A Framework for Evolving Grid Computing Systems

PhD Thesis

Mai Ahmad AL-Fawair

This thesis is submitted in partial fulfilment of the

requirements for the Doctor of Philosophy

Software Technology Research Laboratory

Faculty of Technology

De Montfort University

United Kingdom, England

2009

Abstract

Grid computing was born in the 1990s, when researchers were looking for a way to share expensive computing resources and experiment equipment. Grid computing is becoming increasingly popular because it promotes the sharing of distributed resources that may be heterogeneous in nature, and it enables scientists and engineering professionals to solve large scale computing problems. In reality, there are already huge numbers of grid computing facilities distributed around the world, each one having been created to serve a particular group of scientists such as weather forecasters, or a group of users such as stock markets.

However, the need to extend the functionalities of current grid systems lends itself to the consideration of grid evolution. This allows the combination of many disjunct grids into a single powerful grid that can operate as one vast computational resource, as well as for grid environments to be flexible, to be able to change and to evolve. The rationale for grid evolution is the current rapid and increasing advances in both software and hardware.

Evolution means adding or removing capabilities. This research defines grid evo-

lution as adding new functions and/or equipment and removing unusable resources that affect the performance of some nodes. This thesis produces a new technique for grid evolution, allowing it to be seamless and to operate at run time. Within grid computing, evolution is an integration of software and hardware and can be of two distinct types, external and internal. Internal evolution occurs inside the grid boundary by migrating special resources such as application software from node to node inside the grid. While external evolution occurs between grids.

This thesis develops a framework for grid evolution that insulates users from the complexities of grids. This framework has at its core a resource broker together with a grid monitor to cope with internal and external evolution, advance reservation, fault tolerance, the monitoring of the grid environment, increased resource utilisation and the high availability of grid resources.

The starting point for the present framework of grid evolution is when the grid receives a job whose requirements do not exist on the required node which triggers grid evolution. If the grid has all the requirements scattered across its nodes, internal evolution enabling the grid to migrate the required resources to the required node in order to satisfy job requirements ensues, but if the grid does not have these resources, external evolution enables the grid either to collect them from other grids (permanent evolution) or to send the job to other grids for execution (just in time) evolution.

Finally a simulation tool called (EVOSim) has been designed, developed and

tested. It is written in Oracle 10g and has been used for the creation of four grids, each of which has a different setup including different nodes, application software, data and polices. Experiments were done by submitting jobs to the grid at run time, and then comparing the results and analysing the performance of those grids that use the approach of evolution with those that do not. The results of these experiments have demonstrated that these features significantly improve the performance of grid environments and provide excellent scheduling results, with a decreasing number of rejected jobs.

Declaration

The thesis presented here is mine and original. It is submitted for the degree of Doctor of Philosophy at the Software Technology Research Laboratory (STRL), Faculty of Technology, De Montfort University, United Kingdom.

Acknowledgement

I should like first to thank my God Allah, without whom I could not even have contemplated all that was involved in this course of study.

My heartfelt thanks to my supervisory team, Professor Hussein Zedan and Dr. Omar Aldabbas, for their help and support during the period of my study. Professor Zedan was so much more than my supervisor. His openness to new ideas, his extensive knowledge and his willingness to help me at every turn made him the best advisor and guide I could hope for. But my gratitude is due to him for another reason: he proved to be a father figure to me, helping me to adjust to life in the UK. To you, Professor Zedan, your "daughter" remains forever grateful.

My husband brought me to this country and helped me enormously, both in my studies and in my personal life. I left my own family in Jordan; he proved to fill the roles of mother, father, brother and sister as well as husband, helpmate and friend. My life is immeasurably richer in every way because of him, and without him my experience in the UK would have ended before it began.

There is a wonderful person whose incisive thinking and decisiveness have in-

spired me as long as I can remember, and the veracity of whose opinions has always been borne out in practice: my father. For your unvarying support from my infancy and for your constant stress on the importance of education (to say nothing of your indoctrination of me by calling me "Doctor" since I was a child), I will forever be in your debt, because that is why I am what I am today.

And where would I be without my mother? Your loving care and nurture from the day I was born, as well as your teaching, are responsible for bringing me to this point, as also is your looking after my children during the writing of my PhD thesis, for which I will be forever grateful.

To my brothers, Dr. Mousa and your family and Eng. Malek for your moral support and your duaa, to Ma'en for being as much a friend and confidant as a brother, and to Mohammed and my sister Maimonah for your kindness in looking after my children while I was in the UK (and especially to Mohammed for constantly carrying my son Ameen and to Maimonah for teaching him certain other essentials!): my heartfelt thanks and love to all of you.

My father-in-law, my mother-in-law and all of Omar's brothers and sisters and their families: your constant encouragement, both at home and here in the UK, made my journey towards my goal of a PhD much easier.

And my dear child children Ameen and Rahma: how can I apologise to you for leaving you for three months while I was writing up my thesis? I hope my PhD will serve as an inspiration for you when you grow up, so that you can do better.

Thank you to all the staff in the STRL for your support, and especially to Dr. Antonio Cau and Dr. Helge Janicke for reading my thesis, and to the departmental secretaries.

To all my friends, my deepest thanks for your help and support, and especially for your encouragement during my study.

Publications

1. M. Alfawair, O. Aldabbas, P. Bartels and H. Zedan, "Grid Evolution", *IEEE International Conference on Computer Engineering & Systems (ICCES'07)*, Cairo, November 2007.

2. O. Aldabbas, M. Alfawair, H. Zedan and A. Cau "Temporal Dimension for Job Submission Description Language", *The 7th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS '08)*, Cambridge University, Cambridge, UK, February 2008, pp. 80-88.

List of Abbreviations

API	Application Program Interface
CoD	Code-on Demand
CPU	Central Processing Unit
DSI	Data Storage Interface
DFS	Distributed File System
DNS	Domain Name System
XDR	eXternal Data Representation
FAFNER	Factoring via Network-Enabled Recursion
FTP	File Transfer Protocol
GPFS	General Parallel File System
GSSAPI	Generic Security Services Application Program Interface
GASS	Global Access Secondary Storage
GIIS	Grid Index Information Service

GRAMP	Grid Resource Access and Management protocol
GRAM	Grid Resource Allocation Manager
GRIP	Grid Resource Information Protocol
GRIS	Grid Resource Information Service
GSI	Grid Security Infrastructure
GridFTP	Grid File Transfer Protocol
ICMP	Internet Control Message Protocol
IB	InfiniBand
IETF	Internet Engineering Task Force
GIP	GlobusInformation Provider
IT	Information Technology
I-WAY	Information Wide Area Year
JSDL	Job Submission Description Language
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LOSF	Lots of Small Files
MPPs	Massively Parallel Processors
MDS	Monitoring and Discovery Service
NFS	Network File System
NWS	Network Weather Service

NWICG	Northwest Indiana Computational Grid
OGF	Open Grid Forum
OGSA	Open Grid Service Architecture
OGSI	Open Grid Services Infrastructure
OS	Operating System
PC	Personal Computer
PDAs	Personal Digital Assistants
PKI	Public Key Infrastructure
QoS	Quality of Service
RAM	Random Access Memory
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SIEREN	Sharing Infrastructure and REsources iN Europe
SWAP	Simple Workflow Access Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Networks
WAN	Wide Area Network

Contents

Abstract i

Declaration iv

Acknowledgement v

Publications viii

List of Abbreviations ix

1 Introduction 1

1.1 Research Motivation 3
1.2 Research Methodology 6
1.3 Research Question 7
1.4 Contributions 8
1.5 Success Criteria 9
1.6 Thesis Outline 10

2 Background and Related Work 12

2.1 Introduction 13
2.2 Grid Computing 14
2.2.1 Features and Objectives of Grid Computing 17
2.2.2 Characteristics of Grid Environments 20
2.2.3 Grid Architecture 21
2.2.4 Grid Types 24
2.3 History of Grid Evolution 27
2.3.1 Grid Generations 27
2.3.1.1 First Generation 28
2.3.1.2 Second Generation 29

2.3.1.3 Third Generation 29
2.3.2 Stages of Development 30
2.3.2.1 Cluster 30
2.3.2.2 Intra-grid 31
2.3.2.3 Extra-grid 32
2.3.2.4 Inter-grid 32
2.4 Grid Technology Infrastructure 33
2.4.1 Globus Alliance 33
2.4.1.1 Security 34
2.4.1.2 Execution Management 35
2.4.1.3 Data Management 35
2.4.1.4 Information Services 35
2.4.2 Open Grid Forum 37
2.4.2.1 Job Submission Description Language 37
2.4.2.2 GridFTP 38
2.5 Current Efforts in Grid Computing 45
2.6 Resource Broker (Scheduler) 46
2.6.1 Condor G 48
2.6.2 Nimrod/G 50
2.7 Migration 51
2.7.1 Weak Migration 52
2.7.2 Strong Migration 54
2.8 Summary 54

3 Grid Evolution **55**
3.1 Introduction 55
3.2 What Does Evolution Mean? 57
3.3 Importance of Grid Evolution 57
3.4 Internal Evolution 59
3.5 External Evolution 61
3.5.1 Permanent Evolution 62
3.5.2 Just-in-Time Evolution 65
3.6 How Evolution Occurs? 69
3.7 Summary 71

4 Computational Model for Evolvable Grid **72**
4.1 Introduction . . . 73
4.2 Objects . . . 74
4.2.1 Grid . . . 74
4.2.2 Node . . . 74
4.2.3 Application Software . . . 77
4.2.4 Data . . . 79
4.2.5 Grid Job . . . 80
4.3 Models . . . 81
4.3.1 Timing Model . . . 82
4.3.2 Communication Model . . . 82
4.3.3 Termination Model . . . 84
4.3.4 Failure Model . . . 85
4.3.5 Migration Model . . . 88
4.4 Strategies . . . 89
4.5 Summary . . . 91

5 Evolution-Based Grid Design **92**
5.1 Introduction . . . 93
5.2 Structure of Evolution-Based Grid Design . . . 93
5.3 User . . . 95
5.4 Portal . . . 96
5.5 Evolution-Based Resource Broker . . . 96
5.5.1 Discoverer . . . 98
5.5.2 Communicator . . . 103
5.5.3 Evolution Maker . . . 104
5.5.4 Failure Recoverer . . . 118
5.6 Information Service . . . 120
5.7 Node . . . 120
5.8 Grids . . . 122
5.9 Monitor . . . 122
5.9.1 Internal Monitoring . . . 123
5.9.2 External Monitoring . . . 126
5.10 Summary . . . 128

6 EVOSim: A Simulation Tool for Grid Evolution 129
6.1 Introduction . . . 129
6.2 EVOSim Analysis and Design . . . 131
6.2.1 EVOSim Components . . . 131
6.2.1.1 Entities . . . 131
6.2.1.2 The System . . . 135
6.2.2 EvoSim Use Cases . . . 135
6.2.3 Sequence Diagram . . . 143
6.3 EVOSim Implementation . . . 147
6.4 Simulation Validation . . . 152
6.5 Summary . . . 157

7 Evaluation 158
7.1 Introduction . . . 158
7.2 Experimental Environment . . . 159
7.3 Grid Experiments without Evolution . . . 161
7.4 Grid Experiments with Internal Evolution . . . 163
7.5 Grid Experiments with Permanent Evolution . . . 166
7.6 Grid Experiments with Just-in-Time Evolution . . . 168
7.7 Summary . . . 171

8 Conclusions and Future Work 172
8.1 Summary . . . 172
8.2 Contributions . . . 175
8.3 Future Work . . . 177

References 178

List of Figures

2.1 Layers of Grid Architecture [37] . . . 22
2.2 The Stages of Grid Development . . . 30
2.3 Globus Toolkit Architecture . . . 34
2.4 MDS Architecture . . . 36
2.5 GridFTP with NaradaBrokering [53] . . . 45
2.6 Condor-G Mechanism for Executing a Job on Globus Managed Resources . . . 49
2.7 Nimrod/G Architecture . . . 50

3.1 External Evolution . . . 62

5.1 The Evolution-Based Grid Design . . . 95
5.2 Resource Broker Design . . . 99
5.3 Resource Discoverer Algorithm . . . 100
5.4 Evolution Maker Algorithm . . . 105
5.5 Internal Evolution Algorithm . . . 106
5.6 Permanent Evolution Algorithm . . . 109
5.7 Just-in-Time Evolution Algorithm . . . 114
5.8 Failure Recoverer Algorithm . . . 119
5.9 Internal Monitoring Algorithm . . . 125
5.10 External Monitoring Algorithm . . . 127

6.1 EVOSim Class Diagram . . . 132
6.2 EVOSim Use Case Diagram . . . 137
6.3 EVOSim Developer Use Case Diagram . . . 138
6.4 Sequence Diagram 1 . . . 144
6.5 Sequence Diagram 2 . . . 146
6.6 Grid Configuration . . . 148

6.7 Node Configuration 149
6.8 Node Specification Configuration 149
6.9 Grid Portal 151

7.1 Job Acceptance 160
7.2 Resources Employment without Evolution 162
7.3 Rejected Jobs with Internal Evolution 164
7.4 Resources Employment with Internal Evolution 165
7.5 CPU Utilisation 166
7.6 Rejected Jobs with Permanent Evolution 167
7.7 Resources Employment with Permanent Evolution 168
7.8 Rejected Jobs with Just in Time Evolution 169
7.9 Rejected Jobs with both External and Internal Evolution 170
7.10 Executed Jobs Comparison 170

List of Tables

2.1 Globus Services 33

6.1 EVO Grid Nodes Specification 153
6.2 Grid1 Nodes Specification 153
6.3 Grid2 Nodes Specification 154
6.4 Grid3 Nodes Specification 154
6.5 Jobs Requirement 154

Chapter 1

Introduction

Objectives

- To describe the research motivation.
- To present the major contribution of this research.
- To explain the research questions.
- To introduce the thesis.

According to Moore's law, individual computers double in processing power every 18 months [73]. According to Storage Law, disk storage capacity doubles every 12 months [73]. According to Gilder's Law, network bandwidth doubles every 9 months [84]. So clearly network bandwidth grows faster than processing power. This exponential growth profoundly changes the landscape of information technology. High speed access to network information becomes the dominant feature of computing in the future.

The term "grid" is taken from the phrase "electrical power grid", which provides universal access to power without regard for where and how the power was produced. In the same way, a computational grid should provide pervasive access to high-end computing resources, without consideration for where and how the jobs are executed. The infrastructure of an electrical power grid makes it possible for a person to access electricity by simply plugging into a wall socket, without being concerned with how and where the electricity used is generated. The concept behind grid computing is similar to that of the power grid [48, 82].

Grid computing has emerged as an ambitious research area that addresses the problem of efficiently using multi-institutional pools of resources. Its goal is to allow coordinated and collaborative resource sharing and problem solving across several institutions so as to solve large scale scientific problems that could not be easily solved within the boundaries of a single institution. The concept of a computational grid first appeared in the mid 1990s, having been proposed as an infrastructure for advanced science and engineering. This concept has evolved extensively since then and has encompassed a wide range of applications in both the scientific and commercial fields [34].

The main goals of grid computing are [34]:

- the provision of a framework for utilising unused computational power as well as unused storage capacity.

- the simplification of collaboration between different organisations by the pro-

vision of direct access to computers, software and storage.

- the provision of access to resources that cannot be accessed locally (known as "remote resources").

- the faster execution of jobs due to their parallel execution on multiple resources (Multi-site computing/Co-allocation).

1.1 Research Motivation

Computing environments have evolved from single-user environments through Massively Parallel Processors (MPPs), clusters of workstations and distributed systems to (most recently) grid computing systems. Every transition has been a revolution, allowing scientists and engineers to solve complex problems and sophisticated applications previously incapable of solving. However every transition has brought new challenges and problems in its wake, as well as the need for technical innovation. The evolution of computing systems has led to the current situation in which millions of machines are interconnected via the Internet with various hardware and software configurations, capabilities, connection topologies, access policies and so forth. The formidable mix of hardware and software resources on the Internet has fuelled researchers' interest in investigating novel ways to exploit this abundant pool of resources in an economical and efficient manner, as well as in aggregating these distributed resources so as to benefit a single application.

Grid computing is diverse and heterogeneous in nature, spanning multiple domains whose resources are not owned or managed by a single administrator. This presents grid resource management [23] with many challenges such as site autonomy, heterogeneous substrate and policy extensibility. The Globus [33, 6] middleware toolkit addresses these issues by providing services to assist users in the utilisation of grid resources. Users are still exposed to the complexities of grid middleware, however, and there is a substantial burden imposed on them in that they must have extensive knowledge of the various grid middleware components in order to be able to utilise grid resources. Such knowledge ranges from querying information providers, selecting suitable resources for the user's job, forming the appropriate JSDL (Job Submission Description Language), submitting the user's job to the resources and initiating job execution.

A central component in grid computing is resource brokering, which insulates the user from the complexities of grid middleware 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 in order to find a single resource with which to execute the job, or scheduling a single job to use multiple resources across one or more sites. As discussed in [51], resource brokering 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, a user-centric broker such as Nimrod-G [14] adheres to the

user's requirements. A key goal of grid computing is to deliver utility of computation as defined by these requirements. Grid resource brokers are thus focused on providing user-centric services. The grid must also cater for multiple users from different sites, who may potentially be interested in the same resource, without knowledge of each other's existence or interests [22, 35].

To maximise the benefits of grid computing, it is essential to be able to discover the resources available on the grid and to map the job to the most suitable resource. In computational grids, the problem of resource discovery and matching the job to suitable resources is a real challenge because of the large number of resources, their heterogeneity, availability and the variety of resource attributes such as CPU load and disk space.

In reality, there are already hundreds of grids around the world, each one created to serve a specific group of researchers or users. The grid computing dream began with talk of creating an all-powerful "grid": one grid composed of many smaller grids joined together, forming a global network of computers that can operate as one vast computational resource. Across the world, researchers and software engineers are working to bring "the grid" closer to achieving the dream [1].

This research focuses on the problem of the grid evolution and how grids can communicate with each other and exchange resources in order to satisfy job requirements and therefore reduced the number of rejection jobs. Grid computing is still a work-in-progress; not much research has concentrated on grid evolution.

1.2 Research Methodology

The research method used in this research is a typical scientific research technique [50], which include the following phases:

1. Literature Review.

   The research literature phase articulated the research question by gathering the information and then studying and observing this information.

2. Modeling.

   Modeling phase is to analyse and study the phenomena that expressed in the research question. So, the model has been built to understand and analyse the phenomena. This model includes the computational model and the grid design.

3. Algorithmic Development.

   In this phase, the novel technique with its algorithm has been established in order to handle and cope with the various issues involved.

4. Prototyping and Evaluation.

   In this phase, the prototype based on the model has been built. The experiments also have been conducted and the results have been tested and evaluated.

1.3 Research Question

The grid is not a silver bullet that can take any application and run it 1,000 times faster without the need for buying any more machines or software. Not every application is suitable or enabled for running on a grid. Grids work better in intensive computing environments.

The research question therefore is

- ***How the grid can evolve in order to meet jobs requirement through obtaining the required resources or sending jobs to other grid to execute?***

  This question spawns the following ones:

  - Transparency and evolution
    - How can the user be insulated from the complexities of middleware, and how can job submission to the appropriate resources be guaranteed?
    - How can the implementation of the construction of one grid coprises of many grid be simplified?
    - How can resource usage using evolution be maximised?
  - Fault Tolerance
    - How can the grid detect failures inside or in grids attached to each other?

* How can the job complete its execution despite hardware or software failures?

1.4 Contributions

The major contribution of this research can be summarised as follows:

- The development of the evolution framework that allows grids to communicate with each other in order to exchange application software, data and even jobs. This framework enhance performance and abilities of the evolvable grid's , as well as the execution of a greater number of jobs and thus the ability to serve more users.

