on the duration of the flight. The control file is a plain text file that describes the order in which these binary files should be processed by the XTO analysis application.

The process of analysing the engine data is listed below:

1) When an aircraft lands in an airport the data is downloaded and read.

2) If no irregularity is detected the aircraft is allowed to continue with its journey and the process is ended. Otherwise if an abnormality is detected it is flagged and marked in the data set.

3) A search through the historical data is initiated to identify the cause of the abnormality and measures are taken to rectify the problem. However if this fails the fourth procedure is executed.

4) The system operator launches a feature analysis session, which uses the XTO application.

5) A diagnosis is made based on the feature detection by this analysis.

The fourth and fifth procedures are where DAME is heavily involved, developing methods and approaches to support these tasks through the use of the Grid. This is to accommodate the large number of aircraft that land at airports across the globe. The Grid facilitates the diagnosis and analysis through its capabilities to spread the computational processes over the geographical distributed resources that it supports.

The need to rapidly diagnose engine data is essential otherwise the aircraft is deemed out of commission and would in turn have a heavy financial burden on the

aircraft operator. Thus not knowing when such abnormalities will occur in a flight, it is important to be able to secure resources on demand. Further, due to the data being sensitive it is also important for the execution to take place on resources that are devoted to single jobs and do not execute third party jobs simultaneously to avoid snooping. Hence this research uses XTO as an exemplar application, used over the Grid with these criteria.

2.3 Grid Resource Brokering

In order to utilise the Grid a user must have the ability to interact with the Grid middleware such as the one predominately used in the Grid community, namely Globus. However Globus provides fundamental service components to enable job submissions, which require the user to have extensive knowledge on how to access and operate these components.

Submitting a job to the Grid is a long and tedious task, which involves in essence the ability to query the information providers, interpret the information returned, filter the appropriate resources that meet the job requirements and select a set of resources for that job. This is then followed by writing a suitable RSL script for the job to be executed at the selected resources, submitting the job, transferring any necessary files and initiating the execution. What is needed is a resource broker that insulates the user from the Grid middleware complexities and submits jobs to the appropriate resources for execution. This is to alleviate the user from the burden of having to know the various mechanisms of the Grid middleware.

A resource broker should perform the task of mapping application job requirements to resource(s) that can meet those requirements. Specifically, brokering is defined as the process of making scheduling decisions involving resources over multiple administrative domains [68]. This can include searching multiple administrative domains to use a single resource or scheduling a single job to use multiple resources at a single site or multiple sites. A Grid broker must be capable of making resource selection decisions in an environment where it has no control over the resources, the resources are distributed, and information about these resources is often limited or stale.

As discussed in [4], resource brokers can be classified into two categories: *system-centric* and *user-centric*. A system-centric broker allocates resources based

on parameters that enhance system utilisation and throughput. Conversely, in a user-centric broker, resource allocation adheres to the user requirements and utility of computation is enhanced compared with system utilisation. Hence user-centric brokering ensures the user's utility takes precedence over system utilisation and throughput.

A key goal of Grid computing is to deliver utility of computation, as defined by the users' requirements, which is why broadly speaking Grid resource brokers are user-centric orientated. Thus this section will discuss the most commonly used and well-known computational resource brokers with their architectures, describing their mechanisms and abilities.

2.3.1 Nimrod/G

Nimrod/G [5, 69], whose architecture is shown in Figure 2.5, has been developed by Monash University, Australia. It is a system based on the concept of computational economy and is designed to run parametric applications on the Grid. The broker requires a user to create a task farm (plan) through the use of its declarative parametric modelling language before a job is passed to the parametric engine. The parametric engine interacts with the scheduler to retrieve resource availability, while the scheduler discovers resource load through the Grid discovery service which uses the Monitoring and Directory Service [33, 34, 50]. Once the parametric engine has confirmed the availability of the user's resource selection, the information is forwarded to the dispatcher to initiate a job wrapper that begins the task of submitting the job to the resource(s).

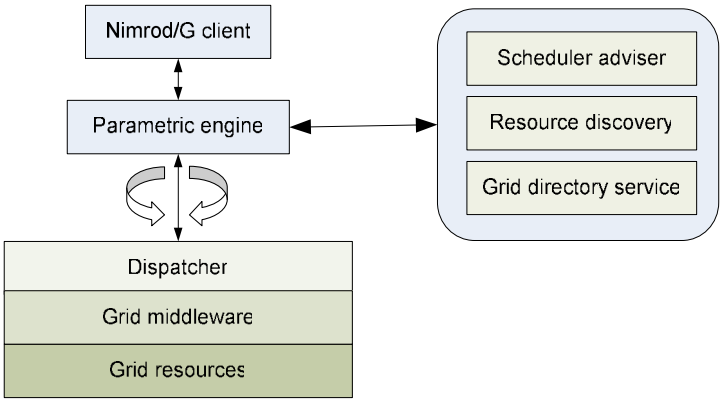

Figure 2.5: The Nimrod/G architecture.

However Nimrod/G does not provide automated resource discovery, a list of resources in the form of Globus Gatekeeper contact strings needs to be set up manually by the user before brokering, hence exposing the user to the Grid middleware complexities. Further Nimrod/G relies on the Globus information service (MDS), which does not offer dynamic resource information such as the local resource managers' queues. This is an entity that is needed in order to ensure the resources would be able to fulfil the job requirement as they are systems that usually govern the resource.

2.3.2. EZ Broker

The EZ broker [70, 71], with its system stack shown in Figure 2.6, is being developed by University of Houston. The broker has two core elements: a *client component* and a *server system*. The client component comprises of a *policy engine* and a *Graphical User Interface* (GUI). The policy engine is a framework that promotes policy-based authentication and access control to Grid resources. It ensures before contacting the resources that user has access rights for their utilisation. The GUI component has been written in the Java programming language and is intuitive with little training required to operate it. It is used to allow the user to provide a description of the resource requirements that are needed by the application.

The server system comprises of a *register* and an *information provider*. The register is designed to act as an index server providing information about hosts that have registered with the EZ broker. The information provider retrieves both static and dynamic information from the Grid resources.

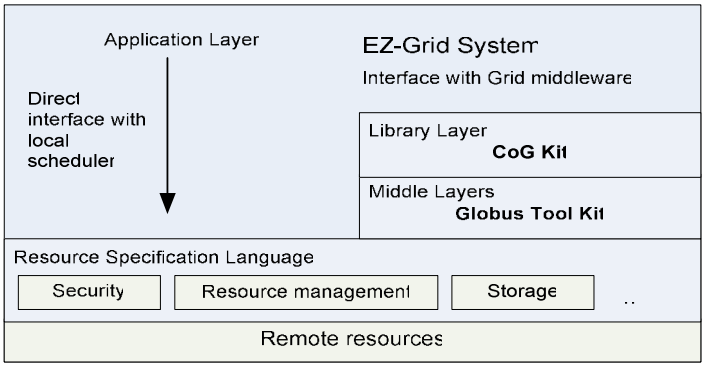

Figure 2.6: The EZ system stack [70].

The EZ system stack (Figure 2.6) shows the interaction of the EZ broker with the Globus toolkit. This is achieved through the use of the CoG Kit [72] to attain services such as security (GSI) and resource management (GRAM).

At present the EZ broker project is a work in progress. It has been proposed to support a wide range of Grid applications. However this would be a long process as there is a broad category of Grid applications each with different requirements and needs. Further brokers should be built with specialised abilities, to provide services to a set of application categories as stated in [73].

2.3.3 Condor-G

Condor-G [74] is a system that allows the user to perform Grid job management operations through the command line aided by the Globus toolkit. The management operations that it supports are listed below:

- Submitting jobs to Grid resources, including their input/output files and any arguments needed to initiate the jobs execution.
- Querying a job's status or cancelling the job.
- The user is informed of job termination or errors that occur during execution via an email.
- It stores a log that provides a history of the job's execution stages.

As shown in Figure 2.7 Condor-G has a Scheduler that responds to the user's request to submit jobs. It does this by creating a GridManager daemon at the job submission machine to handle and manage those jobs. One GridManager daemon handles all jobs for a single user and terminates once all jobs are complete. Each GridManager contacts the modified GRAM that has been configured to support Condor-G at the execution site. This results in the creation of a Globus JobManager daemon, after passing the authentication process by the Gatekeeper.

The JobManager daemon at the execution site connects directly to the GridManager at the submission machine using the Global Access Secondary Storage (GASS) [75]. This is to transfer the job's executable and standard input, output and any error files. The JobManager then submits the job to the local site's resource management system. Updates on the job's status are sent by the JobManager back to

the GridManager, which then updates the Scheduler. All the states for the job are stored in the Scheduler's Persistence Job Queue to keep a history log.

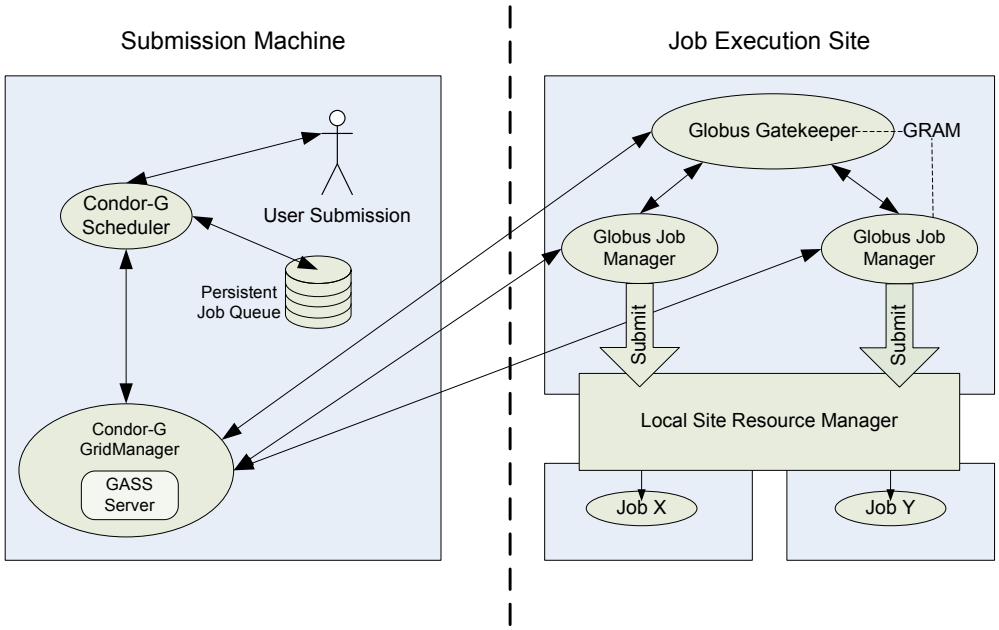

Figure 2.7: Condor-G mechanism for executing a job on the Grid [74].

Condor-G is built on Condor, which inherits the constraint of not supporting co-allocation. The matchmaking mechanism and checkpointing provided by Condor is not incorporated into Condor-G. More importantly the user is exposed to the Grid middleware complexities by having to query the information providers manually and needing to write the appropriate RSL for the job to be executed at suitable resources. Further file transfer is mainly based on the Globus GASS service, which is designed for small lightweight data transfer. For large file transfer the user would need to stage the necessary file using Grid File Transfer Protocol (GridFTP) [37].

2.3.4 Grid Resource Broker

The Grid Resource Broker (GRB) [76] with its architecture shown in Figure 2.8, has been developed by the University of Lecce (Italy). It is mainly based on a portal that is designed to bridge the gap between users and the Grid through the use of the Globus toolkit. The services that it provides are listed below:

- **User profile management,** this provides storage for users to add or remove resources that they are entitled to use on the Grid. It stores both the resource name and the contact string for that resource.
- **Information service,** allows the user to query the MDS, discovering resources that match the job requirement and to find out the state of the resources. It also supports the use of the Network Weather Service (NWS) [77, 78] to provide more dynamic information than that provided by the MDS.
- **Job submission,** it allows for both interactive and batch job submission on the basis that the user specifies the location where the output file should be streamed.
- **Job tracking,** batch job submissions are easily tracked and the user is informed of the different states from pending, active to done or failed.
- **File transfer,** it supports transferring files or directories using multiple parallel streams leveraging on the Globus GridFTP service.

Overall the services provided by the GRB are those supported by the Globus toolkit. It simplifies the user's interaction with these services through its portal by having the user fill in a predefined list of attributes describing what is required by the invoked service. However it does not provide automated resource discovery, decision-making is left to the user and resources are not secured before job submission. Further, during its resource discovery it does not supply local resource management queue information, which is rudimentary when submitting jobs to the host queue.

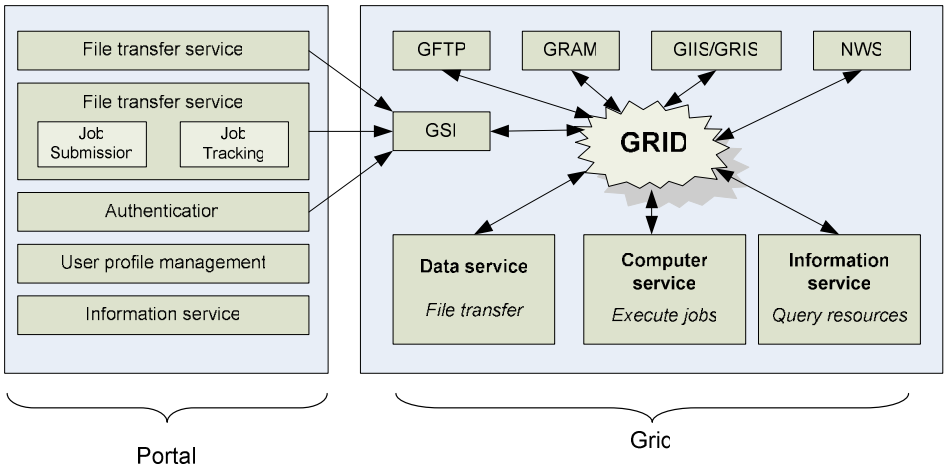

Figure 2.8: The GRB architecture.

2.3.5 AppLeS

The AppLeS [79-81] (Application Level Scheduler) project has been developed by the University of California, San Diego. It primarily focuses on developing scheduling agents for individual applications enabling them to adapt to resource change during execution. The AppLeS agents use both static and dynamic information while selecting a viable set of resources for the applications. The individual steps followed by AppLeS agents are depicted in Figure 2.9 and described below:

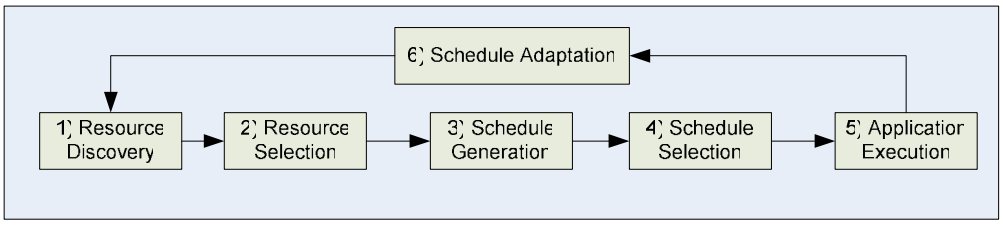

Figure 2.9: The steps that are taken during the AppLeS scheduler.

1) **Resource Discovery** – The AppLeS agents discover the resources which are potentially useful to the application. This is accomplished by taking a list of user's logins and contacting the resources for their availability.

2) **Resource Selection** – The agents identify and select resource sets from among the possible resource combinations.

3) **Schedule Generation** – Given a list of resource sets, the AppLeS agents apply a performance model to determine a set of candidate schedules for the application on the potential targets. The Network Weather Service (NWS) [77, 78] that dynamically monitors the changes in performance of the resources over a set period is also used at this stage.

4) **Schedule Selection** – Given a set of candidate schedules the agents choose the best overall schedule.

5) **Application Execution** – The best schedule is deployed by the AppLeS agents on the target resources.

6) **Schedule Adaptation** – The AppLeS agents are able to account for change in resource availability by looping back to step 1.

The development of an AppLeS compliant application involves the joint collaboration between disciplinary researchers and members of the AppLeS project team. During the collaboration the application scientists provide an original distributed application code. The AppLeS researchers then work with these scientists to modify the application so that it can dynamically be scheduled by the AppLeS scheduler agents. However this process involves the integration of AppLeS agents (that are software components) in which the application code and the agents are combined and not easily separated. In particular, it is difficult to adapt an existing AppLeS application so that it can be used for other applications to make them AppLeS enabled. Thus the AppLeS team have developed templates that embody common characteristics from similar AppLeS-enabled applications. However the AppLeS enabled application integrates an adaptive scheduler agent in the application to form a new self-scheduling adaptive application. An AppLeS template is a software framework developed so that application components can easily be inserted in modular form into the template to create a new self-scheduled application.

Overall AppLeS mainly focuses on scheduling applications onto resources and adapting to the resources changes, with resource reservation not fully exploited. It uses two scheduling engineering strategies [79] namely 1) embed scheduling logic in the application and 2) embed application-specific information in the scheduler through the use of a template. In either case such scheduling strategies are time consuming to build and are not easily re-targeted for other non-similar applications. AppLeS also uses the NWS during its scheduling process which provides a short term prediction of the resource performance. Since the prediction is based on a short term (for example a five minute period) it may work well when the application size is small. However for large applications a NWS based schedule may become unsatisfactory due to the low quality of prediction. Further the NWS itself consumes resource power over long periods of time which can change (intrude upon) the conditions it is attempting to forecast [81].

2.4 Resource Information Providers

Information providers are an integral part of the Grid infrastructure due to its diverse scope and dynamic nature. Resources on the Grid fluctuate as their utilisation and availability continuously change. When a client or a broker enters the process of resource selection the information providers are firstly queried. This is to discover the current state of the resources to determine where to submit a job. Lack of information can severely hinder job execution start time, which is why information providers have a vital role in the Grid. In this section an overview of a range of information providers will be discussed, describing how they operate.

2.4.1 A4 – Agile Architecture and Autonomous Agents

A4 [82] is an homogeneous agent based hierarchical approach to resource discovery developed by the University of Warwick. The mechanism adopted is based on a single component, an agent, which is used to compose the entire system. Each agent has the same set of functionalities in being able to send requests and provide information services.

In Figure 2.10 the hierarchical structure of A4 is shown with different terms used to differentiate the level of the agents in the hierarchy. The term broker in this context is an agent that heads the whole hierarchy. A co-ordinator is an agent that heads a sub-hierarchy and a leaf node is actually denoted as an agent in this description.

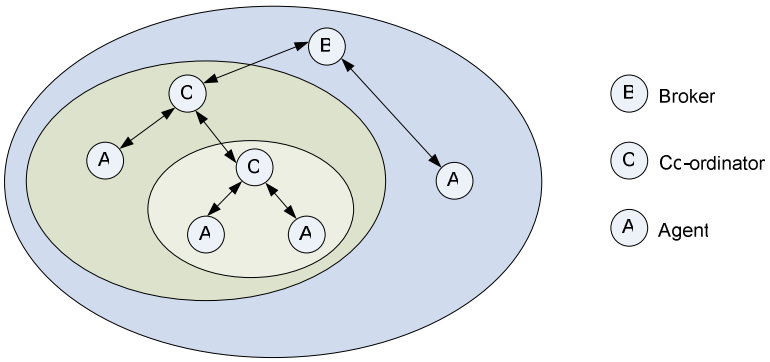

Figure 2.10: The A4 hierarchical structure [82].

When an agent requests a service it will first check its own knowledge to see if it is already aware of the service. If it has the required service information it can contact the target agent directly. Otherwise it will look to its local agents or ask its upper agents until the service information is found. The agent can then connect directly to the target and directly request the service.

The main data type for advertisement and discovery is the Agent Capability Table (ACT) and its basic structure consists of two parts, an agent identity and service information. The process of the service advertisement and discovery corresponds to the maintenance and the look up of the ACTs.

The A4 hierarchy is scalable, however service discovery could be a long process especially if the request traverses through many other agents, which is time critical for applications that require resources on demand. However this has been addressed by the A4 project through the use of two kinds of ACTs a local (L_ACT) and global (G_ACT). Every agent has one L_ACT which stores information about other agents registered with it. The G_ACT is a copy of its upper agent's L_ACT. Thus an agent can have more information of services which allows it to contact them directly without submitting the request to the upper agent. However this creates additional data maintenance workload. Further, Grid resources' status can alter frequently which would also require the agents to correspond to the changes. This could be a strain on the system to populate all the concerned agents with the new information.

2.4.2 Classified Advertisements (ClassAds)

ClassAds [8, 83] are part of Condor, a descriptive language for advertising and acquiring resources. They are generated in the form of property lists constructed as attributes that are used to structure a description. ClassAds have a framework that is based on matchmaking, where entities, a resource provider or a resource requester (consumer), advertise their characteristics and needs through the use of ClassAds. The advertising process is shown in Figure 2.11 and is explained below:

1) The provider and consumer send their ClassAds description to the Matchmaker.

2) The Matchmaker then invokes an algorithm by which matches are identified.

3) To perform the match, the Matchmaker evaluates the expressions in an environment that allows each ClassAd to access attributes of the other. Once the matching phase is complete the Matchmaker notifies the two parties that were matched and sends them the details.

4) The consumer then contacts the provider directly using a claiming protocol to establish a working relationship with the provider.

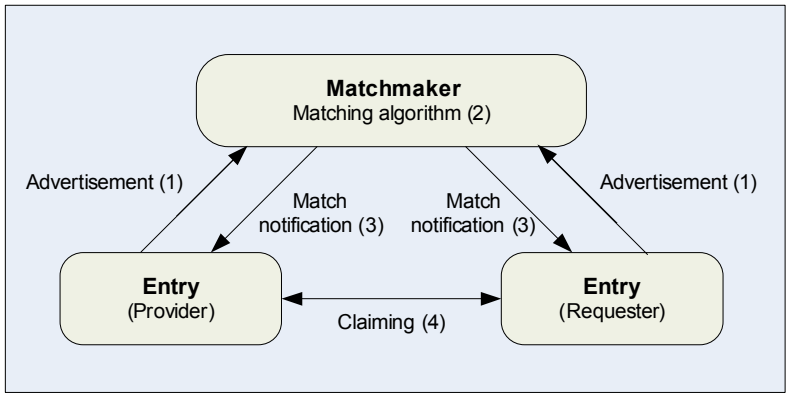

Figure 2.11: The process involved in matching a resource provider to a resource requester through ClassAds [8].

It is important to note that a match does not immediately grant the service to the consumer. Rather the match is a mutual introduction to the advertising entities. Further the state of a resource may be continuously changing and there is a possibility that the Matchmaker establishes a relationship with a stale advertisement. Thus the consumer will still have to further verify the state of the resource before committing to it.

The ClassAds libraries are available as a separate stand-alone package for use in applications other than Condor. Thus ClassAds are used in Grid projects such as the European Data Grid (EDG) [54]. However the EDG only employ ClassAds as a user job resource description language but use the Globus Monitoring and Discovery System [33, 34, 50] (MDS) to gather information about resources.

For time critical applications the Matchmaker framework may not be satisfactory as the consumer's request is dependent on the Matchmaker. The Matchmaker in turn is dependent on a resource provider to publish its information to it. This could end up in the consumer's request waiting indefinitely without knowing if there are any resources to handle the job.

2.4.3 Network Weather Service

The Network Weather Service [77, 78] (NWS) is a system that periodically monitors and dynamically produces short-term resource forecasts based on historical performance measures. It operates a set of performance sensors, mainly for CPU and network resources, from which it gathers readings of regular instantaneous status conditions. It then uses numerical models to generate future forecasts of what the conditions will be for a given timeframe for those resources.

The NWS is built on four different component processes that enable it to provide the forecasts, which are listed below:

- **Persistent State process:** stores and retrieves periodic performance measurements from a persistent storage for a particular resource.
- **Name Server process:** creates a directory that provides information about the location of all hosts on the system and their associated persistent storage.
- **Sensor process:** gathers periodic performance measures from a specified resource.
- **Forecaster process:** generates a predicted performance value for a particular timeframe of a specified resource. It accomplishes this by contacting the name server to learn the location of the persistent storage for the desired resource. It then retrieves the information and applies time-series models to deliver forecast to the client.

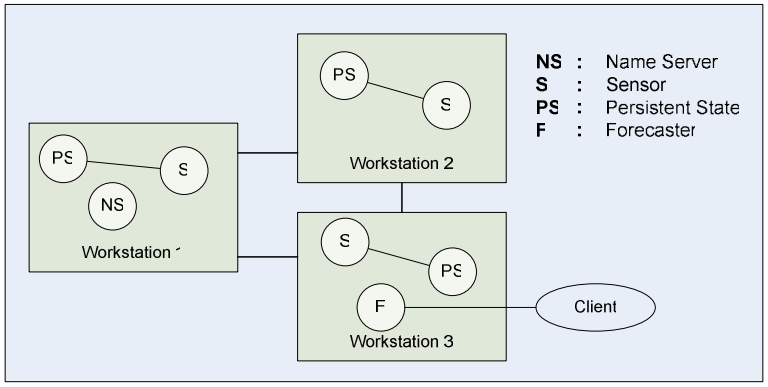

Figure 2.12: The NWS component shown across three workstations [77].

In Figure 2.12 the NWS processes are shown across three workstations. The Name Server resides on only one host in the system. The sensors monitor the performance of the network and processors which then store their measurements in the persistence state repository. The forecast, which also resides on a single host on the system, acts as a proxy for clients who want to query the state of a resource over a period of time.

Overall the NWS provides predicted performance forecasts for mainly CPUs and networks resources. However it does not provide the current state of a resource for applications that would like to schedule their jobs during that time of querying the resources. Further, depending on the level of sampling the sensors carry out this can be intensive on the resource, consuming resource power and affecting the resource's overall performance.

2.4.4 Monitoring and Discovery System (MDS)

The Monitoring and Discovery System [33, 34, 50], (MDS) formally known as Metacomputing Directory Service is part of the Globus toolkit and used to provide the Grid information service. It bestows a standard mechanism for publishing, discovering and accessing information about the state of computational Grid resources. It is decentralised and hierarchical in structure allowing it to be scalable. It also handles both static and dynamic data.

The MDS consists of three main components as shown in Figure 2.13. An Information Provider (IP), Grid Resource Information Service (GRIS) and a Grid Index Information Service (GIIS). The IP in this context represents an interface for any data collection service that gathers information about a particular aspect of a resource such as disk capacity, RAM memory or CPU load. This information is then passed onto the GRIS that deposits and displays the information as entries. The GRIS is a distributed information service than can answer queries about a particular resource by directing the query to the underlying IP. Conversely a GIIS combines arbitrary GRIS services to offer a coherent system image that can be explored or searched by a Grid client. It provides the mechanism for identifying resources of a particular interest. For example it could list all the computational resources available within a particular research consortium. A Grid client can access either the GRIS or GIIS directly depending on the type of retrieval information that is desired.

The Framework for both the GRIS and the GIIS is implemented based on standard APIs defined by the Lightweight Directory Access Protocol (LDAP) [50].

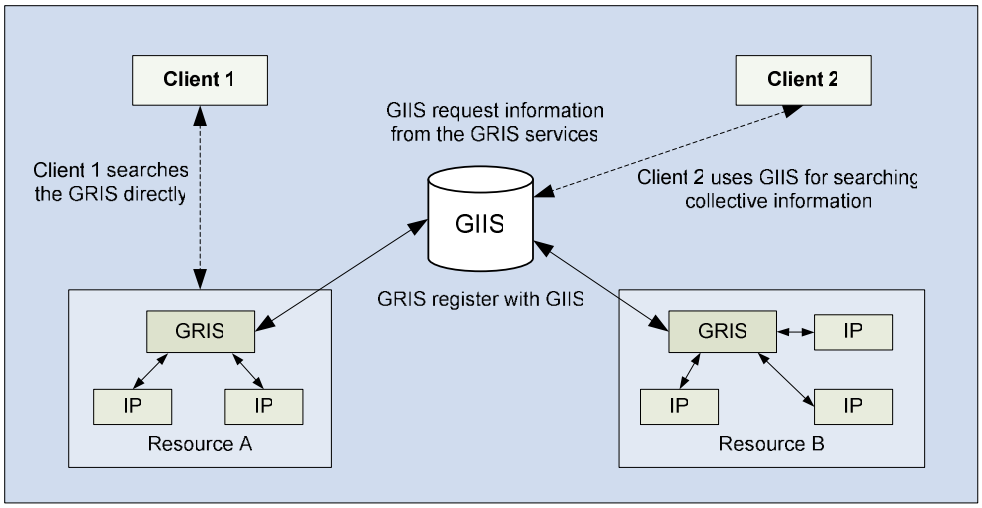

Figure 2.13: MDS architecture.

The MDS can restrict access to certain users by using the Globus GSI to authenticate the authorisation of any queries. It supports both cached information, which is usually out of date and non-cached information which retrieves resource information when queried. It supports this facility by the attribute called Time-To-Live (TTL) which is used in both the GRIS and GIIS. A TTL of 0 indicates that resource information associated to this attribute cannot be cached. A positive TTL value determines the amount of time that the information for a resource is allowed to be provided out of the cache before the information is updated.

The MDS is widely used in the Grid community such as in many e-Science projects and the brokering architectures mentioned in section 2.3. However default installation provides limited dynamic information such as a lack of local resource management queueing details. Nevertheless it allows for such information to be integrated into its system.

2.5 Resource Reservation

Resource reservation [84] is the process of securing resources prior to submitting a job to those resources. This would guarantee the resources reserved for the job are available for utilisation when the job is ready for execution.

To date resource reservation is still a developing area in the Grid with two well-known systems namely Globus Architecture for Reservation and Allocation (GARA) [85, 86] and Maui [12], which are described in this section.

2.5.1 Globus Architecture for Reservation and Allocation (GARA)

The Globus Architecture for Reservation and Allocation (GARA) [85, 86] is a system built on mechanisms provided by the Globus toolkit. Specifically it uses the GSI (Globus Security Infrastructure) for its security and GRAM as part of its resource management [7]. It is a layered architecture that has a single unified interface to communicate with diverse underlying resources but has been shown to work for CPU and network bandwidth. The interface is the same to reserve any type of resources, however GARA only allows for a single resource to be specified per description. For example a client can make a reservation for CPU or network bandwidth, if both are needed the client needs to make two separate requests with two separate descriptions. The reservation descriptions are made through the use of Globus RSLs.