- The design and development of a mixture of user-centric and system-centric resource broker design that uses the evolution framework. This provides a modular technique for handling user requirements and system throughput, as well as the discovery, selection and submission of jobs. The broker insulates the user from the complexities of grid middleware and ensures that jobs are submitted to the appropriate resources for execution.

- The design of a new grid design based on a novel computational model for evolution. This will allow grid evolution to occur seamlessly and at run time when needed.

- The exploration of a new technique for grid monitoring inside and outside

the grid environment, because fault tolerance is an essential characteristic and a necessary function of such environments in order to avoid the loss of computational time. This technique allows grids to detect and handle, and thereby recover from, failure.

- The building of a simulation environment for the evaluation of the framework. The performance of the grid using the approach of evolution then compared and analysed with that of a grid without using this approach. The results of these experiments have demonstrated that these features significantly improve the performance of grid environments and provide excellent scheduling results with a decrease in the number of rejected jobs.

1.5 Success Criteria

The measure of success is that both the model and their supporting algorithm indeed resolve the research question and demonstrate it by the experiments through the prototype. The prototype demonstrates that the output results match the results obtained by manual calculation.

The experiments will perform on the prototype by configuring four grids, each one has its own resources and then applying the evolution by enabling the application software to migrate between those grids after checking policy, CPU speed, disk space and operating system. Jobs also can send to one of those grids to execute. Therefore the results will show the advantages of the evolution and will appear in

the evaluation chapter.

1.6 Thesis Outline

The previous sections have introduced the research presented in this thesis. A summary of the work organise as follows.

Chapter 2 surveys the background and related work on all major research issues covered in this thesis. Firstly, it surveys the architecture, types, characteristics and features of grid computing. The history of grid evolution and generation is then outlined, a review of grid resource brokers is provided and overview of migration given before major projects that utilise the grid are listed. GridFTP and its features are surveyed, the current efforts in grid computing presented and the software evolution is finally summarised.

Chapter 3 describe the techniques to evolution which have been classified as external and internal. These are defined and their performance in grid environments described.

Chapter 4 describes the behaviour of the system, which consists of computational model objects and techniques, by giving the properties of its components' behaviours and their interactions.

Chapter 5 describes the system design, its structure and its components with their functions. This chapter shows how this design is designed to support both external and internal grid evolution. It also describes how the resource broker insulates

the user from grid complexity.

Chapter 6 describes EvoSim, the simulation created to simulate these ideas, as well as compare the performance of the grid environment.

Chapter 7 evaluates the results using the simulation to demonstrate the notions of grid evolution, and describe the results of this evolution.

Chapter 8 concludes the thesis and outlines possibilities for future work.

Chapter 2

Background and Related Work

Objectives

- To review the definition, architecture, types and aims of grid computing.
- To describe grid evolution history and generation.
- To describe grid technology infrastructure.
- To review software evolution concepts.
- To review existing resource brokers (schedulers).
- To review the present state of migration in grid computing.
- To discuss current efforts in grid computing.

2.1 Introduction

As a result of advances in communication technology and the internet, different networks of workstations with various capabilities were connected to each other to form distributed computing systems. These systems possessed both hardware and software capabilities derived from disparate sites, and were often heterogeneous in terms of hardware configuration, operating system and network connection. By using these distributed systems, users were able to solve problems requiring resources that were often not available at their own sites. The smooth functioning and maintenance of these distributed computing systems necessitated the provision of system support tools that provided a range of services such as user interfaces, programming environments, programming language tools, operating system services (including file services), storage services, process spawning, process management, security infrastructures and - most important of all - allocation or scheduling of resources to jobs. The extent and types of these services provided by a distributed system defines the characteristics of that system. A new paradigm called grid computing, that leveraged the vast set of resources available in the distributed computing systems, was developed.

Grid computing is the latest computing environment and has been gaining popularity for the past few years. It can be considered as an extension of distributed computing systems, one in which both the number and heterogeneity of the systems are much greater than usual. It promotes the sharing of distributed resources that

may be heterogeneous in nature, and enables scientists and engineering professionals to solve large scale computing problems. With grid computing, businesses can efficiently utilise computing and data resources and combine them for large capacity workloads. The primary benefit of grid computing is the ability to coordinate and share resources [15, 59]. Many challenges face grid computing; one of them is to find all the required resources for a grid job, resources that may reside in other grid sites. This research proposes the evolution as a solution for this problem.

This chapter surveys the architecture, types, characteristics and features of grid computing in Section 2.2. The history of grid evolution and its generation are then outlined in Section 2.3. Grid technology infrastructure is explained in Section 2.4, a review of grid resource brokers is provided in Section 2.6, current efforts in grid computing in Section 2.5, and a survey of migration is finally summarised in Section 2.7.

2.2 Grid Computing

Distributed systems have been developed since the 1970s. The term describes "a programming model in which processing occurs in many different places (or nodes) around a network" [10]. It can also be defined as a collection of processes, computers or nodes that communicate by passing messages across a network. The many advantages of distributed systems include information exchange, resource sharing, increased reliability and increased performance. The scope of distributed applica-

tions has recently increased dramatically due to the internet. Distributed systems have many unique characteristics, among them variable configuration and partial failure. Traditionally distributed computing has been defined as systems with a fixed number of nodes; these are called static distributed systems. With the advent of peer-to-peer computing and grid computing systems, the number of nodes in distributed systems is constantly changing, and such systems are thus known as dynamic distributed systems. In these systems, processes can join and leave the ongoing computation at any time. Consequently, the set of processes in the system may change from one moment to another.

Grid computing has emerged as a new paradigm for distributed computing, having developed as a mechanism for allowing collections of connected computer systems to form large-scale data and computing networks. It promotes the sharing of distributed resources that may be heterogeneous in nature, so as to enable different application domains including science, industry, engineering, finance and even government to solve large-scale computing problems [34].

The term 'grid' takes its name from the phrase 'power grid', because of its similarity to the concept of a grid supplying electrical power. The power grid is responsible for supplying consistent, dependable and transparent access to an electrical supply. Likewise, grid computing is intended to provide an equally consistent, dependable, and transparent collection of computing resources. The infrastructure of an electrical power grid makes it possible for a person to access electricity by sim-

ply plugging into a wall socket, without being concerned with how and where the electricity being used is generated [48, 82]. In the mid-1990s, Ian Foster and Carl Kesselman attempted a definition of grid computing. They wrote: "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities" [48].

In 2000, Foster, Kesselman and Tuecker redefined grid computing to address social and policy issues: "The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organisation"[37].

Ian Foster [32] gives a simple three-point checklist to define a grid:

1. It coordinates resources that are not subject to centralised control. A grid integrates and coordinates resources and users that exist within different control domains. The grid paradigm enables the coordination and sharing of a large number of geographically dispersed heterogeneous resources.

2. It uses standard, open, general-purpose protocols and interfaces. A grid is built from multi-purpose protocols and interfaces that can be shown in its

architecture.

3. It delivers nontrivial qualities of service. A grid allows its ingredient resources to be used in a coordinated style to deliver various qualities of service such as response time and security.

Grid computing has been defined as *a collection of nodes connected via a network. These nodes may be identified as computational, storage and network resources. They also share services, computing power and other resources such as disk storage database and software applications. They are under the control of different administrative domains with different levels of security. Grid computing is a way to share computing resources.*

2.2.1 Features and Objectives of Grid Computing

Grid computing aims to achieve the following goals [48, 34]:

- The sharing of distributed and heterogeneous computing resources belonging to different organisations.

  Grid computing is the sharing, selection and aggregation of a group of resources such as supercomputers, mainframes, storage systems, data sources and management systems that behave like networks of computation [55, 3]. It promotes the sharing of distributed resources that may be heterogeneous in nature. The primary benefit of grid computing is the ability to coordinate and

share distributed and heterogeneous resources such as Sharing Infrastructure and REsources iN Europe (SIEREN) [46].

SIEREN is a cooperative association between twelve European countries whose purpose is to share their infrastructure and resources.

- The exploitation of underutilised resources.

  In most organisations and companies there are huge numbers of underutilised computing resources. Most of these resources are only busy less than 5% of the time. In most organisations these resources are relatively idle. Grid computing is designed to exploit these underutilised resources and increase the efficiency of resource usage. Users can also rent the resources that reside on the grid for executing their computationally intensive applications, instead of purchasing their own dedicated (and expensive) resources.

- The enablement and simplification of collaboration among different organisations.

  Another capability of grid computing is the provision of an environment for collaboration between organisations. Grid computing enables very heterogeneous and distributed systems to work together, thereby simplifying collaboration between different organisations by providing direct access to computers, software and data storage.

- The provision of a single login service with secure access to grid resources while

protecting security for both users and remote sites.

Grids provide a single login service to all users over all distributed resources using grid authentication mechanisms. They also provide secure access to any information anywhere over any type of network. This is achieved by providing access control mechanisms that govern these resources.

- The provision of resource management, information services, monitoring and secure data transportation.

  The shared resources and networks involved in grid computing are difficult to manage and monitor, but grid computing is able to meet these challenges because of its architecture and protocols.

- The solution of large scale problems.

  Grids are designed to exploit underutilised resources, meaning that they can employ a large number of them to solve a large scale problem; it promising solution for problems like storage and processing of massive amounts of data that even mainframe computers can not handle such as weather forecasting. These resources may be high capability devices such as high capacity disk storage and high performance computing.

- The provision of fast results, delivered more efficiently.

  Grid computing obtains results quickly and more efficiently, because it enables parallel processing; it may also have high capability devices. With grid

computing, businesses can efficiently utilise computing and data resources and combine them for large capacity workloads.

2.2.2 Characteristics of Grid Environments

Grid computing has emerged as a new paradigm for distributed computing. Traditionally distributed computing has been defined as systems with a fixed number of nodes. With the advent of grid computing, the number of nodes in distributed systems is constantly changing, and such system are thus known as dynamic distributed systems. It has the following characteristics [43]:

- Heterogeneous resources.

  Resources have highly heterogeneous configurations: there are different architectural platforms, different CPU speeds, memory hierarchy layouts and capacities, disk space and software configurations.

- Dynamic resource availability.

  The absence of tight controls on volunteer computing-based grid environments can cause a resource to withdraw from the computation at any time based on the wishes of its owner. Resources might non-deterministically join or leave the available set of resources which mean that these resources can join and leave the grid at any time. This renders the grid a best-effort computation medium, by analogy to the internet's best-effort communication medium.

- Dynamic resource load.

  Applications run in a shared environment, where external jobs compete for the same resources. The fluctuations of the machine loads are therefore expected to be high. A machine might be lightly loaded now and heavily loaded the next minute.

2.2.3 Grid Architecture

Traditional distributed technologies do not support an integrated approach to the wide variety of required services and resources, and they lack the flexibility and control needed to enable the type of resource sharing necessary. There is a need to define a grid software infrastructure to support the heterogeneous aspects of the grid. In [36, 37, 41], the grid strongly emphasises interoperability, as it is vital to ensure that virtual organisation participants can dynamically share heterogeneous resources. The grid infrastructure is based on a standard open architecture which facilitates extensibility, interoperability, portability and code sharing. This architecture organises components into layers, as shown in Fig 2.1.

Components within each layer share common characteristics, but can build on the capabilities and behaviours of any lower layer.

1. Fabric Layer

   The fabric layer comprises the resources in the grid. This resource can be either a logical (such as a distributed file system, computer cluster or distributed

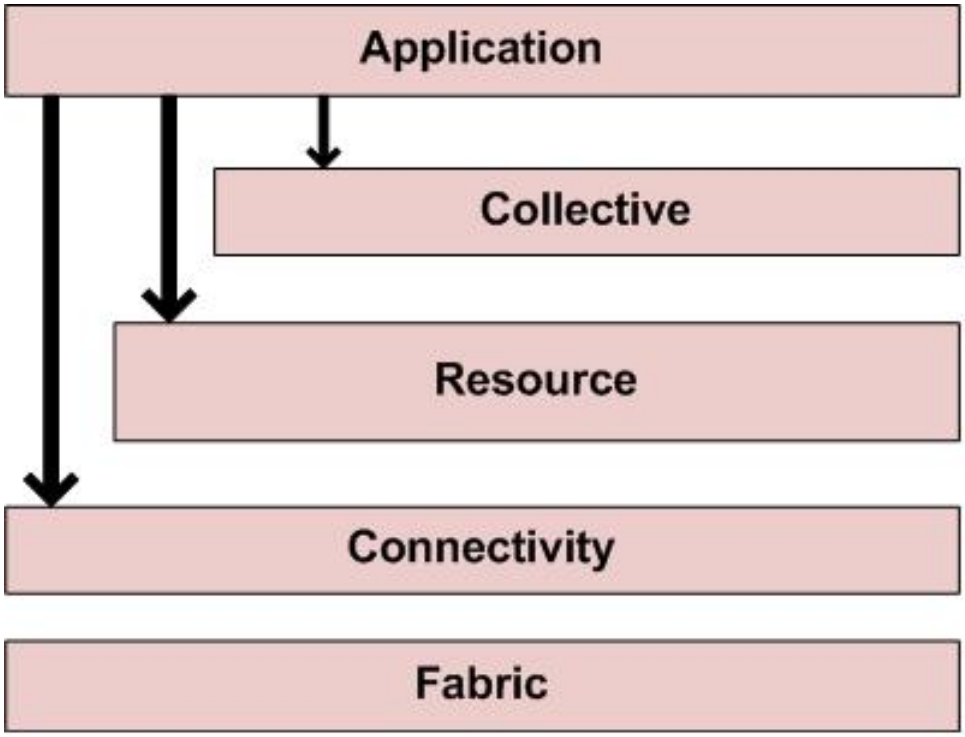

Figure 2.1: Layers of Grid Architecture [37]

computer pool) or a physical resource (such as a computational resource, storage system, catalogue, network resource or sensor). This layer provides the lowest level of access to actual native resources, and implements the low-level mechanisms that allow those resources to be accessed and used. More specifically, those mechanisms must include at least state enquiry and resource management mechanisms, each of which must be implemented for a large variety of local systems.

2. Connectivity layer

   The connectivity layer provides the core communication and authentication protocols required for grid-specific network transactions. These protocols pro-

vide cryptographically secure mechanisms by which to verify the identified grid users and resources. Many communication protocols in the connectivity layer are drawn from TCP/IP protocols stack such as IP [63], ICMP [62], TCP[64], UDP [61] and DNS [57].

3. Resource Layer

   This layer builds on the connectivity layer in order to implement protocols that enable the use and sharing of individual resources such as the Grid Resource Access and Management protocol (GRAMP) used to allocate and monitor resources.

   More specifically, two fundamental components of this layer are information protocols, for querying the state of a resource by calling fabric layer functions to control and access resources, and management protocols, used to negotiate access to a shared resource.

4. Collective Layer

   This layer provides protocols such as the Grid Resource Information Protocol (GRIP) [37] for interacting across collections of resources. In other words, it focuses on the coordination of multiple resources. It includes directory, coallocation, scheduling, brokerage, monitoring and diagnostics, data replication, software discovery, community accounting and payment services.

5. Application Layer

This is the final layer in grid architecture, and contains the user applications that operate in a grid environment. It includes languages and frameworks. These frameworks may themselves define protocols such as Simple Workflow Access Protocol (SWAP) [78], services, and/or an Application Program Interface (API).

2.2.4 Grid Types

From the perspective of application, grids [49, 51] have generally been classified into three categories based on the solutions they seek to provide: computational grids that provide access to heterogeneous resources, data grids that provide data services such as storage, management and data access, and service grids that supply services not provided by any single machine, as described below:

2.2.4.1 Computational Grid

A computational grid is a collection of computing resources that may represent computers on multiple networks and locations, heterogeneous platforms and separate administrative domains with several owners. The purpose of a grid is to run very large applications such as weather forecasting and stock market management [34]. In this type of grid, most machines are high performance servers. The resources are aggregated so as to act as a unified processing resource. The increase in the speed and reliability of networks, in addition to the use of distribution protocols, is an important factor that has led to the use of grids, which enable users

to remotely utilise resources owned by different providers. Computational grids fall into the categories of distributed supercomputing and high through-put computing, depending on how they utilise resources. In the former, the grid executes the jobs in parallel on various resources, reducing completion times. Jobs requiring this category of grid are those that present huge problems. By contrast, high throughputs increase job completion rates. Grid computing can solve the most complex as well as the most vital problems that face scientific and industrial organisations and even governments. The possible applications of grid computing range from the financial (analysing the value of investments, scaling businesses and leveraging existing hardware in investments and resources) through product line (reducing designs, numbers of products and operational expenses) and scalability (creating scalable and flexible enterprise IT infrastructure) to the scientific (connecting research teams located in different geographical areas, accelerating research and production processes) [2].

The Northwest Indiana Computational Grid (NWICG) [60] is an example of a computational grid. It offers three universities high speed and high bandwidth, and consists of computational, storage and data resources.

2.2.4.2 Data Grid

In this type of grid, a large amount of data is distributed and/or replicated to remote sites, potentially worldwide. In general, a data grid refers to a system responsible for storing data and providing access to parties authorised to share it. In other words, data grids provide an infrastructure for creating new information data repositories

such as data warehouses or digital libraries that are distributed across several networks [51]. The aims of data grids overlap with those of heterogeneous distributed database systems, which deal with various kinds of database management systems such as hardware, operating systems and network connections distributed across a heterogeneous environment. Data grid provides infrastructures to support data storage, discovery, handling, publication and manipulation. Enterprise data usually possesses the characteristics of large scale, dynamism, autonomy and distributed data sources. In the academic world, there is a desire to share expensive experimental data in order to coordinate research. In business the need for data sharing is even more urgent. Reasons include the requirement for business data to be in real-time for applications such as e-marketing, where it is very important to maintain an up-to-date and consistent product catalogue. The objective of data grids as presented in [27] is to integrate heterogeneous data archives into a distributed data management "grid" in order to identify services for high performance, distributed, data-intensive computing, and to enable users to elicit relevant information from the distributed databases.

Data grids are compatible with computational grids and can integrate storage and computation. One practical application of a data grid is the EU's DataGrid [71]. This is a project financed by the European Union that relies upon developing computational grid technologies allowing distributed files, databases, computers, scientific tools and devices. The main goal of DataGrid is to build the next gen-

eration of computing infrastructure in order to develop and test the technological infrastructure that will enable the sharing of large-scale databases.

2.2.4.3 Service Grid

This type represents and enables a set of services provided by a set of resources. Service grids fall into three categories: on-demand service grids enable real time interaction, collaborative service grids aggregate various resources to provide new services, and multimedia service grids supply the infrastructure for real-time multimedia applications.

This research is considering and concerning with both computational grid and data grid.

2.3 History of Grid Evolution

This section outlines the history of grid evolution by describing the stages that grids passed through on as they evolved into grid computing.

2.3.1 Grid Generations

This section explains the evolution of grids. Three different generations have been identified [72].

2.3.1.1 First Generation

The first approach to grids was known as metacomputing, meaning that grids started as projects to link supercomputing sites. As early as in 1992, L. Smarr et al., described the concept of metacomputing [75], which was built on several geographically distributed supercomputers connected by fast speed networks. Metacomputing has been applied to large computing simulations, remote instrument/sensor control and large dataset management. The authors also elaborated the analogy between the metacomputer and the PC and suggested the concept of utility computing. This is the original concept of grid computing, which arose directly from metacomputing but with more emphasis on the variety of resource sharing and the addition of high-level services such as scheduling. The objective of these projects was to provide computational resources to the maximum extent commensurate with high performance applications. Examples of such projects are FAFNER (Factoring via Network-Enabled Recursion) [72, 81] and I-WAY (Information Wide Area Year) [72, 25]. *FAFNER* and *I-WAY* attempt to produce a metacomputing environment.

FAFNER was a project that factorised a computationally intensive challenge related to digital security, working on any platform with a web service. It won an award in the TeraFlop supercomputing challenge in SanDiego in 1995 [81].