The GARA architecture, illustrated in Figure 2.14, is made up of four layers. The high-level layer is an interface for the user to describe the applications requirements and what resources are needed. The GARA layer allows reservation requests to be described in a unified way and uses a data structure called a handler to communicate with local or remote LRAM (Local Reservation and Allocation Managers). The LRAM incorporate GRAM facilities to authenticate and authorise users before reservation is allowed. LRAM are also responsible for translating all incoming requests, so that they can be presented to the actual resource manager which provides the reservation. The resource managers in the resource manager layer are responsible for enforcing the reservation by communicating with the lower level resources such as CPUs. Even though GARA may seem ideal for reserving resources, as it supports both immediate and advance reservation, it depends on the local resource manager to support this facility. Consequently not all local resource manager systems support reservation which is why GARA has become obsolete and out of commission.

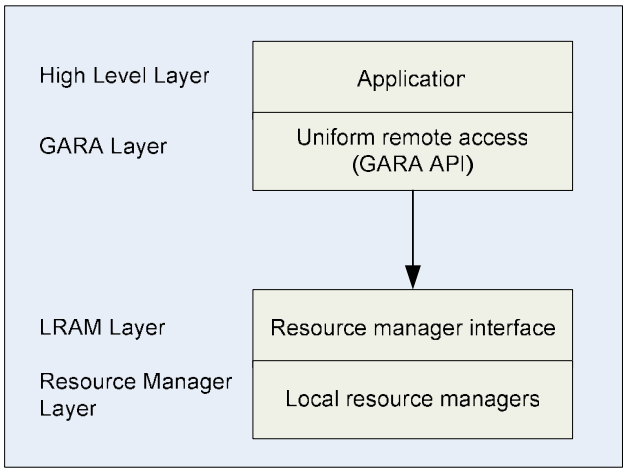

Figure 2.14: The GARA framework (Adopted from [87]).

2.5.2 Maui

Maui [12] is an external local resource manager, meaning it works in conjunction with a site's existing resource manager. It operates with all major local resource managers such as SGE, PBS, LSF and LoadLeveler to extend their capabilities and enhance their scheduling effectiveness.

Maui has received much attention in the Grid due to advance reservation capabilities that it supports. In general it allows a site to set aside a block of resources for various purposes such as cluster maintenance, special group projects or individual user jobs. Reservation is made through the use of resource expressions which indicate both resource quantity and type conditions which must be met by resources to be considered for inclusion in the reservation. The reservation can be configured to support revocable or irrevocable allocation. With irrevocable reservation the resources will be available at the required time regardless of existing or future workloads. However if revocable reservation is chosen the reservation would be released if a higher priority request arrives after the allocation.

Maui provides its services by exclusively controlling the site's local resource managers as it controls and dominates the scheduling. It only offers advance reservation and not immediate reservation.

2.6 Summary

This chapter sets the context of the thesis by firstly presenting Grid resource management and how it differs from the traditional distributed systems. It also looked at the *de facto* Grid resource management middleware toolkit namely Globus and the services it provides. Then an overview of Grid projects was provided, this then led the DAME XTO application and a description into how aircraft engine data is analysed. A survey of Grid resource brokers was discussed with their architectures and the facilities they support. This was followed by an overview of a broad set of information providers describing how they operate and the type of information they support such as static, dynamic or a combination of both. Finally the chapter ended with the two well-known resource reservation systems (GARA and Maui) and how they interact with the resources. The chapter provides the necessary conceptual foundation of the whole thesis.

Chapter 3

Grid Resource Broker Architecture using the Three-phase Commit Protocol

In this chapter an overview of the process of submitting a job directly to a local scheduler (the common approach in distributed systems) and through the Grid will be discussed in Section 3.1 This is to highlight the need and importance for a Grid resource broker to insulate the user from the Grid middleware complexities. It will then be followed by a review of current Service Level Agreements (SLA) on the Grid in Section 3.2, with a discussion of the SNAP (Service Negotiation and Acquisition Protocol) framework [10]. Section 3.3 will describe the SNAP-based Grid resource broker architecture with its components. This is then followed by the need to secure resources in Section 3.4 describing the two-phase commit protocol, which will lead to an enhanced version namely the three-phase commit protocol that reserves resources on demand. Finally Section 3.5 summarises the chapter.

3.1 Overview of Job Submission

A description of the stages for submitting a simple batch job directly to a local scheduler (the common approach used in distributed systems) and the submission of the same job through the Grid will be explained in Section 3.1.1 and 3.1.2 respectively. This is to highlight the significance of the need for a resource broker and outline the processes it encapsulates to insulate the user from the Grid middleware complexities.

There are many types of Grid jobs ranging from parameterised, interactive, batch, and MPI (Message Passing Interface) just to name a few. However for this research batch jobs are used, which are the type handled in the DAME (Distributed Maintenance Aircraft Engineering) project [9] as explained in Section 2.2. This

research uses SGE 5.3 (Sun Grid Engine) [19] for its local scheduler and Globus 2.4 [2, 3] as the Grid middleware toolkit. A simple batch job that requires to be executed at a single queue associated to a CPU will be used to demonstrate the process of submission, firstly directly to SGE and then to the same queue through the Globus Grid middleware toolkit. The batch job used in this example for simplicity and ease of explanation requires to run for a duration of one hour and calculates a list of factorial numbers. The example will illustrate the differences in submitting a job and the importance of a Grid resource broker.

3.1.1 Job Submission Directly to a Local Distributed Scheduler

Submitting a batch job directly to a local distributed scheduler (in this case SGE) can be carried out by two different and simple methods. However before these methods are explained in either case the user would have to describe his/her job through a SGE script. The script usually consists of the run time duration, the directory location where the job's executable files are stored and the location of any data the job depends on. Further, in a distributed system the user is familiar with the architecture and its capabilities which simplify the process of job submission.

In the first method the user can submit his/her job directly to a selected queue of his/her choice for execution. This is on the basis that the queue can handle the job which is determined by the execution run time duration stated in the job's script. Before submitting the job the user can query the available queues to know their loads and decide upon a favourable choice.

The second method is to submit the job directly to the SGE scheduler. In this method the job is sent to a spool area waiting for the SGE scheduling interval to allocate the job to a queue based on its description.

3.1.2 The Process of Submitting a Grid Job

The submission of the same batch job to a remote site on the Grid differs considerably from that explained in Section 3.1.1, even though the underlying system is the same i.e. the job would eventually be handled by the local scheduling

queue at the remote site. However a Grid user would have to pass through several stages before being able to submit his job. Further the user would have to know more information than just the queue load: such as storage capacity, architecture type, etc as the resources are governed by different administrators and have different specifications, which the user may not be familiar with. Thus the submission of a Grid job for execution at a remote resource involves four main stages and these are described below:

Stage 1: Job requirements

In this first stage a user needs to describe his/her job's requirements such as the hardware platform operating system, number of CPUs with preference of type and speed, minimum RAM and disk space, etc. Other issues include the execution start time and its duration, the location of both the executables and any data the job depends on. This stage is pivotal as it will influence the following stages, determining where the job will eventually be submitted for execution.

Stage 2: Resource Discovery

After the job requirement has been specified the *first step* in this stage is to identify a list of resources for which the user has the credentials for their utilisation. This step filters out the implicit constraints; a job will not run at a resource if submitted without the user having the authorisation. The most common solution is to keep a record of which resources a user is entitled to use; this is simply achieved by having a list of account names, resources and passwords written down in a log [68] and kept secure. This method has problems as it requires the user to manually process the filtering of the resources that are appropriate for a job, which suffers from fault tolerance as the list can be long and it can be a tedious task.

The *second step* is eliminating the resources from the list generated in the previous step that do not meet the job's minimum requirement constraints, specified in Stage 1. This process is not extensively mentioned in the literature but has credible benefits. It enhances efficiency in avoiding unnecessary processing associated with contacting resources that do not have the ability to handle the job's requirements.

The *final step* in this stage is to contact the information providers associated with those resources that the user has the credentials to use and are able to meet the job's minimum requirements. The purpose of contacting the information providers is to query the current state of those resources and report back to the requester allowing for a decision to be made as to where to submit the job. The predominant information provider that is commonly used in the Grid community, as mentioned in Section 2.4.5, is the MDS which provides both static and limited dynamic information and is also used as part of this research.

Stage 3: Resource Selection

Once all the queries have reported back from the information providers, the *first step* in this stage is to extrapolate and interpret the data returned, which is usually in the form of a list of attributes with corresponding values. The *second step* is to evaluate the states of the resources and generate a list from those resources that are capable of handling the execution of the job. The *third step* is to co-allocate the job across a set of chosen resources which satisfy the job's requirements (that is if more than one resource is needed). At this point the information provider, for the nominated resource (as only one is required for this example), could be contacted to confirm its status has not changed before submitting the job. However a further option to guarantee that the resource will remain available for the job even after confirmation is to reserve the resource prior to submission.

Stage 4: Job Submission

Once a resource has been selected for the job and reserved if necessary, the *first step* in this stage is to transfer any data files and executables to the remote resource. The *second step* is to submit an RSL (Resource Specification Language), which provides the details on how to execute the job, which queue to use and which will also initiate the execution.

Overall submitting a job through the Grid, the user needs to have extensive knowledge on how to operate the various Grid technologies. This ranges from querying the information provider to the creation and submission of the RSL. Conversely submitting the same job directly to a local distributed scheduler, a

user only creates a script describing the job and either allocates it to a specific queue or allows the scheduler to handle the allocation.

3.2 Grid Service Level Agreements

Grid computing has relied on "best effort" as a guiding principle of operation [88]. However users require some form of commitment and assurances on top of the allocated resources, which is sometimes referred to as Quality of Service. Commitments and assurances are implemented through the use of Service Level Agreements (SLAs) which ensures Grid applications' job requirements are met. SLAs are bilateral agreements which define a mutual understanding and establishes assurances, setting expectations and obligation relating to a service that a provider will supply a user [89]. Further, SLAs allow a service provider to differentiate itself from its competitors and will obligate it to achieve its promises to the user. According to the vision of the European Commission [90], SLA technology will be of central importance in building robust next generation Grids.

There have been a number of attempts at defining SLAs and management architectures for both Web and Grid management. Architectures from Sahai *et al* [88], Leff *et al* [91] and Dinesh *et al* [92] concentrate on SLAs within commercial Grids. The service language used is that presented by Ludwig *et al* [93]. The Global Grid Forum (GGF) has defined WS-Agreements [94]; an agreement-based Grid service management specification designed to support Grid service based on the Open Grid Service Architecture (OGSA) [49]. Two other related works are automated SLA monitoring for Web services [95] and analysis of service level agreement for Web services [96]. Contract negotiation within distributed systems has been the subject of research where business-to-business (B2B) service guarantees are needed [97, 98]. The mapping of natural language contracts into models suitable for contract automation [99] exist but has not been applied to the Grid environment, neither has it been applied as SLAs. An approach for formally modelling e-Commerce [100] exists at a higher level than the research by Ludwig *et al* [93]. Further the GGF Grid Resource Allocation Agreement Protocol (GRAAP) Working Group [101] is interested in SLA terms for resource reservation, but has not yet put forward any design for what these SLAs should look like.

What is required is a modular framework primarily focusing on the Grid and supports end-to-end SLA management [102]. A framework has been proposed by the pioneering researchers of Grid technology at the Argonne National Laboratory [45]. The framework is referred to as SNAP (Service Negotiation and Acquisition Protocol) [10] which is used for this research and is explained in the following section (Section 3.2.1).

3.2.1 Service Negotiation and Acquisition Protocol

Sharing resources in Grids is complicated, in that it requires the ability to bridge the differing policy requirements of the resource owners, in order to create a consistent cross-organisational policy domain that delivers the necessary capability to the end user, while respecting the policy requirements of the resource owner. Further complicating the management of Grid resources is the fact that Grid jobs often require the concurrent allocation of multiple resources. This need for simultaneous resource usage necessitates a structured framework in which resources can be co-ordinated across administrative domains.

Early work on resource management in networks and Grids has led to the development of a range of management abstractions and interfaces specialised to the different classes of entities that need to be managed. For example, integrated [103] and differentiated services [104] have been developed for networks, a Grid Resource Allocation Manager (GRAM) for computational resources [3] and Storage Resource Manager (SRM) functions for storage [105]. However, these domain-specific approaches become increasingly inappropriate as more sophisticated application, demand increased levels of control of the resources.

While common, interoperable mechanisms are a necessary basis for Grid resource management, addressing these issues of "plumbing" is not sufficient. Even with standardised interfaces, the fact that different organisations operate their resources under different policies is a significant impediment to being able to use Grid enabled resources. The user needs to understand and effect resource behaviour, often requiring assurances or guarantees on the level and type of service being provided by the resources. Conversely the resource owner wants to maintain local control and discretion over how the resource can be used. Not only does the owner want to control usage policy, he/she often wants to restrict how much policy information is

exposed to the user. A common means for reconciling these two competing demands is through the use of SLAs. Thus there is a need for a management framework that can be applied to a range of resources and services in a uniform fashion, which supports mechanisms by which agreements between interested parties can be established and asserted through the use of SLAs to gain mutual understanding and assurances.

The Service Negotiation and Acquisition Protocol (SNAP) [10] framework addresses the issues covered above in this section. Hence SNAP is motivated by the requirements in a Grid environment to reconcile the needs of the user with those of the resource providers through the use of SLA. The user's requirements are examined and resource providers that can support such requirements are identified. An agreement between the user and the provider is established to ensure that the user's application job will be performed and the specified requirements met. Note that the use of an SLA also ensures that the user knows what the resources can be expected to deliver without necessarily requiring any detailed knowledge of local resource provider policies, which the resource provider may not be willing to share. SNAP further addresses the issue that multiple resources owned by different providers may be required by a single application job and a single SLA across multiple sites may not be possible. SNAP solves this problem by decomposing management functions into different types of SLAs that can be composed incrementally, allowing for co-ordinated management across the desired resources, which is achieved through a layered formation. Also note that in this research a Grid resource broker (which is described in Section 3.2.2) is used within the SNAP framework with an architecture overview of the broker depicted in Figure 3.1. The SNAP's layered framework is described below:

- The first layer is the Task Service Level Agreement (TSLA) in which a user provides a specification of his/her job requirements. These requirements may include the architecture type, operating system and version, number of CPUs with their description (speed and version), RAM size and storage capacity. The requirements are communicated to the broker for processing through an interface such as a Grid portal that forwards the user's description to the second layer. Hence the TSLA ascertains the user's job requirements which will influence the

proceeding layers and dictate the type of resources chosen for the user's application job.

- The second layer is the Resource Service Level Agreement (RSLA). This relates to resource discovery, decision-making on the appropriate resources that have the capability to meet the job's needs and ensuring the resources are available for utilisation. This also includes acknowledging any policy restrictions such as the user's privilege to use a resource, which is shown in Figure 3.1 by an arrow labelled "Policies" pointing to the broker.

- The third layer is the Binding Service Level Agreement (BSLA), which associates the job to the resources and initiates the execution of the job at the resources.

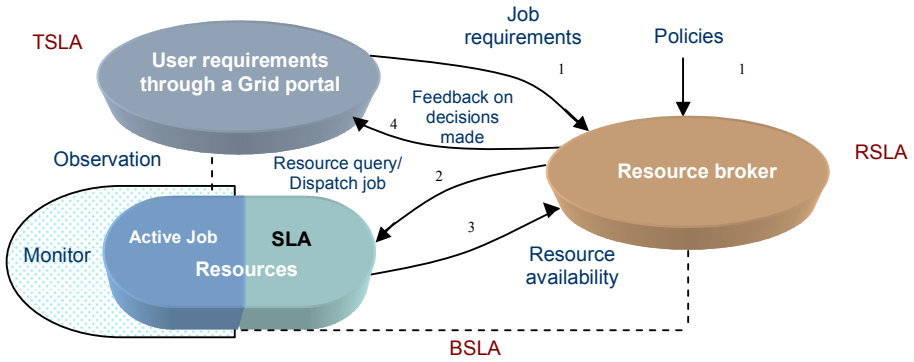

Figure 3.1: An overview of the SNAP-based resource broker.

It is important to state that, in this research and in the SNAP framework, the term "negotiate" is used conceptually, without the use of recursively refining and bargaining on agreements. Thus the method of acquiring and gaining resources for a job is conducted by finding resources that match the resource description attributes provided by the user with those that are advertised by a resource provider. This forms the basis for accepting or rejecting a resource during the selection process as to where to submit the job as well as any assurances specified by the provider, such as the job will not be interrupted during its execution. An analogy to this form of acquiring resources is for example buying an electrical item such as a laptop. The customer knows the type and specification of the laptop he/she would like to

purchase i.e. screen size and resolution, processor type and speed, graphics card memory, etc. The customer would then explore what is on offer at the various stores and what type of warranty (assurance) is provided by the vendor such as one or two years on site service. Having explored the various deals the customer would select the most favourable offer from one of the stores. Conversely the other approach would be to have a custom built laptop, which would involve bargaining what features should be included. In contrast it is inevitable that the latter would take longer than the previous approach to obtain the merchandise as it is not ready to purchase off the shelf.

Due to the nature of real time applications such as that in the DAME project it is necessary to obtain resources with minimum time and to avoid potentially unbounded periods of time where negotiation involves recursive refinement of agreements.

Additionally as shown in Figure 3.1, the architecture facilitates mechanisms to monitor the progress of jobs, which has been incorporated into the broker [106], in order to ensure agreements are honoured and not violated. However this is out of the scope of this thesis.

Overall the SNAP framework is an appropriate method in the design and implementation of a user-centric resource broker, since it provides the means to acquire resources that meet the user's job requirements through the decomposition of SLAs in a layered formation. Further, SNAP can also be mapped onto a range of existing local resource managers, to deploy its beneficial capabilities without requiring wholesale replacement of existing infrastructures.

3.3 SNAP-based Grid Resource Broker Architecture

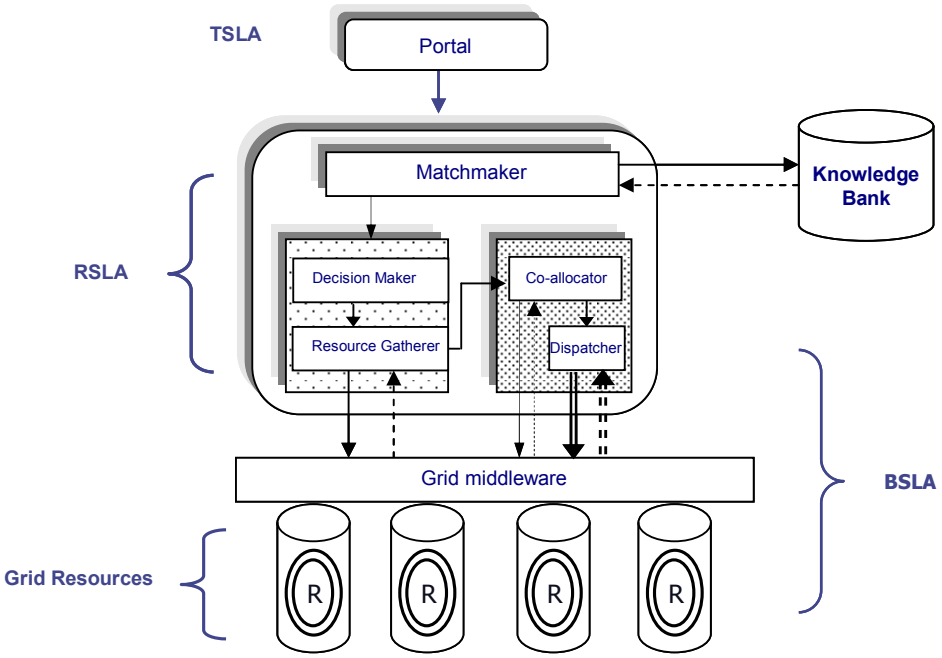

Figure 3.2: A Grid resource broker architecture within the SNAP framework.

As outlined in Section 3.1.2 the process of submitting a Grid job is a long and tedious task, requiring substantial knowledge in how the various mechanisms of the Grid operate. With the use of a user-centric resource broker its primary goal is to adhere to the user's resource requirements and to insulate the user from the Grid middleware complexities. Further, it also serves to alleviate the user from the burden of having to know the various processes and intricacies of the Grid. As yet there is no single broker system that fully caters for these aspects or the job submission stages outlined in Section 3.1.2, where the user can provide the job's resource requirements and for the brokering system to handle the various stages of discovering, filtering, nominating and submitting the job to the resources without exposing the user to the Grid middleware.

With the current brokering systems, such as those outlined in Section 2.3, the user needs to interact with the Grid middleware even after the job requirements are provided to the broker. For example Nimrod/G [5, 69] requires a user to create a task farm (plan) through the use of its declarative parametric modelling language before a job is passed for processing. Hence it does not provide automated resource discovery, a list of resources in the form of Globus Gatekeeper contact strings need to be set up manually by the user before brokering takes place. This is also the case with the Grid Broker [76], in that it does not provide automated resource discovery. Furthermore in this system the decision making process is left to the user as to where to submit a job without the system verifying if the resources are available for use. With Condor/G [74] again the user needs to query the information providers manually and needs to write the RSL for the job to be executed at the appropriate resources. As for AppLeS [79-81] in order for a user to submit a job using this system, the user's job application source code needs to be modified and recompiled or filtered through a template (see Section 2.3.5 for further details) to be compliant with the AppLeS scheduling agents. Once this time consuming process is complete, the user is still required to provide an input, stating a list of resources' contact details that the user has the credentials to utilise, to enable the broker to query the resources' status.

The broker architecture that has been developed in this research and is also used by the DAME (Distributed Maintenance Aircraft Engineering) project [9] is shown in Figure 3.2, which is a user-centric resource broker that is based on the SNAP framework and hence is given the name SNAP-based Grid resource broker. The SNAP-based Grid resource broker differs from the current existing Grid resource brokering systems that have been mentioned in Section 2.3. It is distinguished from the rest in that it insulates the user from the Gird middleware by automating the process from the point of receiving the user's resource requirements right through to the submission for execution at the appropriate resources.

The SNAP-based Grid resource broker is used to submit DAME XTO (eXtract Tracked Order) applications. The number of resources required by an XTO application job varies, as it corresponds to the amount of control files that need to be processed. In the simple case scenario (which is used for illustration purposes and to simplify the understanding) 10 resources are required, with the resource specification listed in Table 3.1.

Specification	Details
Job Type	XTO
CPU type	Intel
Minimum CPU Speed	1.4 Ghz
CPU count	10
Minimum RAM	256 Mbytes
Minimum storage capacity	2 Gbytes
Operating system type	Linux
Operating system version	2.4
Directory for the input files	"/home/cserv1_a/pg/mhh/XTO/Control10_files/"
Start time	Instantly (minimum time possible)
Duration	60 minutes

Table 3.1: The resource requirements for an XTO job application.

The resource requirements are provided to the broker through a portal [107], which is a web based user interface. The interface as shown in Figure 3.2 denotes the TSLA part of the SNAP framework as it obtains the requirements for the job which will traverse through the architecture to ensure any resources acquired follow these criteria. The remainder of the SNAP-base Grid broker architecture components are described in the following subsections (Sections 3.3.1 – 3.3.5).

3.3.1 Knowledge Bank

The second layer of the SNAP framework (RSLA) incorporates a *Matchmaker*, which is supplied with the job's requirements in the form of attribute strings ascertained through the user interface. The Matchmaker serves as a component that translates the user's job requirements into a Structured Query Language (SQL) statement prior to contacting the Knowledge Bank (KB). The KB is a data repository holding static information on each of the resourcea a user is entitled to access and is stored in a MySQL database [108]. The information returned to the Matchmaker after querying the KB is a list of resources that meet the user's resource requirements and which the user is entitled to use.

The information stored in the KB as attributes include the CPUs type and speed, the operating system, memory, storage capacity, etc. The relationship tables that are used in the KB are shown in Appendix A. Additional attributes are also used to keep a history of past performance behaviour of a resource. Relating to the latter, resources are classified as low/high priority according to whether they meet a pre-defined level of performance, at present it is based on reliability, i.e. the likelihood of a resource crashing during the execution of a job. For example if a resource often crashes, it is likely to be classified as low priority.

The KB is a significant component in the SNAP-based Grid broker architecture when compared to current brokering architectures such as those mentioned in Section 2.3, as it helps in several ways:

- To facilitate the broker to filter out all resources that could handle the job's requirements prior to contact, to enhance efficiency in avoiding unnecessary processing of resource contact.
- To alleviate the user from the burden of keeping a log of the resources with their attributes and manually farming the task before contacting the broker such as the case in Nimrod/G [5, 69].
- To facilitate the broker in supporting automated discovery of a resource's dynamic status. Currently this is not supported by any Grid resource brokers without the intervention of the user.
- To store a history profile of past performance of resources, to enable the broker to differentiate and categorise resources into different levels.
- Further an analogy to the KB is a telephone directory where information stored directs to a particular service that caters for a user's requirements.

The KB is developed to support the concept of having such a component provide the benefits outlined above. However more significantly it supports the RSLA in the SNAP framework to discover resources suitable for the user's job requirements since there has been no mechanism proposed for this process, apart from the mundane manual method.

Initially all records are manually entered into the KB by an operator, which is the case in most data repository systems [109]. Therefore there are issues of how the KB should be updated and how frequently, as well as what defines the value of the history profile attributes and what further attributes could be included to provide an enhanced description of the past performance of a resource. All these issues are beyond the scope of this thesis as the KB was primarily developed to show its significance and to absorb its benefits. However there are several suggestions how these issues can be addressed:

- **How the KB could be updated and how frequently:** The static information of Grid resources rarely changes. For example the CPU speed, storage capacity, etc are usually upgraded after a year or longer. Updating the static attributes that specify the description of the resources in the KB could be achieved by requesting the information provider or its administrator to inform the KB when an upgrade does occur. Another option could be when the *Resource Gatherer* (which is described in Section 3.3.3) queries the information provider to gain the resource's current dynamic status, it may also update the KB of any changes it has detected concerning the resources static attributes. Alternatively the KB could query all resources known to it periodically, for example once a week, during off peak times (e.g. overnight) to inspect whether there are any changes that need to be acknowledged. Even though the changes in the static information of a resource occurs after a long period of time, when it actually occurs may be unknown, which is why the one week period has been suggested.

- **What defines the history profile attribute:** The history profile attribute is designed with the thought of storing information concerning whether a resource is likely to crash during an execution of a job. This will help determine if it is reliable. At present the attribute called Mean_history_profile in Table A.3 (Appendix A) stores the mean values of either 1 or 0, where 1 represents reliable and 0 unfavourable. The Mean_history_profile is derived from the attributes Num_crashed_jobs and Num_uncrashed_jobs (also shown in Table A.3), which store the frequency of the number of crashed and uncrashed jobs during execution

in these attributes respectively. However this value is inputted when a resource is entered into the KB based on what is known about the resource from past experience. The values in these three attributes are required to be updated after each job run for any given resource, as they will determine the reliability of the resource during run time.

- Other possible history profile attributes: There are a number of possible attributes that could be included into the KB. This inevitably would assist the *Decision maker* (described in Section 3.3.4) to compose a better prediction and judgement of a resource in terms of its performance and behaviour. The attributes that could be included are whether a job would be interrupted or suspended during execution, would a job complete on time and does a resource degrade in performance during run time.

The Condor's Matchmaker [8, 83] described in Section 2.4.2, may seem to resemble the KB in performing a match between the user's resource requirement and a resource description. However the Condor Matchmaker differs as it depends on the resource provider to advertise its resource description through the use of ClassAds, before a match can take place between a resource and a user's resource requirement (which also needs to be in the form of ClassAds). This means the resource provider is inclined to support ClassAds and needs to generate a new advertisement when it next becomes unoccupied. Further the Condor Matchmaker has no notion of a resource's past history performance.

3.3.2 Decision Maker

Once the information is received by the Matchmaker from the KB, it is forwarded to the Decision Maker, which is also part of the RSLA. The Decision Maker evaluates the information and categorises the potential resources into two categories by tagging them as either *blue* or *white*. This corresponds to their significance, i.e. that some resources are reliable and valuable (blue) while others are acceptable (white). The tagging is based on their history profile and if the Mean_history_profile attribute is in favour of the resource it is tagged with blue otherwise with white. An

analogy to this labelling classification is in the network industry where differentiated services [104] are used to prioritise IP packets.

At present in the context of Grid resource brokering architectures there is no component that provides the functionality of the Decision Maker, pre-processing the resources into different categories. This helps in many ways such as providing a prediction of the resources behaviour based on their categorisation. It also aids the Co-allocator (Section 3.3.4) to select the resources to accommodate for the job based on their categorisation.

3.3.3 Resource Gatherer

The Resource Gatherer, which is also part of the RSLA, contacts the information providers on all the candidate resources passed to it by the Decision Maker (described in Section 3.3.2). This is to query the information providers for each of the resources to gather dynamic information on their status. The dynamic information that is gathered includes whether the resources are available or occupied, number of jobs in the pending queue and the load of the resource.

The Information Provider that is used in this research is the Monitoring and Discovery System (MDS) [33, 34, 50]. It has been chosen in comparison to those discussed in Section 2.4, as it is widely used in the Grid community, for example in many e-Science projects [46], the European Data Grid (EDG) [54], and it is also part of the Globus toolkit [2, 3] which is the predominant Grid middleware. Further, it supplies both dynamic and static information in relation to the resources, it also facilitates the extension and inclusion of additional dynamic information which is provided by the default installation, such as the retrieval of local resource management queues. The ClassAds [8, 83] resource discovery was not adopted, as it relies on several dependencies and can cause indefinite delay in forming a match between the user requirements and a resource as discussed in Section 2.4.2. The Network Weather Service (NWS) [77, 78] focuses on disseminating performance predictions, based on recent events in relation to the time when it was queried and does not provide the current status of a resource (see Section 2.4.3 for further details). Finally the A4 agent based hierarchical approach to resource discovery (discussed in Section 2.4.1) is scalable. However a discovery could be a long

process especially if the request traverses through many other agents, which is time critical for applications that require resources on demand.

3.3.4 Co-allocator

It is often the case that a Grid application job requires multiple resources to be allocated simultaneously as with the DAME XTO (further examples are those mentioned in Section 2.2). The multiple allocation is handled by a co-allocator [6] and its role is to select the most appropriate and available resources for the job. The allocation of the resources may be performed immediately or in the future, depending on the need of the application job. For example a real time application job such as DAME XTO requires resources on demand. Conversely non critical time application jobs such as those for the Grid Particle Physics simulation [110], can be scheduled in advance as they are not bound to any time critical schedule.