I-WAY was a project that linked many high performance computers by using existing high bandwidth networks without building new ones. It was designed to deal with high performance applications needing fast, interconnected and powerful

resources. The project developed a computational resource broker as one of its innovations [25].

2.3.1.2 Second Generation

The I-WAY project paved the way for the second generation to provide a distributed infrastructure for grid applications. It helped to make the grid more ubiquitous by using high bandwidth network technology to support a large scale data and computing infrastructure by focusing on grid middleware providing a set of standardised interfaces to a diversity of services, hiding their heterogeneous nature. In this second generation, many core software technologies have evolved, including Globus [33] and Legion[20, 58]. Peer to peer techniques have also appeared during this period.

2.3.1.3 Third Generation

The second generation paved the path for interoperability to be used to achieve large scale computation and resource sharing. The third generation allows flexible distributed collaboration and resource sharing by using a standard interface to provide functionality by using a service-oriented model. Many features are enabled through such developments, including the ability to dynamically configure and reconfigure itself and protect itself from attack. This thesis is developing the mechanism for an evolution of the grid in order to supply these features which have hitherto been lacking.

2.3.2 Stages of Development

With respect to the topology perspective, the grid has been classified into clusters, intra-grids and extra-grids [37]. However, it is debatable whether these classifications denote the stages of development of grids that are born as clusters, evolve into intra-grids and finally become extra-grids, as shown in Fig. 2.2.

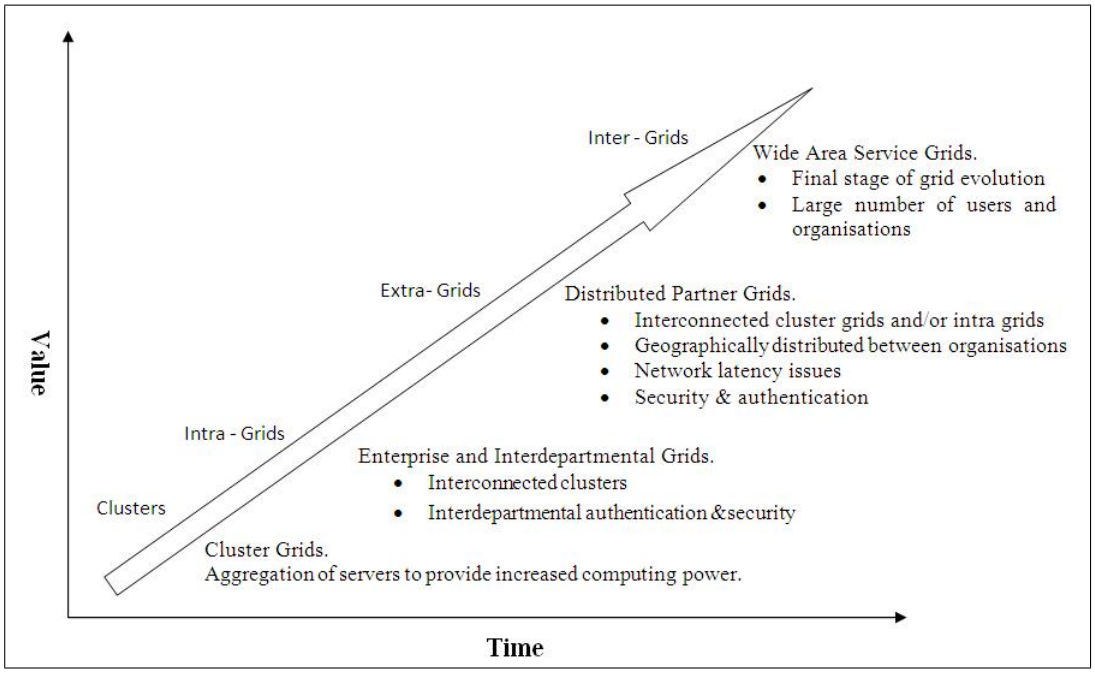

Figure 2.2: The Stages of Grid Development [37].

2.3.2.1 Cluster

Clusters are critical to the evolution of a grid; they are the smallest grids in size and scope. They are aggregations of servers in order to provide increased computing power, as compared to stand-alone computers. Cluster computing is built on unit processors and commodity operation systems. Clusters are designed to

solve problems for particular groups of people within the same department. They are implemented within campus intranets by incorporating PC's, data and servers to maximise the use of computing resources and to increase user job throughput. Cluster grids can therefore operate within a heterogeneous environment consisting of mixed server types, operating systems and workloads. Resources can be accessed at a particular known point in the grid, which has only one job queue [37].

2.3.2.2 Intra-grid

Intra-grids comprise inter-connected clusters. Connecting clusters enable the creation of enterprise or interdepartmental grids. Sometimes referred to as campus grids, these are collections of cluster grids; they enable inter-organisational cooperation. Intra-grids allow departments to share resource collection under common policies without necessarily having to address the security and global policy management issues associated with global grids. For example, managers of IT departments in an organisation might have dedicated computing resources. An intra-grid would collect these resources from managers so that they would be shared by all concerned, even though they are owned by different people. This can be achieved through polices implemented within the environment. Sharing on this larger scale captures unused resources and makes them available to all authorised grid users.

2.3.2.3 Extra-grid

Interconnected cluster grids and/or intra-grids, referred to as extra-grids, are geographically distributed between enterprise organisations. They are also referred to as partner grids. These connect two or more intra-grids and, therefore, have several security domains. Every grid is autonomic and has its own access policies. Security and resource management become very difficult but also more important in such an environment. Extra-grid grids are set up between companies acting in partnership, and between these networks and their customers. In this grid implementation, Virtual Private Networks (VPN) are used to make resources available.

Vendors use various terms to refer to extra grids [34]. IBM, for instance, uses "Extraprise grids" to enable sharing of resources with external partners through VPNs. Likewise, the term "Partner grids" is used by platform computing to define grids between organisations with similar industries that need to collaborate on projects in order to reach a common goal.

2.3.2.4 Inter-grid

Inter-grids are wide area service grids. These are infrastructure services, and sometimes known as global grids. They are collections of Intra-grids and cluster grids connected by the internet, and offer a global view by virtue of distributed systems, especially in academia, where team members may belong to collaborating but geographically dispersed systems. Inter-grids are the final stage of the grid evolution.

Name	Service	Description
GSI	Security	Remote Authentication Services
GRAM	Resource Management	Grid Resource Management
Data management	Transfer data	Manage data using GridFTP
MDS	Information	Grid Meta Directory Service
GEM	Executable Management	Managing location if executables

Table 2.1: Globus Services

2.4 Grid Technology Infrastructure

The grid multilayer architecture involves a modular system structure; it makes it easier to interchange parts and procedures between different vendors. Because equipment is supplied by several vendors, interoperability becomes critical and high standards are necessary. For standardising grid specifications, protocols and interfaces, the Globus alliance and Open Grid Forum (OGF) were established, as explained below.

2.4.1 Globus Alliance

Globus Alliance [6] is an international collaboration of organisations and individuals conducting research for the development of fundamental grid technologies. Globus Alliance introduced open source software called Globus Toolkit for building grid systems and applications. The Globus toolkit provides a set of basic facilities needed for grid computing, such as security, execution managers, data management and information services. These are summarised in table 2.1 and shown in Fig. 2.3.

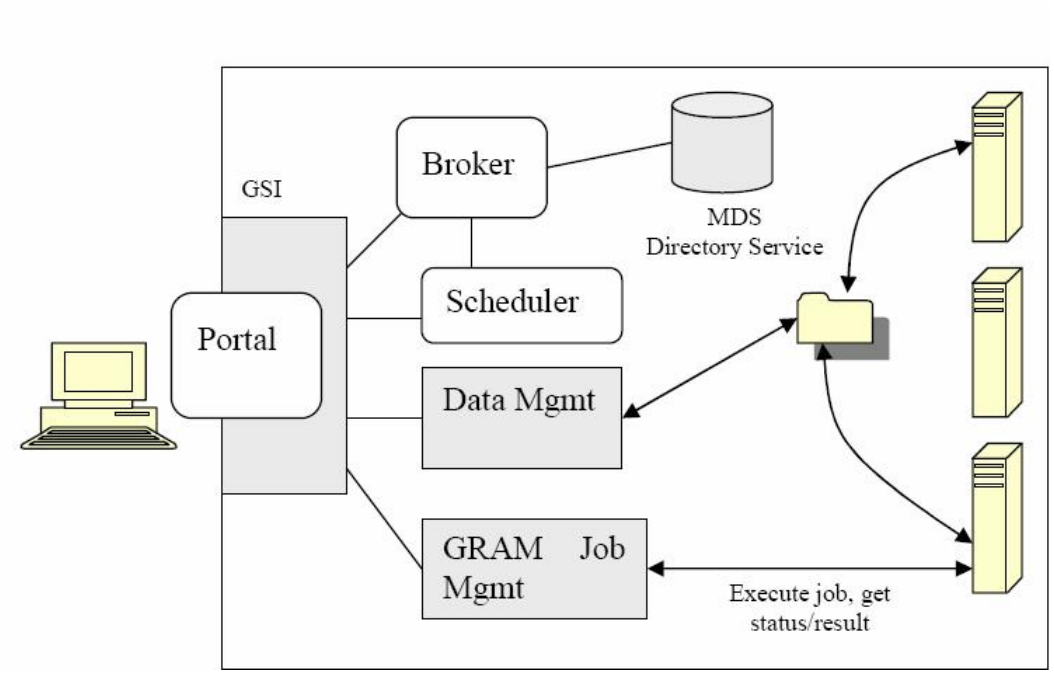

Figure 2.3: Globus Toolkit Architecture
[6].

2.4.1.1 Security

Security is an important requirement for grid computing. In any grid environment there must be mechanisms to provide security, including authentication, authorisation and data encryption. The Globus Toolkit has a component, Grid Security Infrastructure (GSI), for providing robust security mechanisms. GSI provides a public key infrastructure for authenticated and private communication, as well as authenticating users to ensure they are entitled to use the resources in a particular domain.

2.4.1.2 Execution Management

This is the core of Globus toolkit services. The Grid Resource Allocation Manager (GRAM) is a basic library service that provides remote execution together with its status management. It can convert a user request for resources into commands that local computers can understand, meaning that it provides secure and reliable services to launch jobs on particular resources, check their status and retrieve the results when they are complete [49].

2.4.1.3 Data Management

Moving data to various nodes within the grid needs secure and reliable methods. The Globus toolkit contains a data management component that provides such services. This component uses GridFTP [**?**].

2.4.1.4 Information Services

Grid resources require mechanisms such as Monitoring and Discovery Service (MDS) for discovering and publishing their status and configuration information. MDS [49, 47] simplifies grid information services by providing access to static and dynamic information regarding resources. It determines the resources that match the job requirement, as well as discovering the state of those resources by using MDS components, as shown in Fig. 2.4 and described as follows.

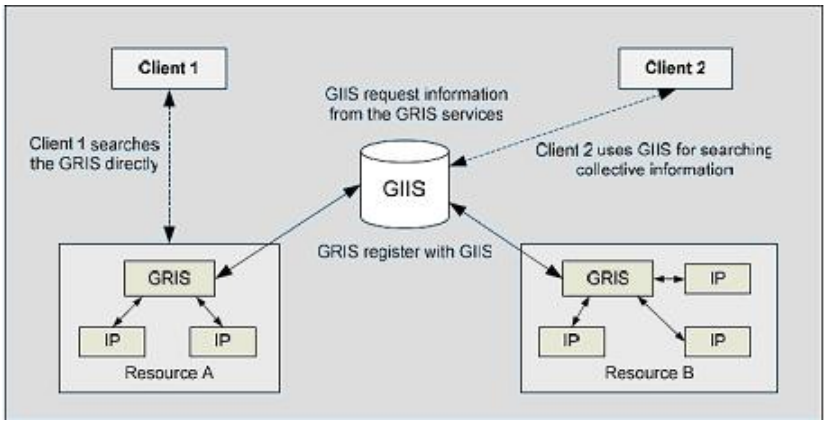

Figure 2.4: MDS Architecture

- **Information Provider (IP)**, which interprets the properties and status of local resources to the format defined in the schema and configuration files. It 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.

- **Grid Resource Information Service (GRIS)**, is a distributed information service that can answer queries regarding a particular resource by directing the query to the underlying IP. GRIS deposits and displays the information as entries.

- **Grid Index Information Service (GIIS),** a storehouse that contains indexes of resource information registered by GRIS and other GIISs.

- **MDS client**, which is based on the Lightweight Directory Access Protocol

(LDAP) client command to search for resource information in grid environments.

2.4.2 Open Grid Forum

Open Grid Forum (OGF) [31] is a public community forum for the discussion of grid technology issues. Currently more than 400 academic and industrial organisations from around the world subscribe. The goals of OGF include the creation of an open process for the development of grid specifications and agreements, and the establishment of grid architecture documents and best practise guidelines. Various research groups within OGF have created many standards such as Open Grid Service Architecture (OGSA) to present a service-oriented view of the shared physical resources or services supported by theses resources, Open Grid Services Infrastructure (OGSI) to define mechanisms for creating and managing grid services (this acts as technical specification for implementing grid services), GridFTP and JSDL [7, 4] ; many other issues are currently being worked on.

2.4.2.1 Job Submission Description Language

Job Submission Description Language (JSDL) [7] is an XML-based description language designed by the Open Grid Forum (OGF) as a standard language in which to describe a job's requirements. It describes the requirements of computational jobs for submission to resources, especially in grid environments. It can, however, also

describe requirements that are not fundamental to grid computing, which means that it is not restricted to the latter. The specification of this language states that it can be mixed with other languages to realise a more complex functionality. To facilitate the expression of job requirements as a set of XML elements, JSDL consists of a vocabulary and normative XML Schema. JSDL was inspired by two main scenarios. The first is that there are many job management systems adapted to different organisations, while each system has its own language for describing job requirements. This makes it difficult for the system to manage jobs, and requires organisations to prepare the job submission documents. The second scenario is that the job submission description may be transformed or passed between systems to match the resource requirements for the job. JSDL is being used in many existing projects such as the Globus toolkit [6].

2.4.2.2 GridFTP

GridFTP provides secure, efficient data movement in grid environments. It is an extension of the standard FTP protocol for grid computing [56, 5, 12, **?**, 13].

Distributed scientific and engineering applications require the following:

- Transfers of large amounts of data (terabytes or petabytes) between storage systems.

- Access to large amounts of data (gigabytes or terabytes) by many geographically distributed applications and users for such purposes as analysis and

visualisation.

Developed by the Open Grid Forum, GridFTP was designed to provide reliable, efficient and secure access, and to transfer huge amounts of data between distributed resources in the grid using many facilities such as multistreamed transfer, autotuning and globus-based security. It is a kind of service offered by grid computing.

GridFTP is very important in this work because it can be the basis for grid evolution through the use of aspects of migration.

The FTP protocol was attractive for the following reasons:

- It is one of the most common data transfer protocols, as well as being the most likely candidate to meet a grid's needs.

- It includes many features, such as its provision of a well-defined architecture and the fact that it is used extensively.

- It is a widely implemented and well-understood Internet Engineering Task Force (IETF) [44] standard protocol.

- Its support for third party transfers, which means its ability to directly transfer between two servers in client/server and other transfers.

- Numerous groups have added various extensions through the IETF. Some of these extensions would be particularly useful in grids.

GridFTP has the following features:

- Grid Security Infrastructure (GSI) and Kerberos support.

  Robust and flexible authentication, integrity, and confidentiality features are critical when transferring or accessing files. GridFTP must therefore support GSI and Kerberos authentication. It provides this capability by implementing the Generic Security Services Application Program Interface (GSSAPI) [54] authentication mechanisms.

- Third-party control of data transfer.

  In order to manage sets of large data for large distributed communities, it is essential to control of transfers between storage servers by providing third party. GridFTP provides this capability by adding GSSAPI security to the existing third-party transfer capability defined in the standard FTP. Third party operation allows a user at one site to initiate, control and monitor a data transfer operation between two other parties (source and destination).

- Parallel data transfer.

  Using multiple Transmission Control Protocol (TCP) [65] streams in parallel (even between the same source and destination) on wide-area links can improve aggregate bandwidth over using a single TCP stream. GridFTP supports parallel data transfer through FTP command extensions and data channel extensions.

- Striped data transfer.

Data may be striped or interleaved across multiple servers. Striped transfers provide further bandwidth improvements over those achieved with parallel transfers. GridFTP includes extensions that initiate striped transfers, which use multiple TCP streams to transfer data that is partitioned among multiple servers.

- Partial file transfer.

  The transfer of partial files required from many applications such as high-energy physics analysis. However, standard FTP supports the transfer of complete files or the transfer of the remainder of a file starting at a particular offset. GridFTP introduces new FTP commands to support transfers of subsets of file.

- Automatic negotiation of TCP buffer/window sizes.

  Using automatic negotiation of TCP buffer/window size is an important in achieving maximum bandwidth with TCP/IP, thus improving the transfer performance, especially in a wide area. GridFTP extends the standard FTP to support both manual settings and automatic negotiation of TCP buffer sizes for large files and for large groups of small files.

- Support for reliable and restartable data transfer.

  Reliable transfer is important for many applications that manage data. Fault recovery methods are needed for handling such faults as transient network

failures and server outages. The FTP standard includes basic features for restarting failed transfers that are not widely implemented. The GridFTP protocol exploits these features, and extends them to cover the new data channel protocol.

- Integrated instrumentation.

  The protocol calls for restart and performance markers to be sent back. Moreover, there are new functionalities and recent developments added to GridFTP to take advantage of the latest network technologies and transport protocols.

- GridFTP Pipelining.

  When the dataset is large, it will be broken up into many small files. This causes a problem known as "lots of small files" (LOSF). The solution to this problem is known as pipelining [12]. Pipelining approaches the LOSF problem by trying to minimise the amount of time between transfers. Pipelining allows the client to send transfer commands at any time. The server processes these requests in the order they are sent. Pipelining improves the performance of LOSF transfers.

- GridFTP over UDT.

  User Datagram Protocol (UDT) [42] is an application-level data transport protocol using UDP to transfer bulk data. It achieves good performance on high-bandwidth, high-delay networks in which TCP has significant limitations.

Because of this, it can provide significantly higher end-to-end performance than GridFTP over TCP, especially on wide area networks.

- Split DSI for GridFTP.

  GridFTP has a Data Storage Interface (DSI) that interacts with storage systems, accepting requests such as get, put and stat and performing the necessary functions based on the underlying storage system. DSI interface has been implemented to achieve split TCP functionality. In the previous implementation, a GridFTP server could act either as an end or as an intermediate server, which is very restrictive from the point of view of production using GridFTP servers. GridFTP generalises by allowing DSI to perform different functions based on the input it receives. For example, the DSI "get" command could be either to contact the underlying storage directly or to forward the data to another GridFTP server. This can help both to overcome some of the TCP limitations and to avoid some bottleneck links.

- GridFTP Where there is FTP (GWFTP).

  GWFTP is an intermediate program used to act as a proxy between existing FTP clients and GridFTP servers. Users can connect to GWFTP with their favourite standard FTP client, and GWFTP will then connect to a GridFTP server on the client's behalf. This process allows the use of ordinary FTP clients to invoke operations on GridFTP servers.

- Network provisioning.

  Network provisioning technologies must be integrated into a scalable architecture that can provide on-demand setup of channels at varying bandwidth resolutions. This allows for binding of GridFTP transfers to optical paths.

- GridFTP over Infiniband.

  InfiniBand (IB) defines a point-to-point interconnect architecture that leverages networking principles (switching and routing) to provide a scalable, high performance server. InfiniBand provides transport services for upper layer protocols and supports flow control and quality of service to provide ordered, guaranteed packet delivery. Using Infiniband enables GridFTP to take advantage of the recent developments innfiniband to achieve high performance on wide area networks.