Providing assurances for the execution start time of a job is attained through reservation and application jobs such as those for the Grid Particle Physics simulation simplify the process of co-allocation as it can be planned in advance. Further there are several mechanisms available that can help advance reservation and co-allocation namely GARA (Globus Architecture for Reservation and Allocation) [85, 86] and Maui [12], even though they have their limitations as mentioned in Section 2.5. However immediate reservation complicates the co-allocation as the resources have to be secured on demand with limited time. Different forms for immediate co-allocation and reservation are listed below:

- Acquire surplus resources than that required by the user's job in case the reservation is unsuccessful on some resources. However the Grid is moving into an economically driven infrastructure [69, 111] where resources would be leased in exchange for money [112-114]. Adopting this approach would severely drain an organisation or an individual budget wastefully on resources that are not required by the job but have been reserved as an extra provision.

- Secure at least *N* out of *M* requested resources [1] (where N is less than M) for the user's job and then gradually assign more resources over time. This is adequate for non real time application jobs and where the

outcome of the job does not impact on subsequent procedures after its completion. However this is not acceptable when all resources are needed in order for the job to begin execution. For example in a visualisation application where both computational and rendering resources are required, the job cannot begin without the complete set of those resources requested. Further, other application jobs like DAME require all resources to be available simultaneously and the outcome of the job is expected to be delivered under constraint time as it will have an impact on subsequent procedures.

- Secure only the necessary and correct number of resources for the job. This would enable the simultaneous execution of the entire job and without wastefully using unnecessary resources. Hence this is the approach adopted by this research.

The process of nominating resources during co-allocation is also part of the RSLA and is based on those resources that are unoccupied. The Co-allocator first prioritises its selection by choosing resources that have a good history profile, which were tagged as blue by the Decision Maker. If there are insufficient blue tagged resources to handle the job it moves on to the less prominent resources which were tagged as white until the correct number of resources have been reached for the job. Once it has selected the resources to handle the job it secures them for utilisation through the use of immediate reservation. Note that the Co-allocator does not need to evaluate the resources based on whether they have the appropriate hardware specification to handle the application job's requirements as this has already been dealt with by the KB prior to contacting the resources.

However in the worst case scenario where insufficient resources are available, the application job will not be able to run. The user is informed of the situation and the broker responds based on the feedback from the user, for example the user could request the broker to wait until there are sufficient resources or to abort the task.

3.3.5 Dispatcher

Once the resources are secured the final procedure, which is part of the BSLA is carried out by the *Dispatcher*, which binds the job to the resources. The Dispatcher firstly generates the appropriate RSL (Resource Specification Language) for the nominated resources. It then transfers any data files that the job depends on for it to be processing with any execution scripts. This is then followed by the initiation of the job through the submission of the RSLs which in turn begins to run the application job.

3.4 Securing Resources

Local resource management systems are centralised and all resources are governed under a single administrator. However the Grid needs to cater for multiple users from different sites, who may potentially be interested in the same resource, without each other's knowledge of their existence or interests [6, 7]. Thus, as indicated in Section 3.3.4, once resources have been nominated for utilisation, the Co-allocator then needs to secure the resources through immediate reservation. This process is complicated on the Grid not only by the time constraints but also with the resources spread over several domains and the potential third party users who may be interested in the resources nominated by the broker. The current approach adopted for resources on demand in the Grid is the two-phase commit protocol [115]. This protocol is not related to that which has the same name used for example in databases [116, 117]. The steps for the Grid two-phase commit protocol are explicitly described below:

1) Identify resources appropriate for running the application job and secure (reserve) the chosen resources. This is part of the RSLA in the SNAP framework.

2) Submit the application job for execution. This is part of the BSLA in the SNAP framework.

In the remainder of the thesis a broker following this protocol, without the additional enhancements discussed in Section 3.4.2, is referred to as a simple SNAP broker. Section 3.4.1 outlines, by means of a scenario, the need for a more sophisticated approach that will lead to the description of the proposed three-phase commit protocol solution in Section 3.4.2.

3.4.1 Motivating Scenario

Once resources have been secured, the simple SNAP broker can transfer any necessary data and submit the job for execution. However, as the following scenario illustrates, difficulties can arise in successfully securing resources.

Having received a user's request to run an application job and details of its requirements, a resource broker probes various sites that encompass resources, which could cater for the user's request. The contacted resources return their current status at varying intervals [118, 119], as they are distributed and independent from each other. At the point of receiving a response from all providers, the broker moves into a state of co-allocating the job based on the information gathered. Before submission of the job, the resources must be reserved. However time has elapsed since their last confirmation and other candidates (unknown to the broker) may have committed to some or all the resources it decided to use. If this occurs prior to reservation, alternative resources must be identified to replace those which are no longer available and the co-allocation process must be revisited. This process could repeat itself and could lead into an oscillation between the broker and the resources without a successful job submission. Even if the job is eventually submitted successfully, such a scenario could significantly delay the execution start time.

Note: in order to avoid deadlock if some resources that have been nominated to host the job fail during reservation then all resources for that specific attempt are released [43]. This does not only avoid deadlock but in an economic model it allows the Co-allocator to reassess the available resources and plan according to the budget available to the user.

3.4.2 Three-phase Commit Protocol

The key problem highlighted by the scenario above (Section 3.4.1), is that information obtained about resources from their information provider may be out of date by the time the Co-allocator makes a decision as to where the job should run and proceeds to attempt to reserve the chosen resources. This problem arises, since the broker does not know if the status of a resource changes until it re-contacts its information provider. An efficient solution will be to receive a signal from each resource if its status changes rather than needing to contact the information provider on each chosen resource.

The proposed protocol follows the two-phases of the simple SNAP broker, indicated in Section 3.4. However the first phase is separated into two parts, to enhance the current two-phase commit protocol and strengthen it through the use of probes. The probes are software sensors to enable rapid update of changes in the status of the resources that are under consideration for utilisation by the broker. Hence the name three-phase commitment protocol, with its phases listed below:

1) Identify resources appropriate for running the application job (RSLA) and simultaneously initiate probes to provide rapid updates of the resources status.

2) Secure (reserve) the chosen resources (RSLA).

3) Submit the application job for execution (BSLA).

Specifically, when the Resource Gatherer (Section 3.3.3) queries the information providers on candidate resources, it simultaneously transmits probes to the resources, thereby entering into the first phase of the commitment. The purpose of the probes is to enable the broker to be kept updated while waiting for all queries to return at their various times. This helps to reduce the likelihood of the oscillation situation outlined earlier as it provides a constant view for the broker of the resources' status. This allows the broker to remain up-to-date while the information providers report back to the broker. The approach of having the information

providers broadcast resource status is more efficient than having the broker repeatedly contacting the information providers for updates after the initial contact.

The probes are created by spawning a thread that connects to a server which is associated with the resources that are governed by a scheduler (in this research the scheduler used is SGE). Currently the information provider, Monitoring and Discovery System (MDS) [33, 34, 50], provides information on request and does not broadcast the information. The server has been developed to enhance the current system by providing this facility. The information which is broadcast is not sensitive as it is that which can be retrieved when querying the information provider anonymously. The information disseminated serves the purpose of informing the broker if the resources have been taken or those that were occupied have become available. The broadcast of the information has a low (almost negligible) overhead as it is a minor modification to SGE i.e. using the signals generated by SGE prolog and epilog facilities indicating the start and end of a job respectively. The prolog and epilog facilities are provided by most schedulers such as PBS and LSF.

Once all information provider queries have reported back to the broker and updates from the probes are acknowledged, the information is forwarded to the Co-allocator, which executes the second phase of the commitment, by nominating resources to handle the job. It then informs the probes associated with the nominees to request the resource's information provider to evolve into the amber state. On such a request the information provider would reserve the resources and display the amber state to present an indication to any candidate interested in its use that it has entered a transition phase. This indicates to another user that though the resource is not active, it is unavailable.

Other active probes are not destroyed until after the final phase has been completed as this facilitates the flexibility to use them if the second phase fails on some resources.

Once the resources are secured through the use of immediate reservation during the amber state, the third and final phase is executed by the Dispatcher which upgrades the resource's status from the amber to the red state during the submission of the job to the resources. This phase binds the job to the resources and their information provider signals to any incoming client that the resources are active and have been committed. Figure 3.3 shows the evolution of the resources from being categorised by the broker to the final commitment stage. The protocol ensures other

candidates interested in the same resources are aware of the resource's status through its colour coded system. Hence the three-phase commit SNAP broker is expected to provide a performance enhancement over the simple SNAP broker in terms of the time interval between submission (to the broker) of user requirements and the job beginning execution, as it avoid having to re-contact the resources' information provider after the initial contact as it uses the probes for updates.

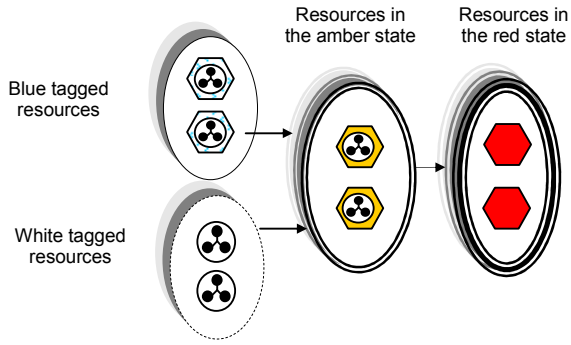

Figure 3.3: Resources are categorised by the broker as either blue or white depending on priority. Resources are then asked to evolve into amber for them to be secured and then into the final commitment stage (red state).

3.5 Summary

The chapter begins by providing an overview of submitting a batch job directly to a local scheduler, the approach commonly used in distributed systems then through the Grid. This outlines the difficulties and the extensive knowledge a user requires in order to submit a job through the Grid. It also highlights the need for a Grid resource broker to insulate the user from the Grid middleware complexities. This is followed by a review of current Grid SLAs (Service Level Agreements) that leads on to the discussion of the SNAP (Service Negotiation and Acquisition Protocol) framework. The SNAP's three layers are described, namely TSLA, (Task Service Level Agreement) RSLA (Resource Service Level Agreement) and BSLA (Binding Service Level Agreement). The chapter also describes the SNAP-based Grid resource broker architecture with its various components including the use of the Knowledge Bank that helps to automate the contact of resources. Then the need to secure resources is discussed, with the use of the two-phase commit protocol. A

scenario to state the need for a more sophisticated approach is presented and the chapter ended with a description of the three-phase commit protocol.

Chapter 4

Performance Evaluation of the Three-phase Commit Protocol

In this chapter a performance evaluation of the SNAP-based (Service Negotiation and Acquisition Protocol) Grid resource broker using the three-phase commit protocol compared to the simple SNAP broker (both described in Section 3.4) will be presented. Section 4.1 provides an overview of the experiments and objectives for both the Grid test-bed and the White Rose Grid (WRG) environments. Section 4.2 will describe the Grid test-bed, the experimental design and performance results. Section 4.3 will follow the same structure as the previous section with the focus on the WRG. This will also include a discussion of the challenges during the deployment for both brokers on the WRG. Section 4.4 compares and evaluates the experiments carried out on both environments and finally Section 4.5 summarises the chapter.

4.1 Overview of the Experiments and Objectives

As discussed in Section 3.4.2, the three-phase commit SNAP broker is expected to provide a performance enhancement over the simple SNAP broker in terms of the time interval between submission (to the broker) of the user's job requirements and the job beginning execution. In particular, the vision of resource status provided by the probes used in the three-phase commit protocol are expected to provide an enhancement by ensuring that decisions are made on the basis of up-to-date information. Specifically, the experiments involve a comparison of the performance of the three-phase commit SNAP broker, compared to the simple SNAP broker on two different environments, the Grid test-bed and then the White Rose Grid (WRG).

The experiments carried out on both environments are designed on the basis of two common objectives which are:

- To show that the simple SNAP broker and the three-phase commit protocol have been successfully implemented.
- To investigate the behaviour of the three-phase commit SNAP broker when scenarios occur in which a performance enhancement over the simple SNAP broker is expected.

The experiments were firstly carried out on a local Grid test-bed as it is an environment used for developing, testing and evaluation Grid technology research. These experiments were then carried out on the WRG, a large distributed Grid infrastructure to validate the results obtained on Grid test-bed. The WRG infrastructure (which is further described in Section 4.3) spans across three administrative sites and holds true Grid attributes described in Section 2.1, such as site autonomy and a heterogeneous substrate. Even though the same experiments were carried out on both environments the parameter boundaries differed due to the different infrastructure size. This will be discussed when describing the experiments for each environment. Further the DAME (Distributed Aircraft Maintenance Environment) XTO (eXtract Tracked Order) [9] jobs are used for all experiments and named user's jobs throughout the chapter.

4.2 Experiments on the Grid test-bed

The experiments were performed on a Grid test-bed consisting of 10 machines. Each machine has a Pentium IV processor (1.2GHz) with 256MB RAM. The operating system is Linux 2.4. Globus 2.4 [2, 3] and Sun Grid Engine 5.3 (SGE) [19] are installed on all machines. There is a Grid Resource Information Service (GRIS) associated with each of the machines. Communication occurs with a fast (100 Mbps) LAN network. This is acceptable, since network bandwidth and latency issues are not addressed in these experiments, although the possibility of varying information providers response times is considered. The main attributes that affect

the information provider response time are the number of concurrent users accessing the service and the load on the resources [118, 119].

4.2.1 Grid test-bed Experimental Design

The experiments carried out can be described in terms of two scenarios:

> **Scenario 1:** The resources appropriate for the user's job are taken and the broker must wait until they become free before submitting the user's job.
>
> **Scenario 2:** While the broker is in the process of making a decision as to where the user's job should be submitted another third party job is submitted (see below for a more detailed description).

The first scenario provides a setting in which the effectiveness of the probes in providing a vision of the resources can be investigated. Specifically, the user's job to be submitted requires 3 resources. Each of the two brokers (simple SNAP broker and the three-phase commit SNAP broker) is considered in turn. Each is given access to only 3 of the resources and on each of these a third party job is running for a fixed duration.

Two experiments are performed, based on this scenario. In the first experiment, the third party job submitted occupies all 3 resources and has a fixed duration of 30 seconds, which is adequate time to ensure the resources are unavailable when their information provider is initially contacted by the brokers. This job is submitted immediately before the broker is initiated to start. The information provider response time (i.e. the GRIS response time) is then varied between 10 and 90 seconds. This is sufficient to ensure the two brokers can cope with the effects of a real environment such as the White Rose Grid (WRG) [11] which has a normal response time that can run up to 90 seconds. Additionally, this time may vary considerably depending on, for example, the number of users concurrently accessing the same GRIS [118, 119] and the load on the machine. On the Grid test-bed the information provider response time is fairly stable at 8 seconds for the GRIS on each machine. The default installation of the information provider was changed to ensure non cached

information is retrieved. Using cached information would result in a quicker response time but consequently the information retrieved may be out of date.

The response time was varied by adding a variable delay into the code. The time taken between the broker beginning execution and the broker becoming aware that the resources are free is then recorded, in addition to the time taken before the user's job begins execution and the number of information provider contacts made.

In the second experiment based on scenario 1, there is no artificial delay in the information provider response time. Instead, the duration for which the resources are unavailable, which was previously fixed at 30 seconds, is varied between 30 and 110 seconds incrementing by 10 seconds at each interval. This variation is sufficient to help investigate the effectiveness of the probes and the impact it has on the number of information provider contacts made by the brokers.

For the simple SNAP broker, as soon as the information providers inform the broker that the resources are free, the time is noted and stored. For the three-phase commit SNAP broker, the time is stored when a signal (generated by the probes) has been obtained from all three resources that they are free.

The experiment based on scenario 2 is used to investigate the three-phase commit SNAP broker's performance when resources are initially free but their status changes during co-allocation, as before, the broker requires three resources. In this experiment the broker has access to all 10 resources. Note that if a resource is taken during co-allocation, the simple SNAP broker only becomes aware of this when it re-confirms with the resources before committing and reserving them. In order to highlight the scenario whereby the broker is required to repeatedly contact the information provider and attempt to co-allocate the user's job, additional third party jobs are submitted at time intervals chosen to coincide with each attempt at co-allocation that the simple SNAP broker makes. Additionally, the resources taken are chosen to be the highest priority available so that there is always a conflict between third party jobs and the broker. This experiment is used to determine whether the vision of the resources that the probes provide in the three-phase commit protocol do indeed enable the broker to obtain fast enough updates to decrease the likelihood of oscillation between broker and resources.

Overall the two scenarios used for the experiments are chosen primarily to outline the benefit of the use of the probes in the three-phase commit protocol and to demonstrate their effectiveness compared to the simple approach. Further, certain

factors such as the limited amount of resources and the time restrictions in using them narrowed the scope for more realistic scenarios to be evaluated. However this has been addressed in chapter 5 through the use of mathematical modeling and simulation.

4.2.2 Grid test-bed Experimental Results and Discussion for Scenario 1

The results for the first experiment relating to scenario 1 on the Grid test-bed are shown in Figures 4.1, 4.2 and Table 4.1. Figure 4.1 shows the time taken between the broker beginning execution and the broker becoming aware that the resources are free. Figure 4.2 shows the time taken between the broker beginning execution and the user's job beginning execution. Finally Table 4.1 shows the number of contacts each broker made to the information provider to gather resource status updates. All the results for this experiment are based on the function of the information provider response time.

The three-phase commit SNAP broker becomes aware that the status of the resources has changed much faster than the simple SNAP broker as a consequence of the use of probes (shown in Figure 4.1). The three-phase commit SNAP broker compared to the simple SNAP broker gains a performance improvement of 26% when the information provider response time is 10 seconds and 50% when it is 90 seconds (both percentages are calculated using equation 1).

$$\text{Percentage improvement} = \frac{\text{simple SNAP broker} - \text{three-phase commit SNAP broker}}{\text{simple SNAP broker}} * 100. \quad (1)$$

Clearly this shows when the information provider response time increase so does the performance of the three-phase commit SNAP broker. This is related to the probes used by the three-phase commit SNAP broker and not the traditional process of querying the information provider used by the simple SNAP broker. Subsequently, the user's job begins execution sooner when the three-phase commit SNAP broker is used (Figure 4.2) providing a performance improvement of 13% for 10 seconds and 39% when it is 90 seconds (both percentages are calculated using equation 1). Usually, the longer the information provider response time, the longer it takes before

either broker is aware that the resources are free. For the three-phase commit SNAP broker, there is only one contact with the information provider, as shown in Table 4.1, and then the broker relies on the probes for its updates. Hence increasing the response time has little effect until it exceeds the 30 second period for which the resources are taken. For the simple SNAP broker, the effect is apparent even for quicker response times. However, note that the time taken before this broker becomes aware of the change in resource status is shorter when the response time is 40 seconds than when the response time is 30 seconds. This is due to the fact that when the response time is 30 seconds, the simple SNAP broker needs to contact the information provider three times before it is aware of the change in status, while if the response time is 40 seconds, only two contacts are required, as shown in Table 4.1.

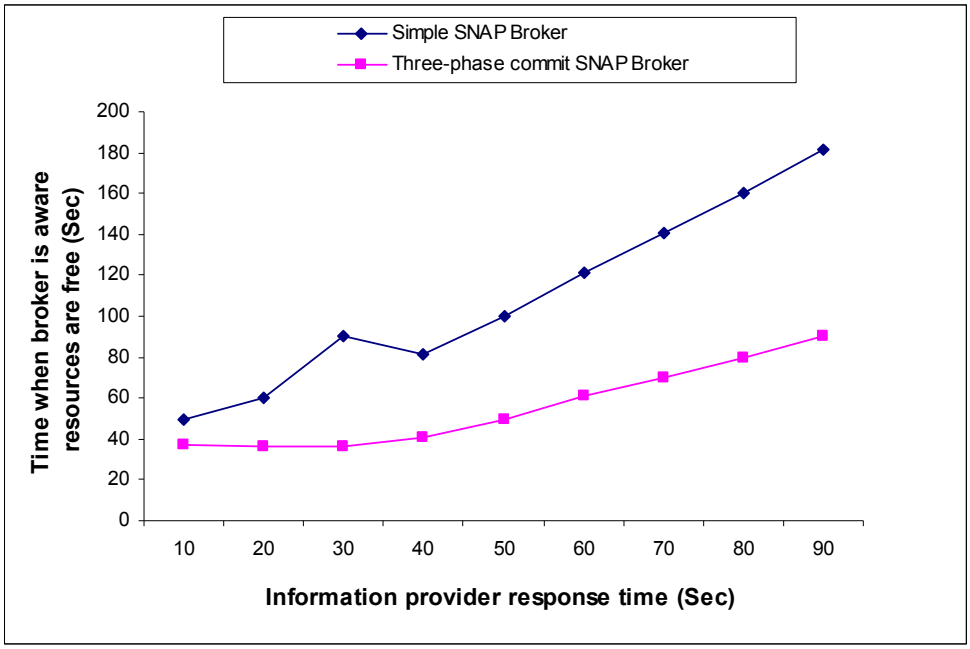

Figure 4.1: Time taken for broker to determine resources are free, as a function of information provider response time.

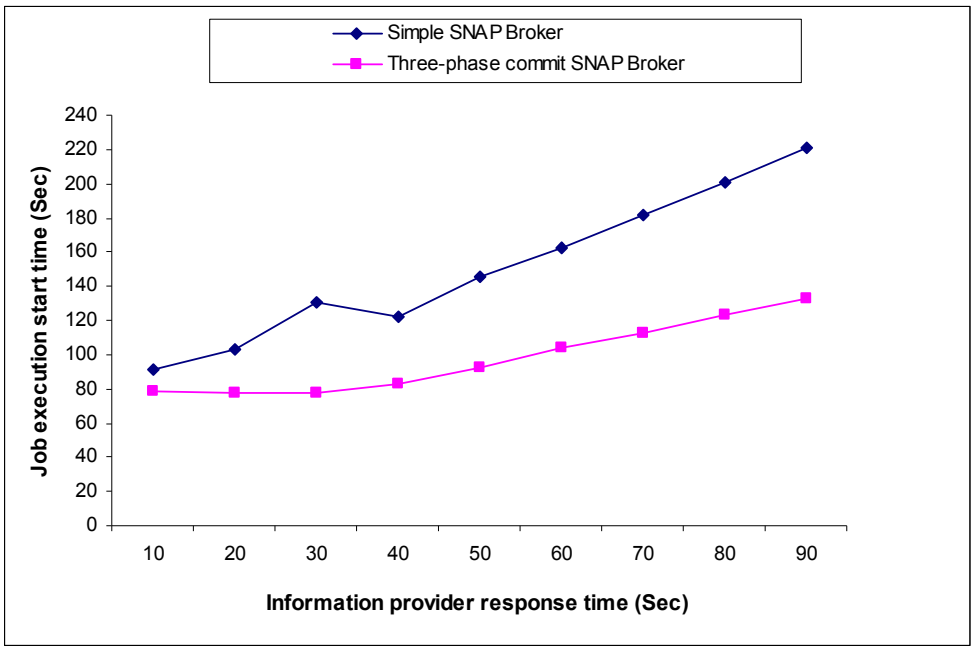

Figure 4.2: Time taken for the user's job to begin execution, as a function of information provider response time.

There is a similarity in the performance trend for both Figure 4.1 and 4.2. This is related to the fact that the difference between the two brokers is the timing when they become aware the resource's status has changed. Once they discover the resources are freed (Figure 4.1), they follow the same procedure in reserving and submitting the user's job, hence both brokers incur the same cost of 42 seconds which is the total time for reservation and job submission to the resources.

As stated earlier the three-phase commit SNAP broker only needs to contact the information provider once and then relies on the probes for updates. However the simple SNAP broker needs to make at least two contacts before it is aware the resources are free as shown in Table 4.1. The number of contacts it makes to the information provider increases if the response time is less than the time duration in which the resources are occupied by the third party job.

The results for the second experiment relating to scenario 1 are shown in Figures 4.3, 4.4 and 4.5. The attributes evaluated in this experiment are similar to that of experiment 1 discussed above. However the experiment differs in that it is based on the function of time for which the resources are unavailable. Figure 4.3 shows the time taken until the broker became aware that the resources were free, Figure 4.4 shows the time taken until the user's job begins execution.

Figure 4.5 shows the number of contacts each broker made to the information provider to gather resource status updates. As before, the three-phase commit SNAP broker exhibits improved performance due to the use of probes. It becomes aware the resources are free quicker than the simple SNAP broker by an average performance improvement of 22% (calculated by equation 2).

$$\text{Average percentage performance improvement} = \sum_{i=1}^{N} \frac{SSB_i - 3CSB_i}{N.SSB_i} * 100 \quad (2)$$

Where SSB_i = Simple SNAP broker and $3CSB_i$ = three commit SNAP broker.

Subsequently, the user's job begins execution sooner when the three-phase commit SNAP broker is used providing an average performance improvement of 13% (calculated by equation 2)

Information Provider Response time	Simple SNAP Broker	Three-phase commit SNAP Broker
10	5	1
20	3	1
30	3	1
40, 50, 60, 70, 80 and 90	2	1

Table 4.1: Number of contacts made by each broker to the information provider as a function of information provider response time.

Figure 4.3 is similar to Figure 4.4 in its performance trend, this is again related to the fact that both brokers incur the same cost for the reservation and the submission of a user's job on the nominated resources, which is on average 42 seconds. Further, the difference in time between the two brokers in Figures 4.3 and 4.4 is on average 20 seconds. This is related to the fact that the experiments are based on the function of time for which the resources are unavailable and not on the information provider response time as was the case for experiment 1 in scenario 1. Hence the simple SNAP broker becomes aware the resources are released shortly after they become freed, despite this fact the three-phase SNAP broker still outperformed the simple SNAP broker, as it uses the probes.

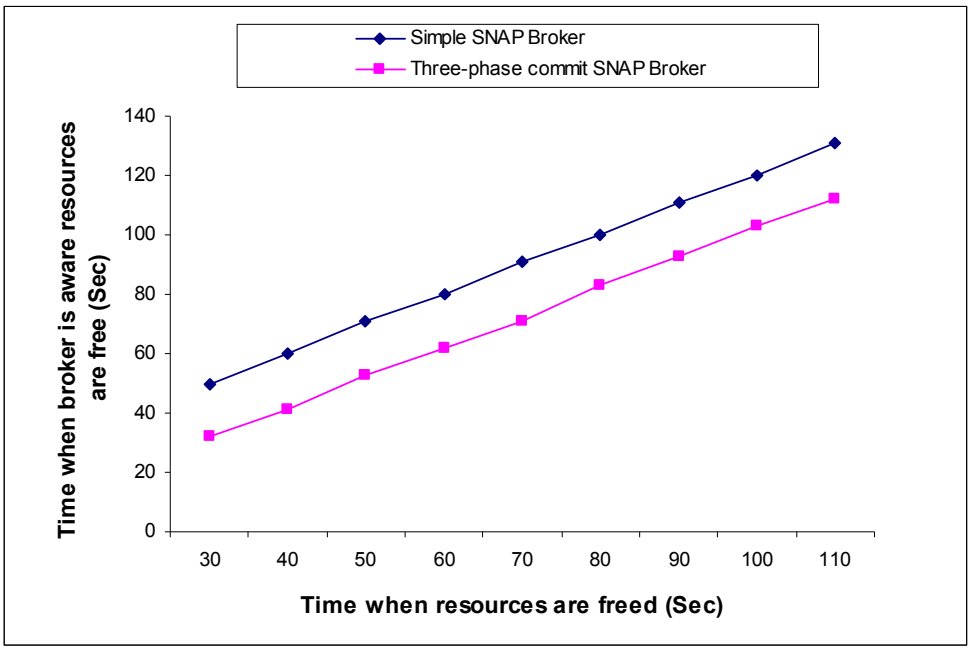

Figure 4.3: Time taken for the broker to determine resources are free as a function of time for which resources are unavailable.

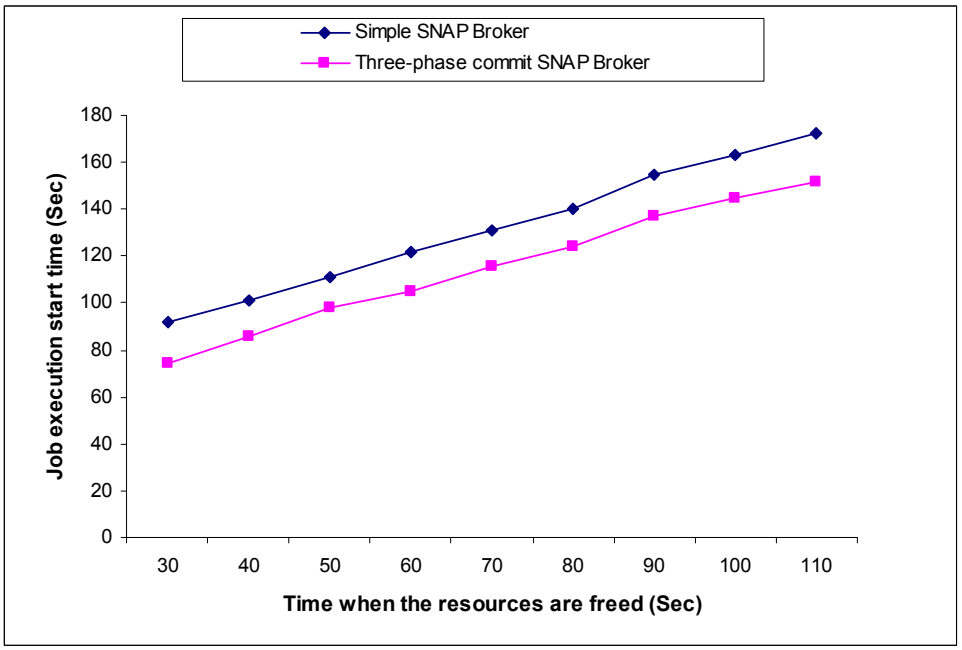

Figure 4.4: Time taken for user's job to begin execution as a function of time for which resources are unavailable.

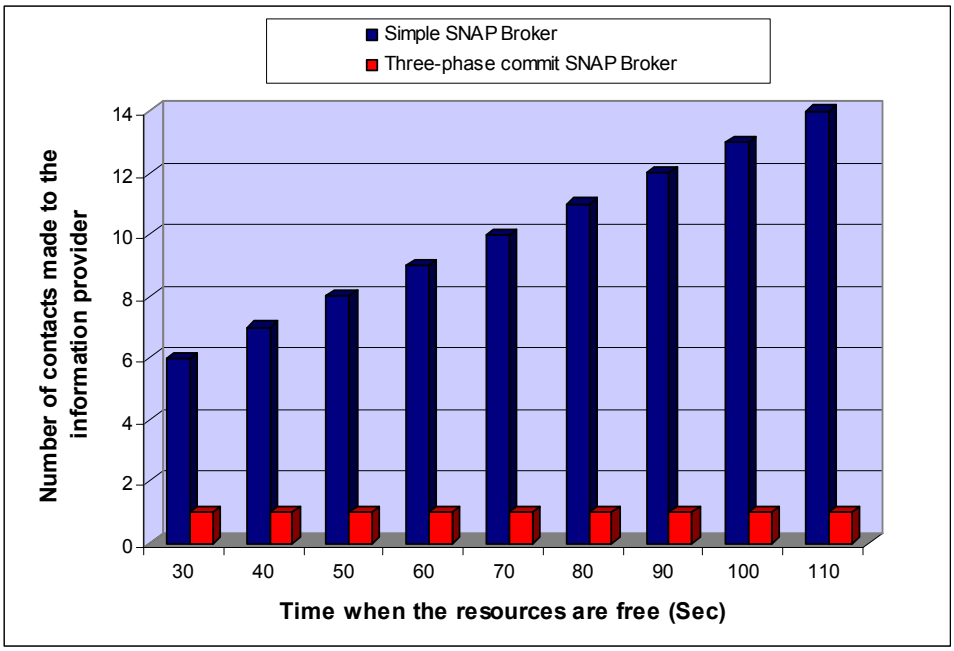

Figure 4.5: Number of contacts made by each broker to the information provider as a function of the time for which the resources are unavailable.

The number of contacts made to the information provider (shown in Figure 4.5) by the simple SNAP broker increases, as the time in which the resources are occupied by third party jobs increased. However results in experiment 1 scenario 1 shown in Table 4.1, the number of contacts made to the information provider stabilised with only two contacts compared to experiment 2 scenario 1 (shown in Figure 4.5). This is related to the fact that experiment 2 scenario 1 is based on the function of time for which the resources are unavailable.

4.2.3 Grid test-bed Experimental Results and Discussion for Scenario 2

The results of the experiment relating to scenario 2 on the Grid test-bed are shown in Figures 4.6 and 4.7 and are based on the function of number of additional third party jobs submitted while the broker is in the process of co-allocating the user's job. Figure 4.6 shows that the three-phase commit SNAP broker takes just over 40 seconds to submit and begin the execution of the user's job, irrespective of the number of third party jobs submitted. Since the probes ensure rapid updates of the

status of the resources, the three-phase commit SNAP broker is aware that a resource has been taken very quickly after it occurs and consequently chooses an alternative resource. It successfully submits before any other resources are taken and provides a performance improvement of 36% when 1 resource is taken and 73% when 7 resources are taken (both percentages are calculated using equation 1) compared to the simple SNAP broker. Clearly, the simple SNAP broker takes longer when more resources are taken, since it is unable to identify changes in resource status fast enough to successfully submit the job before more resources are taken.

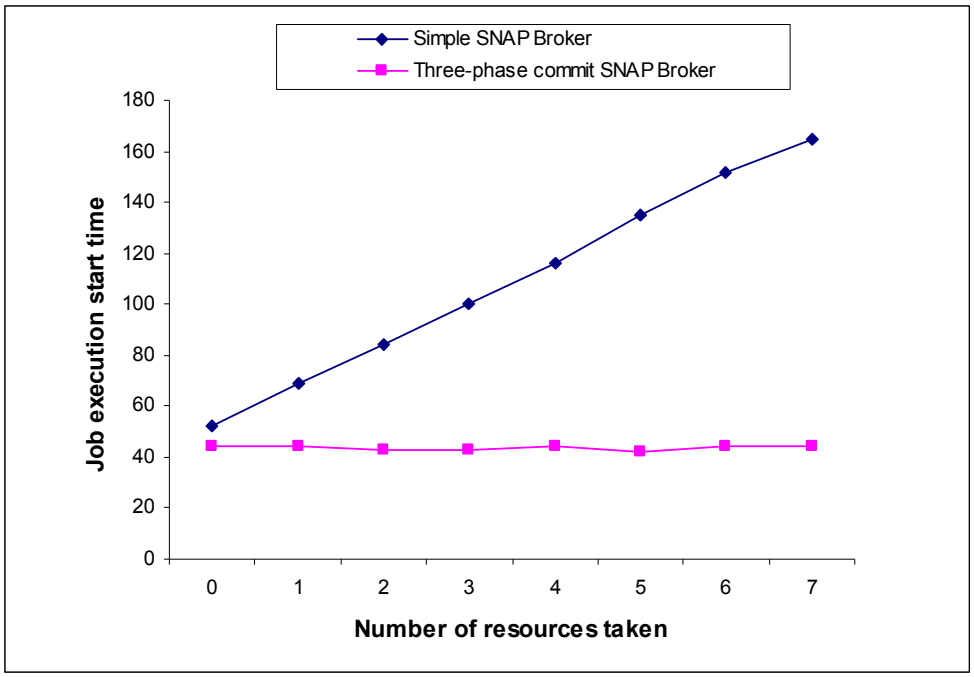

Figure 4.6: Time taken to begin the user's job execution as a function of the number of additional jobs submitted.

Figure 4.7 shows the number of information provider contacts each broker makes before the user's job is submitted. The three-phase commit SNAP broker only makes the initial contact however for the simple SNAP broker the number of contacts to the information provider increases as the number of resources are taken. This is due to the fact that when it attempts to reserve the chosen resources before submission it is unsuccessful as some are taken. Thus it needs to gather new information to find out which resources are still available as it is unable to know the

status of the other resources without having to re-contact their information provider. This consequently impacts on the start time of the user's jobs as seen in Figure 4.6.

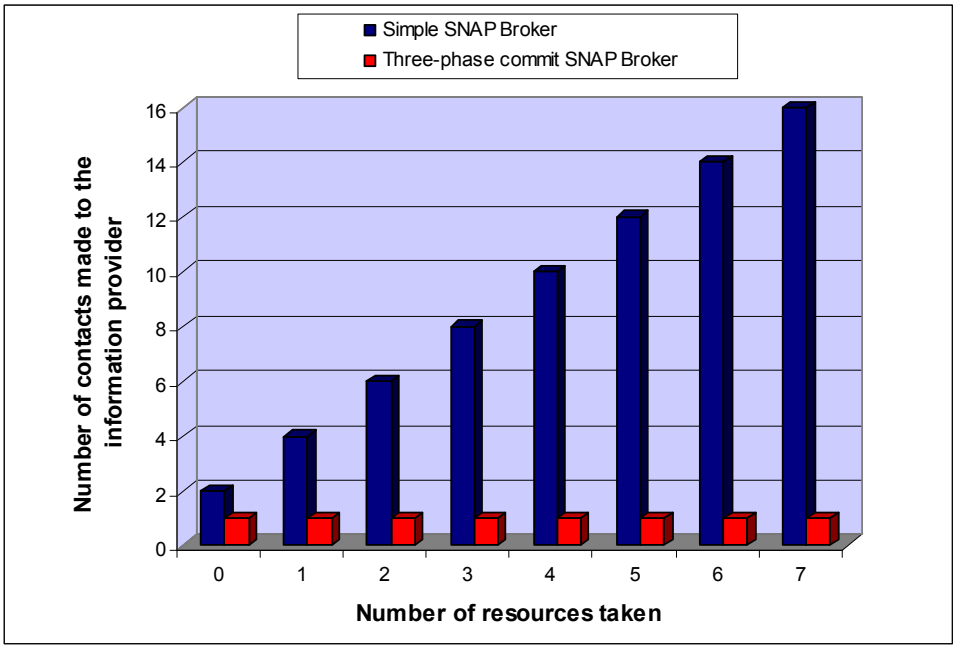

Figure 4.7: Number of contacts made by each broker to the information provider as a function of additional jobs submitted.

4.3 White Rose Grid Experiments

The Grid test-bed is mainly used for development as well as testing and evaluating Grid technology research. However it is a local environment which does not fully include Grid resource management issues such as site autonomy, heterogeneous substrate and, more importantly, the distribution of resources across different sites. The White Rose Grid (WRG) addresses these issues as it is a virtual organisation comprising of three universities: the Universities of` Leeds, York and Sheffield [11]. Its purpose is to deliver stable well-managed services and to provide:

- Cost effective resource management for high-end computing.
- An infrastructure to support research projects which will benefit from access to powerful computing resources.

- Support for scientific communities.
- A market assessment and outreach to industry.

The WRG is heterogeneous in terms of its underlying hardware and operating system. Two large compute nodes are situated at Leeds (Maxima and Snowdon), one at York (Pascali) and another at Sheffield (Titania). These nodes are connected by a fast network and offer significant heterogeneous computational facilities. Figure 4.8 shows the architecture of the WRG and depicts all machines at the various sites. The specification of these compute nodes is described below:

- **Snowdon,** a Beowulf 256 CPU running at 2.2GHz and 2.4Ghz Intel Xeon processors.
- **Maxima,** Sun Fire 6800 server (20 Ultrasparc 3Cu, 44GB memory, 100GB Storage), 5 Sun V880 servers and 2 TB storage.
- **Pascali**, Sun Fire 6800 server (20 Ultrasparc 3 Cu, 44GB memory, 100GB Storage), 1 Sun V880 server and 1TB storage.
- **Titania,** 10 Sun Fire V880 Servers (8xUltrasparc Cu 900MHz, 32GB) and 2TB Storage.

Maxima, Pascali and Titania are built from a combination of large symmetric memory Sun servers and storage backup running on Solaris, whereas Snowdon comprises a Linux/Intel based computer cluster connected with Myricom Myrinet. The middleware infrastructure is enabled through the use of Globus 2.4, while Sun Grid Engine 5.3 (SGE) [19] handles the local job scheduling and is also configured to meet the needs of the users at each site. The WRG is a production Grid and its resources are widely used by various projects such as DAME, Grid Visualisation Middleware (also called gViz) [120] and Modelling and Simulation for e-social Science (MoSeS) [121] just to name a few. Thus, for the purpose of the experiments, a resource from each of the four machines across the three sites on the WRG is provided. This is acceptable as the objective of the experiments is to investigate whether the three-phase commit SNAP broker still provides its enhancement over

the simple SNAP broker over a large distributed infrastructure in a real Grid environment.

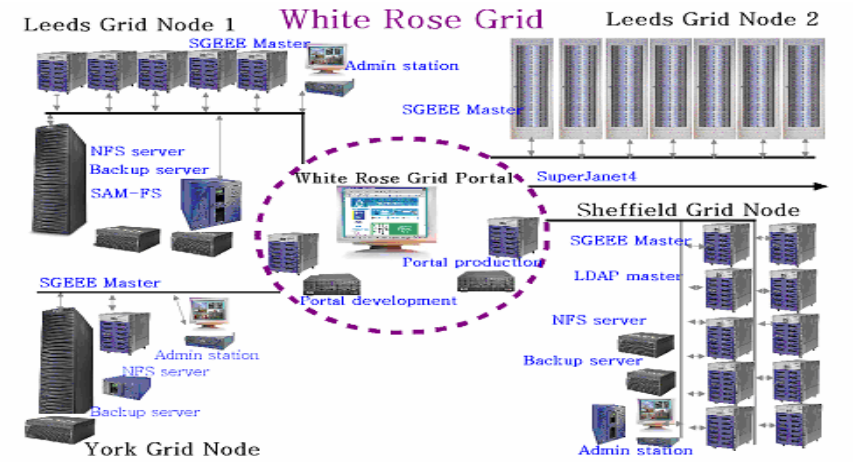

Figure 4.8: The White Rose Grid Architecture [11].

4.3.1 White Rose Grid Deployment Challenges

There were several challenges that had to be overcome in order to deploy the three-phase commit SNAP broker on the WRG, which the simple SNAP broker also incurred as it has identical features apart from the inclusion of the three-phase commit protocol. The challenges can be broken down into two parts which are interrelated. The first relates to the technologies in place and the second to the human collaboration across the three sites. Both are discussed below respectively.

There are two main technologies used across the three sites namely the Grid middleware tool kit which is Globus 2.4 [2, 3] and the local resource management system which is Sun Grid Engine (SGE) 5.3 [19]. The modifications to the technologies are discussed in Appendix B. However performing these modifications to cater for the deployment of the three-phase commit SNAP broker in relation to the Globus services is a consistent and a repetitive process across the three sites, as the technology is standard and all packages have the same structural location, whichever platform they are installed on, Linux or Solaris. Conversely even though SGE 5.3 is the same version across the three sites the configuration of each version can be different. This is related to the heterogeneous substrate mentioned in Section 2.1 where each site's administrator has different preferences in configuring the local

resource management systems. Hence it requires the administrator from each site to provide their SGE site configuration and the queues set up. Despite gaining this information it is still a tedious task to modify SGE, for example the directory location where certain services are located, as they are not consistent on each platform. Overall to perform the necessary modifications to Globus it required a deep and intricate understanding of the technology; once this is gained the modifications are repeated similarly across the three sites as it is a standard and consistent technology. However with SGE due to each site's administrator preference of configurations and setup, a modification on one site would not necessarily mean the same procedural technique would be used on other sites.

The human collaboration between the various sites was the most difficult challenge of all, which was surprising as it was anticipated it would be the compatibility of the technologies such as their configuration that might be problematic. However this was a beneficial experience with several recommendations for future collaboration on such a scale as the WRG. The challenges encountered are listed below:

- **First deployment:** This was the first type of deployment on the WRG that pushed the boundaries of the existing technologies and required them to be modified. This complicated the deployment as there was no past wisdom that could be used. Hence no documents were available to suggest favourable approaches in collaborating between the three sites.

- **Different administrators:** Each of the three sites has its own administrator that maintains the site's resources. Each administrator has various depths of understanding in how the Grid technologies operate and each have different working hours (some are full time while others are part time). This slows the progress as some tasks are suspended until the administrator is on duty. Further, a considerable amount of time was required to explain to the administrators the intricate details of the technologies to enable them to perform the tasks.

- **Permissions:** The WRG is a production Grid, used by several projects. The access permissions to sensitive files and programs are limited and require their modifications to be performed by the site's administrator.

This is time consuming as you are dependent on the administrator to perform each modification. Further not all tasks can be done at once as it is an incremental process that requires testing throughout. If it is discovered during testing a feature does not fully operate, it must be relayed to the administrator for amendment and this could be a recursive process.

- **Distance:** This is one of the major obstacles in the deployment as communication mainly occurs through e-mail and phone calls. Ideally it would be preferable to be with the administrator at the site, as it helps to illustrate concepts through visual aid (diagrams) or execute commands and discuss the outcome. However regular travelling to each site is costly and requires a lot of time and energy.

- **Different infrastructures:** Even though this has been mentioned as part of the technology challenges, the three sites' administrators had to correspond with each other, to ensure the outcome of some modifications occurred correctly. However as each site had different configurations, in particular SGE, the modifications made had to be further fine tuned at some sites due to their infrastructure.

The recommendations for collaborating on such a scale as the WRG are listed below:

- Ensure each site administrator's technical background knowledge is known, preferably before the deployment of a broker. This will help when explaining the reasoning behind the modifications to the technologies.

- Find out which is the best form of communication, whether through e-mail or phone. It is usually the case that for those with a strong technical background an e-mail will be sufficient to request the modifications and for those without a telephone call is recommended as it helps to explain through verbal communication.

- One of the most important aspects to note when deploying a broker on different sites is that administrators maintain resources that are used by several projects and are highly sceptical to altering or modifying any existing setting. It is strongly recommended that the modifications are shown to be fully operational on other similar scale resources. Thus it is best to perform modifications on a test-bed (which is already the case at Leeds) then move to the production resources that are based at the local site (e.g. Snowdon and Maxima for the work in this thesis). This is for several reasons: the individual who is deploying the broker is situated at the local site and is familiar with the administrator, he/she is on site to rectify any problems and is there to thoroughly test the modifications. This approach provides the administrators at other sites with the confidence to perform the requirement specified as it has already been shown to work on other similar scaled resources.

- Having permissions access to sensitive files and programs runs the risk of compromising the security at the various sites, this is why the deployment occurred through the site's administrator. Hence without doubt it would be preferable to be with the administrator when the modifications take place. However it would be time consuming to travel for each modification needed. Therefore the approach taken is to write a list of instructions that needs to be carried out, send a copy to each administrator at the various site and arrange a convenient time to visit them to perform the major tasks. The small tasks can then be performed through e-mail or phone.

4.3.2 White Rose Grid Experimental Design

The issues of how well the three-phase commit SNAP broker performs compared to the simple version, over a large distributed Grid infrastructure such as the White Rose Grid (WRG) is the subject of the experiments described in this section. As with the experiments on the Grid test-bed (Section 4.2) the WRG experiments will investigate whether the resource status provided by the probes used in the three-phase commit protocol still provides an enhancement by ensuring that decisions are made on the basis of up-to-date information. Hence the experimental design for the

WRG follows the two scenarios used in the Grid test-bed (Section 4.2.1) with the various parameter values increased to reflect the WRG environment.

In scenario 1 for both experiments the user's job requires four resources from three sites, one from each of the WRG machines to investigate the effectiveness of the probes across the distributed three sites. In scenario 2 the user's job requires three resources, however in this experiment a combination of all the allocated WRG resources and eight Grid test-bed resources are used. This is due to the nature of the experiment requiring more resources than the actual number required by the user's job, as it investigates the performance when resources are initially free but their status changes during co-allocation. Also the combination of the WRG and Grid test-bed resources is beneficial as it will ensure the brokers can cope with the two different scalable environments simultaneously.

In relation to the first experiment for scenario 1 the additional third party job that is submitted to make the resources unavailable has a fixed duration of 40 seconds. This job is submitted immediately before the broker is executed. The information provider response time (i.e. the GRIS response time) is then varied between 30 seconds and the maximum period before the Monitoring & Discovery System (MDS) [33, 34, 50] on the WRG times-out which is 360 seconds. The value for each run in this experiment is incremented by 10 seconds for each interval. On the WRG the information provider response time from all sites is fairly stable at 27 seconds for the GRIS on average. Hence this is why the third party job for this experiment has a length of 40 seconds, to ensure the resources are unavailable during the first contact each broker makes to the information provider. In the second experiment based on scenario 1, there is no artificial delay in the information provider response time. Instead, the duration for which the resources are unavailable, which was previously fixed at 40 seconds, is varied between 40 and the maximum time the broker will wait before it times-out if it does not find any available resources (which is 360 seconds). Again the value for each run in this experiment is incremented by 10 seconds for each interval.

In the experiment based on scenario 2 the broker has access to 12 resources (from both the WRG and the Grid test-bed). This time the broker requires three resources and it prioritises the WRG resources from any of the three sites. As with the Grid test-bed this is to highlight the scenario whereby the broker is required to repeatedly contact the information provider and attempt to co-allocate the user's job, additional

third party jobs are submitted at time intervals chosen to coincide with each attempt at co-allocation that the simple broker makes. Additionally, the resources taken are chosen to be the highest priority available so that there is always a conflict between the additional jobs and the broker. This experiment is used to determine whether the vision of the resources that the probes used in the three-phase commit protocol do indeed enable the broker to obtain fast enough updates to decrease the likelihood of oscillation between broker and resources.

4.3.2.1 White Rose Grid Experimental Results and Discussion for Scenario 1

The results for the first experiment on the WRG relating to scenario 1 are depicted in Figure 4.9 which shows the time taken between the broker beginning execution and the user's job beginning execution as a function of the information provider response time. Both brokers became aware the resources were freed on average 15 seconds prior to that shown in Figure 4.9, for each time interval, as both brokers incur the same cost for reservation and submission of the user's job to the resources.

Further, for the simple SNAP broker the number of information provider contacts remained constant at two for 50 seconds and all remaining subsequent time intervals. However three contacts are made for the 30 and 40 seconds as the first two occur while the resources are also still occupied, whereas for the three-phase commit SNAP broker only one contact to the information provider is recorded throughout the experiment. The three-phase commit SNAP broker begins the execution of the user's job much sooner than the simple SNAP broker as a consequence of the use of the probes. The three-phase SNAP broker provides a performance improvement of 46% when the information provider response time is 30 seconds and 49% when it is 360 seconds (both percentages are calculated using equation 1). For the three-phase commit SNAP broker, there is only one contact with the information provider, hence increasing the response time has little effect until it exceeds the 40 second period for which the resources are taken. For the simple SNAP broker, the effect is apparent even for faster response times. However the time taken before this broker becomes aware of the change in resource status is shorter when the response time is 50 seconds than when the response time is 40 seconds. This is due to the fact that when the response time is 40 seconds, the simple SNAP broker needs to contact the information provider three times before it is

aware of the change in status, while if the response time is 50 seconds, only two contacts are required.

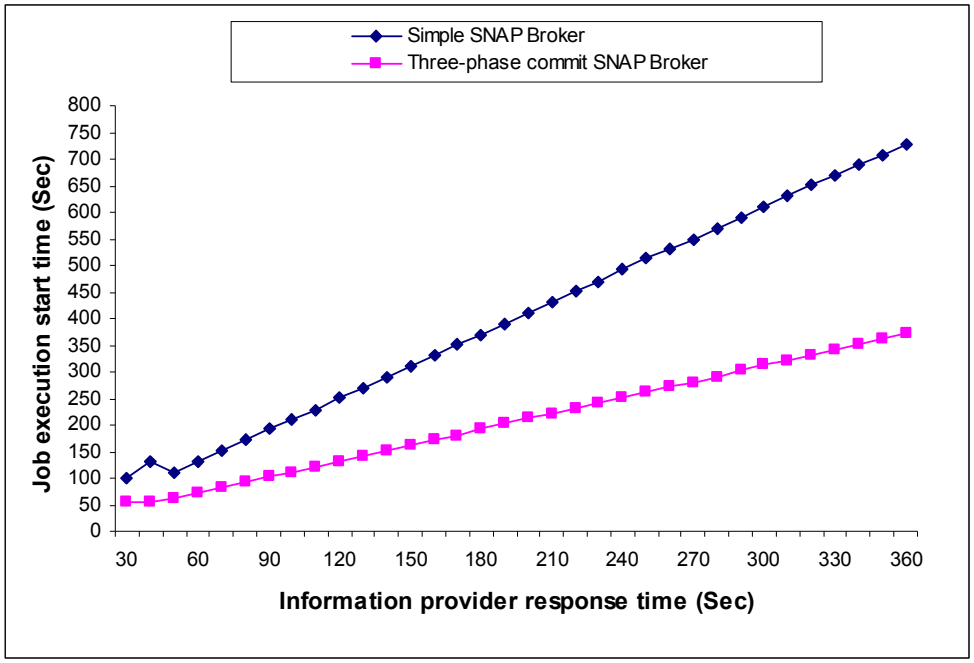

Figure 4.9: Time taken for the user's job to begin execution as a function of information provider response time.

The first set of results for the second experiment relating to scenario 1 on the WRG are shown in Figures 4.10 and 4.11. Figure 4.10 shows the time taken until the user's job begins execution, while Figure 4.11 shows the number of information provider contacts made by each broker. Both are given as a function of the time for which the resources are unavailable, with each run incrementing by 10 seconds during each interval. As shown in Figure 4.10 the user's job begins execution sooner when the three-phase commit SNAP broker is used providing an average performance improvement of 18% (this is calculated using equation 2). As in experiment 1 scenario 1 both brokers became aware the resources were freed on average 15 seconds prior to that shown in Figure 4.10 for each time interval. The user's job execution start time using the simple SNAP broker is related to the broker repeatedly contacting the information provider to find out when the resources are released, which can be seen by looking at the performance trend for both Figures 4.10 and 4.11. The performance trend of this experiment, regarding the time taken until the user's job begins execution (Figure 4.10), is qualitatively different to the

results obtained on the Grid test-bed (Figure 4.3), does not show any similarities. The reason why the results are not monotonic for the simple SNAP broker on the WRG is interconnected to several factors which are as follows:

- The third party job occupying the resources may vary slightly in their completion time as it is an attribute that cannot be controlled due to the various sites' independent infrastructure.

- The information provider associated to each resource across the three sites responds individually and on average within 27 seconds. A miss in picking up the change in resources status would mean for it to be acknowledged during the next call.

- Finally, in order for the simple SNAP broker to proceed in its submission of the user's job it must receive a response from all information providers that the resource they contacted is unoccupied.

Combining the above three factors with the information provider being a dominant issue. Further, the duration of the resources being occupied by third party jobs and the experiment runs incrementing by 10 seconds for each interval, influences the start time of the user's job. For example Figure 4.10 shows almost the same job execution start time of 200 seconds (with a discrepancy of 1 second) for three different consecutive times when the resources are freed (140, 150 and 160 respectively). This consecutiveness is repeated numerously throughout Figure 4.10. Thus two third party jobs with a 10 seconds duration gap between their completions can occur within the information provider response time.

However at times there can be a miss in picking up the completion of all third party jobs which would mean it is acknowledged the next time the information provider is invoked by the broker. This is why there is the same job execution start time for three different consecutive times when the resources are freed.

The effect of the above three points combined with the incremental duration time of third party jobs occupying the resources can also be supported by further runs of this experiment, which have been carried out to further investigate the serration in Figure 4.10. The additional runs have the third party jobs incrementally completing

every 20 and 30 seconds during each interval as shown in Figure 4.12 and 4.13 respectively. With the 20 seconds duration only one third party job can complete within the information provider response time. However at times there can be a miss such as that shown in Figure 4.12, with the user's job execution start time being 146 seconds for two different consecutive times when the resources are freed after 80 and 100 seconds. Further in Figure 4.13, where the third party jobs complete incrementally every 30 seconds for each time interval. The majority of time there is no miss by the information provider in picking up that the resources are free, apart from the leap between 160 and 190 seconds when the resources were freed. Hence the performance trend in Figure 4.13 is monotonic than Figure 4.10 and 4.12. Note that the final value on the x axis is 340 seconds in Figure 4.13 as the next value would be 370 seconds which exceeds the brokers 360 second acceptance time before timing out.

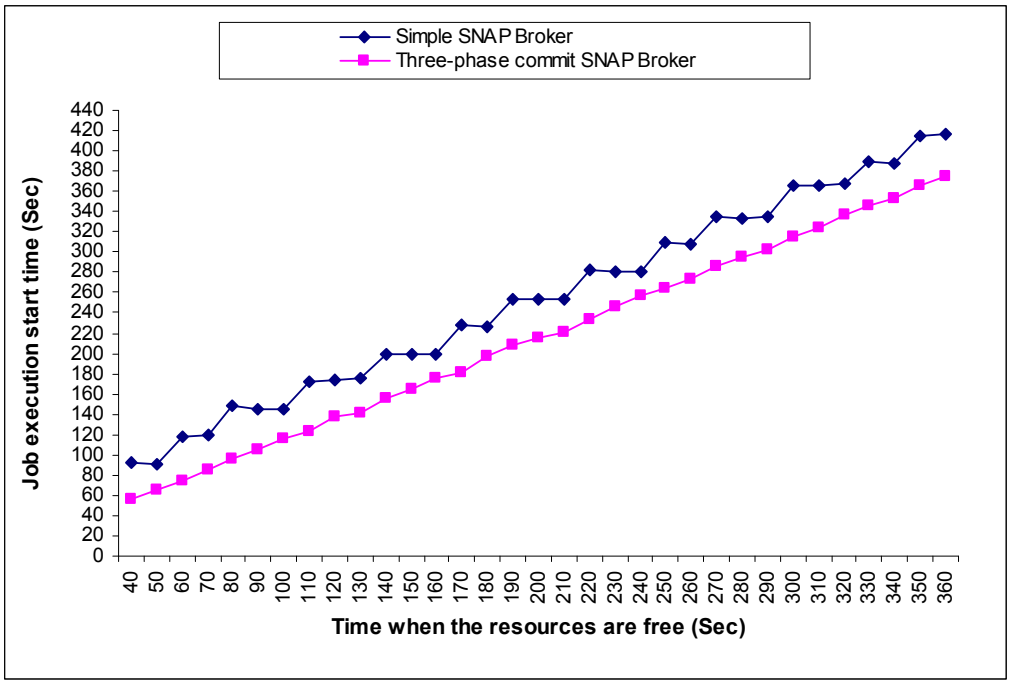

Figure 4.10: The time taken for the user's job to begin execution as a function of the time for which resources are unavailable, with an incremental duration of 10 seconds for each interval.

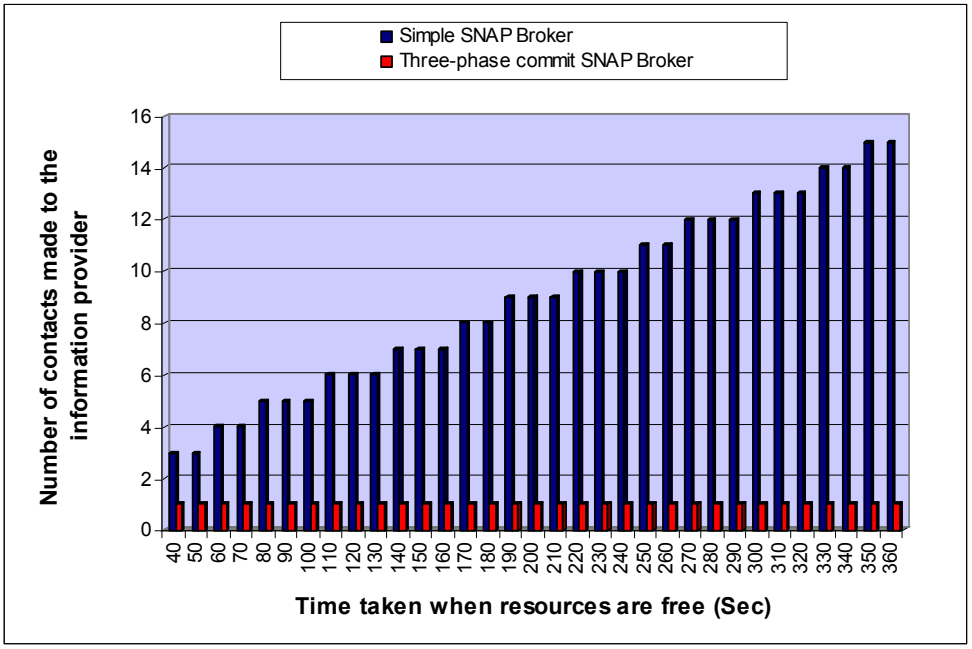

Figure 4.11: Number of contacts made by the broker to the information provider as a function of the time for which resources are unavailable, with an incremental duration of 10 seconds for each interval.