GridFTP in reality

NaradaBrokering [52, 53] is a distributed messaging middleware designed to run on large cooperating broker nodes networks. Because NaradaBrokering's communication is asynchronous communication, it can publish messages at a very high rate. It exploits GridFTP facilities and capabilities such as fragmentation/reliable delivery service.

Fig. 2.5 shows the integration between GridFTP and NaradaBrokering. The architecture embodies two approaches, proxy and router. The key component of the proxy approach is the remote GridFTP server, which simulated by NB Agent A. A

GridFTP client contacts NB Agent A, which then forwards all requests to the NB Agent B that contacts the available GridFTP server. The router approach is used to upload GridFTP functionality with NaradaBrokering; GridFTP functionality requires two socket connections, control channel and data channel. control channel is used for transfer commands (i.e. "put", "get"), data channel for transfer data.

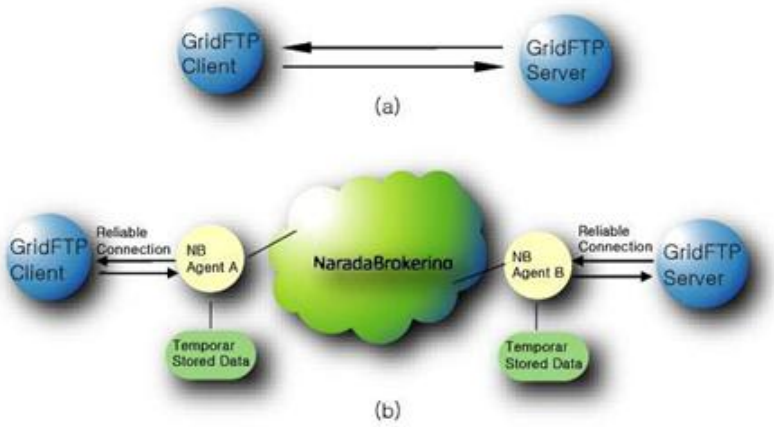

Figure 2.5: GridFTP with NaradaBrokering [53]

2.5 Current Efforts in Grid Computing

There are various development being carried out within grid computing. These can be classified into the following categories:

1. Grid-technology projects, which are involved in the development of grid-enabling technologies such as middleware and hardware (e.g. Globus [49]and Condor [38]).

2. Testbeds projects for developing and maintaining working test beds using ex-

isting grid technology such as [66], DOE SCIENCE GRID [40] and GRIDPP [30].

3. Field-specific applications projects, which are devoted to exploring and harnessing grid technologies in the context of specific fields of scientific research such as BIOINFORMATICS [70], Earth System Grid II [45] and ECOGRID [68].

4. @home, used for distributed or internet-based computing projects such as Compute Against Cancer [67], Folding@home [69] and SETI@home [74].

2.6 Resource Broker (Scheduler)

The Resource Broker, sometimes called Scheduler, is one of the main components of grids. It plays an important role in building an effective grid environment by scheduling user applications onto grid resources to achieve certain performance goals such as minimising execution times and communication delays, as well as the use of resources, maximising resource utilisation and reliability, and load balancing. The broker is responsible for the discovery and selection of resources and for application execution by transferring job input files to the resources and sending job output back to users.

These tasks are:

- Resource Discovery.

The role of discovery is that of determining a list of authenticated resources available in the grid. This list is usually obtained by searching its database containing information about the resources. Resource discovery mechanisms use a single database (the centralised approach) or a set of databases (the distributed approach); they reside at various places in order to obtain information about the available resources regarding logical entities such as application software, operating systems, policies and data, and physical entities such as CPU speed and architecture, current loads, and networks, in order to compile a list of resources that can meet the application requirements. An example of such a database is the Monitoring and Discovery Service (MDS) in Globus [24, 21, 43, 55].

- Resource Selection.

  Once the resource list has been assembled, the optimal resources that meet the user's requirements such as cost are selected from it.

- Job Execution.

  Once the job and resources have been selected, job input files are transferred to and run on the resources. When the job is finished, the broker informs the user.

  Unpredictable situations that need addressing may occur during job runtime, so the job monitoring role is to check a job's execution and detect failure or

unexpected incidents. An example of such monitoring is the Network Weather Service (NWS) [85].

Much work has been done on resource brokering in grid computing and many scheduling algorithms have been introduced to deal with grid applications. The following subsections outline some of these.

2.6.1 Condor G

Condor G is a centralised high throughput grid scheduling system developed by the University of Wisconsin, USA. It allows users to manage heterogeneous resources running Globus through the command line aided by the Globus toolkit.

The management operations that it supports are [38]:

- submission of jobs to grid resources, including their input/output files and any arguments needed to initiate a job's execution.
- querying a job's status or cancelling a job.
- informing the user via email of job termination or errors that occur during execution.
- storing a log that provides a history of a job's execution stages.

As shown in Fig. 2.6, once the user submits the job the scheduler responds by creating a GridManager daemon at the job submission machine to handle and manage this job. One GridManager daemon handles all jobs for a single user, and terminates

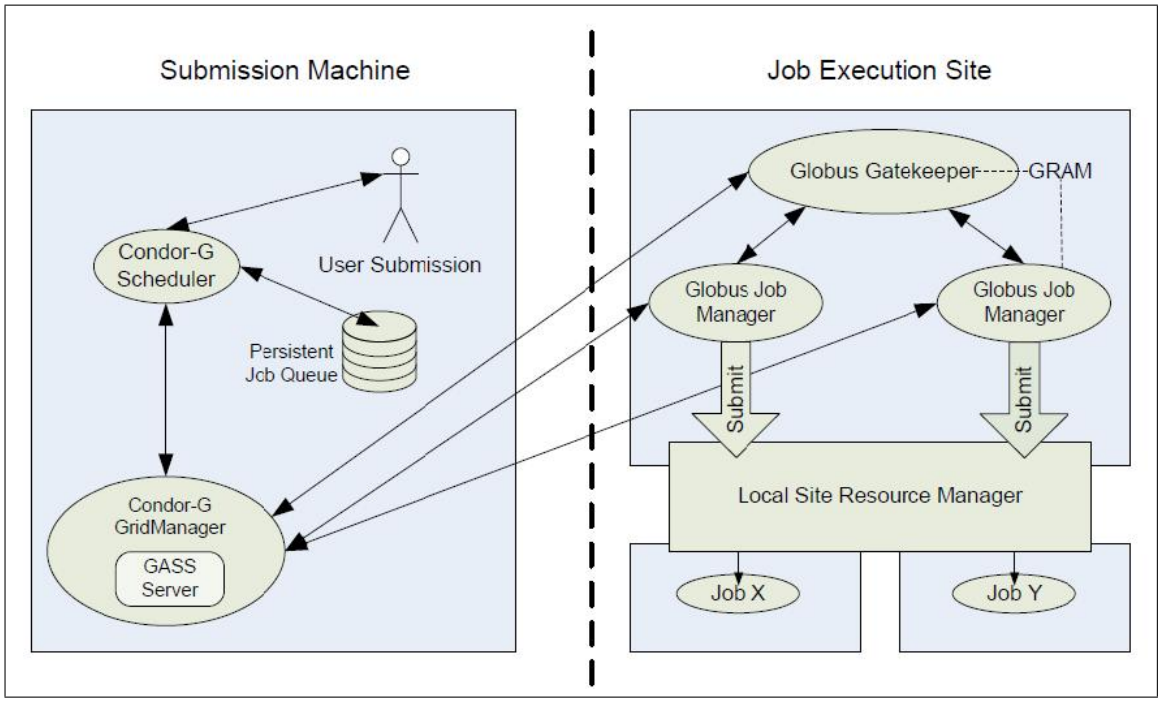

Figure 2.6: Condor-G Mechanism for Executing a Job on Globus Managed Resources [38].

once they are all completed. Each GridManager contacts the modified GRAM that has been configured to support Condor-G at the execution site. After passing the Gatekeeper's authentication process, a Globus JobManager daemon is created. The JobManager daemon at the execution site connects directly to the GridManager at the submission machine using the Global Access Secondary Storage (GASS) [9] in order to transfer the job's executable and standard input and output files as well as any error files. The JobManager then submits the job to the local site's resource management system. The JobManager sends updates on the job's status 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. However, Condor G does not support checkpointing, migration of serial grid applications, ap-

plication models, policy based, co-allocation and advance reservation. In addition, file transfer is mainly based on the GASS service rather than on GridFTP.

2.6.2 Nimrod/G

Nimrod/G is a hierarchical system based on the concept of computational economy, and is designed to run parametric applications on grids. It has been developed by Monash University, Australia [14]. Nimrod/G architecture is shown in Fig. 2.7. In

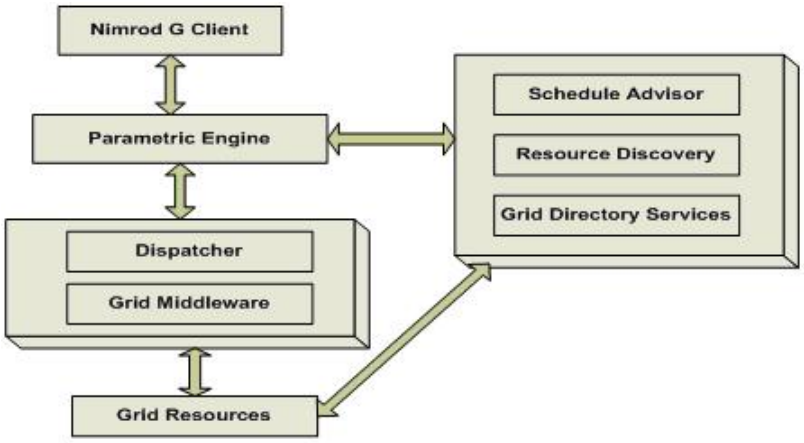

Figure 2.7: Nimrod/G Architecture

this architecture, the user (i.e. the client) creates an experimental plan by using declarative parametric modelling language before a job is passed to the parametric engine. The latter interacts with the scheduler to retrieve resource availability, while the scheduler discovers resource load and status through the Grid's directory service (which uses the Monitoring and Directory Service) and selects the resources that meet the user's requirement. Once the parametric engine has confirmed the availability of the user's resource selection, the information is forwarded to the dispatcher

to initiate the job on the selected resource by job wrapper, which is responsible for beginning the execution of the job on the resource and returning the result to the parametric engine via the dispatcher.

Nimrod/G supports resource reservation, co-allocation and load balancing through periodic rescheduling. However, it does not support exposing migration, fault tolerance, and policy based scheduling.

2.7 Migration

Migration is the capability to move physical or virtual computational resources (software code, portable notebook PC's, running objects, mobile agents and data) from one location to another across a local or global network. Migration is a central, very broad concept used in distributed computing; it can be subdivided into personal, computer and computational migration. In the first of these, the user can do the work at locations remote from physical hardware, which users do not need to carry around with them, but can start work in one location and carry it on anywhere else throughout the world regardless of machine type. Examples include web-based email accounts. Computer migration is concerned with moving an actual piece of computer hardware such as portable notebook PCs and personal digital assistants (PDAs) from one location to another. Computational migration addresses the movement of software [83, 29]. This thesis is concerned with the latter. Computational migration can also be referred to as control migration, data migration, link migra-

tion and object migration [17]. Control migration supports moving a control thread such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) from one machine to another and back again. Data migration allows the data required by the process to be passed through the network. An example is Java RMI call the method name, its arguments and return type, packed and transmitted to and from the remote machine. Link migration, referring to the transmission of the ability to move objects (codes) between different servers, is the fundamental concept behind distributed objects. It is also the basic concept of distributed computing. Code migration (mobile computation) has come into popular use because it provides the capability to link software components at runtime. This means that a software component can move around and execute on different servers across the network. From the viewpoint of execution state, code migration can be divided into two groups, weak and strong migration [18].

2.7.1 Weak Migration

Weak migration allows code to move across nodes. The codes sometimes have initialisation data attached but without execution states (i.e. the state of the computation is lost at the originating node). One illustration of a weak migration system is a Java applet; others are Code-on Demand (CoD) and Remote Execution/Evaluation[29, 77].

Remote evaluation is derived from the client-server and virtual machine styles

[39]. A client component has to know how to control the remote process; the client thereupon sends the instruction to a server component at the remote site, which in turn executes the code using the resources available there. The client then receives the result from the sever. The remote evaluation style assumes that the code provided will be executed in a protected environment so that it will not affect other clients served by the same server beyond the capacity of the resources being used [28]. One of the advantages of remote evaluation is the ability to customise the services as server components, which provides improved extensibility and customisability and better efficiency when the code can adapt its actions to the environment inside the server. A good example of Remote Execution Evaluation is grid computing. Code-on-Demand style systems are used when a client component has access to a set of resources but does not know how to process them. It therefore sends a request to remote server to acquire the code of "know-how". Once the code is received, the client executes it locally. The advantages of Code-on-Demand are the adding of features to a pre-deployed client, which provides enhanced extensibility and configurability, and better user-perceived performance and efficiency, when the code can adapt its actions to the client's environment and interact with the user locally without remote interactions [28]. A famous example of Code-on-Demand is Java applets.

2.7.2 Strong Migration

Strong migration refers to the capability of a computational environment to migrate the code and execution state (the context of execution) to restart at a new resource. The execution state includes running code, program counter, saved processor registers, return addresses and local variables. A group of coordinating threads runs in a process on the client, and then accesses the remote resource by the invocation of a remote process. The advantage of strong migration is that an application can decide to move between locations while it is processing information, presumably in order to reduce the distance between it and the next set of data it wishes to process. In addition, the reliability problem of partial failure is reduced because the application state is only in one location at a time.

2.8 Summary

This chapter survey the background and related work on related research issues covered in this thesis. This background expressed the fast growth in hardware, software and network bandwidth which articulated the need for grid evolution which present as a significant problem and it also illustrated the need for increasing the CPU utilisation which is one of grid aims.

Chapter 3

Grid Evolution

Objectives

- To present what grid evolution means.
- To discuss the importance of evolution.
- To describe external and internal evolution as it applies to grids.
- To discuss when evolution occurs.
- To describe permanent and Just-in-Time evolution.
- To describe the benefits of grid evolution.

3.1 Introduction

Grid computing came about in the 1990s when researchers were seeking a way of sharing expensive computing resources and experimental equipment. The concept

evolved from the metacomputer, which enabled the use of independent computers as a single supercomputer [75, 72, 34], and will play a key role in e-business as a resource sharing and collaboration platform. In order fully to fulfil this function it must be pervasive, dependable, consistent and inexpensive to access [34].

It is worthwhile to note that the grid concept has evolved and is still developing. The idea of grid applications started with the tapping of computing power; hitherto, most commercial applications as well as commonly held knowledge about this concept are still limited to SETI@Home [74].

In reality, there are already huge numbers of grids distributed around the world, each one created to serve a particular group of scientists or other users [1]. Grid evolution has found to be a good mechanism by which to tackle the problem of catering for users from all disciplines. In the proposed research, this notion and demonstrate how to integrate its two types, external and internal evolution , has fully explore in a unified way.

This chapter will begin by describing the meaning of evolution in Section 3.2, detailing the importance of grid evolution in Section 3.3 and explaining internal evolution in Section 3.4. Section 3.5 describes the two types of external evolution, Permanent and Just-in-Time, while the last section explains how evolution occurs.

3.2 What Does Evolution Mean?

Evolution means adding or removing capabilities. In grid computing, evolution means adding new functions and/or equipment or deleting unusable resources that affect performance in some nodes. Within grid computing, migration can be of two distinct types: external which representing evolution externally (i.e. between the original grid and the cooperating grids) and internal (i.e. within a single grid).

3.3 Importance of Grid Evolution

Grid computing is becoming increasingly popular, with applications in many areas such as science, engineering and business. Grids are an evolving area of computing, one in which standards and technology are still being developed to enable this new paradigm. The grid environment must therefore be flexible in order to be able to change and evolve. Grid evolution has found to present a significant problem because the static nature of current software/hardware evolution techniques and their ill-defined processes render them inadequate. The evolution of grids is important because:

- they need to increase their computing power and disk space.
- new software is constantly being developed in order to cope with rapid software development.
- new hardware must be added.

- they must stay abreast of the latest developments.
- scientists need to share huge amounts of data worldwide.
- they must serve scientists and others as much as possible.
- grids need to communicate with each other.
- their performance or reliability may have to be improved.
- they must support QoS by adding new features.
- failures must be repaired.

The new technique for grid evolution should allow evolution to happen seamlessly and at run time. To solve these problems, the concept of internal and external evolution had introduced. Internal evolution occurs inside the grid boundary by migrating special resources such as application software from node to node inside the grid. External evolution happens outside the boundary of a given grid.

There are triggers enforce evolution such as fulfil job requirements,failure and QoS. Therefore, the two types of grid evolution, external and internal, have many benefits such as:

- Addition of greater functionality to evolvable grids, giving them more properties because additions may include new or special properties not hitherto contained in the grid.
- Execution of more jobs through the addition of features.

- The serving of more users through the improvement of grid characteristics.

- Because the grid has accrued new tools or equipment, it will be more likely to be able to fulfil its application requirements, thereby reducing the number of rejected jobs.

- Enhanced performance as regards the running of applications by migrating data to the execution host, thus reducing communication, and the solution of load balancing problems.

- Improvement of grid quality through the acquisition of equipment and tools.

Both types of evolution must be realised by efficient and modular techniques which are described below.

3.4 Internal Evolution

Internal evolution is mainly a process that evolves the internal structure and resources of a given grid. In other words, internal evolution deals with the function of a single grid. It involves migration inside a grid. Migration is the capability to move or migrate application software and data across grid nodes. This allows the grid to perform more functions, as well as enabling the fulfilment of requirements for grid applications, enhancing the performance of running applications by migrating data to the execution host (thus reducing communication), and providing a solution to

load balancing problems. All these reasons support grid evolution. There are also many reasons why it is possible that the required node may not be found:

- The node required to meet the job requirements may currently be busy.
- The node that meets the job hardware requirement does not have the required application software and/or data.
- The node has many resources (application software and data) and therefore does not allow the addition of new application software or data.

When such resources migrate between nodes inside a grid, the destination node acquires new application software or data, thus evolving to perform more functions and execute more jobs. That's what is meant by node evolution, which evolves the node inside the grid by migrating application software or data from that node to another in the same grid. This evolves the node without changing the properties of the grid as a whole, because the latter already possesses this application software or data and they are not coming as a new to the grid but they are a new for the node and fulfil user job requirements. As a result, serve more users and lead for grid internal evolution. The other evolutionary mechanism is the addition of new application software or data by installing it directly into the node or by migrating it from other grids. This type of migration may occur between two grids by migrating application software from a node outside the grid to one inside it. This addition leads to the evolution of the whole grid by the addition of new products, as a result of which it evolves externally as described in the next section.

3.5 External Evolution

The other type of evolution is called external evolution. This evolution occurs when any hardware, application software or data is added to or removed from the grid if affecting its node performance. In this case, the whole grid evolves externally. When new hardware such as a node is added, the grid evolves. The new node may have a particular function or specification such as speed or capacity. As a result, new components and abilities are added to the grid. This type of evolution extends the boundary of the grid and increases its computing power and disk space.

The grid can also evolve externally to include, for example, another grid, without affecting its integrity (i.e. the structure and resources). Fig. 3.1 shows the boundaries of an evolved grid that has grown by joining another grid (the "Joined Grid") and acquiring two nodes (laptop and data storage). The grid boundaries are thus increased by its acquisition of new resources or grids. This increase may fluctuate according to need; the new addition may just as easily be removed again when that need passes.

When application software or data that affects the grid's performance is removed, the grid evolves. This removal improves the performance of a grid by increasing its speed and disk space. It can thus be seen that external evolution may be either Just-in-Time or Permanent.