Figure 4.11 shows the number of information provider contacts made by each broker. The three-phase commit SNAP broker makes one contact only and then subsequently the probes are used to provide an update of the resources' status. On the other hand, the simple SNAP broker needs to repeatedly contact the information provider as it has no vision of the resources' status without having to repeat this process until it reaches the stage when it becomes aware the resources are available.

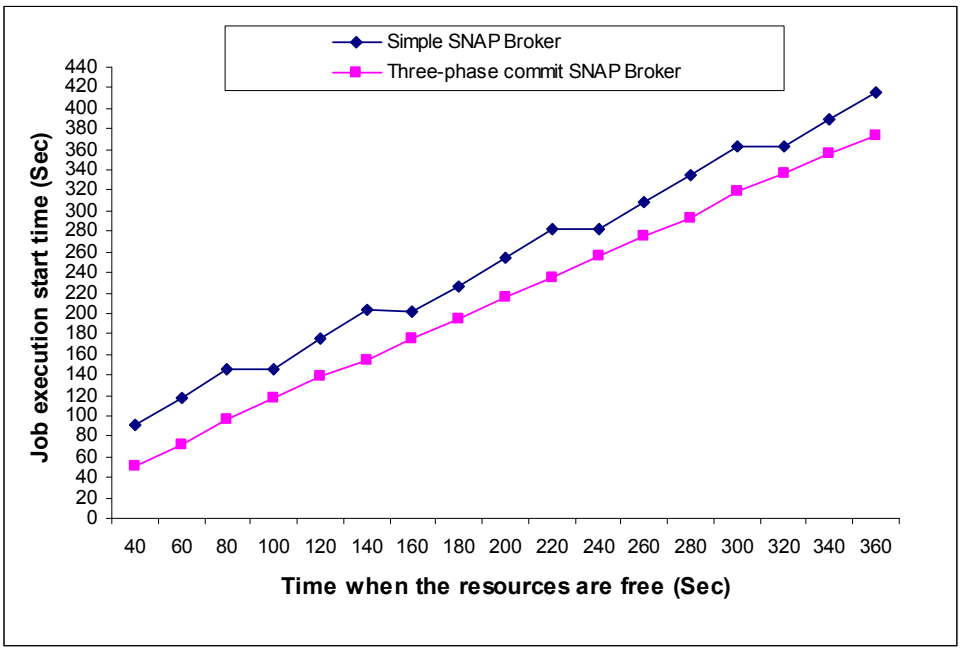

Figure 4.12: The time taken for the user's job to begin execution as a function of the time for which resources are unavailable, with an incremental duration of 20 seconds for each interval.

4.3.2.2 White Rose Grid Experimental Results and Discussion for Scenario 2

Figures 4.14 and 4.15 show the results for the experiment carried out on the WRG combined with the Grid test-bed resources, in relation to scenario 2. Figure 4.14 shows that the three-phase commit SNAP broker takes just over 50 seconds to submit and begin execution of the user's job, irrespective of the number of other third party jobs submitted. Since the probes ensure rapid updates of the status of the resources, the three-phase commit SNAP broker is aware that a resource has been taken soon after it occurs (due to the use of the probes) and consequently chooses an alternative resource. It successfully submits the user's job all to the WRG resources as it is able to adjust swiftly to changes in the status of the resources before any other resources are taken. Further, the three-phase commit SNAP broker compared to the simple broker provides a performance of 21% when 1 resource is taken and 91% when 7 resources are taken (both percentages are calculated using equation 1). Clearly the simple SNAP broker takes longer when more resources are taken, since it is unable to identify changes in resource status fast enough to successfully submit

the user's job before more resources are taken. Further, unlike the three-phase commit SNAP broker the simple SNAP broker is unable to submit the user's job all to the WRG resources as the number of resources taken increases. Hence when 0 or 1 resource are taken, the simple SNAP broker is able to submit the user's job all to the WRG resources, but when 2 and 3 resources are taken it combines both WRG and Grid test-bed resources. Finally when 4 or more resources are taken it is only able to submit to the Grid test-bed. This is a consequence of not having the probes to provide rapid updates of resource status changes.

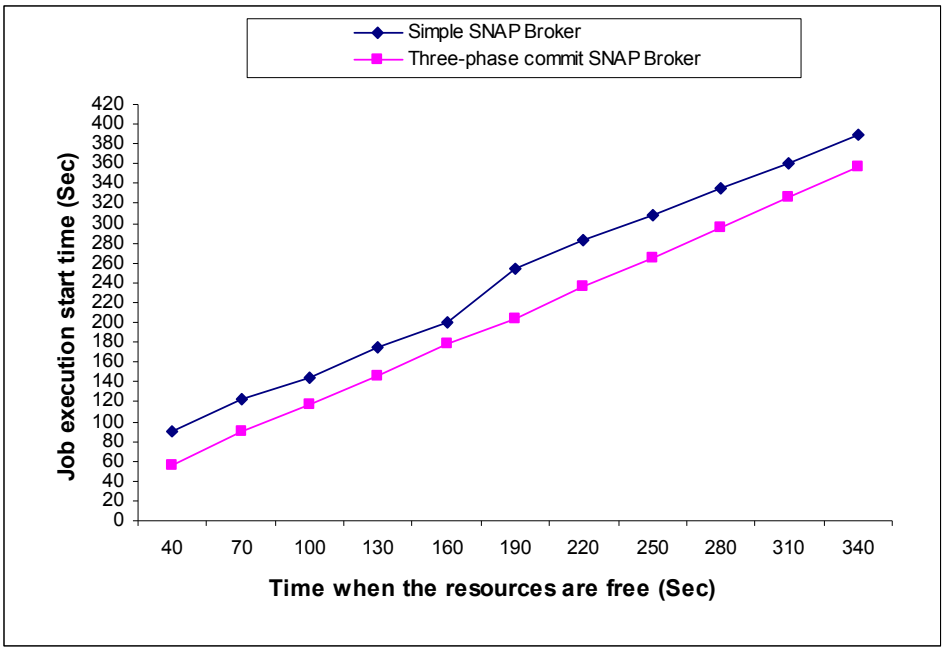

Figure 4.13: The time taken for the user's job to begin execution as a function of the time for which resources are unavailable, with an incremental duration of 30 seconds for each interval.

Figure 4.15 shows the number of information provider contacts each broker makes before the user's job is submitted. The three-phase commit SNAP broker only makes the initial contact, however for the simple SNAP broker the number of information provider contacts increases as the number of resources are taken. This is due to the fact that when it attempts to reserve the chosen resources before submission it is unsuccessful as some are taken. Thus it needs to gather new information to find out which resources are still available as it is unable to know the

status of the other resources without having to re-contact their information provider. This consequently impacts on the start time of the user's jobs as seen in Figure 4.14.

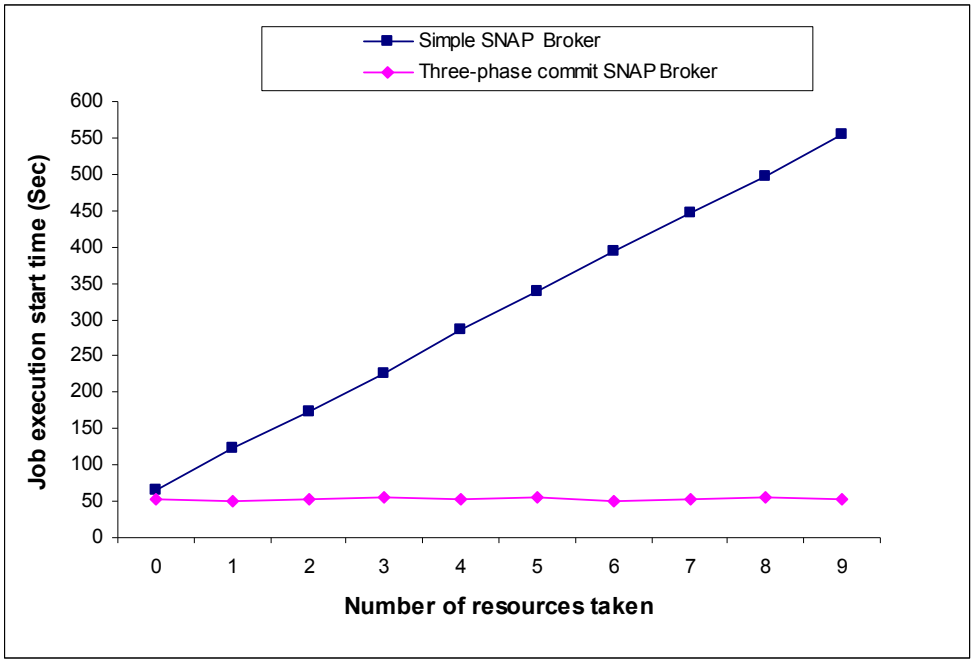

Figure 4.14: Time taken to begin job execution as a function of number of additional jobs submitted.

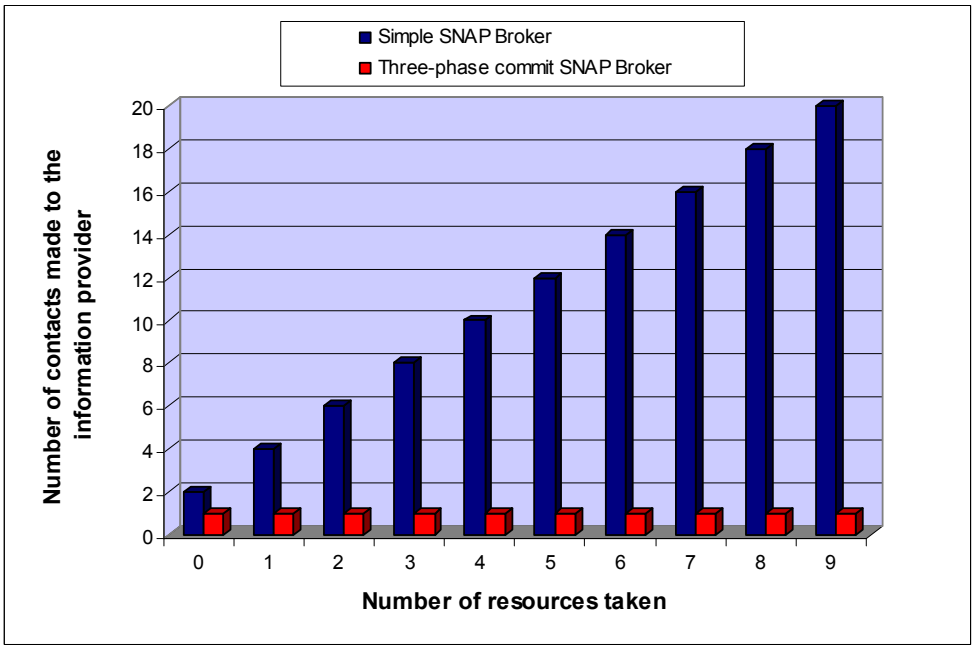

Figure 4.15: Number of contacts made by the broker to the information provider as a function of additional jobs submitted.

4.4 Overall Evaluation of the Experiments

The experiments discussed in Sections 4.2.2 and 4.2.3 for the Grid test-bed and Sections 4.3.3 and 4.3.4 for the WRG have demonstrated that the three-phase commit SNAP broker has access to fast updates on the status of resources, enabling a performance enhancement in a number of specific scenarios. The experiments illustrate the value of using the three-phase commit protocol. This has been achieved by considering specific scenarios, where the vision of resources provided by the use of probes enables faster submission of a user's job than would otherwise be possible.

In both the Grid test-bed and the WRG the three-phase commit SNAP broker out-performs the simple SNAP broker. However on the WRG there is an increase in performance compared to the Grid test-bed. The performance increase is related to the information provider response time on the WRG being over three times longer in comparison to that on the Grid test-bed. Further, the three-phase commit SNAP broker gains its performance from the point when it receives the user's job requirements until just before it submits the user's job to the resources after reserving them. Both the simple and the three-phase commit SNAP broker incur the same cost when submitting the user's job to the resources after nominating and securing the resources.

The number of contacts made to the information provider by the simple SNAP broker is related to the information provider response time and the duration of the third party jobs, initially occupying the resources. For example long third party job duration in relation to short information provider response time means a high number of contacts to the information provider by the simple SNAP broker. Conversely short third party job duration with long information provider response time means fewer contacts to the information provider by the simple SNAP broker. This can be seen in both experiments in scenario 1 and on both environments (WRG and the Grid test-bed). A good example is shown in Figure 4.5, where the information provider response time remains constant throughout the experiment, however as the duration of third-party jobs occupying the resources during each interval increases so does the number of contacts made to the information provider. The reason behind this is that the simple SNAP broker needs to continuously contact the information provider until the resources are unoccupied, as it does not have the advantage of the probes used by the three-phase commit SNAP broker that provide a

constant vision of the resources' status. Further, the number of contacts made to the information provider is also influenced by not being able to swiftly adjust to the resources' status. Further, the number of contacts made to the information provider is also influenced by not being able to swiftly adjust to the resources' status. For example resources that were initially available when the simple SNAP broker contacted their information provider were then subsequently taken by third party jobs, while the broker is in the process of co-allocating the user's job. The effect of this has been demonstrated through the experiment in scenario 2 on both the Grid test-bed and the WRG as shown in Figures 4.7 and 4.15 respectively. However the three-phase commit SNAP broker was not affected by the resources changing status during co-allocation as it uses the probes to provide rapid updates of the resource's status changes and consequently is able to swiftly adjust and nominate resources still available. This subsequently also affected the user's job start time as shown in Figures 4.6 and 4.14 on the Grid test-bed and the WRG respectively.

Studies have also been carried out to investigate when the three-phase commit SNAP broker will exhibit lower performance than the simple SNAP broker. In the worst-case scenario (unfavourable to the three-phase commit protocol) where resources required by the broker are idle and there is no competition for their use the three-phase commit SNAP broker will always perform just as well as the simple SNAP broker. The simple SNAP broker will not outperform the three-phase commit SNAP broker as the protocol used by the latter broker follows the same procedure as the simple SNAP broker, explained in Section 3.4.2, with the first phase strengthened. Further there is no overhead cost associated with setting up the probes since this occurs concurrently when initially contacting the Monitoring and Discovery System (MDS) [33, 34, 50].

The three-phase commit SNAP broker is most effective in providing a significant performance enhancement over the simple SNAP broker under normal average traffic conditions, where there is a flow of requests for resources by other Grid users on the system. This is related to the probes providing the broker with rapid updates of the resource status as they occur and not depending on the traditional approach used in the simple SNAP broker. Further, as shown in Figure 4.1, 4.2 (both are testgrid results) and Figure 4.9 (WRG results) the three-phase commit SNAP broker still provides a significant performance enhancement over the simple SNAP broker when there is a delay in the information provider response time. In fact the simple

SNAP broker suffers considerably when the information provider response time increases: again this is related to not having access to the probes. However both the simple and the three-phase commit SNAP broker find it difficult to secure resources in circumstances where there is an extremely high volume of users competing for the same resource. Nevertheless the three-phase commit SNAP broker still outperforms the simple broker as it will be updated immediately when resources are free, allowing the broker adjust swiftly and to make a decision on the resources to reserve. Hence the probes have a vital role in providing the three-phase SNAP broker the competitive advantage in knowing the status of resources.

It is important to note that although there is an associated overhead with non-cached information obtained from the information provider, the three-phase commit protocol needs to query the latter only once. Unlike the simple SNAP broker, that does not have the advantage of the probes and consequently needs to repeatedly contact the information providers. This has not shown an impact on either of the environments infrastructure. However if it was for a prolonged sustained period of time it would have a strain on the load of the information provider. This is avoided by the three-phase commit SNAP broker as it uses the probes to gain its updates of the resources status.

4.5 Summary

The chapter begins by giving an overview of the experiments that are carried out on both environments, the Grid test-bed and on the White Rose Grid (WRG). The experiments on both environments had the same objective, which is to investigate the behaviour of the three-phase commit SNAP broker when scenarios occur in which a performance enhancement over the simple SNAP broker is expected. This was then followed by a description of the Grid test-bed, its experimental design and a discussion of its results. The results show that the three-phase commit SNAP broker outperformed the simple version. This was achieved by the aid of the probes used in the three-phase commit protocol that provided a rapid update of the state of the resources enabling the broker to adjust swiftly to the changes.

A description of the WRG was then given followed by the challenges encountered when deploying the simple and three-phase commit SNAP broker. The deployment was beneficial as many collaboration techniques were gained, such as the

significance of ensuring a prototype version for any new protocol, that pushes the boundaries of existing technologies and requires their modification, should be demonstrated at least on a test-bed environment before its deployment on a production Grid. This is to show hesitant administrators that maintain production resources with several projects depending on the infrastructure, the credibility of the protocol's performability and reliability. A description of the WRG experimental design with a discussion of its results was then presented. The results again showed the three-phase commit SNAP broker outperformed the simpler version. The chapter ended with an overall evaluation of the results on both environments.

The next stage in the research is to examine the performance of both the simple and the three-phase commit SNAP broker through mathematical modelling and simulation, which is shown in chapter 5. This would allow for a wider number of resources and varying traffic conditions to be considered than that evaluated in this chapter due to the limitation of physical resources available for the experiments

Chapter 5

Performance Evaluation using Mathematical Modelling and Simulation of the Three-phase Commit Protocol

Chapter 3 discussed a simple SNAP-based resource broker and a more sophisticated SNAP-based broker, following a three-phase commit protocol. Chapter 4 presented experimental results, taken from the Grid test-bed and the White Rose Grid (WRG) [11], showing the three-phase commit SNAP broker providing a performance enhancement over the simple SNAP broker, in terms of the time interval between submission (to the broker) of the user's job requirements and the job beginning execution. However the experiments were constrained by the number of physical resources available for the experiments, which also limited the type of traffic conditions used. Thus the purpose of this chapter is to use simulation validated by mathematical modelling, to evaluate the performance of the SNAP resource brokers. This approach allows a wide range of possible traffic conditions to be considered. The traffic model on which the analysis is based is expressed using queueing theory [122].

Simulation is defined as "the imitation of the operation of a real-world process or system over time" [123]. Mathematical modelling is defined as "an abstract model that uses mathematical language to describe the behaviour of a system" [124]. Simulation is beneficial in many ways such as:

- A real Grid infrastructure does not provide a repeatable and controllable environment for experimentation and evaluation of scheduling strategies, which simulation accommodates.

- Simulation is effective in working with very large hypothetical problems that would otherwise require involvement of a large number of active users, which is difficult to co-ordinate and construct on a real Grid environment for investigation purposes [125].

This chapter is organised as follows. Section 5.1 describes both the simple SNAP broker and the three-phase commit SNAP broker protocol as a list of sequences to aid the mathematical modelling expression and simulation development. Section 5.2 describes the traffic model in which different traffic conditions are used to evaluate the SNAP brokers. The mathematical modelling is presented in Section 5.3 and is followed by the Simulation description in Section 5.4. Section 5.5 presents the experimental results and discussion of those carried out with parameter values obtained from the Grid test-bed, followed by the same experiments with parameter values from the White Rose Grid. This is to validate the overall performance of both SNAP brokers and verify the three-phase commit SNAP broker still maintains it enhancement over the simple SNAP broker with parameters from a large Grid environment. Section 5.6 provides an overall evaluation of the experiments carried out in this chapter. Section 5.7 ends the chapter with a summary.

5.1 SNAP Brokers Protocol

In Section 3.3 the SNAP-based Grid resources broker architecture is described and depicted in Figure 3.2. The architecture forms the basis for both the simple and the three-phase commit SNAP broker protocols. However to aid the mathematical modelling and simulation Figure 5.1 shows the SNAP-based Grid resource broker architecture components with numbers labelling the steps in the simple SNAP broker protocol which correspond to its protocol description listed below:

1) Having received the user's job requirements, the Matchmaker contacts the Knowledge Bank (KB), which returns the attributes for the resources the user has access to and that are capable of supporting the user's job.

2) The Matchmaker forwards the information to the decision maker, which prioritises resources, tagging them blue and white, corresponding to "high priority" and "adequate" respectively.

3) The Decision Maker passes this information onto the Resource Gatherer.

4) The Resource Gatherer contacts the Monitoring and Discovery System (MDS) [33, 34, 50] (the GRIS (Grid Resource Information Service) on each resource) to obtain up-to-date dynamic information about 'candidate' resources. In this step, probes are set up. This only occurs the first time step 4 is carried out. Probes do not need to be set up subsequently. These are only used in the simple SNAP broker, to support fast reservation of resources.

5) The dynamic information about the resources is passed to the Co-allocator, which makes a decision as to where the job should run. If insufficient resources are available, the Co-allocator informs the Resource Gatherer and step 4 is repeated. Otherwise the Co-allocator reserves the chosen resources. If this is unsuccessful (e.g. because other third party users have taken one or more of the chosen resources), return to step 4.

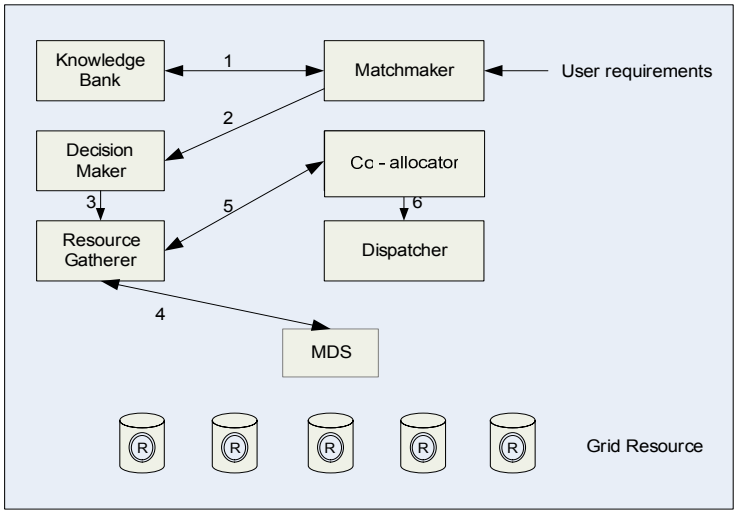

Figure 5.1: The SNAP-base Grid broker components and their interactions.

Once the resources have been reserved, the Dispatcher transfers any necessary data files that the user's job relies on and submits the user's job for execution, which is shown in step 6 in Figure 5.1.

The three-phase commit SNAP broker works according to the following protocol, (first three steps are as above for the simple SNAP broker).

4) The Resource Gatherer contacts the MDS (the GRIS on each resource) to obtain up-to-date dynamic information about 'candidate' resources. In this step, probes are set up. This only occurs the first time step 4 is carried out. Probes do not need to be set up subsequently. These probes listen for any changes in the status of resources. In addition they are used to support reservation.

5) The dynamic information about the resources is passed to the Co-allocator.

6) The Co-allocator makes a decision as to where the user's job should run. If insufficient resources are available, the Co-allocator waits until the probes inform it that sufficient resources are available to support the user's job.

7) If any resources are taken by other third party users, the broker is made aware of this by the probes. If this occurs, step 6 is repeated. Otherwise the Co-allocator reserves the chosen resources. If this is unsuccessful, return to step 6.

Once the resources are reserved, the Dispatcher (as with the simple SNAP broker) transfers any necessary files the user's job depends on and submits the user's job for execution.

5.2 Traffic Model

In order to model the behaviour of the SNAP-based resource brokers, a traffic model, which enables different realistic traffic conditions to be considered, is required. The approach taken here is based on queueing theory [122], there are other approaches such as Petri Nets [126]. However the queueing theory is the most appropriate choice for the traffic model as it is based on queues which is what is required for this work. Initially a simple model was chosen, in order to enable a simple comparison between simulation and analytical results. This model is presented in Section 5.2.1. The model was then extended to enable evaluation of the

SNAP-base resource brokers under more realistic traffic conditions than was possible with the simple model. The extended model is presented in Section 5.2.2.

5.2.1 Simple Traffic Model

It is assumed that each (single CPU) resource is a server and has a corresponding single-server queue, with no restrictions on its queue capacity, which is the case with local schedulers such as Sun Grid Engine (SGE) [19]. Jobs are independently submitted to each queue, with random inter-arrival and service times (i.e. each queue is an M/M/1 queue). The broker needs to reserve resources and submit jobs within this environment. The model is depicted in Figure 5.2. The following parameters are used within this model:

Number of Processors (Servers)	P
Mean Service Time	T_s
Mean Arrival Rate	λ

Note that, with this model, the traffic on different resources does not display any correlation. Hence many realistic scenarios are not encapsulated by this model (e.g. third party users submitting jobs that require more than 1 resource). This is addressed in the extended model, discussed in the following section.

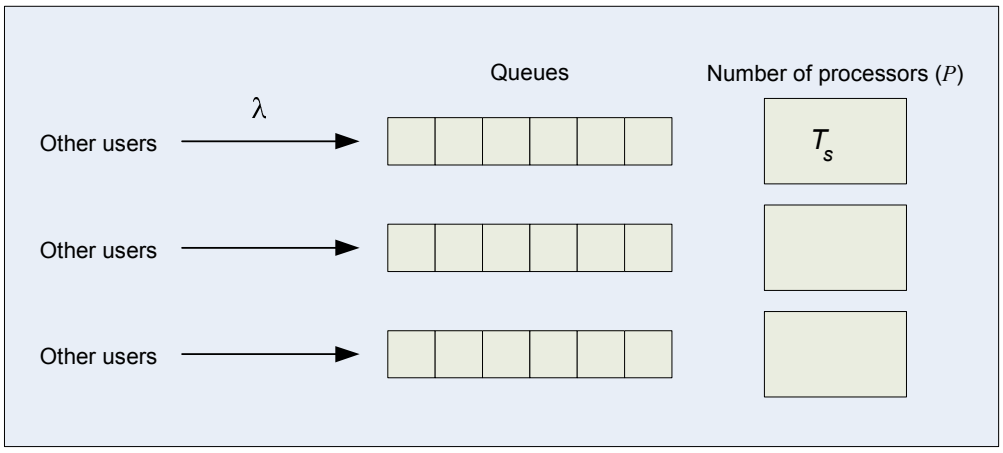

Figure 5.2: Queueing system used in simple traffic model.

5.2.2 Extended Traffic Model

In order to account for correlations between traffic on different resources, a multi-server queue is introduced. As before (in the simple traffic model), each server has an associated queue. However, incoming third party jobs to the system enter a multi-server queue. Each job has a service time (T_s) and number of resources (R) required associated with it. There are a number of possibilities regarding how jobs at the front of the multi-server queue are dealt with. For example:

1) When a job is at the front of the multi-server queue, it is sent to local queues in a round-robin fashion, i.e. (for P resources) queue 1, queue 2, … queue P, queue 1…

2) When a job is at the front of the multi-server queue, it is sent to local queues corresponding to randomly chosen resources.

3) When a job is at the front of the multi-server queue, it is sent to local queues with the least number of jobs.

4) When a job is at the front of the multi-server queue, it is sent to local queues with the least waiting time.

The jobs arriving in the multi-server queue have random inter-arrival and service times, while the number of resources required for a particular job is chosen at random from the set (1, 2, 4, 8, 16 and 32). This is illustrated in Figure 5.3. Only up to 32 resources has been allowed to be requested by any individual third party job, hence 25% of the systems resources, which is a high value for a single job on a multi-user system. However these jobs are generated to create traffic flow into the system for the brokers to try to reserve resources under competing conditions by third-party users.

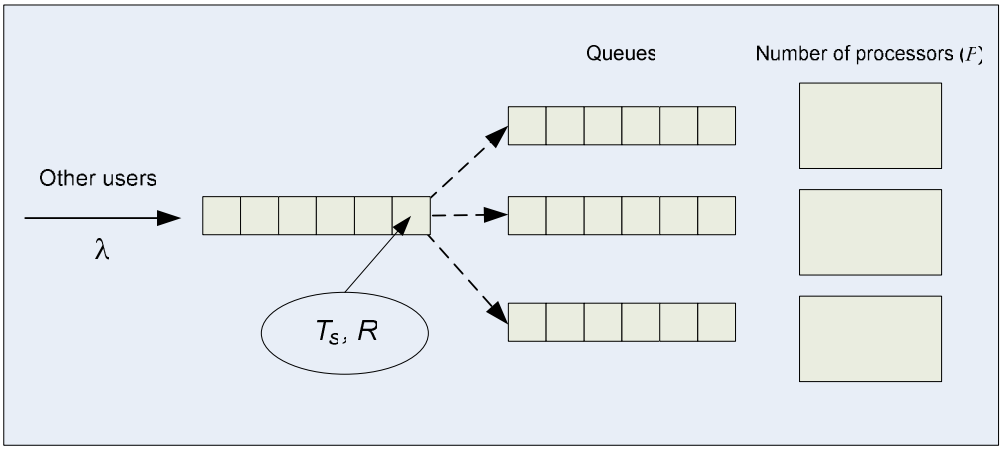

Figure 5.3: Queueing system used in extended traffic model.

5.3 Mathematical Modelling

This section presents a mathematical analysis, carried out in order to provide expressions that enable the performance of the SNAP based brokers to be evaluated. This mathematical modelling was undertaken collaboratively with other members of the Grid group at the University of Leeds, namely Dr Iain Gourlay and Dr Karim Djemame. Specifically, expressions are obtained, enabling the average time taken (for each broker) between receiving a user's job requirements and resources being reserved for the user's job. Note the time taken to submit the job once resources have been reserved is not included in the analysis, since it is not dependent on which of the protocols (discussed in Section 5.1) is used. The performance evaluation is carried out using the simple traffic model, presented in Section 5.2.1. The results produced will enable the simulation and analytical results to be compared. If the results from these two approaches agree, this provides evidence to support the validity of further simulation results based on the extended traffic model, where the analytic approach would be impractical.

The following parameters are used in the analysis of the SNAP brokers. The parameters allow the SNAP brokers to be evaluated in various Grid environments such as the Grid test-bed and WRG as the steps referred to are those given in Section 5.1, where the SNAP broker protocols are presented.

J: Number of resources required by the user's job submitted.

t_{KB}: The time taken for steps 1-3 of the broker protocols.

t_{mds}: The time taken for step 4 of the broker protocols.

t_{dec}: The time taken by the Co-allocator to decide where the job should run.

t_{res}: The time it takes to reserve resources.

Section 5.3.1 presents the analysis for the simple SNAP broker and the corresponding analysis for the three-phase commit SNAP broker is given in Section 5.3.2.

5.3.1 Simple SNAP Broker

Step 5 (described in Section 5.1) takes $t_{dec} + t_{res}$. It is assumed that

$$t_{mds} = t_{mds(1)} + t_{mds(2)} \tag{3}$$

Here $t_{mds(1)}$ is the time, for gathering resource information, during which any change to resource status (on any resource being contacted) will be picked up. While $t_{mds(2)}$ is the time during which changes to status will not be identified as it is the time in which the status of the resource is being generated, prepared to be sent to the broker and received by the broker.

The total time the broker takes, from receiving user requirements until resources are reserved, in the absence of other traffic is,

$$E\left(t_{simple}\left(no\ traffic\right)\right) = t_{KB} + t_{mds} + t_{dec} + t_{res}\,. \tag{4}$$

Note that the above time does not account for the possibility that certain steps may need to be repeated. Referring to the broker protocol, for step 5, there is an overhead associated with the possibility that step 4 needs to be repeated due to insufficient resources being available the first time the MDS is contacted. Each time this is carried out, the time taken is $\left(t_{mds} + t_{dec}\right)$. Let the average time taken to complete this operation be $E\left(t_{mds/dec}\right)$. Note that

$$E\left(t_{mds/dec}\right)=\overline{n}\left(t_{mds}+t_{dec}\right). \tag{5}$$

Here, $\overline{n}$ is the mean number of times that there are insufficient resources available when the MDS is contacted. This means that $\left(t_{mds}+t_{dec}\right)$ must be replaced by $E\left(t_{mds/dec}\right)$ in equation (4).

There is an additional overhead associated with the possibility that a chosen resource is taken prior to successful reservation. In order to explain the effect of this overhead on equation (4), consider the following scenario. Suppose an event, *A* occurs, taking time T_A. If the event fails, then it must be repeated until it succeeds. Let the probability that the event fails on a given trial be P_f. In that case, the entire process takes an average time given by,

$$E\left(t_{total}\right)=\left(1-P_f\right)T_A\sum_{n=1}^{\infty}P_f^{n-1}n \tag{6}$$

The summation in equation (6) is equal to $1/\left(1-P_f\right)^2$. Hence,

$$E\left(t_{total}\right)=\frac{T_A}{\left(1-P_f\right)} \tag{7}$$

In the case under consideration, event *A* corresponds to contacting the MDS, making a decision as to where the job should run and reserving resources. The time taken (T_A) is therefore $\left(E\left(t_{mds/dec}\right)+t_{res}\right)$. Failure is caused by one or more chosen resources being taken by third party users, prior to the completion of reservation. For the simple traffic model, the probability that a resource that is free at time t_0 is still free at time t_1 is $e^{-\lambda\left(t_1-t_0\right)}$ where λ is the mean arrival rate. Here the time interval Δt involved ranges from after the $t_{mds(1)}$, when the MDS is contacted, until reservation is successfully completed. Hence,

$$\Delta t=t_{mds(2)}+t_{dec}+t_{res} \tag{8}$$

Hence, the probability that one of the J resources chosen to run the job is taken prior to successful reservation is given by,

$$P_{fail} = 1 - e^{-\lambda J \Delta t} \tag{9}$$

This leads to the following expression for the average time the simple SNAP broker takes from receiving the users' requirements until resources are successfully reserved.

$$E\left(t_{simple}\right) = t_{KB} + e^{\lambda J \Delta t}\left(E\left(t_{mds/dec}\right) + t_{res}\right) \tag{10}$$

5.3.2 Three-phase Commit Broker

In the case of the three-phase commit SNAP broker, the MDS is contacted only once, since the probes are used to provide the broker with a vision of the resources. However, there is still an overhead associated with the possibility that other users could take resources prior to the successful completion of the decision as to where the job should run. Hence t_{dec} is replaced with $E\left(t_{dec/three-phase}\right)$. In addition, once the broker makes a decision, it may fail to successfully reserve resources, if another user takes one or more of the chosen resources, during the time interval t_{res}. Using the same reasoning as given above for the simple broker to calculate the effect of this overhead, leads to the following expression for the average time it takes the three-phase commit broker to successfully reserve resources.

$$E\left(t_{three-phase}\right) = t_{KB} + t_{mds} + e^{\lambda J t_{res}}\left(E\left(t_{dec/three-phase}\right) + t_{res}\right) \tag{11}$$

5.4 Simulation

While there exists a large number of simulation languages (such as Parsec [127] and JiST [128]), simulation libraries (such as SimJava [129] and DSOL [130]) and application specific simulators (such as OMNeT++ [131]), there exists very few tools for simulating Grid computing environment. The most notable ones are MicroGrid [132], Simgrid [133] and Gridsim [125], however these tools do not support scheduling performance evaluation nor the functionalities of the SNAP

resource brokers (in specific the three-phase commit protocol). Therefore the discrete event simulation tool used in obtaining the results (shown in Section 5.5) has been developed from scratch. This tool, written in Java, adopts the process interaction approach to discrete-event simulation. The SNAP resource brokers' objects (in the physical system) are represented by logical processes. Interactions among physical processes (events) are modelled by timestamped messages exchanged among the corresponding logical processes. The programs developed for the simulation are executed using the traditional sequential simulation protocol (Global Event List) [134].

The objectives of the simulation experiments are twofold: 1) an observation of the behaviour of the broker in terms of time to reserve resources, and 2) a comparison between the analytical and simulation results. Specifically, the simulator can be used to consider a wide range of traffic conditions, using both the simple and extended traffic models presented in Section 5.2. Experiments using the simple model can be used to assess the validity of the results through comparison with the analytical results. Experiments can then be carried out, using the extended traffic model, enabling the effect of correlation of traffic across resources to be evaluated. In each case, the performance of the three-phase commit SNAP broker relative to the simple SNAP broker is of significant interest. The simulation results are obtained over n runs (n being large[*]) i.e. the mean value of n runs by each SNAP broker for each point in the plots presented in Section 5.5. This is to calculate the standard deviation as well as gain a 95% significant interval.

5.5 Experiments and Results

This section presents the experiments designed to study the behaviour of the SNAP brokers under a range of traffic conditions. The approach taken is to compare analytic and simulation results, obtained using the simple traffic model. This is used to validate the results obtained using the simulator. Further results will then be obtained through simulation using the extended traffic model. Hence all performance evaluation comparison of the SNAP brokers will be based on the simulation results.

[*]Up to 1000 runs

All the experiments will firstly be carried out with parameters obtained from the Grid test-bed (its specification description is given in Section 4.2), then the same experiments will be carried out with values from the White Rose Grid (its specification description is given in Section 4.3). This is to follow the same approach carried out in chapter 4, ensuring the three-phase commit SNAP broker still provides a performance enhancement over the simple SNAP broker in a large Grid environment, however with this chapter it investigates the broker using various traffic conditions.

In addition to the experiments discussed in this section, some preliminary tests were carried out to check the components of the simulator. A description of these and the results obtained can be found in Appendix C.

5.5.1 Design of Experiment 1

This experiment involved the SNAP brokers submitting a user's job into the system using the simple traffic model, each broker is considered separately. The mean service time and mean inter-arrival time are 300 Sec and 350 Sec respectively. These values are acceptable as the purpose is to ensure traffic is generated on the system and also allow the opportunity for the SNAP brokers to attempt reserving resources i.e. the system is not over flooded with jobs, which is not the norm. The total number of resources on the system is 128. It is assumed that all 128 resources are appropriate for the user's job to be submitted by the SNAP brokers. Firstly, simulation is used to obtain values for $E(t_{mds/dec})$ and $E(t_{dec/three-phase})$ in the simple and three-phase commit SNAP brokers respectively for both the Grid test-bed and the WRG. This avoids the need to derive the values through long mathematical expressions as they can be easily obtained through simulation. The values are shown in Appendix D.

The average time each broker takes to submit the user's job is measured as a function of number of resources (1, 2, 4, 8, 12 and 16) required by the user's job. There are two aspects related to these values, the first is that the resources request do not exceed 16, the justification is that the next incremental value would be 32 (25% of the systems resources). However even though this value (32 resources) is allowed to be requested by third party jobs (see Section 5.2.2), it is not the norm for a single job to request such amounts on a multi-user system. Further the SNAP brokers need

to reserve the resources simultaneously, unlike the third party jobs which do not and are generated to create traffic flow. Hence such a request would incur a high waiting time as 25% of the systems resources are required under competing conditions. The second aspect is the value 12 in the sequence of resources requested by the SNAP brokers, the reason for this is that there is a large gap between 8 and 16 resources request, which is why the value is included.

The simulation and modelling results, for the simple and the three-phase commit SNAP brokers are then compared. If good agreement is obtained between simulation and modelling then this will support the validity of further results, obtained through simulation using the extended traffic model, for experiment 2 discussed below in Section 5.5.2.

5.5.2 Design of Experiment 2

This experiment is closely related to experiment 1, except that the extended traffic model is used in place of the simple traffic model. The mean inter-arrival time to the multi-server queue is:

$$\frac{1}{\lambda} = \frac{350\overline{R}}{128} \tag{12}$$

Here $\overline{R}$ is the average number of resources required by third party jobs being submitted to the multi-server queue. In this case, since R is randomly chosen from the set (1, 2, 4, 8, 16 and 32), $\overline{R}$ is 10.5. The mean service time associated with these jobs is 300 seconds.

There are several options that can be used to allocate third party jobs at the front of the multi-server queue to the local queues as discussed in Section 5.2.2. However not every option can be chosen for the evaluation of the simple SNAP broker compared to the three-phase commit SNAP broker, as it would be time consuming and the purpose of the experiments is to investigate if the three-phase commit SNAP broker still outperforms the simple SNAP broker under realistic traffic conditions. Thus two approaches have been chosen namely round-robin and random. The reason for choosing these two is that round-robin follows an orderly sequential approach in allocating resource for jobs. Conversely the random approach does not follow an orderly sequence and is unpredictable in its resource allocation. Thus this would

provide a good insight into how the SNAP brokers would cope in two different extreme traffic conditions.

5.5.3 Experimental Results and Discussion based on the Grid test-bed Parameters

The results and discussion for experiments 1 and 2 are presented in this section based on the Grid test-bed parameters which are those presented in [135], the parameters used are listed below:

- Time taken to connect to Knowledge Bank (KB): 0.268 Sec.
- Time taken to filter out resources appropriate for the user's job and return the results to the broker: 0.017 Sec.
- Time taken for the broker to prioritise resources and tag them blue and white, corresponding to "high priority" and "adequate" respectively: 0.003 Sec.
- Hence t_{KB}= 0.288 Sec.
- Contacting the MDS takes 8 sec on average. Changes to resource status are picked up in the first 4 Sec. Hence, $t_{mds(1)} = t_{mds(2)} = 4$ Sec.
- The time taken to decide on where to submit the user's job is 0.007 Sec. Hence, t_{dec} =0.007 Sec.
- Reservation takes 2.881 sec. Hence, t_{res} = 2.881 Sec.

Figure 5.4 and Figures 5.5 show a comparison, for experiment 1, of the simulation results with the analytic results for both the simple SNAP broker and the three-phase commit SNAP broker respectively with the Grid test-bed parameters. The average time taken from receiving user requirements until resources are successfully reserved are plotted as a function of the number of resources required by the user's job. The average inter-arrival time is 350 seconds and the average service time is

300 seconds. As can be seen from the Figures (Figures 5.4 and 5.5), the simulation and analytical results show good agreement with slight discrepancies. This can be expected when comparing simulation to analysisl [123, 134], however most importantly the performance trend is the same.

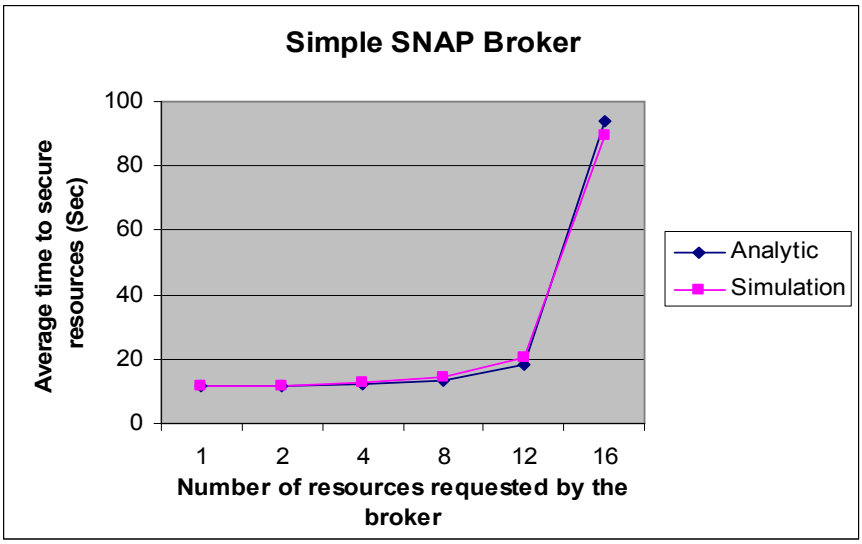

Figure 5.4: Simulation compared to analytical results, showing the average time the simple SNAP broker takes to reserve resources, with the Grid test-bed parameters. The mean inter-arrival time and mean service time are 350 Sec and 300 Sec respectively.

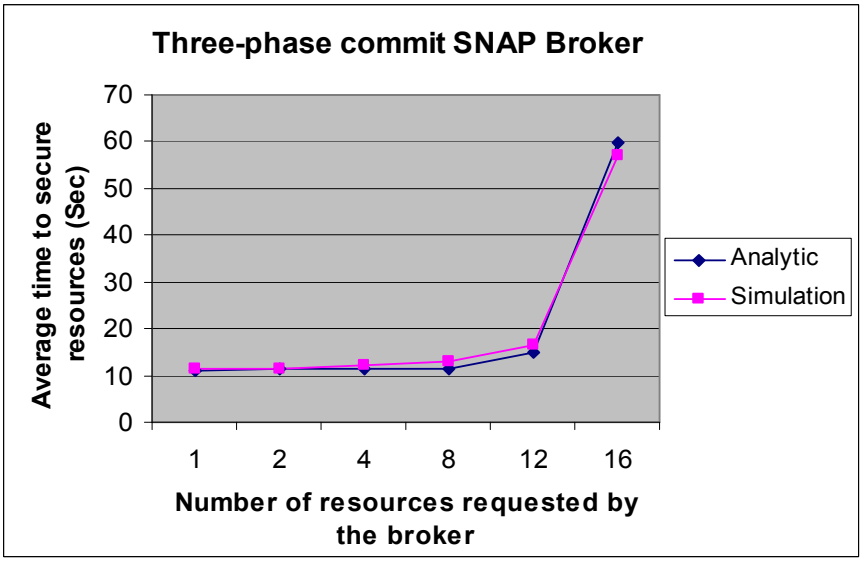

Figure 5.5: Simulation compared to analytical results, showing the average time the three-phase commit SNAP broker takes to reserve resources, with the Grid test-bed parameters. The mean inter-arrival time and mean service time are 350 Sec and 300 Sec respectively.

In Figures 5.6 and 5.7, the same results as shown in Figures 5.4 and 5.5 are used to compare the performance of simple SNAP broker and the three-phase commit SNAP broker. Both the simulation and the analytic results indicate that the three-phase commit SNAP broker provides a performance enhancement, particularly as the number of resources increases to 8 or more. Specifically the performance enhancement for 8, 12 and 16 resources is 14%, 19% and 36% respectively (the percentages are calculated using equation 1), resources below 8 carry a performance improvement of less than 4%. This is to do with the fact that as the number of resources required by the brokers increases so does the probability that some of the selected resources will be taken by third party users before successfully reserving the resources. Thus with the simple SNAP broker if it fails to secure a selected set it must incur the cost of re-contacting the MDS. However with the three-phase SNAP broker the probes provide updates of the resources' status and the broker can adjust without having to incur the cost of re-contacting the MDS. Another factor that will influence the chances of securing resources is the duration of the information provider response time, this effect will be highlighted by the results in the next section (Section 5.5.3) where the WRG parameters will be used (specifically, the MDS response time is higher on the WRG).

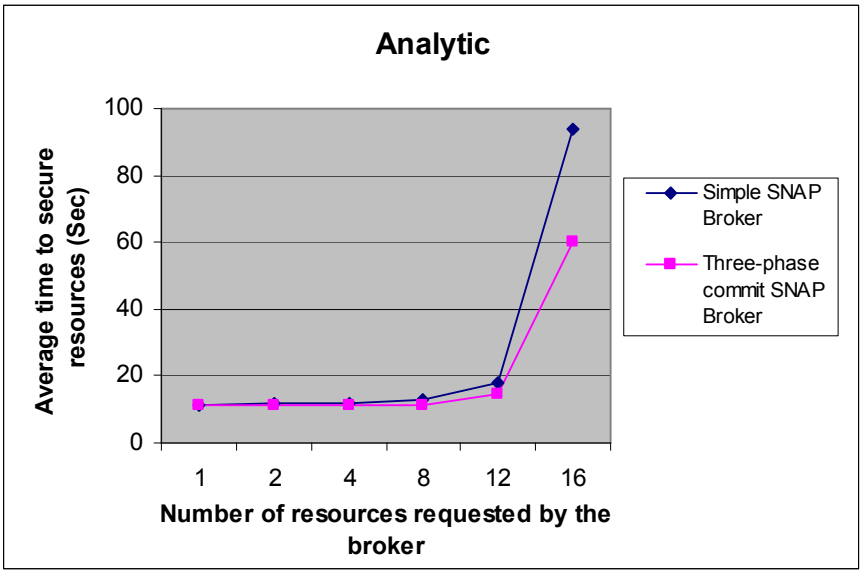

Figure 5.6. Analytical results showing the average time the simple SNAP broker takes in comparison to the average time the three-phase commit SNAP broker takes to reserve resources, with the Grid test-bed parameters. The mean inter-arrival time and mean service times are 350 Sec and 300 Sec respectively.

The results for experiment 2 have been obtained, using the round-robin and random approaches to distribute third party jobs using the extended traffic model. Figures 5.8 and 5.9 show these results for round-robin and the random approach respectively, with the mean inter-arrival time given by equation (12) and the mean service time of 300 Seconds. The three-phase commit SNAP broker clearly outperforms the simple SNAP broker in both traffic conditions when the resources requested increase to 12 or more. This is to do with the fact (indicated above for experiment 1) that as the number of resources requested by the brokers increase so does the probability some of the selected resources will be taken by third party users before successfully reserving the resources.

Overall both the simple and the three-phase commit SNAP broker perform well in a system that distributes third-party jobs using round-robin compared to the random approach. This can be explained by the fact that the round-robin systematically runs through the list of resources and allocates the first set that are available to third-party users as they enter the system. Further, the round-robin method does not revisit the same set of resources until it has fully progressed through the list and looped back. It is also important to note that all the resources in the experiments are set with equal priority and both brokers will nominate the first set of resources that are unoccupied in the list and attempt to reserve them. Hence both the brokers and the round-robin traffic algorithm follow an orderly systematic approach to selecting resources. However the random approach is sporadic in its resource selection for third-party jobs. Conversely both brokers follow the same procedure in selecting their resources as described above when discussing the round-robin approach. Consequently the brokers find it difficult to secure resources as the conflict between the brokers and third party jobs increases substantially. This is due to the fact that the random algorithm does not follow any orderly approach and it disregards the fact of having visited a resource in its previous nomination. Further, as a whole three-phase commit SNAP broker outperforms the simple broker in either traffic conditions (round-robin or random), this is due to the use of the probes that provide raid updates of the resources status as they occur. Further due to the random traffic condition both SNAP brokers incur a higher delay in reserving the resources compared the round-robin again this is related to the methods the two distribute the traffic. Also note that the SNAP brokers take longer to reserve the resources for both round-robin and the random traffic model compared to the simple traffic model

used in experiment 1. This is related to the jobs submitted by third party users requiring more than one resource which are correlated across to different resources by the extended traffic model, while in the simple traffic model only one resource is required by each job.

The performance enhancement of the three-phase commit SNAP broker compared to the simple SNAP broker, based on the round-robin traffic condition for 12, and 16 is 18% and 31% respectively (the percentages are calculated using equation 1), resources below 12 carry a performance improvement of less than 6%. In regards to the random traffic condition the performance enhancement of the three-phase commit SNAP broker compared the simple SNAP broker for 12 and 16 resources is 12% and 18% respectively, resources below 12 carry a performance improvement of less than 3%.

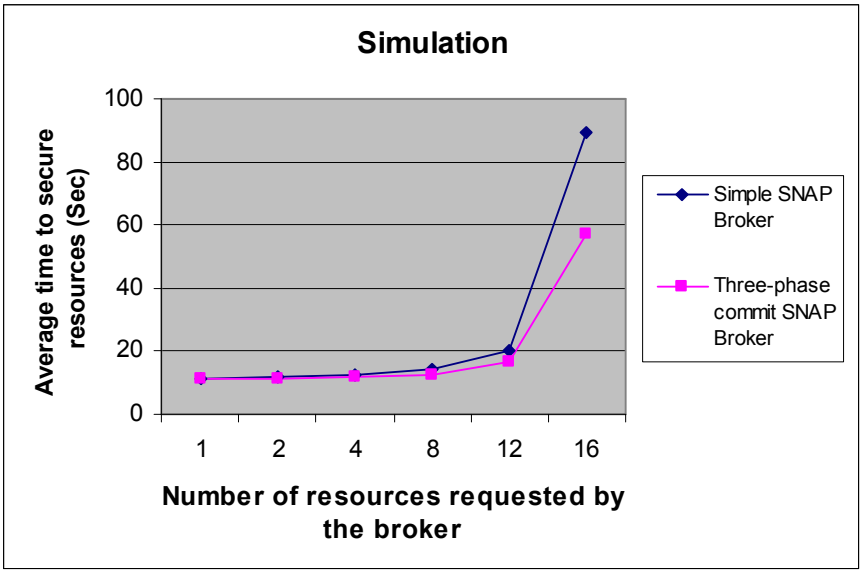

Figure 5.7: Simulation results showing the average time the simple SNAP broker takes in comparison to the average time the three-phase commit SNAP broker takes to reserve resources, with the Grid test-bed parameters. The mean inter-arrival time and mean service time are 350 Sec and 300 Sec respectively.

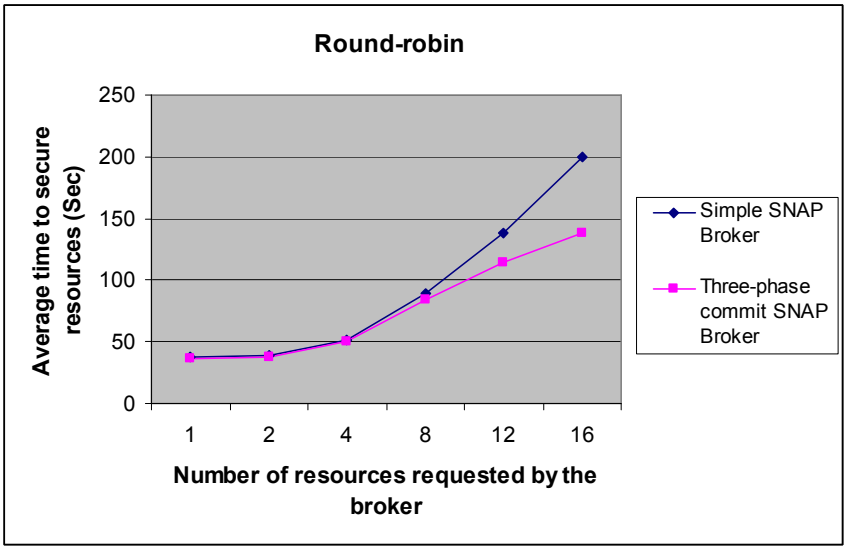

Figure 5.8: Simulation results showing the average time the simple SNAP broker takes in comparison to the average time the three-phase commit SNAP broker take to reserve resources, with the Grid test-bed parameters, when the extended traffic model is used for round-robin resource selection.

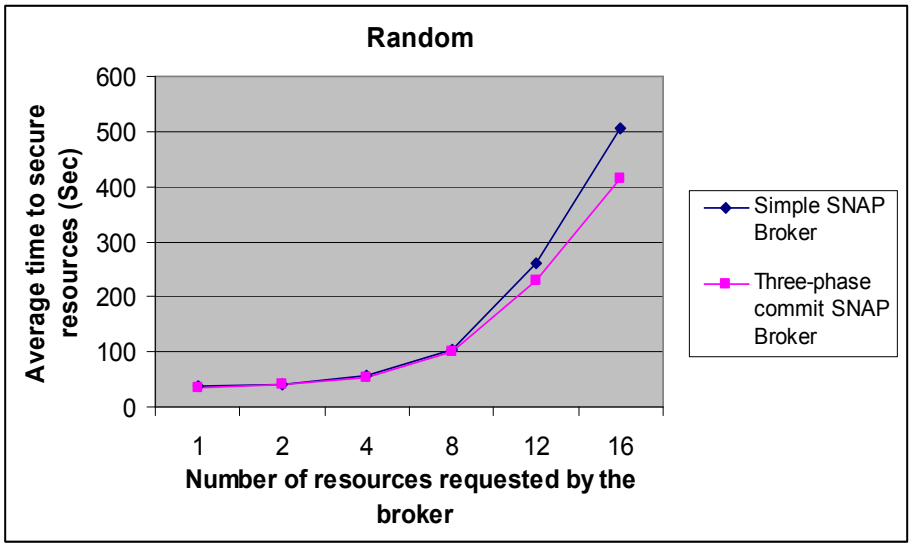

Figure 5.9: Simulation results showing the average time the simple and three-phase commit SNAP brokers takes to reserve resources, with the Grid test-bed parameters, when the extended traffic model is used for random resource selection.

5.5.4 Experimental Results and Discussion based on the White Rose Grid Parameters

The results and discussion for experiment 1 and 2 are presented in this section, which are based on the White Rose Grid (WRG) parameters. The parameters are the same as those on the Grid test-bed apart from two which are:

- Contacting the MDS takes 27 Sec on average. Changes to resource status are picked up in the first 17 Sec. Hence, $t_{mds(1)}$ = 17 Sec and $t_{mds(2)}$ = 10 Sec.
- Reservation takes 1.779 Sec. Hence, t_{res} = 1.779 Sec.

Contacting the MDS on the WRG incurs a higher cost than on the Grid test-bed as it is a much larger infrastructure despite having higher speed CPUs. However the reservation is purely directed at a resource (CPU) which is why it is quicker on the WRG than the Grid test-bed (due to the CPU speed). The other parameter values are the same as the broker resided on the same machine for the experiments carried out on the Grid test-bed and the WRG.

Figure 5.10 and Figure 5.11 show a comparison, for experiment 1, of the simulation results and the analytic results for both the simple SNAP broker and the three-phase commit SNAP broker respectively, using the WRG parameters. The average time taken from receiving user requirements until resources are successfully reserved are plotted as a function of the number of resources required by the user's job. The average inter-arrival time is 350 seconds and the average service time is 300 seconds. As can be seen from the Figures (Figures 5.10 and 5.11), the simulation and analytical results show good agreement with slight discrepancies, as with the Grid test-bed (see Section 5.5.3).

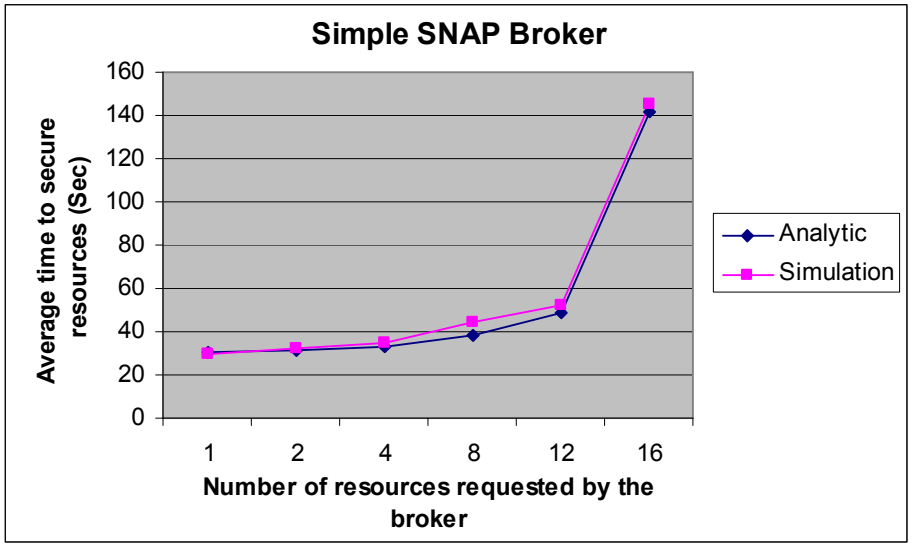

Figure 5.10: Simulation compared to analytical results, showing the average time the simple SNAP broker takes to reserve resources, with the WRG parameters. The mean inter-arrival time and mean service times are 350 Sec and 300 Sec respectively.

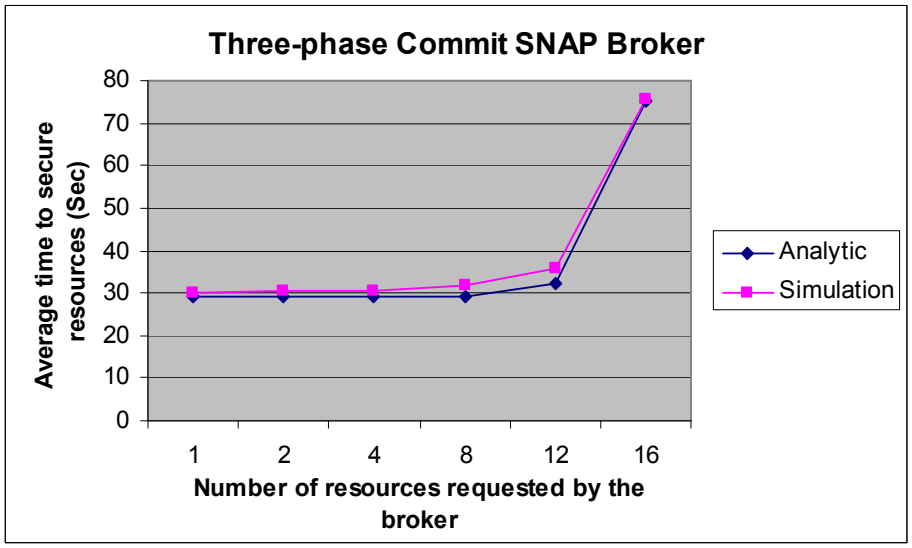

Figure 5.11: Simulation compared to analytical results, showing the average time the three-phase commit SNAP broker takes to reserve resources, with the WRG parameters. The mean inter-arrival times and mean service times are 350 Sec and 300 Sec respectively.