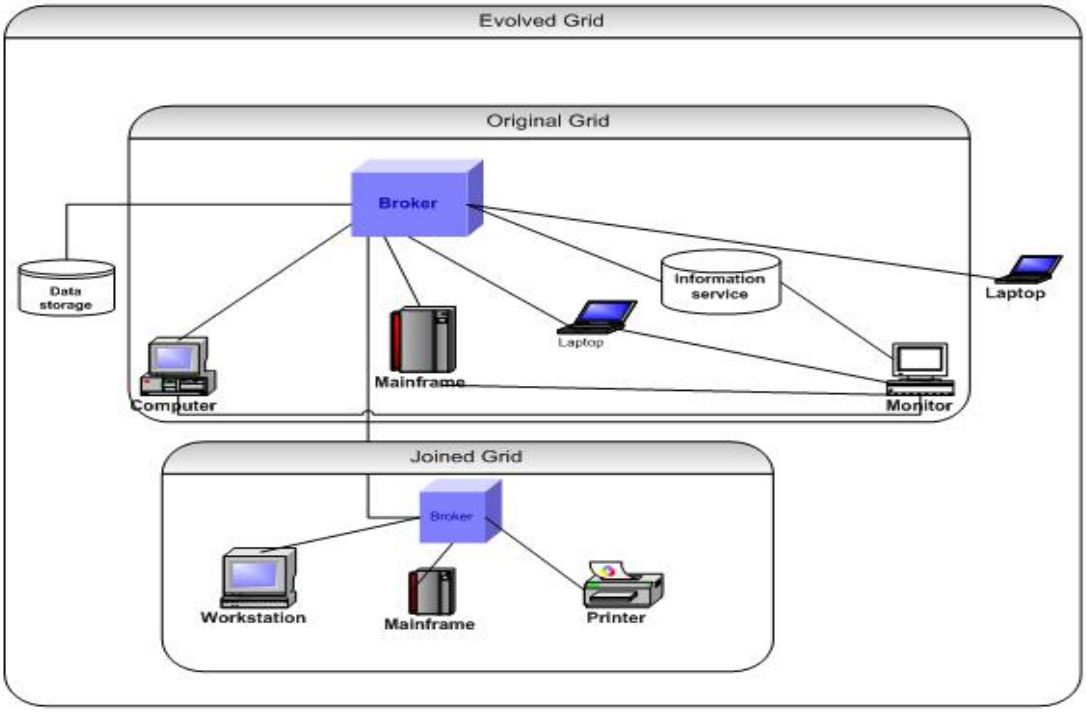

Figure 3.1: External Evolution

3.5.1 Permanent Evolution

Permanent evolution occurs if any addition or removal is permanent. Addition involves nodes, application software and/or data, while removal entails application software that affects the grid's performance. If it is necessary for any new node to be added to the grid, it must register itself with the resource broker. This registration requires a message from the node to the resource broker containing the node name, availability time (the time during which this node will be available for use by the grid), and all node specifications:

- Hardware specification for the node, including CPU speed, CPU count, virtual memory, physical memory and available disk space.

- Application software, containing the node's operating systems and all the application software available on the node.
- Data, containing all the data available on the node, as well as the size of that data.
- Available time

Upon receipt of this message, the resource broker will register this node with the grid, saving all its information in the information service to be used by a resource broker when required. On the other hand, it may be new application software or data that is to be added to the grid node. This addition occurs in one of two ways. The application software or data can be manually installed in the node or it can be migrated, which means moving the application software or data from one node to another, either in the same grid or in another one. If the migration is from outside the grid it is a permanent evolution, occurring when the original resource broker communicates with another and asks it for data or application software, taking one or both to fulfil its job requirements. This constitutes a permanent evolution of the original grid. But if it occurs from node to node within a grid, it called internal evolution as has just been described. Regarding the migration of application software or data from a node outside the grid, when the resource broker notices a need for new application software to execute one or more of the submitted jobs, it sends a broadcast message to the other cooperating resource brokers to request the required product. The message contains the application software name and version and/or

the data name and size. Once the other resource brokers receive this message, they search their grids for that product.

They may reply with one of the following answers:

- "No, I have not".
- "Yes, I have, but I cannot give it to you." The requester ignores both of these messages.
- "Yes, I have all you have requested." (This message is a reply to a request for application software or data or both) In this case the requester takes a copy of this application software and/or data or migrates it, installs it on the required node and sends the job to that node for execution.
- "Yes, I have some of what you requested." (This message is a reply to a request for both application software and data). In this case, the requester attempts to collect the required application software and data from different grids, install it on the required node and send the job to that node for execution.

If the resource broker has asked for both application software and data, it starts searching all the received messages with priority for messages saying "Yes, I have all what you requested", but if there is no such message, it searches for ones saying "Yes, I have part of what you requested" and tries to assemble the required application software and data from different grids. However, if no positive responses are received, or if the resource broker has asked for application software or data and did not receive

any positive replies ("Yes, I have all what you requested"), it performs Just-in-Time evolution.

3.5.2 Just-in-Time Evolution

The notion of Just-in-Time evolution had been introduced, which means the evolution of the grid only when it is needed at run time, by enabling the job to be executed on other cooperation grids. This adds great flexibility to the operation of a grid. This need arises dynamically from the system itself, and happens when:

- the grid does not have the required node; or
- the required node does not allow the addition of application software and/or data; or
- the required data and/or application software is available on other grids but does not allow migration or copying; or
- the grid does not have all the requirements necessary for the job; or
- the grid is busy.

This type of external evolution involves the original grid and one or more others supporting the requisite evolution. This requires knowledge regarding those cooperating grids whose resource brokers have permitted communication to the original resource broker. The grids involved communicate with each other in order to exchange the required nodes, application software or data so as to satisfy the job requirements.

This communication occurs between the resource brokers and is managed by the original broker. If the latter proves unable to fulfil the job requirements, it sends a broadcasting message to all cooperating grids and asks them for all those requirements (node, application software and data). This message must contain a product name with full specifications and the estimated time that product takes to use. For nodes, this specification includes node CPU speed, CPU count, disk space, operating system and other factors, depending on the node type. For application software it includes the software's name and version, and for data, the data name and size. This resource broker may receive one of the following answers:

- "No, I do not have any required resources". The requesting broker ignores such messages.

- "Yes, I have all the required resources". The requestor sends the job to this respondent in order to execute the job on this node and return the result when the time for the job to be executed has passed.

- "Yes, I have some of your required resources", this will be one of the following:

  - Node with application software.If the respondent has a node with application software, the requesting broker continues searching the other messages sent by the other resource brokers for the remaining necessary data. If this is found on another grid, the resource broker migrates it to the grid that contains the required node and application software in

order to download this data and execute the job. If it is not found, the requesting broker apologises to the user that the grid cannot execute the job.

– Node with data.

If the respondent has a node with data, the requestor continues searching the remaining messages for the required application software. If it is found on another grid, the broker migrates it to the grid that contains the required node and data in order to download this application software and execute the job. But if it is not found, the requesting broker apologises to the user that the grid cannot execute the job.

– Node only.

If the respondent only has a node, the requestor continues searching the remaining messages for one that contains both the required application software and data. If such a message is found, the broker migrates the software and data to the grid containing the requisite node in order to download them and execute the job. If they are not found in the same message, the requestor searches for them separately and, if it finds them on separate nodes, it migrates them both to the grid containing the requisite node, installs them on this node and executes the job. If they are not found, however, the resource broker apologises to the user that the grid cannot execute the job.

– Application software with data.

  If the respondent has application software with data, the requestor continues searching the remaining messages for the required node. If found, the broker migrates the software and data to this node and sends the job to it for execution. If not found, the broker apologises to the user that the grid cannot execute the job.

– Application software.

  If the respondent only has application software, the requesting broker continues searching all the messages from other resource brokers for those that contain both the required node and data. If found, the requestor migrates the application software to the grid that contains the required node and data in order to download this software and execute the job. If node and data are not found in the same message, it searches for them separately. If it finds them, it migrates the application software and data to the grid containing the required node, installs them on this node and executes the job. If it does not find them, the requestor will apologise to the user that the grid cannot execute the job.

– Data.

  If the requestor finds only data, it continues searching all the messages from other resource brokers for ones containing the required node and application software; if found, the broker migrates the data to the grid

containing the required node and application software in order to download this data and execute the job. If not found in the same message, it searches for them separately. If it finds them, it migrates the application software and data to the grid containing the required node, installs them on this node and executes the job. If not found, the requesting broker apologises to the user that the grid cannot execute the job.

External evolution may occur outside the grid boundary, meaning that other grids may evolve themselves without any arrangements with the original grid. This evolution is useful for those grids that communicate with evolvable grids, asking them for nodes, application software and/or data in order to execute jobs. As a result, such evolution indirectly supports external evolution.

3.6 How Evolution Occurs?

When the grid receives the job with all its requirements, the resource broker searches the information service for the job requirements. The result will be one of the following; depending on these results, evolution will be activated.

1. The grid has all the job requirements on the same node.

   In this case the resource broker sends the job to this node and asks it to execute this job by the stipulated execution time, after which the resource broker stops the job, retrieves the result and confirms it to the user.

2. The grid has all the job requirements scattered across different nodes in the same grid, and all migration policies allow migration of the required products.

   In this case, the resource broker collects these requirements from the other nodes, inserts them on the required node and sends the job to this node for execution. This collection process leads to internal evolution.

3. The grid has some of the requirements, namely node and either application software or data.

   In this case, the resource broker determines whether or not this node's policy allows the addition of application software and/or data. If it does not, the broker sends a broadcasting message to other cooperating resource brokers and asks them for all the job requirements, but if migration is allowed, the broker sends a broadcast message to other cooperating resource brokers and ask them for the required application software and/or data.

4. The grid has none of the job requirements.

   In this case, the resource broker sends a broadcast message to other cooperating resource brokers asking them for all the job requirements of node, application software and data that exist on the same node at the stipulated execution time. This is Just-in-Time evolution.

To support evolution, a special computational model and design is needed. Therefore, a novel computational model with grid design has been designed to en-

courage the evolution of grid computing. These will be described in the next two chapters.

3.7 Summary

This chapter describes the technique for the evolution of grid computing in both external and internal types. This evolution allows the creation of a single grid comprising many connected grids acting as one powerful grid that can operate as one vast computational resource. It also allows grid environments to be flexible, to be able to change and to evolve. This chapter also described how evolution occurs seamlessly and at run time through the migration technique which using the GridFTP in order to migrate the required application software and/or data in order to fulfil job requirements.

Chapter 4

Computational Model for Evolvable Grid

Objectives

- To present computational model objects.
- To describe the models that manage and control these objects.
- To describe how the required application software and/or data migrate from one grid node to another.
- To present how the computational model supports grid evolution.

4.1 Introduction

Grid computing systems are the latest computing environments and have been gaining popularity for the past few years. They can be considered as extensions of distributed computing systems, but in which the number and heterogeneity of the systems are much higher. Users, by plugging their systems to Grid computing systems can potentially use the vast number of services that are available in the Grid similar to the way in which electrical appliances can draw power from the electrical power grid. Grids also provide opportunities for collaborative computing, in which users across the grid can collaborate towards solving large applications.

The computational model is a blueprint of a computation performed on a particular architecture. Our unit of computation is an object perceived by an object-oriented paradigm. These objects cooperate, communicate and collaborate with each other. The computational model comprises objects, models and strategy; the objects comprises of grids, nodes, data, application software and jobs, while the models consist of communication, termination, timing, failure and migration models.

This chapter describes the computational model's objects in Section 4.2, the models in Section 4.3, and the strategy in Section 4.4.

4.2 Objects

The main objects for the computational grid will be described in this section. These objects include grids, nodes, application software, data, jobs and applications.

4.2.1 Grid

A grid is a collection of nodes connected via a network and managed by a **resource broker**. It promotes the sharing of services, computing power and resources such as disk storage databases and software applications. In the computational model and design, a grid can allow communication with other grids in order to exchange application software and/or data, and even the job if the chosen grid does not contain the job requirements in its domain. In this case it sends the job to another grid for execution. In the model, each grid has an ID and its own policy.

4.2.2 Node

A node is the most basic component in grid computing. It is a collection of work units that can be shared and that can provide some capabilities. The grid nodes usually differ in speed, capacity, architecture and operating systems. Communication between nodes is achieved via network capabilities such as LAN and WAN.

Nodes are responsible for receiving, executing and returning the results of jobs. Each node has its own application software and data that fulfils the user's requirement in order to execute the job. The model assume that all grid nodes can commu-

nicate with each other in grid environments in order to exchange jobs, application software and data that can migrate between grid nodes when required. They must therefore contain middleware application software. Each node in the grid sends a periodic heartbeat to the monitor responsible for managing and controlling these nodes. This heartbeat contains a timestamp, node name, job status (if the node is running a job) and other optional information.

Node can be either logical node or physical node; the latter can contain one or more logical nodes [80].

4.2.2.1 Logical Node

Logical nodes are designed to perform management roles in grid environments; they can manage different kinds of grid components such as Condor Pool, Distributed File System (DFS) or Distributed Computer Pool [37]. as well as the relationships between them.

In the model, the resource broker and grid monitor acts as a logical node.

4.2.2.2 Physical Node

Physical nodes are categorised according to their function in the grid. The two most common types of physical node are computation and storage nodes. Sometimes these nodes are also called “resource”, “members”, “donors” or “host” among many other names, but they all mean “node”.

1. **Computation Nodes**

Computational nodes are the machines deployed and exploited according to their hardware and processing capabilities. The node could be a cluster, mainframe, high performance computer or a desktop PC. These capabilities are provided mainly by the various kinds of processor architecture. Each computation node has its own processor architecture and its own hardware specifications such as speed, software platform compatibility and internal memory.

Computation nodes can be exploited in many ways; a grid job can be executed on a grid node instead of running on a local machine outside the grid. Parallel jobs can be executed on many processors, whether they are located on the same grid node or on many. Parallel jobs are executed in this way because they have been designed so that their work can be divided into separate parts in order to run on different processors. Some other jobs may require repeated execution on many computation grid nodes [49]. Parallel jobs by nature play an important role in increasing scalability: the number of separate work units produced by the job will consequently increase the number of computation grid nodes which can most efficiently be used for the job's execution, which saves some of the time taken for the job's execution. In other words, the type of parallel jobs and the role of the grid computation nodes in executing the jobs work units improve the scalability of the grid.

2. **Storage Nodes**

Since the function of the computation node is the processing of jobs, other

machines are responsible for storing and providing data. These machines are called storage nodes. The most common storage type is secondary storage using hard disk drives or other permanent storage media such as tape drives. Implementing secondary storage in a grid has the advantage of improving the capacity, performance, sharing and reliability of data exchange within that grid. There are different kinds of file systems which will handle the storage and organisation processes for the data across the nodes of the grid network. These popular and common network file systems include Network File System (NFS), Distributed File System (DFS) and General Parallel File System (GPFS). It is advisable that the data capacity of all the nodes, especially the data storage ones, be mounted on a particular type of network file system.

3. **Special Nodes**

   Grid computing can use resources of types other than those pertaining to job execution. A grid administrator may sometimes create new artificial resources to execute certain jobs. For example, some machines may be designated for use only for medical research.

4.2.3 Application Software

Application software is one or more software programs, such as scientific and multimedia applications, that are installed on grid nodes. These programs are collections of instructions describing tasks or set of tasks to be carried out by a node. Appli-

cation software is loaded into the node's RAM and is executed in the CPU.

The most basic function for application software is to use the grid environment to allow the job to run on an available node on the grid. Application software is responsible for executing and running the job. It allows the end user to accomplish one or more specific jobs by employing the capabilities of a computer in fulfilling the requirements of a job the user wishes to perform. Each application software has requirements such as disk space, CPU speed and operating system.

Application software can be classified according to usage. For example:

- Middleware applications enable a node to communicate with other components.
- Scientific applications allow users to perform scientific experiments and calculations.
- Management applications allow users to manage projects, documents and databases.
- Word Processing applications allow users to create and edit documents.
- Programming languages applications allow the compilation and running of several programs.
- Multimedia Software enables the creation of images, audio and video.
- 2D/3D Graphics applications let them draw and create graphics and figures.

- Operating system applications.

  In the model, application software has to be able to receive the executable file (i.e. the job), run it and produce the result. It can also migrate between nodes inside or outside the grid depending on its policy in order to enable the required node to meet the user's requirements for the execution of the job. This capability reduces the number of rejected jobs. The software also may be uninstalled from the node by the grid if it affect its performance; the grid can nonetheless save the original copy of it without installing it. When the need for this application software arises again, the grid reinstalls it. This need may arise when the job requires this application software and no other node has it, or the node that does have it is busy.

4.2.4 Data

Data is a piece of information stored in a grid node, and is used by application software to fulfil certain tasks. The application software can therefore access data either on a local node or remotely. The data (such as stock market financial data) may already be stored on one or more nodes, or it may come with the user job. In the model, each stored data has a unique name in order to allow users to describe it. The data is able to migrate from node to node within or outside the grid, depending on its policy, which may allow data to migrate, copy itself, be updated and changed, or be static.

4.2.5 Grid Job

A grid job is typically submitted for execution by an appropriate node on the grid. It may perform a calculation, execute one or more system commands, operate machinery or move or collect data. Sometimes the job has one of many other names such as "transaction", "work unit", "submission", all of which mean the same thing.

In the system, users need to describe their job requirements. These include job name, required software applications, required data, execution time and resource specifications (CPU count, speed, operating system, physical and virtual memory). In the system, jobs can also migrate from node to node within the grid environment in order to complete the job if a failure occurs.

The collection of jobs that fulfil the whole task is called the grid application. For example, a grid application can be the simulation of business scenarios such as stock market development that require a large amount of data as well as a high demand for computing resources in order to calculate and handle the great number of variables and their effects [49].

The emergence of grids is due to the needs of large-scale computing infrastructures for solving major computing and data-intensive problems in the fields of science, engineering, industry and business. The grid infrastructure provides different kinds of support to a wide range of applications which can be categorised as follows [11]:

- **Distributed supercomputing support** allows applications to use grids in

order to reduce their completion time.

- **High- throughput computing support** allows applications to use unused processor cycles in grids for loosely coupled or independent tasks in order to increase aggregate throughput.

- **On-demand computing support** allows applications to use resources in the grid that cannot be cost-effectively or conveniently located locally.

- **Data-intensive computing support** allows applications to use grids to gather information from distributed data repositories and databases.

- **Collaborative computing support** allows applications to use grids to establish human to human interactions via a virtual space.

- **Multimedia computing support** allows applications to use grids to deliver contents assuring end to end QoS.

4.3 Models

Models manage and control system objects to achieve all the desired objectives of the grid system. These are communication, termination, timing, failure and migration models. The role of each describes as follows.

4.3.1 Timing Model

Time is an important and interesting issue in grid computing, the aim of which is the exploitation of underutilised resources to achieve faster job execution times. Each node in the design has its own internal clock. Every node must therefore periodically synchronise its clock with that of the resource broker using network time protocol, and when they join the grid they also specify availability time; the period during which they will be available. All cooperating grids are available continuously from the time they join the grid until they need to leave it. If any participating grids need to do this, they must complete all jobs that are referring to the original grid.

The timing model is responsible for maintaining and controlling system time, thus preventing jobs from running for longer than they are allowed to, as well as assisting in handling failure. The execution of a job needs a certain period of time; most resources use time as a charging unit because it is easily quantifiable. It is therefore possible for users to provide expected times for job completion.

4.3.2 Communication Model

All objects in a grid system need some form of communication in order to perform their activities with the desired flexibility. Whether it is as simple as transferring a single packet between a client and server, or as complex as advanced coordination issues carried out over a network between hundreds of nodes, communication is essential.

There are two fundamental ways for grid nodes to communicate with each other regarding jobs: by passing messages known as Remote Procedure Calls (RPCs) [79], and by using shared memory. Message passing is a more popular paradigm than shared memory because it allows easier communication between multiple processor architectures, and has a larger number of supporting applications and software tools. When coordination is involved, each of these communication forms has a complimentary coordination style; message passing uses control-driven coordination (procedures and function calls are made between two processes, irrespective of whether they are local to a single machine or are hosted on different machines), and shared memory uses data-driven coordination (communication carried out by placing data inside the shared memory).

In the model, communication is responsible for transmitting a message from one object to another. The purpose of this communication is to exchange and transmit information and data between these objects. Information transmission is also necessary when dependencies are present. Communication also helps discover failures. Communication between sending and receiving objects is performed via RPC which may be either synchronous or asynchronous.

The model uses the latter type, which means a non-blocking send. In asynchronous communication, the use of send and receive operations do not block synchronous communication. Asynchronous communication is an alternative form that may be useful in situations when it is possible for an object to retrieve replies later.

Communication provides the means of coordination and cooperation between objects.

Because the design is client/server, the client makes requests to a grid service using a remote procedure call. When the request has been carried out, notification is sent back to the client, who can meanwhile make a new remote procedure call to that same service. RPC is a message-passing protocol that provides high-level communications. It is built on the eXternal Data Representation (XDR) protocol that standardises data representation in remote communications. This protocol converts the parameters and results of each RPC service provided. RPC consists of two distinct structures, the call message and the reply message. In this model, the client makes a procedure call (call message) to request a service from the server. When the request arrives, the server performs the requested service, and sends a reply (reply message) back to the client.

4.3.3 Termination Model

After a job has been submitted, it starts running on the nodes until termination. Usually a job finishes due to conditions such as normal, failure or user termination. The model uses conventional (as opposed to distributed) termination because most tasks in grid computing are executed in parallel, which is the preferred method. It allows objects working in parallel to complete one task; when any object has done so, it does not wait for the others to finish theirs. This type of termination saves

time and allows objects that have finished their tasks to embark on others while the remaining objects are still completing their jobs.

4.3.4 Failure Model

Failures in distributed systems can be unpredictable in that they can leave the job in one of many possible failed states. Failure entails the desired state or condition is not being arrived at. In the grid computing environment the probability of failure is high because the grid aggregates an immense amount of hardware and software components. Depending on the system's size, the probability of failure will therefore increase.

- The grid consists of extremely heterogeneous objects, which can lead to failure when they interact.
- Grid environments are extremely dynamic, with components constantly joining and leaving the system.
- Detecting failures in a dynamic and heterogeneous system such as a grid is very difficult.

The failure may be due to software or hardware crashes, process failure, communication delays, network failures or system performance degradation. Failures in grid computing are partial, meaning that some components fail while others continue to function. When hardware or software faults occur, jobs may produce incorrect

results or stop before they are complete. The model assumes that all system components, whether hardware or software, may fail at any time. Hardware failure may be in a node or network or in communication. Fault tolerance is an essential characteristic and a necessary function for grid environments in order to avoid the loss of computation time. Two mechanisms provide a fault tolerance model: failure detection and failure handling.

Checkpointing schemes allow a job to continue from the point of failure, avoiding the necessity of rerunning the whole job. This model is mainly used in computing systems to store the current state of the job. By switching to an earlier checkpoint, a system can reload the previous state and resume computation from the point of failure. Besides recovery, checkpointing also enables other features such as job migration, which allows a failed job to continue on another machine from the point of failure. When failure occurs, job migration is the best way to complete the submitted job.

In the design, the system can detect and handle, and thereby recover from, failure. Detection inside the grid happens at the point when each node sends a regular heartbeat to the monitor. The heartbeat contains a timestamp, node name, job status (if the node is running a job) and other optional information. If the monitor does not receive an expected heartbeat from any node, it puts that node in a SUSPECTED state, and sends an "are-you-alive" message to the node. If the node responds with a message, the monitor puts the node back in the ALIVE state

and continues as normal, but if it does not, the monitor puts the node in a FAILED state. This message helps the monitor detect the failure and recover it. When the node sends a heartbeat to the monitor, the latter compares the current state of the job with the last checkpointed state; if it is the same, the monitor assumes that the job has failed. If such failure occurs, the monitor informs the resource broker involved to take one of the following actions:

- Migrate the job in its current state to another suitable resource on the list as found by the resource broker in the resource discovery stage, in order to restart execution of the job from its last state.

- If only failed resources have appeared in the list, or if all the listed resources are currently busy, the resource broker will try to find another resource in another participating grid that can complete the job. If such a resource is not found it, it sends a message to the user to say that the grid cannot execute the job because of failure.

In order to detect failure outside the grid, when the original grid need a resource from another grid (i.e. external evolution), it sends a requesting message to all participating grids, to which they must all reply. If a particular grid does not do so, the original grid put it in a SUSPECTED state and sends an are-you-alive message to it. If reply is received, the original grid returns it to the ALIVE state, but if it does not, it is put it in a STOPPED state to prevent it from being used. If, however, any other grid has a job for the original grid (i.e. Just in Time evolution),

the model assume that all these grids have a failure recovery capability and will therefore periodically send a message to the original grid to inform it of the status of its job. If the original grid does not receive this message, it assumes the job has failed. It therefore finds another resource that can execute this job from its last checkpointed state.

4.3.5 Migration Model

Migration is the capability of physical or virtual computational resources (software codes, portable notebook PC's, running objects, mobile agents and data) to move from one location to another across local or global networks. This very broad concept is central to distributed computing. It can be subdivided into personal, computer and computational migration. Computational migration addresses the movement of software [29, 83], with which the present work is particularly concerned. It can also be referred to as control, data, link or object migration [17].

Control migration supports moving a control thread from one machine to another and back again. Data migration allows the data required by the process to be passed through the network. Link migration is the fundamental concept of distributed objects, which transmits the end point of ability to move objects (codes) between different servers. It is also the basic concept in distributed computing.

Code migration (mobile computation) has come into popular use because it provides the capacity to link software components at runtime. This means that a soft-

ware component can move around on different servers across the network in order to execute tasks. From the point of view of the execution state, code migration can be weak or strong [18]. Migration is a central concept of grid computing. This model use weak and strong migration. Weak migration means to allow the code to move across nodes so that it sometimes has initialisation data attached, but never the execution state (in other words, the state of the computation is lost at the originating node). This happens when the user submits the job to a grid (i.e. to a resource broker), and also when the resource broker submits the job to grid nodes or to any adjoining grids. Strong migration is the capability of a computational environment to migrate the code and execution state (the context of execution) to restart at a new resource. The execution state includes running codes, program counters, saved processor registers, return addresses and local variables. This happens between grid nodes and between the cooperation grids in cases of partial failure, to evolve the grid or when the resources are not available at the current node for any reason. In the model, strong migration can occur when the application software and/or data is migrated from one node to other inside or outside the grid. This migration will depend on the resource strategies that occur during both external and internal evolution.

4.4 Strategies

Strategies or policies are sets of rules and principles stated by one or more owners or administrators of grid or resources that stipulate how those grids or resources

can be accessed and used. They determine how a job should be completed, how the resources should used, how security is implemented in the grid and how the grid manages and protects its resources. Policies in the design can be on both resources and grids.

- **Grid policies**

  These are formulated by grid administrators and is enforced by resource brokers. They determine how the grids are accessed, used and how they manage their resources. They also define how grid resources are selected - for example, they determine the choice of the lowest load resource from the suitable resources list in order to execute the job.

- **Resource policies**

  Resource owners have the right to determine their resource's governance policies and define how their resources can be used. Those strategies will be on node, data and application software. Resources policies are stored in the information service and are used by resource brokers. Resources strategies can be static or dynamic. A static policy is fixed: it does not allow data and/or application software to migrate to other nodes in the grid environment. In other words, the node's data is read-only and the node's application software is use-only. By contrast, a dynamic policy gives resource owners more options by which to decide their polices for all node components (data and application software) by determining the policy for each component separately. This

strategy may allow to data and application software to migrate from node to node in grid environments.

4.5 Summary

This chapter described the components of the system, which consists of computational model objects, models and strategies. This chapter also sets out the behavioural properties of the system's objects and the interactions between them by describing how those objects interact using the models listed. It described how the components can communicate in asynchronous communication and using the RPC technique. This chapter also explained how the objects terminate using the conventional termination model, and how these objects can migrate using the GridFTP.

Chapter 5

Evolution-Based Grid Design

Objectives

- To explain evolution-based grid design, its components and their functions.
- To describe how this grid design supports both types of evolution.
- To show how grids communicate with each other.
- To describe how resource broker with portal insulate the user from the complexities of the global grid.
- To describe how the design monitors the grid environment.
- To describe how this design deals with failure.

5.1 Introduction

Grid computing has emerged as a new paradigm for distributed computing. It promotes the sharing of distributed resources that may be heterogeneous in nature and belong to different administrative domains, and it enables scientists and engineering professionals to solve large scale computing problems.

Researchers have become greatly interested in dynamic distributed systems because of new computing paradigms such as grid and peer-to-peer computing, which have increased in popularity because dynamic distributed systems are especially useful for solving large-scale problems requiring vast computational power; such systems allow processes to move and run on whatever resources are suitable.

The design is based on the computational model for grid computing, which is discussed in the previous chapter. This chapter will give an overview of the design structure in Section 5.2 and a description of the components of the grid design and their functions which are user in Section 5.3, evolution-based resource brokers and their design in Section 5.5, information service in Section 5.6, node in Section 5.7, grid in Section 5.8, and monitor in Section 5.9.

5.2 Structure of Evolution-Based Grid Design

The design of distributed systems refers to their structure in terms of separately specified components. Many approaches to distributed computing, including client-

server architecture and peer to peer computing, have been proposed. The proposed system uses client/server architecture because it is the most suitable type for distributed systems and heterogeneous environments [26].

Client/server is a network computational architecture consisting of clients and servers that run on the software and hardware appropriate to their functions. They are separate logical entities that work together over a network to accomplish a task. A client is a front-end that the user sees and interacts with; it is a computer system that sends a message over the network to another system (the server) to access remote services. The server is a computer system responsible for receiving a request from the client and performing all the coordination necessary to fulfil that request. There are two types of client/server architecture: two-tier and three or multi-tier. In two tier architecture, the user interface is usually located on the client, which is a fat client because most of the necessary applications are located on it. The client sends a request to the server, which processes it and returns the results. Three-tier architecture usually consists of a client, an application server (the "server") and a data server (the database). The client runs the user interface while the server runs process management services and the database provides database management functions. The client sends the request to the server, which then forwards it to the database that returns the data to the server. The design uses the three-tier model.

Fig. 5.1 shows an evolution-based grid design which consist of a user, portal, resource broker, information service, a number of nodes, a monitor and other grids.

In what follows will describe the role of each.

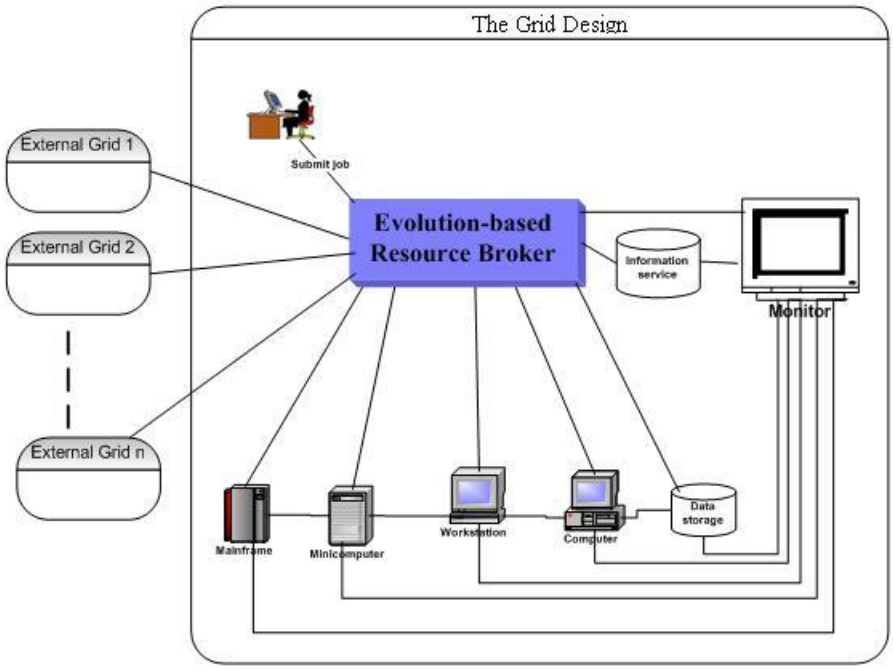

Figure 5.1: The Evolution-Based Grid Design

5.3 User

The users have jobs that must be executed using the grid's resources. Users access the grid through the grid portal, a software interface running on their own computer. The user must describe the job requirements through this portal in order to send these requirements to the resource broker, which will find the node that meet these requirements in order to execute the job. The user should describe the node's hardware specification such as CPU count and speed, required application software, required data and job execution time. Users have to define contact information such

as an e-mail address so as to enable grids to contact them and inform them of the state of their jobs.

5.4 Portal

Grid portal is a virtual computing resource acting as interface employed by grid users to access grids; the portal is the effective gateway to grids. A portal has various implementations that hide a grid's complexity from users through a simple interface. Web and grid portals provide similar services to users: for instance, web browsers provide a single interface through which to access internet resources, whereas grid portals are used to describe and submit jobs that will use the services provided by grid nodes or cooperation grids. In the design, grid users need no knowledge of job description languages because the portal allows users to describe their jobs simply, by just filling in the required fields for describing their jobs. This portal will then translate these requirements into the job submission description language in order to allow resource brokers to understand them and find a suitable resource.

5.5 Evolution-Based Resource Broker

A resource broker is a central component in grid computing, being linked with all other grid components. The resource broker developed in this research differs from the existing ones because it is based on the migration framework and enables the grid

to communicate with others. It thereby controls external and internal evolution. Its importance is due to its key role in building an effective dynamic grid environment by automating the user's job submission process and node selection.

The basic tasks performed by a grid resource broker include node discovery and selection, migration of application software, data and/or jobs if required, node reservation, job transferral to nodes and the return of the result output to the user. In general, all access to the nodes available in the grid is controlled by a resource broker, who also controls all jobs submitted to its node or to other grids. The resource broker therefore has the job of registering and updating any new hardware or software, which is why it is linked to all other grid components. It must have all the information regarding all the nodes registered with it; it must therefore source all this information from information services. It must also know all the grids that can cooperate with it and who have given communication permission to this resource broker; the latter will have saved this list of grids in its information service and updated it when any new cooperative event occurs. All these transitions are invisible to the user, who views the grid as a single huge and powerful computer.

Migration feature enables the satisfaction of requirements for grid jobs, as well as supporting grid evolution, enhancing the performance of running jobs by migrating data to the execution host (thus reducing communication), and solving load balancing problems. Migration is the solution in each of the following cases:

- Nodes that meets the job hardware requirements but do not have the required

application software.

- Nodes that meets the job hardware requirements but do not have the required data.

- Nodes that meets the job hardware requirements but have neither the required application software nor the data.

In order for the resource broker to fulfil all these functions, it requires a special design. The Evolution-based Resource Broker therefore consists of many components, as shown in Fig 5.2.

These are discoverer, communicator (including reserver, user informer and dispatcher and receiver), evolution maker (including internal, permanent and just-in-time evolution), and failure recoverer.

5.5.1 Discoverer

Discoverer functions start after the resource broker has received the job requirements including the required hardware specification, application software and/or data and execution time from the user. The discoverer then communicates with the information service, sending a query regarding the availability at the required time of the necessary hardware, software and data. It does this by performing the following algorithm (shown in Fig. 5.3):