In Figures 5.12 and 5.13, the same results as shown in Figures 5.10 and 5.11 are used to compare the performance of simple SNAP broker and the three-phase

commit SNAP broker. Both the simulation and the analytic results indicate that the three-phase commit SNAP broker provides a significant performance enhancement, particularly as the number of resources increases to 4 or more. The performance enhancement for 4, 8, 12 and 16 resources is 18%, 29%, 35% and 51% respectively (the percentages are calculated using equation 1), however resources below 4 carry a percentage improvement less than 8%. This is to do with the fact (indicated in Section 5.5.3) that as the number of resources required by the brokers increases so does the probability some of the selected resources will be taken by third party users before successfully reserving the resources. However in comparison to the same experiment carried out based on the Grid test-bed parameters, shown in Section 5.5.3 the increase in performance by the three-phase commit SNAP broker is seen when fewer resources are requested based on the WRG parameters. This is related to the increase in MDS response time on the WRG, hence the greater the response time the more likely resources will be taken by third party users.

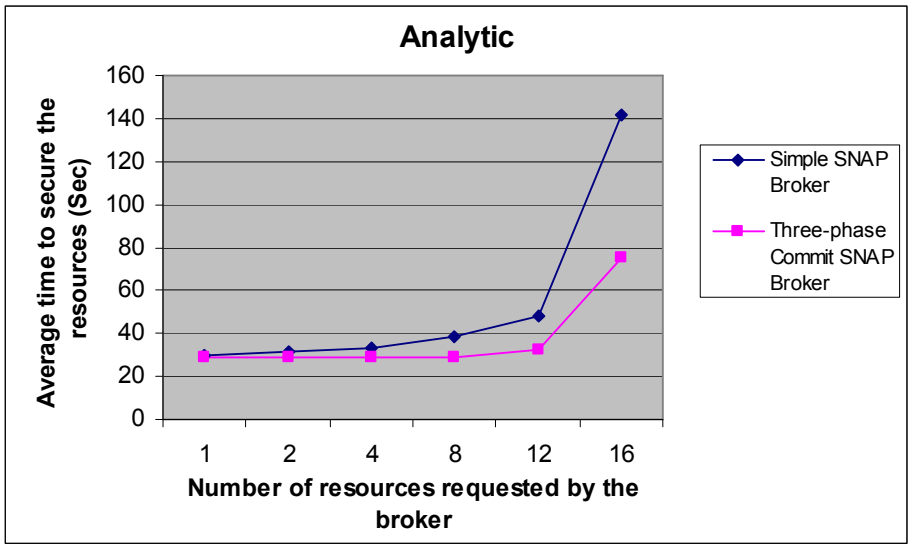

Figure 5.12: Analytical results showing the average time the simple SNAP broker takes in comparison to the average time the three-phase commit SNAP broker takes to reserve resources, with the WRG parameters. The mean inter-arrival time and mean service times are 350 Sec and 300 Sec respectively.

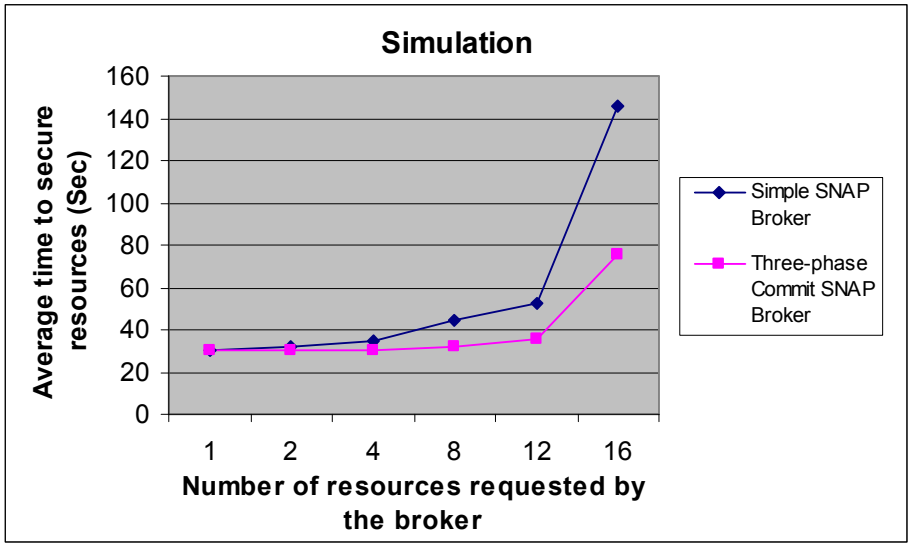

Figure 5.13: Simulation results showing the average time the simple SNAP broker takes compared the average time the three-phase commit broker takes to reserve resources, with the WRG parameters. The mean inter-arrival time and mean service times are 350 Sec and 300 Sec respectively.

The results for experiment 2 based on the WRG parameters have been obtained using the round-robin and random approaches to distribute third party jobs with the extended traffic model, i.e. the approaches discussed in section 5.2.2. Figures 5.14 and 5.15 show the results for the round-robin and the random approach respectively, with mean inter-arrival time given by equation (12) and mean service time of 300 seconds. The three-phase commit SNAP broker outperforms the simple SNAP broker in both traffic conditions when the resources requested increase to 8 or more, again this is related to the increase of competition when more resources are required. The performance increase is seen when fewer resources are requested compared to the same experiments carried out based on the Grid test-bed (Section 5.5.3). This is related (as with the case for experiment 1 based on the WRG parameters) to the increase in MDS response time.

The performance enhancement gained by the three-phase commit SNAP broker compared to the simple broker based on the round-robin traffic condition for 8, 12 and 16 resources is 26%, 31% and 48% respectively (the percentages are calculated using equation 1). However below 8 resources it provides a performance less than 7%. In regards to the random traffic condition the performance enhancement the three-phase commit SNAP broker compared to the simple SNAP broker provides for 8, 12 and 16 resources is 24%, 28% and 43% respectively (the percentages are calculated using equation 1). However for resources below 8 it provides a performance less that 3%. Further, unlike experiment 2 based on the Grid test-bed (where the y-axis differed for the round-robin traffic condition (Figure 5.8) and the random (Figure 5.9) i.e. 0 – 250 Sec (round-robin) and 0 – 600 Sec (random)), on the WRG the y-axis for the round-robin (Figure 5.14) and the random (Figure 5.15) are the same i.e. 0 - 900 Sec. This is related to the increase in the MDS response time on the WRG. Hence this shows when the MDS response time increases it becomes difficult for the simple SNAP broker to secure resources in either the round-robin or the random traffic conditions, while the three-phase commit SNAP broker is able to maintain its enhancement over the simple SNAP broker due to the use of the probes.

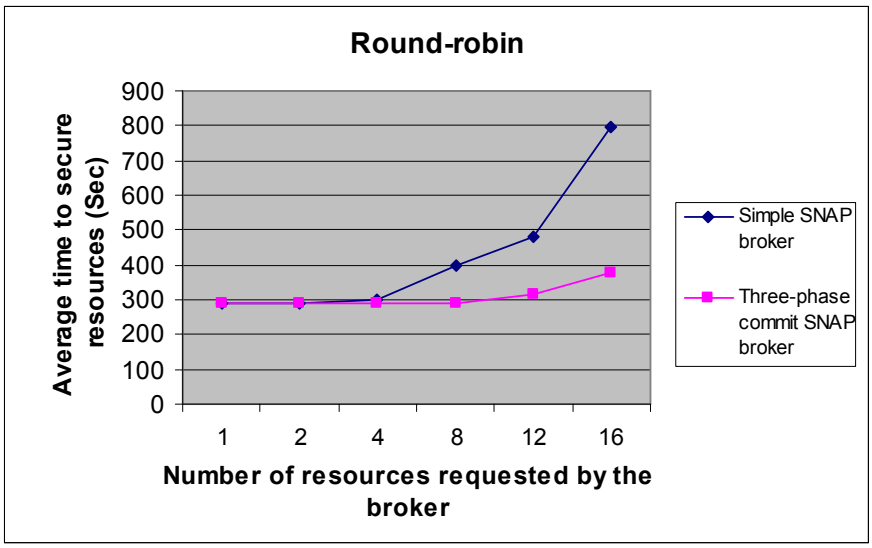

Figure 5.14: Simulation results showing the average time the simple SNAP broker and three-phase commit SNAP broker takes to reserve resources, with the WRG parameters, when the extended traffic model is used for round-robin resource selection.

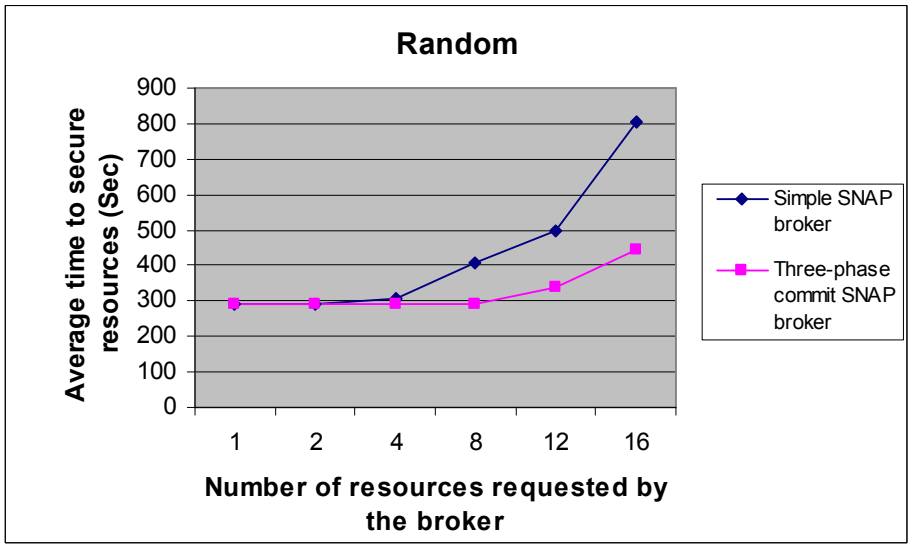

Figure 5.15: Simulation results showing the average time the simple SNAP broker and three-phase commit SNAP broker takes to reserve resources, with the WRG parameters, when the extended traffic model is used for random resource selection.

5.5.5 Overall Evaluation of the Experiments

The experiments carried out in this chapter using parameters from the Grid test-bed (Section 5.5.3) and from the WRG (Section 5.5.4), compared the three-phase commit SNAP broker to the simple SNAP broker using different traffic conditions, i.e. the simple traffic model and the extended traffic model which includes round-robin and random selection. This has shown that the three-phase commit SNAP broker outperforms the simple SNAP broker both in the simple and the extended traffic model.

The experimental results obtained based on the parameters from the WRG, shows the performance improvement gained by the three-phase commit SNAP broker is more significant than that gained with the parameters from the Grid test-bed. This is for both the simple and the extended traffic model. The performance increase is related to the information provider response being larger than that on the Grid test-bed. This is also the same reason for the performance improvement of the experiments in chapter 4 when comparing the results from Grid test-bed to that on the WRG. Further, for both the Grid test-bed and the WRG the simple traffic model provides the highest performance increase followed by the extended traffic model with round-robin than by the random approach This is when comparing the performance increase provided by the three-phase commit SNAP broker to the simple SNAP broker. This is related to the fact that in the simple traffic model there is no correlation in the traffic. However the performance decreases slightly for the round-robin comparing it to the simple traffic model this is related to the correlation in its job traffic. As for the random it suffers for the same reason as the round robin in that is its jobs are correlated by the extended traffic model and further its job allocation is sporadic i.e. it does not follow an orderly sequence in resources allocation.

5.6 Summary

In this chapter mathematical modelling and simulations were used to evaluate the performance of the simple SNAP broker compared to the hree-phase commit SNAP broker. As the experiments carried out in chapter 4 were constrained to the number of physical resources available for the experiments, which also limited the type of traffic conditions used, using simulation (which was validated through mathematical modelling) allowed for the flexibility to investigate the SNAP broker under more realistic traffic conditions and with more resources than that used in chapter 4.

The chapter begins by presenting the SNAP-base Grid resource broker architecture components showing the sequence taken in the simple SNAP broker protocol. This is followed with a description in the format of a sequence of steps for both the simple SNAP broker and the three-phase commit SNAP broker protocol, to aid the development of the mathematical modelling and simulation. The chapter then describes both the simple and extended traffic models that are used to generate traffic conditions both SNAP brokers would be evaluated under. This is then followed with a description of the mathematical model of both the simple SNAP broker and the three-phase commit SNAP broker. A description of the simulation tool is then provided, which was developed specifically for the experiments in this chapter. The chapter then presents the experiment and performance results, firstly for parameters obtained from the Grid test-bed then with the parameters from the WRG (White Rose Grid). Both sets of experiments show the three-phase commit SNAP broker outperformed the simple SNAP broker, with a performance increase for the experiments based on the WRG parameters compared to those based on the Grid test-bed. This is related to the increase in the information provider response time on the WRG. Further as the information provider is high (such as that on the WRG) and the resource request is high (in the case for the experiment in this chapter it was 16) the simple SNAP broker finds it difficult in the simple traffic model and the extended which included round-robin and random to secure resources which is why there is a significant performance enhancement for the three-phase commit SNAP broker in all traffic conditions. Additionally both the simulation and analytical show good agreement for the simple traffic model which provided the foundation to evaluate the three-phase commit SNAP broker to the simple SNAP broker based on simulation to obtain the results for the extended traffic model.

Chapter 6

Conclusion and Future Work

6.1 Summary

The research in this thesis presents the development of a user-centric resource broker that is based on the SNAP framework. The work shows the performance evaluation of the SNAP-based resource broker using the traditional two-phase commit protocol (referred as the simple SNAP broker within the thesis) compared to the proposed and developed three-phase commit protocol (referred to as the three-phase commit SNAP broker within the thesis). The performance evaluation has been carried out on a Grid test-bed, the White Rose Grid (WRG) and through mathematical modelling and simulation, with the results showing, in all evaluations, that the three-phase commit SNAP broker outperforms the simple SNAP broker.

Chapter 2 begins by reviewing Grid resource management and the attributes that differentiate it from traditional distributed systems, which are site autonomy, heterogeneous substrate, policy extensibility and co-allocation. This is followed with a discussion of the two broad categories of distributed resource management approaches namely Network Batch Queuing Systems (NBQS) and Wide Area Scheduling Systems (WASS). This showed that both NBQS and WASS did not have the full necessities to facilitate Grid resource management. NBQS are typically designed for a single administration domain making them autonomous to a single site. Conversely WASS usually crosses over several sites with each system in this category targeting to serve a particular purpose. For example Condor [24-26] is designed to support high-throughput computation by taking advantage of idle compute resources while Gallop [21] is designed to facilitate parallel application. Overall there is not a single NBQS or WASS that provides a solution to all Grid resource management attributes. Before describing the Grid resource management toolkit namely Globus [2, 3], the components (which are constructed through hierarchical layers) that integrate the Grid infrastructure and also server to aid the

resource management are described. The layers consisted of a Grid fabric, Grid middleware, user level middleware and a Grid application layer.

Globus the *de facto* Grid resource management toolkit that is being used by major global Grid development teams including the UK e-Science projects [46] is described. The toolkit comprises of the Globus Security Infrastructure (GSI) [30], the Monitoring and Discovery System (MDS) [33, 34, 50] to provide information of the resources status, Grid File Transfer Protocol (GFTP) and more importantly Grid Resource Allocation Manager (GRAM) for resource management. This followed by providing an example of major Grid projects which lead to the description of Distributed Aircraft Maintenance Environment (DAME) [9] which has been used as an exemplar for this research due to the need for resources on demand.

A review of Grid resource brokers is then provided with a description of their main components, their architectures and limitations. This then followed with a review of resource information providers which are an integral part of the Grid due to its divers scope and dynamic nature. Finally Chapter 2 looked into the two well-known resource reservation systems the Globus Architecture for Reservation and Allocation (GARA) and MAUI [12].

Chapter 3 firstly provides an overview of the process of submitting a job to a local distributed resource management system compared to the same submission through the Grid middleware. This was to provide an insight into the Grid middleware complexities and the need for a Grid resource broker to insulate the user from the difficulties. This then followed with an overview of Grid Service Level Agreements and the Service Negotiation and Acquisition Protocol (SNAP) [10], before describing the developed SNAP-based Grid resource broker architecture. The architecture's components were then described individually. This then followed with a scenario to highlighting the need to secure resources which consequently lead to the development of the three-phase commit protocol that was described in detail.

Chapter 4 presents empirical performance evaluation results from experiments carried out on a Grid test-bed which showed the three-phase commit SNAP broker outperformed the simple SNAP broker by up to 54% in certain scenarios. The same experiments were then further carried out on the White Rose Grid (WRG) [11] which also showed that same effect with the three-phase commit SNAP broker providing a performance improvement of just below 75%.

Chapter 5 further evaluated the SNAP brokers through mathematical modelling and simulation, this allowed a wider traffic condition to be considered than that used in chapter 4 due to the availability of physical resources for the experiments which also restricted the traffic conditions considered. The experiments were firstly conducted with the parameters obtained from the Grid test-bed and then repeated with the parameters obtained from the WRG (this was to validate the results based on the Grid test-bed). Again the three-phase commit SNAP broker outperformed the simple SNAP broker, both in the simple traffic model and the extended (which included round-robin and random) and in both parameter environments (Grid test-bed and WRG). Additionally both the simulation and modelling show good agreement for the simple traffic model which provided the foundation to evaluate the three-phase commit SNAP broker to the simple SNAP broker based on simulation to for the extended traffic model.

6.2 Contributions

The contributions of the research work in this thesis is summarised in the following points:

- The design and development of the three-phase commit protocol that secures resources on demand, which follows the traditional two-phase commit protocol. However the first phase is separated into two parts to enhance the current two-phase commit protocol and strengthen it through the use of probes. The probes provide rapid update of the resource status, allowing a broker to adjust swiftly to the changes which are highlighted through a scenario in Section 3.4.1.

- The development of a SNAP-based Grid resource broker that insulates a user from the Grid middleware complexities. The broker ascertains the user's job requirements through a user interface, filters out the appropriate resources that have the capability for the user's job through the use of a Knowledge Bank (KB). It prioritises the resources based on their past performance and then contacts the resources information

provider to gain their dynamic status. It then nominates a set of resources and secures them through the three-phase commit protocol before submitting the user's job for execution. The SNAP resource broker differs from current broker system as with the current systems the user needs to interact with and is exposed to the Grid middleware complexities, even after the job requirements are provided to the broker. For example with Nimrod/G [5, 69] it requires a user to create a task farm (plan) through the use of its declarative parametric modelling language before a job is passed for processing. Hence it does not provide automated resource discovery, a list of resources in the form of Globus Gatekeeper contact strings need to be set up manually by the user before brokering takes place. This is also the case with the Grid Broker [76], in that it does not provide automated resource discovery. Furthermore in the Grid Broker system the decision making process is left to the user as to where to submit a job without the system verifying if the resources are available for use. With Condor/G [74] again the user needs to query the information providers manually and needs to write the Resource Specification Language (RSL) for the job to be executed at the appropriate resources. As for AppLeS [79-81] in order for a user to submit a job using this system, the user's job application source code needs to be modified and recompiled or filtered through into a template to be compliant with the Apples scheduling agents. Once this time consuming process is complete, the user is still required to provide an input, stating a list of resources contact details that the user has the credentials to utilise, to enable the broker to query the resources status. Further most importantly in comparison to other brokering systems the SNAP broker ensures the resources are reserved prior to the user's job submission which is done through the use of the three-phase commit protocol. Existing brokering currently do not support reservation to secure resources prior to the user's job submission. This is to ensure other third party users unknown to the broker do not utilise the broker's nominated resources, while it is in the process of submitting the user's job, which will delay the execution start time of the user's job as it will be placed in the pending queue.

- The proposal and use of a Knowledge Bank (KB), a data repository that stores static information of resources as attributes which provides a description of their characteristics. This helps in many ways such as supporting automated resource discovery. It facilitates the SNAP brokers (both the simple and the three-phase commit version) to filter out all resources that can handle a job's requirements prior to contacting the resource's information provider, avoiding unnecessary processing of resource contacts. Also it alleviates the user from the burden of having to keep a log file of the resources with their associated descriptions that they are entitled to use. Further in addition to keeping a description of the resource specification, additional attributes are also used to keep a history of past behaviour performance of a resource. Relating to the latter, resources are classified as low/high priority according to whether they meet a pre-defined level of performance, at present it is based on reliability, i.e. the likelihood of a resource crashing during the execution of a job. For example if a resource often crashes, it is likely to be classified as low priority.

- Empirical performance evaluation of the simple SNAP broker compared to the three-phase commit SNAP broker on a local Grid test-bed. The evaluation showed that the three-phase commit SNAP broker outperformed the simple SNAP broker and in certain scenarios with an enhancement improvement of 54%.

- The deployment of both the simple SNAP broker and the three-phase commit SNAP broker onto the WRG. The deployment was challenging from two aspects, the technical, which mainly suffered from site autonomy and heterogeneous substrate. The second was human collaboration which was related to geographical distance and the fact it was the first type of deployment on the WRG, which required to push the boundaries of the existing technologies by modifying existing settings, enhancing existing services and adding a new protocol (three-phase commit protocol). The deployment could only be achieved by close collaboration and Co-operation of the various site's administrators. Further despite the technical and collaboration challenges encountered

the deployment shows that the three-phase commit SNAP broker works on a real distributed Grid infrastructure.

- Empirical performance evaluation of the simple SNAP broker compared to the three-phase commit SNAP broker on a real distributed Grid infrastructure, the WRG. This is to validate the experiments carried out on the local Grid test-bed. Again the three-phase commit SNAP broker outperformed the simple SNAP broker, the increase in performance compared to the Grid test-bed is related to the increase in information provider response time. Further the simple SNAP broker will not outperform the three-phase commit SNAP broker as the protocol used by the latter broker follows the same procedure as the simple SNAP broker. The WRG experiments also validated to show that there is no overhead cost associated with setting up the probes since this occurs concurrently when initially contacting the Monitoring and Discovery System (MDS) [33, 34, 50, 136]. It is also important to note that the use of the three-phase commit protocol is scalable compared to repeatedly contacting the information providers which is the case in the simple SNAP broker. This is due to the fact that the three-phase commit only makes the initial contact to the information providers and then uses the probes for updates.

- Mathematical modelling and simulation of both the simple SNAP and the three-phase commit SNAP broker. This evaluates the SNAP brokers under a wider traffic conditions than that possible on the Grid test-bed and the WRG. With the experiments carried out on the Grid test-bed and the WRG, the three-phase commit SNAP broker outperformed the simple SNAP broker.

6.3 Future Work

There are many ways to further extend the work presented in this thesis. The most appealing ones are listed below:

- In the thesis the use of a Knowledge Bank (KB) was proposed and used within the SNAP broker architecture to mainly aid automated resources

discovery and to alleviate the user from having to keep a log file of the resources he/she is entitled use. The KB could be enhanced by including attributes that characterise a resource based on its past performance such as the likelihood of it crashing during job execution or its performance degradation. This would enhance the Decision Maker component in the SNAP broker architecture to better prioritise the resources as more information is know about the resources.

- The option to leave the experiments running on the WRG for a week or even longer to assess the performance of the three-phase commit protocol under real Grid conditions was explored. However this option was not practical as the WRG is a production Grid with several projects and with many users that use its resources. Further, the experiments that were carried out on the WRG and presented in this thesis had to be authorised by the directors of the three sites. This was related to the nature of the three-phase commit protocol requiring certain technologies to be modified such as those discussed in Appendix B, which meant a limited time was given to complete the experiments. Hence experiments that would run under real Grid conditions would require a large number of resources (only a limited amount was provided for the experiment) and time. This consequently would hinder the progress of projects that have a contract in using the resources and which also fund the maintenance of the infrastructure.

- After the deployment of the three-phase commit protocol on the WRG and the completion of the experiment that clearly shows the performance gain in the use of the three-phase commit protocol compared to the simple approach, the option of installing the protocol on all resources on the WRG was investigated. However it was concluded that it is not viable as the man power and the technical knowledge would not be available for long term maintenance since the protocol was part of the research carried out for this thesis.

- The three-phase commit protocol has been developed to reserve Grid resources on demand. However this protocol can be used in other fields

such as online train booking systems or other reservation systems. This should be explored and investigated.

- The performance evaluation of the SNAP brokers through mathematical modelling and simulation is performed using the average information provider response time. However additional experiments could be carried out using random information provider response time within the boundaries of the average response time. This would help to provide a better realistic environment setting than that used in the current work. Other queueing strategies in the extended traffic model could be investigated such as when jobs are sent to local queues with the least number of jobs or to local queues with the least waiting time. Further other modelling tools such as Petri Nets [126] could be used to study the system's bottlenecks.

- It would be ideal to evaluate the SNAP brokers on a global scale that crosses over many continents and uses several hundred resources. This is beneficial in two ways: 1) It would provide an even better comparison of the of the three-phase commit SNAP broker compared to the simple SNAP broker than that shown in chapter 4 and 5 due, to the geographical distance and quantity of resources. 2) Despite the latency issue that would arise when comparing the SNAP brokers over many continents, the three-phase commit will still outperform the simple SNAP broker. This is related to the fact that the simple SNAP broker has to repeatedly contact the information providers for updates of resources status. This will generate a high volume of traffic due to the number of resources evaluated and will also slow its decision making process as it needs to wait for all information providers to reply which can be long as seen when comparing the Grid test-bed to that of the WRG. Thus with the evaluation crossing over several contestants it would mean an even longer delay than that on the WRG. In regards to the three-phase commit SNAP broker it only needs to contact the information providers once then it relies on the probes to provide any resources status updates, further this will create less traffic than having to repeatedly contact the information provider as with the case in the simple SNAP broker.

Appendix A

The tables shown below (Tables A.1 – A.3.) are those used in the Knowledge Bank (KB). Table A.1 stores user's information such as the login details and password, Table A.2 lists which resources a user is entitled to access while Table A.3 stores the resources specification.

Users(UserName, Sign_in_id, Sign_in_password)

```
CREATE TABLE Users
        (UserName          CHAR(50)          NOT NULL
         Sign_in_ID        CHAR(20)          NOT NULL
         Sign_in_password  CHAR(20)          NOT NULL
         PRIMARY KEY       (Sign_in_ID))
```

Table A.1: User's information details.

Accounts(Sign_in_id, Resource_name)

```
CREATE TABLE Accounts
        (Sign_in_ID          CHAR(20)                    NOT NULL
         Resource_name       CHAR(20)                    NOT NULL
          PRIMARY KEY        (Sign_in_ID, Resource_name))
```

Table A.2 Storing a user's identification and the resource names entitled to it.

Resource (Resource_name, Host_name, Site_address, CPU_count, CPU_speed, CPU_version, RAM_total, Storage_capacity, OS_type, OS_version, Num_crashed_jobs, Num_uncrashed_jobs, Mean_history_profile)

```
CREATE TABLE Resource
        (Resource_name          CHAR(20)        NOT NULL
         Host_name              CHAR(20)        NOT NULL
         Site_address           CHAR(20)        NOT NULL
         CPU_count              INTEGER         NOT NULL
         CPU_speed              INTEGER         NOT NULL
         CPU_version            INTEGER         NOT NULL
         RAM_total              INTEGER         NOT NULL
         Storage_capacity       INTEGER         NOT NULL
         OS_type                CHAR(20)        NOT NULL
         OS_version             CHAR(10)        NOT NULL
         Num_crashed_jobs       INTEGER         NOT NULL
         Num_uncrashed_jobs     INTEGER         NOT NULL
         Mean_history_profile   INTEGER         NOT NULL
          PRIMARY KEY           (Resource_name))
```

Table A.3 Store the resources details. The acronym OS represents Operating System. The attribute OS_version uses a character type as the version could include more than one decimal point i.e. 2.2.1

Appendix B

The Monitoring and Discovery System (MDS) [33, 34, 50] is used for the broker's architecture to gather dynamic information, which is extended from the default installation. Specifically, the MDS is deriving information from the local resource manager, Sun Grid Engine (SGE) [19]. An attribute is added to each GRIS (Grid Resource Information Service) to indicate the different transitions the resources evolve into, from normal, amber to the red state. The Time-To-Live (TTL) for each information provider is set to zero so that any query would retrieve fresh information and not cached information, which could be out of date.

The MDS provides information on request, which does not facilitate the broadcast or streaming of information regarding the changes of resources' status. This facility is necessary to enable active probes working on behalf of the broker to be kept updated. A solution to this is to create a server that acts as an interface to SGE, which enables the streaming of resources changes through a port associated to the server. When a job starts or completes the information is broadcast to anyone listening on that port. This is supported by the in-built function of SGEs, Prolog and Epilog, which inform the server when a job has started and ended respectively and is secure as only information that can be obtained during anonymously querying the information provider is broadcast.

Netscape Directory Software Development Kit 4.0 [137] is used to query the GRIS on each resource. Taking this approach is more efficient than having to search the GIIS (Grid Index Information Service) hierarchy, which could have resources that may not be able to cater for the task requirements. Note that the broker typically does not want information about every resource, since the KB has been used to identify only resources that are capable of supporting the application. Hence this approach helps to avoid unnecessary processing.

The KB is developed using MySQL [108] to store static data and relevant information associated with the users and the resources. JDBC-ODBC (Java Database Connectivity – Open Database Connectivity) is used for the communication between the broker and the KB. Globus 2.4 [2, 3] and CoGkit 1.1a

[72] are used for the security and the binding of the resource with the task. They also support GridFTP [37] and the generation of the RSL (Resource Specification Language) to initiate the execution. GRAM (Grid Resource Allocation Manager) [3] is recompiled and installed after the poling service was modified for updates from every 20 seconds (which is the default installation value) to 1 second. This ensures the various stages of a job process from Pending, Active to Done are recorded by the broker as they occur.

The user interface was developed in the form of a Grid portal built on Web technologies namely XML, and HTML and Java Servlets to transfer the user inputs to the broker for processing.

Appendix C

The following experiments are carried out to ensure that the basic traffic and broker models have been implemented correctly in the simulation. These were carried out prior to the experiments described in chapter 5. The traffic model referred to in relation to the preliminary tests is the simple traffic model.

Test 1

Consider a single resource. The mean service time is fixed at 30 sec. The mean inter-arrival time (λ^{-1}) is varied between 35 and 300 sec. Values for the following parameters will be obtained using both modelling and simulation:

1) Mean number of items in the queue.

2) Mean time spent by an item in the queue.

In this test (and those that follow) the simulation is run enough times to give 95% confidence in the results. If analytical and simulation results agree, then this gives confidence that the queueing model is being applied correctly in both.