- Hardware Specification Verification

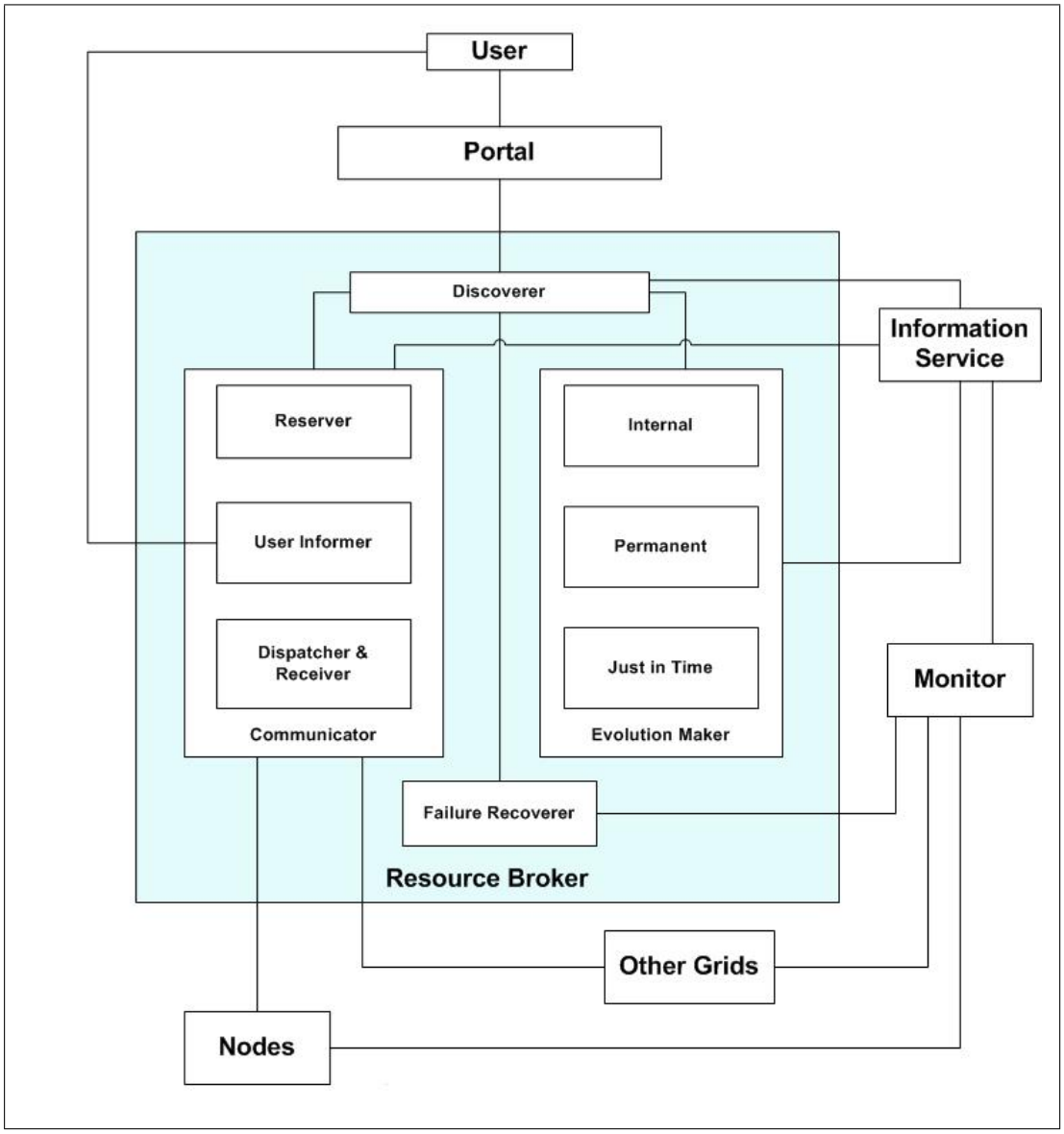

Figure 5.2: Resource Broker Design

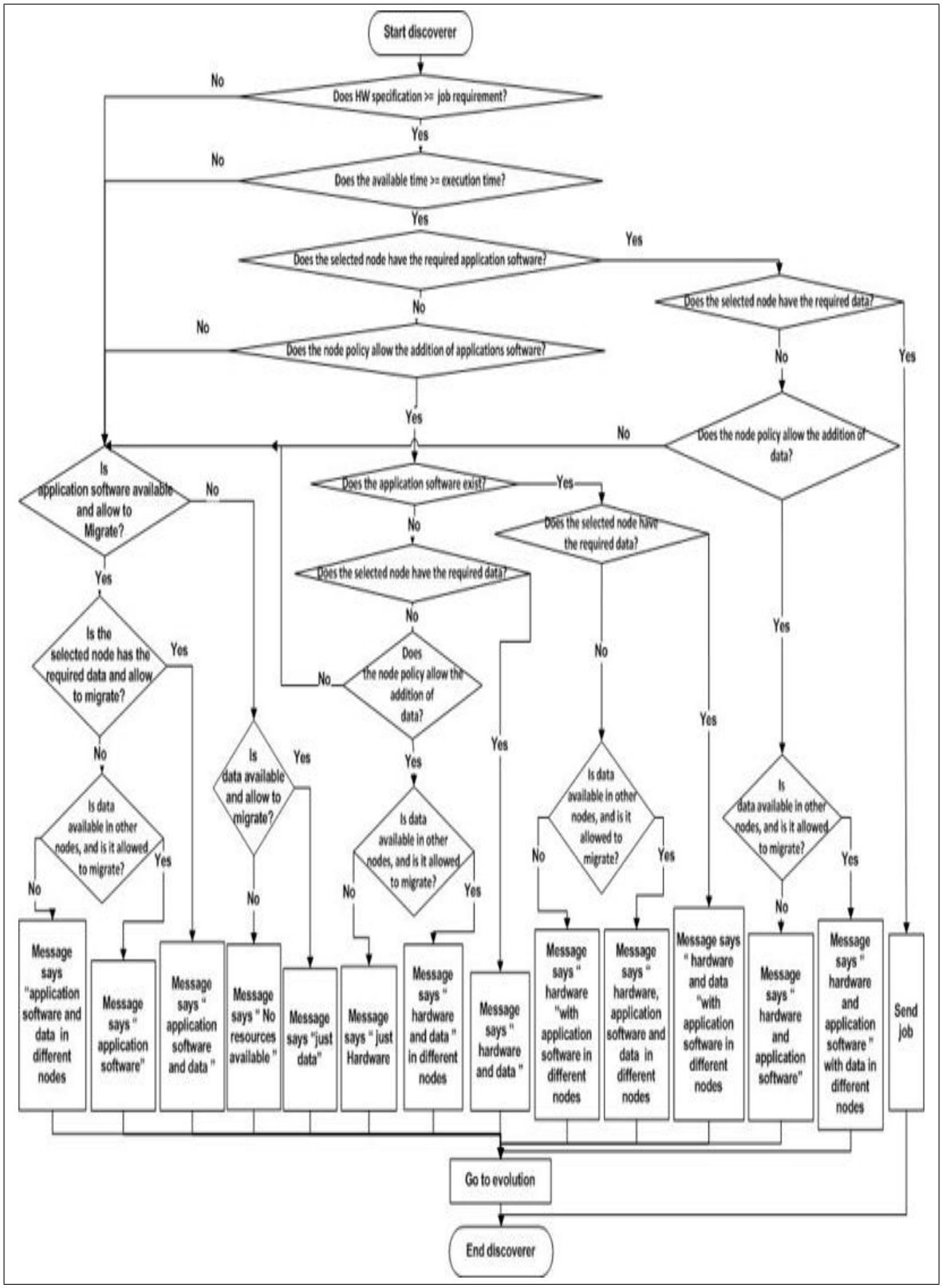

Figure 5.3: Resource Discoverer Algorithm

The discoverer determines whether the given hardware specification is equal to or less than that stored in the information service. If it is, the discoverer compares the available times for the suitable nodes with the time required to execute the job. If these match, it adds them to the list and verifies the other requirements. If the discoverer does not find the required hardware specification, however, it transfers the work to the evolution maker with a message containing what the grid has, if the grid indeed has the required application software and/or data.

- Application Software Verification

  After verifying the hardware specification, the discoverer compares the required application software with that of the listed nodes. If found, the discoverer verifies the required data. If the discoverer does not find the required application software on any of the listed nodes, it transfers the work to the evolution maker with a message detailing what the grid has.

- Data verification

  If the job requires data, the discoverer compares the required data name with that of the listed nodes; if it does not find the required data name, it transfers the work to the evolution maker; otherwise it transfers it to the communicator.

If the discoverer need to transfer the work to the evolution maker, it performs the following before this transformation:

The discoverer searches the information service for the existence of and policies regarding all the job requirements (e.g. do they allow addition, migration or copying?) and then sends the result of this search to the evolution maker as one of the following messages:

1. The grid has hardware only.
2. The grid has hardware and data on the same node.
3. The grid has hardware and application software on the same node.
4. The grid has hardware and application software on the same node and data on different nodes.
5. The grid has hardware and data on the same node and application software on different nodes.
6. The grid has hardware, application software, and data on different nodes.
7. The grid has hardware with application software on different nodes.
8. The grid has hardware and data on different nodes.
9. The grid has application software and data on the same node.
10. The grid has application software and data on different nodes.
11. The grid has application software only.
12. The grid has data only.

13. The grid has no resources available.

5.5.2 Communicator

Once the discoverer verifies that the grid has all the job requirements, it transfers the work to the communicator which consists of reserver, user informer, and dispatcher and receiver.

5.5.2.1 Reserver

The reserver is responsible for reserving the resources that satisfy the job requirements in order to execute the job at the required time. The reserver will put the resource in a busy state for this job in the information service at the required time (the job execution time).

5.5.2.2 User Informer

The user informer is responsible for informing users of anything that happens regarding their jobs (for example, that the grid cannot execute the job, the job has started running on the node, the job has finished, or the job has failed).

5.5.2.3 Dispatcher and Receiver

Once the job start execution time is reached, the dispatcher and receiver send the job input file to the reserved node in order to execute the job until the expiry of the execution time. The job input file contains all the information necessary to explain

how to run the job, such as time, application software and/or data necessary and location if required. Once the job has started running on the node, the dispatcher and receiver mark this node's status as busy in the information service. The dispatcher and receiver then wait until the execution time has elapsed, after which they send a message to the node to stop the job. Whether the job has completed or not, it is stopped; the result is retrieved and the user informed that the job has finished. The node's status is then changed to idle in the information service.

The other function for the dispatcher and receiver is to register any node that need to register with the grid or any grid that need to give a communication permission to this grid. This registration is by receipt of a message from the node or the grid and the subsequent saving of the message's information in the information service.

5.5.3 Evolution Maker

The evolution maker functions appear when the discoverer does not find some or all of the job requirements. In this case it transfers the work to the evolution maker, who attempts to find the requirements needed by one of its evolution maker components, whether internal, permanent or just-in-time. The evolution maker determines if the work is relevant for the internal, permanent, or just-in-time evolution makers by quickly checking the messages from the discoverer; if the requirements are scattered across different nodes on the grid, the work is designated to the internal evolution

maker, if the grid has some of the requirements including the required hardware, the work is given to the permanent evolution maker, and if the grid does not have the required hardware, the work is allocated to the just-in-time evolution maker as shown in Fig. 5.4.

If any of the evolution makers migrate application software or data, it immediately updates the information service by adding this change to the node specification and deleting it from the node that the application software or data was moved from.

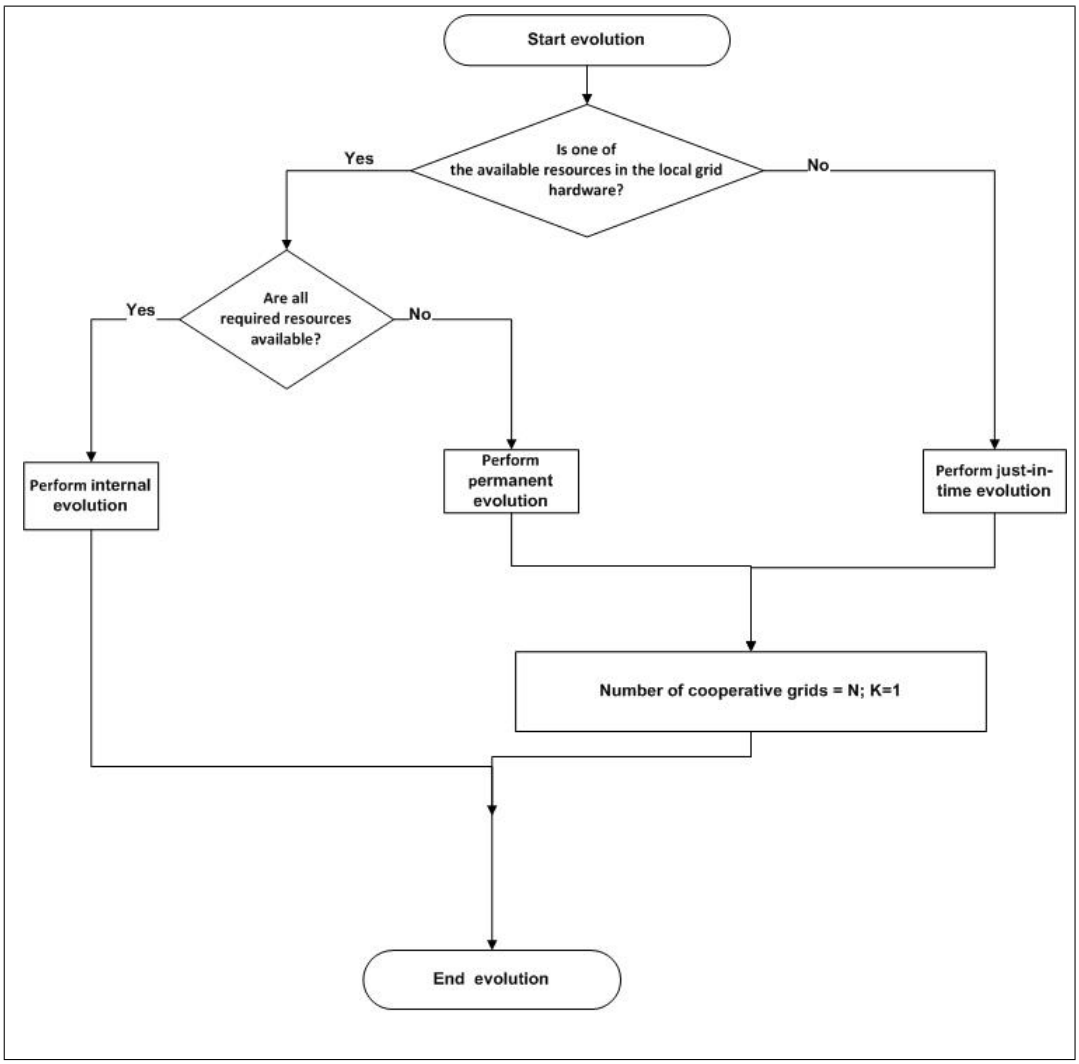

Figure 5.4: Evolution Maker Algorithm

5.5.3.1 Internal Evolution Maker

If the evolution maker determines that the work is suitable for the internal evolution maker, it transfers the work to that component in order for it to perform its work, as described below and shown in Fig. 5.5. This means that the message would have been one of the following:

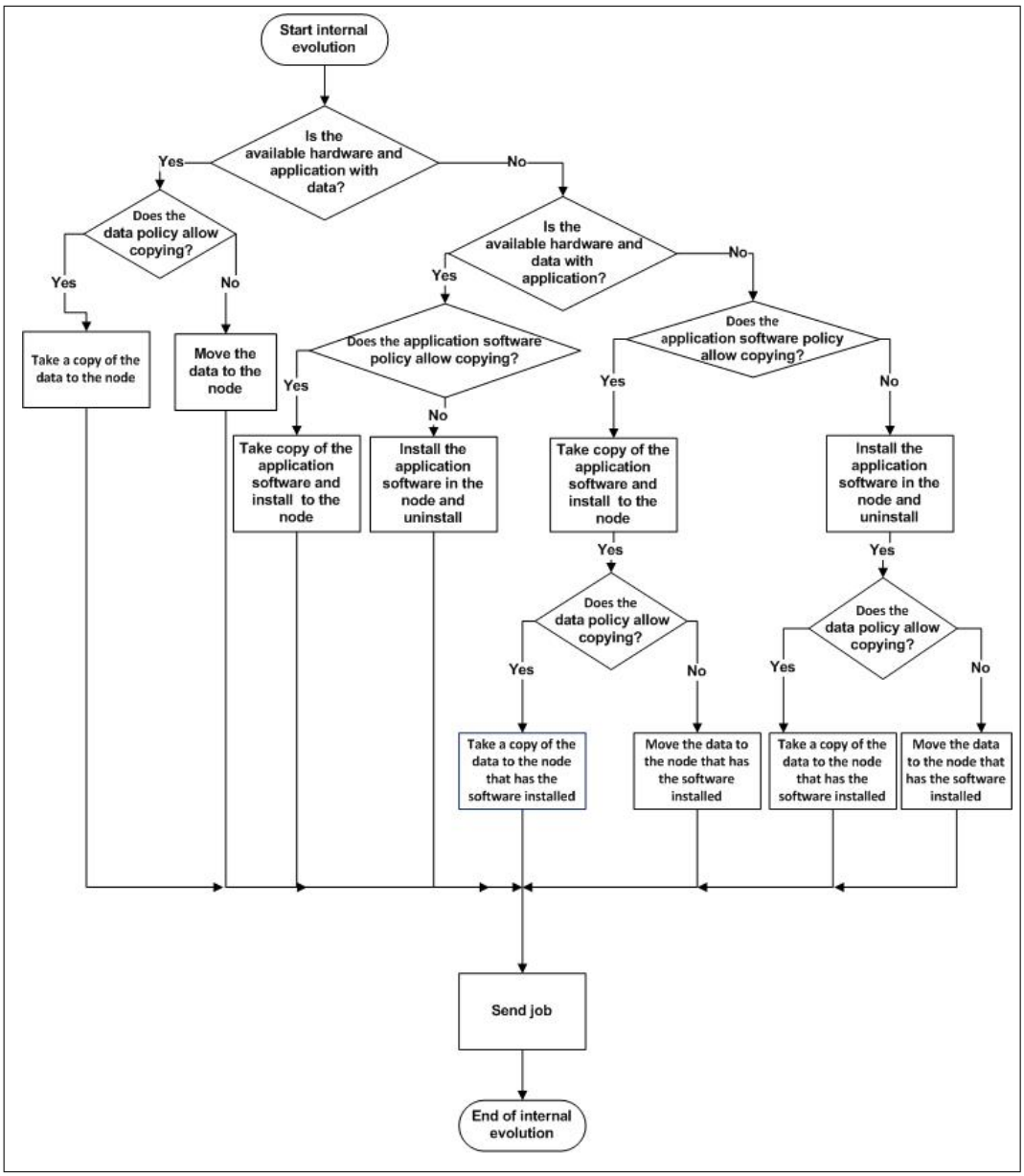

Figure 5.5: Internal Evolution Algorithm

- The grid has hardware and application software on the same node and data on different nodes.

  In this case, the internal evolution maker will instigate internal evolution by migrating the required data from its node to the node that has the required hardware and application software. It then transfers the work to the communicator in order to reserve this node and send the job.

- The grid has hardware and data on the same node and application software on different nodes.

  In this case the internal evolution maker instigates internal evolution by migrating this application software from its node to the node that has the required hardware and data and then transferring the work to the communicator, which reserves this node, sends the job and informs the user.

- The grid has hardware, application software, and data on different nodes.

  In this case, the internal evolution maker instigates internal evolution by migrating the application software and data from their nodes to the node that satisfies the hardware requirements and then transfers the work to the communicator, which reserves this node, sends the job and informs the user.

Note: The evolution maker migrates according to the policy, copying or moving as appropriate. In the design, the priority is for copying. However, if the internal evolution maker does not receive the work from the evolution maker, the work is

allocated either to the permanent or the just-in-time evolution maker.

5.5.3.2 Permanent Evolution Maker

If the evolution maker determines that the work should be allocated to the permanent evolution maker, the later performs its work accordingly as described below and shown in Fig. 5.6. In this case, the message would have been one of the following:

- The grid has hardware and application software on the same node.

  If the permanent evolution maker has found this message, it sends a broadcast message to all the cooperating grids listed in the information service and asks them for the required data. It should receive the same number of replies as there are cooperating grids. If it does not, it informs the failure recoverer. In any case, it searches all the replies for a confirmation that the sender has the required data; if it finds one, it instigates a permanent evolution by migrating this data according to its policy, taking a copy or migrating it from its grid to the node that has the hardware and application software and then transferring the work to the communicator, which reserves this node, sends the job and informs the user.

- The grid has hardware and data on the same node.

  If the permanent evolution maker receives this message, it sends a broadcast message to all cooperating grids listed in the information service asking them for the required application software. It should receive the same number of

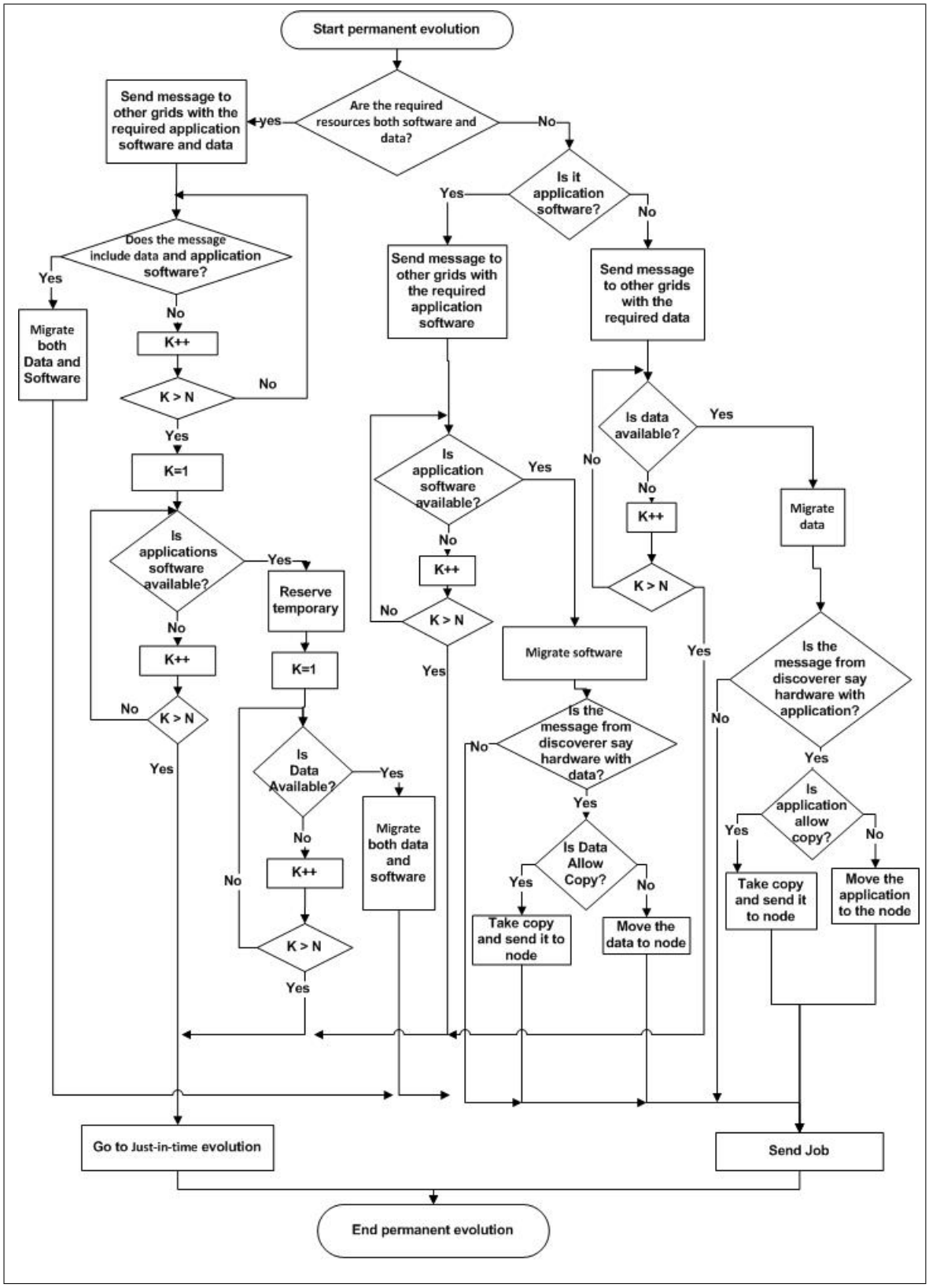

Figure 5.6: Permanent Evolution Algorithm

replies as there are cooperating grids. If it does not, it informs the failure recoverer. In any case, it searches all the replies for a confirmation that the sender has the required application software; if it finds one, it instigates a permanent evolution by migrating this software according to its policy, taking a copy or migrating it from its grid to the node that has hardware and data and then transferring the work to the communicator, which reserves this node, sends the job and informs the user.