Test 2

Assume there is no other traffic entering the system (i.e. the inter-arrival rate is 0). The time taken (from the point when user requirements are received) to submit a job will be obtained through both analytical and simulation approaches and the results compared. Note that the number of resources the job requires does not affect these results. If analytical and simulation results agree, then this gives confidence that the model describing the working of the broker is being applied correctly in both.

Test 3

This is the first experiment to address the integration between the traffic model and the broker model. The mean service time is fixed at 30 sec. The mean inter-arrival time (λ^{-1}) for every resource is fixed at 35 sec. The total number of resources is fixed at 128 and the number of resources required by the job being submitted by the broker (J) is varied between 1 and 100. Results for modelling and simulation are compared.

Results for Test 1

The first question (mean number of items in the queue) is addressed using the following expression.

$$\overline{N}_q = \lambda W , \tag{13}$$

where $\overline{N}_q$ is the mean number of items in the queue and W is the mean waiting time in the queue. The mean waiting time is obtained using (note $\rho = T_s / T_\lambda$),

$$W = \frac{\rho T_s}{(1-\rho)}. \tag{14}$$

Eliminating ρ from these expressions leads to

$$\overline{N}_q = \frac{T_s^2}{T_\lambda^2\left(1-\frac{T_s}{T_\lambda}\right)} \tag{25}$$

and

$$W = \frac{T_s^2}{T_\lambda\left(1-\frac{T_s}{T_\lambda}\right)} . \tag{36}$$

Here $T_\lambda = \lambda^{-1}$ is the mean inter-arrival time.

Figures C.1 and C.2 show both the modelling and simulation results for mean number of items in the queue and mean queue waiting time (both as a function of mean inter-arrival time) respectively.

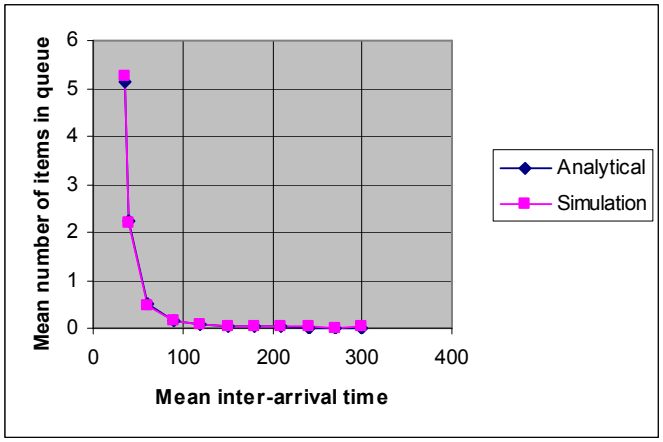

Figure C.1 Comparison between analytical and simulation for mean number of items in the queue as a function of mean inter-arrival time.

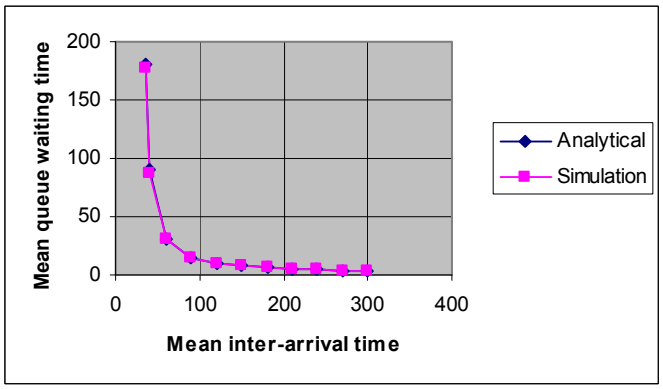

Figure C.2 Comparison between analytical and simulation for mean queue waiting time as a function of mean inter-arrival time.

Results for Test 2

When no other traffic is in the system, the analysis indicates that,

$$E(t_{simple}) = 11.176s \qquad (47)$$

This is given by equation (4).

For the three-phase commit protocol,

$$E(t_{three-phase}) = 11.176s \qquad (58)$$

This is given by letting $\lambda = 0$ and replacing $E(t_{dec/three-phase})$ with t_{dec} in equation (11). As expected, the results are the same for both brokers.

These agree exactly with the times obtained in the simulation.

Results for Test 3

The following expression can be used to determine the probability (P_{NR}) that not enough resources are available to support the job on initial contact with the Monitoring and Directory System (MDS) [33, 34, 50] .

$$P_{NR} = \sum_{K=0}^{J-1} \frac{P!}{(P-K)!\,K!} \rho^{P-K} (1-\rho)^{K} \qquad (69)$$

Here $\rho = \lambda T_s$ is the server utilisation.

The value of P_{NR} can then be estimated through simulation by running it many times for a given value of J and determining the frequency with which enough resources are free. Figure C.3 shows the results for both analytical and simulation.

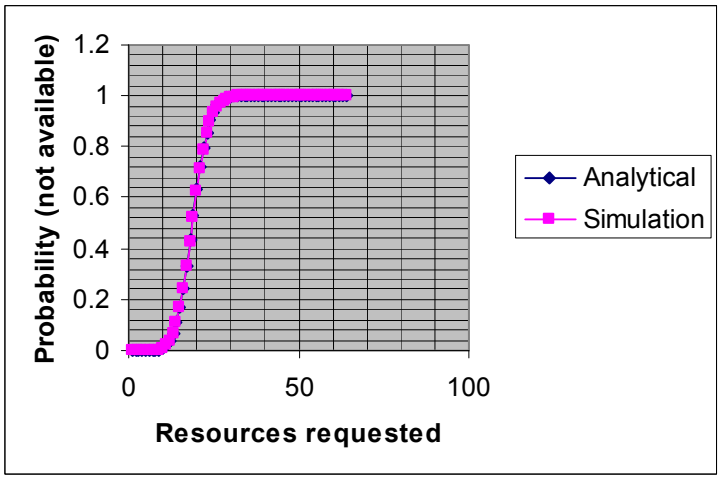

Figure C.3 Comparison between simulation and analytical for the probability that insufficient resources are available when the MDS is first contacted, as a function of number of resources requested (P=128).

Appendix D

Table D.1 shows the values for $E(t_{mds/dec})$ and $E(t_{dec/three-phase})$ obtained through summation with the Grid test-bed parameters. Table D.2 is the same table but the value obtained through the White Rose Grid (WRG) [11] parameters. The values in Table D.1 and D.2 are used to aid the mathematical modelling as discussed in Section 5.5.1

Resources required by the broker	$E(t_{mds/dec})$	$E(t_{dec/three-phase})$
1	8.191161	0.007
2	8.431371	0.007
4	8.719623	0.007
8	9.664449	0.015041
12	14.1804	3.075493
16	68.39579	42.30818

Table D.1: Show the values obtained through simulation based on the Grid test-bed parameters for $E(t_{mds/dec})$ and $E(t_{dec/three-phase})$.

Resources required by the broker	$E(t_{mds/dec})$	$E(t_{dec/three-phase})$
1	27.007	0.007
2	27.007	0.007
4	27.007	0.007
8	27.11503	0.015041
12	30.40988	3.075493
16	80.804949	42.30818

Table D.2: Show the values obtained through simulation based on the WRG parameters for $E(t_{mds/dec})$ and $E(t_{dec/three-phase})$.

References

1. Czajkowski, K., I. Foster, N. Karonis, and C. Kesselman. *A Resource Management Architecture for Metacomputing Systems*. In *IPPS/SPDP '98 Job Sheduling Strategies for Parallel Processing*. 1998. Orlando; FL: Berlin. p. 62-82.

2. *The Globus Alliance. URL: http://www.globus.org/.* 2005.

3. Foster, I. and C. Kesselman, *The Globus Project: a Status Report.* Journal of Future Generations Computer Systems, 1999. **15**(5-6): p. 607-621.

4. Krauter, K., R. Buyya, and M. Maheswaran, *A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing.* International Journal of Software Practice and Experience, 2002. **32**(2): p. 135-164.

5. Buyya, R., D. Abramson, and J. Giddy. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. In *International Conference/exhibition on High Performance Computing in the Asia-Pacific Region*. 2000. Beijing: IEEE Computer Society. p. 283-289.

6. Czajkowski, K., I. Foster, and C. Kesselman. *Resource Co-allocation in Computational Grids*. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*. 1999. p. 219 -288.

7. Foster, I., et al. *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation*. In *International Workshop on Quality of Service*. 1999. London: Piscataway. p. 27-36.

8. Raman, R., M. Livny, and M. Solomon. *Matchmaking: Distributed Resource Management for High Throughput Computing*. In *The Seventh IEEE International Symposium on High Performance Distributed Computing*. 1998. Chicago; IL: IEEE Computer Society Press. p. 140-147.

9. Austin, J., et al., *Predictive Maintenance: Distributed Aircraft Engine Diagnostics*, in *Grid2: Blueprint for a New Computing Infrastructure.*, I. Foster and C. Kesselman, Editors. 2003, Morgan Kaufmann, 2nd Edition, Chapter 5. p. 69-79.

10. Czajkowski, K., I. Foster, C. Kesselman, V. Sander, and S. Tuecke. *SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management*. In *Distributed Systems. Proceedings of Job Scheduling Strategies for Parallel Processing*. July 24 – 26, 2002. Edinburgh, Scotland. p. 153-183.

11. *White Rose University Consortium. URL: http://www.wrgrid.org.uk/.* 2005.

12. David, B.J., *Grid Scheduling with Maui/Silver*, in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. Schopf, and J. Weglarz, Editors. 2003, Kluwer Academic Publishers, Chapter 11. p. 161-170.

13. Buyya, B., S. Chapin, and D. DiNucci. *Architectural Models for Resource Management in the Grid*. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*. 2000: Publisher: Springer Verlag Lecture Notes In Computer Science, Vol. 1971. p. 18–35.

14. Foster, I. and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. 2003: Morgan Kaufmann.

15. Sacks, L., et al., *Active Robust Resource Management in Cluster Computing Using Policies.* Journal Of Network And Systems Management, Special Issue on Policy Based Management of Networks and Services, 2003. **11**(3): p. 329-350.

16. Baker, M., R. Buyya, and D. Laforenza, *The Grid: International Efforts in Global Computing.* International Conference on Advances in Infrastructure for Electronic Business, Science,and Education on the Internet (SSGRR 2000), l`Aquila, Rome, Italy, In CD-ROM, ISBN 88-85280-52-8, July 31 - August 6, 2000.

17. *Platform Load Sharing Facility. URL: http://www.platform.com/products/LSF/.* 2005.

18. Henderson, R.L. *Job Scheduling Under the Portable Batch System*. In *Job scheduling strategies for parallel processing*. 1995. Santa Barbara; CA: Springer-Verlag. p. 279-294.

19. *Sun Grid Engine. URL: http://gridengine.sunsource.net/.* 2005.

20. *LoadLeveler. URL: http://www.hlrn.de/doc/loadl/.* 2005.

21. Weissman, J.B., *Gallop: The Benefits of Wide-Area Computing for Parallel Processing.* Journal Of Parallel And Distributed Computing, 1998. **54**(2): p. 183-205.

22. Chapin, S.J., D. Katramatos, J. Karpovich, and A.S. Grimshaw. *The Legion Resource Management System*. In *Job scheduling strategies for parallel processing*. 1999. San Juan; PR: New York. p. 162-178.

23. Natrajan, A., M.A. Humphrey, and A.S. Grimshaw. *Capacity and Capability Computing Using Legion*. In *Proceedings of the International Conference on*

Computational Sciences-Part I. 2001: Lecture Notes In Computer Science, Vol. 2073. p. 273-283.

24. Epema, D.H.J., M. Livny, R. Van Dantzig, X. Evers, and J. Pruyne, *A Worldwide Flock of Condors: Load Sharing Among Workstation Clusters.* International Journal of Future Generations Computer Systems, 1996. **12**: p. 53-66.

25. Thain, D., T. Tannenbaum, and M. Livny, *Distributed Computing in Practice: The Condor Experience.* Concurrency And Computation: Practice and Experience, 2005. **17**: p. 323-356.

26. Tannenbaum, T., D. Wright, K. Miller, and M. Livny, *Condor - A Distributed Job Scheduler*, in *Beowulf Cluster Computing with Linux*, T. Sterling, Editor. 2002, The MIT Press.

27. Lo, V.M., *Temporal Communication Graphs: Lamport's Process-Time Graphs Augmented for the Purpose of Mapping and Scheduling.* Journal Of Parallel And Distributed Computing, 1992. **16**(4): p. 363 - 377.

28. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid*, in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G.C. Cox, and A.J.G. Hey, Editors. 2003, Wiley, Chapter 6. p. 171-197.

29. Buyya, R., *Economic-based Distributed Resource Management and Scheduling for Grid Computing.* PhD Thesis, Monash University, Melbourne, Australia, April 2002.

30. Welch, V., et al. *Security for Grid Services*. In *The Twelfth IEEE International Symposium on High Performance Distributed Computing*. June 2003. Seattle, WA: IEEE Computer Society Press. p. 48-57.

31. Pearlman, L., V. Welch, I. Foster, C. Kesselman, and S. Tuecke. *A Community Authorization Service for Group Collaboration*. In *Policies for Distributed Systems and Networks; Policy 2002*. 2002. Monterey, CA: IEEE. p. 50-59.

32. Butler, R., et al., *A National-Scale Authentication Infrastructure.* Journal Of IEEE Computer, 2000. **33**(12): p. 60-66.

33. Czajkowski, K., S. Fitzgerald, I. Foster, and C. Kesselman. *Grid Information Services for Distributed Resource Sharing*. In *The Tenth IEEE International Symposium on High performance distributed computing*. 2001. San Francisco, CA: IEEE Computer Society Press. p. 181-194.

34. Fitzgerald, S., I. Foster, C. Kesselman, and G. Von Laszewski. *A Directory Service for Configuring High-Performance Distributed Computations*. In

The Sixth IEEE International Symposium on High Performance Distributed Computing. 1997. Portland; OR: IEEE Computer Society Press. p. 365-375.

35. Allcock, B., et al., *Data Management and Transfer in High-Performance Computational Grid Environments.* Journal Of Parallel Computing, 2002. **28**(5): p. 749-771.

36. Stockinger, H., et al., *File and Object Replication in Data Grids.* Journal Of Cluster Computing, 2002. **5**(3): p. 305-314.

37. Allcock, B., et al., *The Globus Striped GridFTP Framework and Server*. 2005, To be Published in the Proceedings of Super Computing 2005 (SC05), November 2005.

38. Nabrzyski, J., J. Schopf, and J. Weglarz, *Grid Resource Management State of the Art and Future Trends*. 2003: Kluwer Academic Publishers.

39. Berman, F., G. Fox, and A.J.G. Hey, *Grid Computing Making the Global Infrastructure a Reality*. 2003.

40. Natrajan, A., S.A. Grimshaw, A.M. Humphrey, and A. Nguyen-Tuong, *Dispelling Seven Myths about the Grid Resource Management, Technical Report CS-2004-33*. August 2001, University of Virginia Department of Computer Science.

41. Brucker, P., *Scheduling algorithms*. 3rd ed. 2001: Springer-Verlag New York, Inc Secaucus, NJ, USA.

42. Casavant, T.L. and J.G. Kuhl, *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems.* IEEE Transactions on Software Engineering, February 1988. **14**(2): p. 141 - 154.

43. Silberschatz, A., P.B. Galvin, and G. Gagne, *Operating System Concepts*. 7th ed. 2005: Hoboken, NJ: J. Wiley & Sons.

44. Stallings, W., *Operating Systems: Internals and Design Principles*. 4th ed. 2000.

45. *Argonne national Laboratory. URL: http://www.anl.gov/.* 2005.

46. *National e-Science Centre. URL: http://www.nesc.ac.uk/.* 2005.

47. *W3C. URL: http://www.w3.org/.* 2005.

48. *Globus 4 Tutorial URL:http://gdp.globus.org/gt4-tutorial/.* 2006.

49. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G.C. Cox, and A.J.G. Hey, Editors. 2002, Wiley, Chapter 8. p. 217 - 249.

50. Yeong, W., T. Howes, and S. Kille, *Lightweight Directory Access Protocol, IETF RFC 177*. March 1995.

51. *International virtual data Grid Laboratory URL: http://www.ivdgl.org/.* 2006.

52. *European Data Trans-Atlantic Grid URL:http://datatag.web.cern.ch/datatag/.* 2006.

53. *IBM IntraGrid URL: http://www-306.ibm.com/e-business/br/glossary/glossary_i.shtml.* 2006.

54. *European Data Grid. URL: http://eu-datagrid.web.cern.ch/eu-datagrid/.* 2005.

55. *European Grid Solar Observations URL:http://www.mssl.ucl.ac.uk/grid/egso/.* 2006.

56. *CrossGrid URL:http://www.crossgrid.org/main.html.* 2006.

57. *Grid Physics Network URL:http://www.griphyn.org/.* 2006.

58. *Earth System Grid URL:https://www.earthsystemgrid.org/.* 2006.

59. *Fusion Collaboratory URL: http://www.fusiongrid.org/.* 2006.

60. *Information Power Grid URL:http://www.gloriad.org/gloriad/projects/project000053.html.* 2006.

61. *Asia Pasific Bioinformatics Network URL:http://www.apbionet.org/.* 2006.

62. *Grid Datafarm URL:http://datafarm.apgrid.org/.* 2006.

63. *AstroGrid. URL: http://www.astrogrid.org/.* 2005.

64. *CombeChem URL:http://www.combechem.org/.* 2006.

65. *Reality Grid. URL: http://www.realitygrid.org/.* 2005.

66. Nairac, A., N. Townsend, R. Carr, and S. King, *A System for the Analysis of Jet Engines Vibration Data.* Integrated Computing-Aided Enginering, 1999. **6**: p. 53 - 65.

67. King, S.P., *Corporate Standard for Dynamic Analysis – Data Files.* Rolls-Royce Electronic Instrumentation Report Series, Issue 3, Report Number EIR 01315.

68. Schof, J., *Ten Actions When Grid Scheduling*, in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. Schopf, and J. Weglarz, Editors. 2003, Kluwer Academic Publishers. Chapter 2. p. 15 - 23.

69. *The Grid Economy Project. URL: http://www.buyya.com/ecogrid/.* 2005.

70. *The EZ-Grid Project. URL http://www2.cs.uh.edu/~ezgrid/doc.html.* 2005.

71. Sundaram, B. and B.M. Chapman. *Policy Engine: A Framework for Authorization, Accounting Policy Specification and Evaluation in Grids*. In *The Second International Workshop on Grid Computing*. 2001: Lecture Notes In Computer Science. p. 145-153.

72. Laszewski, G., I. Foster, and J. Gawor. *CoG Kits: A Bridge between Commodity Distributed Computing and High-performance Grids*. In *ACM Java Grande 2000 Conference*. 2000. San Francisco, California: ACM. p. 97 - 106.

73. Nahrstedt, K. and J.M. Smith, *The QOS Broker.* IEEE Multimedia, 1995. **2**(1): p. 53 - 67.

74. Frey, J., T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Journal Of Cluster Computing, 2002. **5**(3): p. 237-246.

75. Bester, J., I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. *GASS: A data movement and access service for wide area computing systems*. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems*. May 1999. Atlanta, Georgia: ACM Press. p. 78 - 88.

76. Aloisio, G., M. Cafaro, E. Blasi, and I. Epicoco, *The Grid Resource Broker, a Abiquitous Grid Computing Framework.* Journal of Scientific Programming, 2002. **10**(2): p. 113-120.

77. Wolski, R., N.T. Spring, and J. Hayes, *The Network WeatherSservice: a Distributed Resource Performance Forecasting Service for Metacomputing.* Journal Of Future Generations Computer Systems, 1999. **15**(5-6): p. 757-768.

78. Wolski, R., *Experiences with Predicting Resource Performance on-line in Computational Grid Settings.* ACM SIGMETRICS Performance Evaluation Review, 2003. ACM Press New York, NY, USA **30**(4): p. 41 - 49.

79. Berman, F., et al., *Adaptive Computing on the Grid Using AppLeS.* IEEE Transactions on Parallel And Distributed Systems, 2003. **14**(4): p. 369-382.

80. *AppLeS. URL: http://grail.sdsc.edu/*. 2005.

81. Berman, F. and R. Wolski. *The AppLeS Project: A Status Report*. In *NEC Research Symposium; Heterogeneous Computing and Multidisciplinary Applications*. 1997. Berlin: Siam. p. 1-20.

82. Cao, J., D. Kerbyson, and G. Nudd. *Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing*. In *International Symposium on Cluster Computing and the Grid*. 2001. Brisbane, Australia: Los Alamitos Calif.: p. 311-319.

83. Raman, R., *Matchmaking Frameworks for Distributed Resource Management.* PhD Thesis, University of Wilsconsn - Madison, Department of Computer Science, 2001.

84. Smith, W., I. Foster, and V. Taylor. *Scheduling with Advanced Reservations*. In *International Parallel and Distributed Processing Sysmposium; Fourteenth International Parallel and Distributed Processing Symposium*. 2000. Cancun, Mexico: IEEE Computer Society. p. 127-132.

85. Roy, A., *End-to-End Quality of Service for High-end Applications.* PhD Thesis, The University of Chicago, Department of Computer Science, 2001.

86. Foster, I., A. Roy, and V. Sander. *A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation*. In *The Eighth International Workshop on Quality of Service (IWQOS)*. 2000. Pittsburgh, PA: IEEE. p. 181-188.

87. Roy, A. and V. Sander, *GARA: A Uniform Quality of Service Architecture*, in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. Schopf, and J. Weglarz, Editors. 2003, Publisher Kluwer Academic Publishers. Chapter 23. p. 377 - 394.

88. Sahai., A., S. Graupner, V. Machiraju, and A. van Moorsel. *Specifying and Monitoring Guarantees in Commercial Grids through SLA*. In *Cluster Computing and The Grid; CCGrid 2003*. 2003. Tokyo: IEEE Computer Society. p. 292-301.

89. Burchard, L., M. Hovestadt, O. Kao, A. Keller, and B. Linnert. *The Virtual Resource Manager: an Architecture for SLA-aware Resource Management*.

In *IEEE International Symposium on Cluster Computing and the Grid.* 2004. p. 126-133.

90. Bal. H, et al., *Next Generation Grid(s), European Grid Research 2005-2010. Technical report, Expert Group Report for the EU, Brussel, 2003.* *ftp://ftp.cordis.lu/pub/ist/docs/ngg_eg_final.pdf*.

91. Leff, A., J.T. Rayfield, and D.M. Dias, *Service-Level Agreements and Commercial Grids.* IEEE Internet Computing, 2003. **7**: p. 44-50.

92. Liabotis. I, et al., *Self-organising Management of Grid Environments.* International Symposium on Telecommunications (IST'2003), Isfahan, Iran, August 16-18 2003.

93. Ludwig, H., A. Keller, A. Dan, R. King, and R. Franck, *A Service Level Agreement Language for Dynamic Electronic Services.* Electronic Commerce Research, 2003. **3**(1/2): p. 43-59.

94. Andrieux, A., et al., *Web Services Agreement Specification (WS-Agreement).* 2005 Global Grid Forum.

95. Li-jie, J., M. Vijay, and S. Akhil, *Analysis on Service Level Agreement of Web Services.* HP Laboratory: Technical Report, HPL-2002-180 20020710, 2002.

96. Akhil, S., M. Vijay, S. Mehmet, J. Li Jie, and C. Fabio, *Automated SLA Monitoring for Web Services.* HP laboratory: Technical Report, HPL-2002-191 20020717, 2002.

97. Goodchild, A., C. Herring, and Z. Milosevi, *Business Contracts for B2B.* Proceedings of the CAISE00 Workshop on Infrastructure for Dynamic Business-to-Business Service Outsourcing, 2003. **30**(1613-0073): p. 63 - 74.

98. Ward, C., M.J. Buco, R.N. Chang, and L.Z. Luan, *A Generic SLA Semantic Model for the Execution Management of E-business Outsourcing Contracts.* Lecture Notes In Computer Science, 2002. **2455**: p. 363-376.

99. Milosevic, Z. and R.G. Dromey. *On Expressing and Monitoring Behaviour in Contracts*. In *Enterprise Distributed Object Computing*. 2002. Lausanne, Switzerland: IEEE Computer Society. p. 3-14.

100. Marjanovic. O and M. Z. *Towards Formal Modeling of e-Contracts*. In *Enterprise Distributed Object Computing*. 2001. Seattle, WA: IEEE Computer Society. p. 59-68.

101. *Grid Resource Allocation Agreement Protocol (GRAAP) Working Group, Global Grid Forum (GGF). URL: https://forge.gridforum.org/projects/graap-wg/.* 2005.

102. Czajkowski, K., I. Foster, and C. Kesselman, *Agreement-Based Resource Management (Invited Paper).* Proceedings- IEEE, 2005. **93**(3): p. 631-643.

103. Braden, R., D. Clark, and S. Shenker, *Integrated Services in the Internet Architecture: An Overview, IETF, RFC.* 1994.

104. Blake, S., et al., *An architecture for differentiated services, IETF, RFC 2475.* 1998.

105. Shoshani, A., A. Sim, and J. Gu, *Storage Resource Managers: Essential Components for the Grid*, in *Grid Resource Management: State of the Art and Future Trends*, J. Nabrzyski, J. Schopf, and J. Weglarz, Editors. 2003., Kluwer Academic Publishers. Chapter 20.

106. Djemame, K., M. Haji, and J. Padgett. *SLA Management in a Service Oriented Architecture.* In *In Proceedings of International Conference of Computational Science (ICCSA)*. 2005. Singapore: Lecture Notes in Computer Science, 3483. p. 1282-1291.

107. Russell, D., P.M. Dew, and K. Djemame. *Service-Based Collaborative Workflow for DAME*. In *Proceeding 2005 IEEE International Conference on Services Computing*. 2005. Orlando, Florida, USA. p. 139-146.

108. *MySQL. URL: http://www.mysql.com/.* 2005.

109. Barker, K., B. Porter, and P. Clark. *A Library of Generic Concepts for Composing Knowledge Bases*. In *International Conference on Knowledge Capture, ACM Press New York, NY, USA*. 2001. p. 14 - 21.

110. *Grid Particle Physics. URL:http://www.gridpp.ac.uk/lhcb/.* 2005.

111. *Grid Economic Services Architecture Working Group. URL: https://forge.gridforum.org/projects/gesa-wg/.* 2005.

112. Rajkumar, B., G. Jonathan, and A. David, *An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications.* he Second Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing, 2000. Kluwer Academic Press.

113. Rajkumar, B., A. David, and G. Jonathan, *An Economy Driven Resource Management Architecture for Global Computational Power Grids.* 2000.

The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000).

114. Buyya, R., D. Abramson, and J. Giddy, *A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker.* Future Generation Computer Systems, 2002. Elsevier Science Publishers B. V. Amsterdam, The Netherlands **18**(8): p. 1061 - 1074.

115. MacLaren, J., *Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) report on Advance Reservations:State of the Art. URL:http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html. June 2003.*

116. Atzeni, P., S. Ceri, S. Parabosci, and R. Torlone, *Database Systems: Concepts, Languages, Architectures*. 1999: Mc Graw Hill.

117. Connolly, T. and C. Begg, *Database Systems, A Practical Approach to Desing, Implementation and Management*. Fourth Edition ed. 2005: Addison Wesley.

118. H. N. Lim Choi Keung., J. R. D. Dyson., S. A. Jarvis., and G.R.Nudd. *The Globus Monitoring and Discovery Service (MDS-2): A Performance Analysis*. In *In Proceedings of the 19th UK Performance Engineering Workshop (UKPEW'2003)*. 2003. Warwick, UK. p. 103-116.

119. Zhang, X., J. Freschl, and J. Schopf. *A Performance Study of Monitoring and Information Services for Distributed Systems*. In *The Twelfth IEEE International Symposium on International Symposium on High-Performance Distributed Computing*. 2003. p. 270-281.

120. Brodlie, K., et al. *Visualization in Grid Computing Environments*. In *Proceedings of IEEE Visualization, IEE Computer Society Press.* 2004. p. 155-162.

121. *Modelling and Simulation for e-social Science (MoSeS), URL:http://www.ncess.ac.uk/research/nodes/#moses/.* 2005.

122. Kleinrock, L. and R. Gail, *Queueing Systems: Problems and Solutions*. 1996: Wiley.

123. Banks, J., J.S. Carson, B.L. Nelson, and D.M. Nicol, *Discrete Event System Simulation*, ed. W.J. Fabrycky and J.H. Mize. 2001: Prentice Hall.

124. *Wikipedia Encyclopaedia. URL: http://en.wikipedia.org/wiki/Mathematical_model/.* 2005.

125. Buyya, R. and M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing.* The Journal of Concurrency and Computation: Practice and Experience (CCPE), 2002. **14**(13-15): p. 1175-1220.

126. Murata, T., *Petri Nets: Properties, Analysis and Applications.* Proceedings of the IEEE, 1989. **77**(4): p. 541-580.

127. Bagrodia, R., et al., *Parsec: A Parallel Simulation Environment for Complex Systems.* 1998. IEEE Computer Society **31**(10): p. 77-78.

128. *JiST. URL:http://jist.ece.cornell.edu/.* 2005.

129. *SimJava. URL: http://javasim.ncl.ac.uk/.* 2005.

130. *DSOL. URL:http://www.simulation.tudelft.nl/dsol/dsol/index.html/.* 2005.

131. *OMNeT++. URL: http://www.omnetpp.org/.* 2005.

132. Song, H., et al. *The MicroGrid: a Scientific Tool for Modeling Computational Grids*. In *IEEE Supercomputing(SC 2000)*. 2000. p. 53- 64.

133. Casanova, H. *Simgrid: a Toolkit for the Simulation of Application Scheduling*. In *The First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001)*. 2001. p. 430-437.

134. Law, A.M. and W.D. Kelton, *Simulation Modelling and Analysis*. 2000: McGraw-Hill.

135. Gourlay, I., M. Haji, K. Djemame, and P. Dew. *Performance Evaluation of a SNAP-based Grid Resource Broker*. In *In Proceedings of FORTE'2004 Workshops (1st European Performance Engineering Workshop)*. 2004. Toledo, Spain: M. Nunez, A. Maamar, F.L. Pelayo, K. Pousttchi and F. Rubio (Eds.). p. 220-232.

136. Yeong, W., T. Howes, and S. Kille, *Lightweight Directory Access Protocol, IETF RFC 177*. 1995.

137. *Netscape Directory Software Development Kit. URL:http://www.redhat.com/software/rha/netscape/.* 2005.