- The grid has hardware and application software on different nodes.

  If the permanent evolution maker receives this message, it sends a broadcast message to all cooperating grids listed in the information service asking them for the required data. It should receive the same number of replies as there are cooperating grids. If it does not, it informs the failure recoverer. In any case, it searches all the replies for a confirmation that the sender has the required data; if it finds one, it instigates a permanent evolution by migrating this data according to its policy, taking a copy or migrating it from its grid to the node with the hardware as appropriate, and likewise copying or migrating the application software according to its policy. It then transfers the work to the communicator, which reserves this node, sends the job and informs the user.

- The grid has hardware and data on different nodes.

  If the permanent evolution maker receives this message, it sends a broadcast message to all cooperating grids listed in the information service asking them

for the required application software. It should receive the same number of replies as there are cooperating grids. If it does not, it informs the failure recoverer. In any case, it searches all the replies for a confirmation that the sender has the required application software; if it finds one, it instigates a permanent evolution by migrating this software according to its policy, taking a copy or migrating it from its grid to the node with the hardware as appropriate, and likewise copying or migrating the data according to its policy. It then transfers the work to the communicator, which reserves this node, sends the job and informs the user.

- The grid has hardware only.

  If the permanent evolution maker receives this message, it sends a broadcast message to all cooperating grids listed in the information service asking them for the required application software and data. It should receive the same number of replies as there are cooperating grids. If it does not, it informs the failure recoverer. In any case, it searches all the replies for a confirmation that the sender has the required software and data; if it receives one or more replies, it searches them for the requirements. The replies will be as follows; the list is given in the order of priority in which the permanent evolution maker searches them.

  1. Yes, I have all you ask for.

     If the permanent evolution maker receives this message from any co-

operating grid, it stops searching the other messages and migrates the application software and data depending on their policies as above, after which it transfers the work to the communicator which reserves this node, sends the job and informs the user. If the evolution maker does not receive such a reply, it searches for the following:

2. Yes, I have part of what you ask for.

   If the permanent evolution maker receives this reply from any participating grid, it firstly searches for the required application software in these messages. If found, it reserves it temporarily and searches for the required data. If that is found, it migrates the application software and the data from their respective grids to the node in the original grid that satisfies the job's hardware requirements. It then transfers the work to the communicator, which reserves this node, sends the job and informs the user. If one or both of these requirements are not found, it cancels any reservation it has made and transfers the work to the just-in-time evolution maker.

3. No, I have none of the requirements.

   The permanent evolution maker ignores such messages.

If all the messages that the permanent evolution maker received are "No, I have none of the required resources", the permanent evolution maker transfers the work to the just-in-time evolution maker.

5.5.3.3 Just-in-time Evolution Maker

If the work was transferred from the permanent evolution maker or came from the evolution maker, it is intended for the just-in-time evolution maker. Once this evolution maker receives the work from the permanent evolution maker, it performs its work as described below and shown in Fig. 5.7. If the work came from the evolution maker, the message from the discoverer must have been one of the following:

- The grid has application software and data on the same node.
- The grid has application software and data on different nodes.
- The grid has application software only.
- The grid has data only.
- The grid has no resources available.

If the just-in-time evolution maker has found one of these messages, it sends a broadcast message to all the cooperating grids listed in the information service and asks them for all the job requirements in same node. The cooperating grids reply with one of the following messages, and the just-in-time evolution maker searches them in the following order of priority:

1. Yes, I have everything you ask for. In this case, the just-in-time evolution maker transfers the work to the communicator in order to send the job to

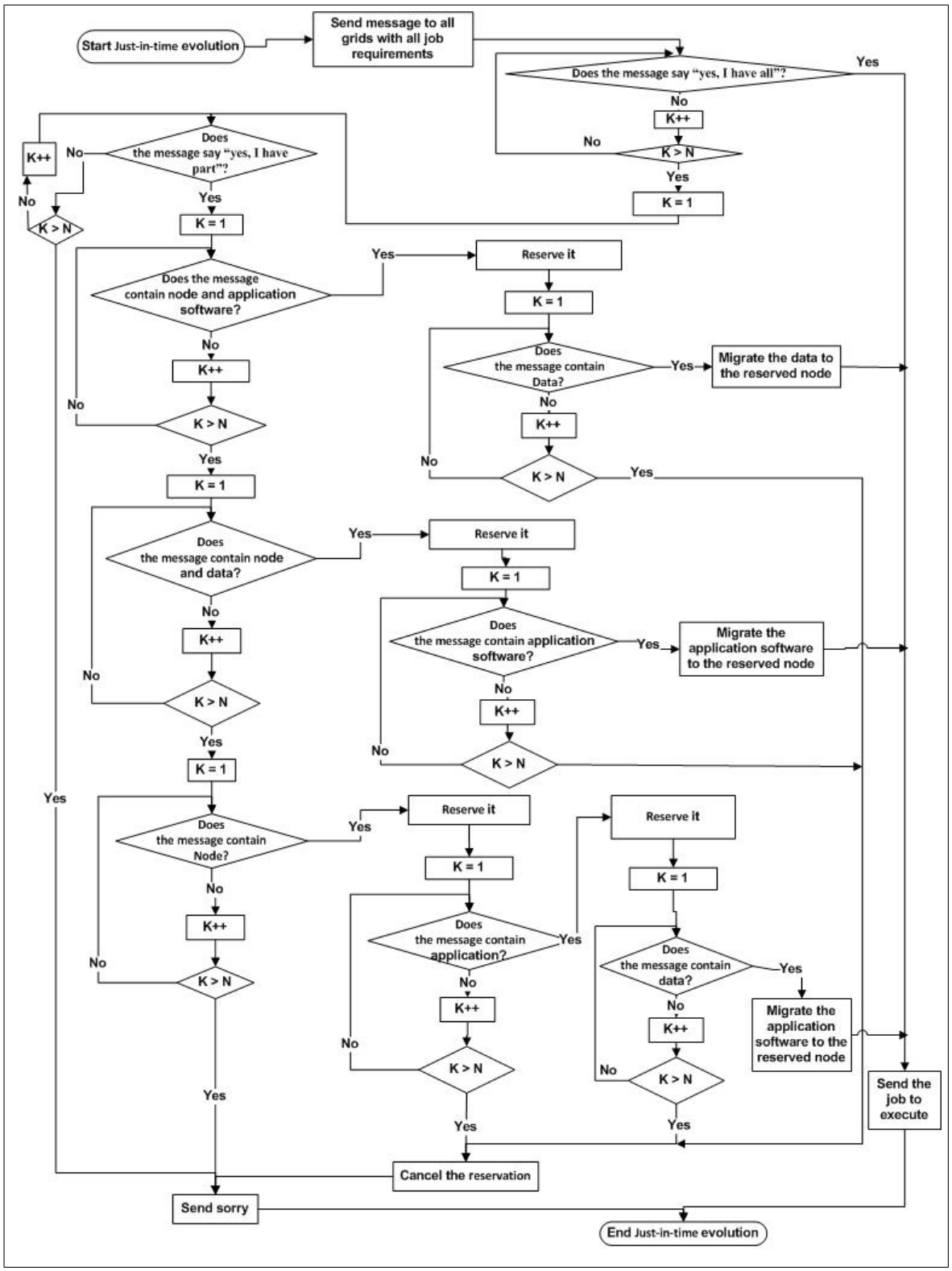

Figure 5.7: Just-in-Time Evolution Algorithm

this grid for execution on this node and return after the execution time has elapsed.

2. Yes, I have have part of what you ask for. This part will be one of the following:

   - Node with application software.

     In this case the just-in-time evolution maker searches the replies from the discoverer. If one is to the effect that the grid has data, it migrates it depending on its policy (if copying is allowed it does so, and if not it migrates the data from the original grid to the grid that has the node containing the application software. It then transfers the work to the communicator in order to send the job to this grid. If the reply from the discoverer is negative, the just-in-time evolution maker searches the replies received from other grids for the required data. If found, it migrates that data to the grid containing the required node and application software in order to execute the job when the communicator sends it it. But if the data is not found, the just-in-time evolution maker tells the user informer which in turn apologises to the user that the grid cannot execute its job.

   - Node with data.

     In this case the just-in-time evolution maker searches the replies from the discoverer. If one is to the effect that the grid has application software, it migrates it depending on its policy (if copying is allowed it does so,

and if not it migrates the software from the original grid to the grid that has the node containing the data. It then transfers the work to the communicator in order to send the job to this grid. If the reply from the discoverer is negative, the just-in-time evolution maker searches the replies received from other grids for the required software. If found, it migrates that software to the grid containing the required node and data in order to execute the job when the communicator sends it. But if the application software is not found, the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job

- Node with nothing.

  In this case the just-in-time evolution maker searches the replies from the discoverer. If one is to the effect that the grid has application software and data on the same node, it migrates them depending on their policies (if copying is allowed it does so, and if not it migrates them from the original grid to the grid with the node. It then transfers the work to the communicator in order to send the job to this grid. If the reply from the discoverer is negative, the just-in-time evolution maker searches the replies received from other grids for the required software and data in the same message. If found, it migrates them to the grid containing the required node in order to execute the job when the communicator sends

it. But, if it is not found them together in the same message, will search for them separately, if it found them, it will migrate them to the grid that has the required node and then transfer the work to the communicator. But, if it is not found them, the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job.

- Application software with data.

  In this case the just-in-time evolution maker may use this message if it finds a previous one containing hardware. But if the hardware is not found, the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job.

- Application software.

  In this case the just-in-time evolution maker may use this message if it finds a previous one containing hardware and data. But if these are not found, the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job.

- Data. In this case the just-in-time evolution maker may use this message if it finds a previous one containing hardware and application software. But if these are not found, the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job.

3. No, I do not have what you need.

   If the just-in-time evolution maker receives this message, it ignores it.

If all the replies from the cooperating grids are (No, I do not have what you need), the just-in-time evolution maker tells the user informers which in turn apologises to the user that the grid cannot execute its job.

5.5.4 Failure Recoverer

The work of the failure recoverer will be activated in one of the following two cases and shown in Fig 5.8:

- If the permanent evolution maker or just-in-time evolution maker does not receive the expected number of replies from other grids to their requests, it informs the failure recoverer, which sends a broadcast Are-You-Alive message to all grids listed in the information service. If any grid does not reply positively, the failure recoverer puts it in a Failed state in the information service.

- If the monitor detects any failure in the nodes or in other grids, it informs the failure recoverer accordingly and gives it the job status with the remaining time to execute the job. The failure recoverer then contacts the discoverer to find another node on which to complete the job from its last checkpoint state.

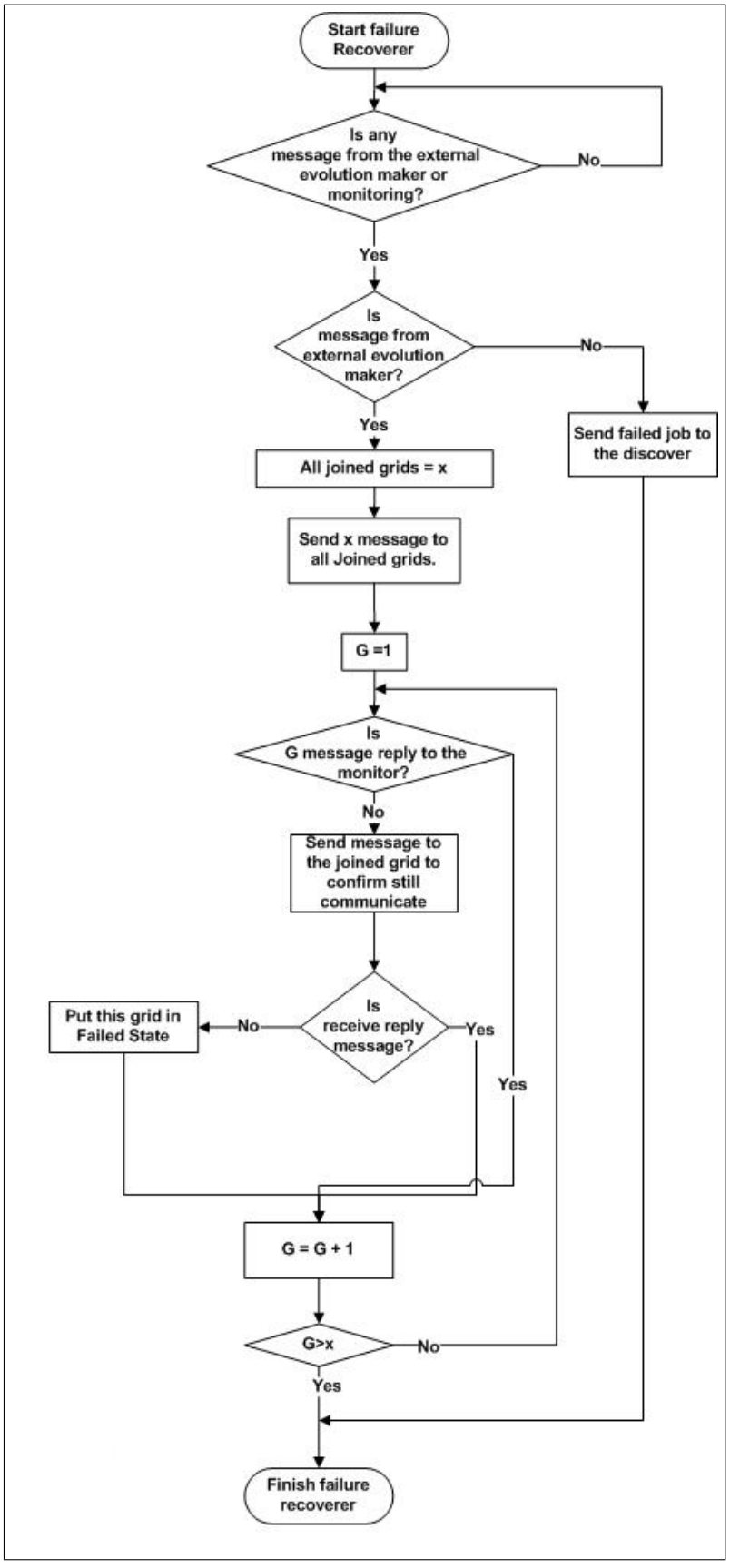

Figure 5.8: Failure Recoverer Algorithm

5.6 Information Service

The information service is an indispensable component of grid computing. It is an index service containing information regarding all the nodes in the grid and all the jobs running on those nodes.

This information can be either static or dynamic; the former regards the hardware specification and the operating system, and dynamic information concerns the node's available time, disk space, the job currently running, application software, and data.

This information is updated by the monitor and the resource broker, and is used by the latter in order to execute the job. The other type of information that the information service should contain is a list of all the grids that have given their own grid communication permission, in order to use them to execute jobs or support evolution by migration as described in the relevant chapter (chapter 3). This list of grids will be updated by the resource broker who has received permission to communicate.

5.7 Node

Nodes are the most basic components in the grid. A node is a collection of work units that can be shared and can provide some capabilities. Each node has to communicate with a resource broker, a monitor and with other nodes in the grid environment

in order to help other nodes to do the work requested by the resource broker so as to reduce the resource broker's bottleneck problem and exchange application software and data by migration. Each node that must be added to the grid must register itself with the resource broker and send all its specifications with its available times to the grid. In order to communicate, it must contain middleware application software. This communication occurs via network capabilities. Each node has its own application software, policy and data. A node is therefore able to execute a job. Once the nodes receive the monitor's broadcast, each node sends its heartbeat containing a timestamp, node name and notification of its status to the monitor in order to update its information in the information service at regular intervals. The status notification includes job name, job status, job start time and job execution time if the node has a job. Nodes in the design are responsible for receiving jobs that come from the resource broker. The node will thereupon recognise the job by saving its name and generating a work space directory structure designed to help both software applications and the node itself to find the data they need to support the job's execution. Then, by using migration if defined, the node will perform the migration actions, whereupon it will be capable of executing the job. The node allows the resource broker to control the jobs by stopping them after their completion of the execution time.

5.8 Grids

Grids are collections of nodes connected via networks and managed by resource brokers. They promote the sharing of services, computing power and resources such as disk storage databases and software applications. In the design, grids can grant permission to communicate with other grids in order to exchange application software and/or data and even jobs if the chosen grid does not have the job requirements in its domain. In such cases it sends the job to other grids for execution.

5.9 Monitor

The environment in which grid computing operates is dynamic in nature because grid nodes are heterogeneous and belong to different organisations and administrative domains. The availability of nodes also varies over time. Grid jobs are, moreover, complicated and extremely large, so grid computing faces difficulties in managing the lifetime of jobs. To manage jobs at run time, a dynamic environment requires a sound structure to deal with any eventuality such as hardware or software failure and node unavailability that occurs after job submission and during execution. The design deals with this by adding a monitor in order to observe the grid environment and manage incidents such as these, and develop the resource broker to solve this problem.

The main functions of the monitor in the design are:

- Monitoring grid jobs by tracking job execution states.
- Detecting node failures in the grid environment in order to solve them.
- Detecting communication failures that may occur in other grids.

To deal with this, the monitor has been developed a monitor to carry out both internal and external monitoring.

5.9.1 Internal Monitoring

The monitor periodically sends a broadcast message to all nodes in the grid. The broadcast contains an *Are-You-Alive* message asking for job status if the node is running a job. The monitor then waits for a response from the nodes. Before sending this message, the monitor checks the information service to find out the nodes' numbers and those nodes running jobs, ensuring that all nodes have replied, and all that are running jobs have sent job status reports.

The responses contain timestamps, node names and available times; those nodes that are running jobs also send job names and statuses, job start and execution times. The monitor will firstly ensure that all nodes have replied. If any have not, the monitor sends another message to the same node to confirm its status. If it does not reply again, the monitor puts it in a FAILED state in the information service. The monitor also ensures that all nodes running jobs have sent their job statuses, repeating the request if they have not done so. If a node replies with its job's status, the monitor compares it with the previous state. If it is the same, the monitor asks

the node to send the job status again for confirmation, and if not the same, it saves the new state in the information service. But if the same, the monitor inform user that the job has failed. This message helps the monitor detect and solve failures.

If the failed node was running a job, the monitor calculates the remaining execution time by using the following equation:

`Completed time = time stamp - job start time.`

`Remaining time = job execution time - completed time.`

If the monitor does not receive the expected respond from any node, it considers this node as having failed, immediately putting it in a FAILED state in the information service and informing the resource broker of this failure by sending the failure recoverer a message containing the node name and the job that was running on this node with its last status and the time remaining. The algorithm for internal monitoring shown in Fig 5.9.

Once the failure recoverer receives this message it contacts the discoverer, which does one of the following:

- It migrates the job to another suitable node in the grid, in order to start the job from the point of failure.

- If no other nodes are available in the grid, or if all the other nodes are busy, the broker attempts to find a suitable node in another grid by sending a broadcasting message to all participating grids asking them for all job requirements. If found, it sends the job with its last status for completion, but if not found,

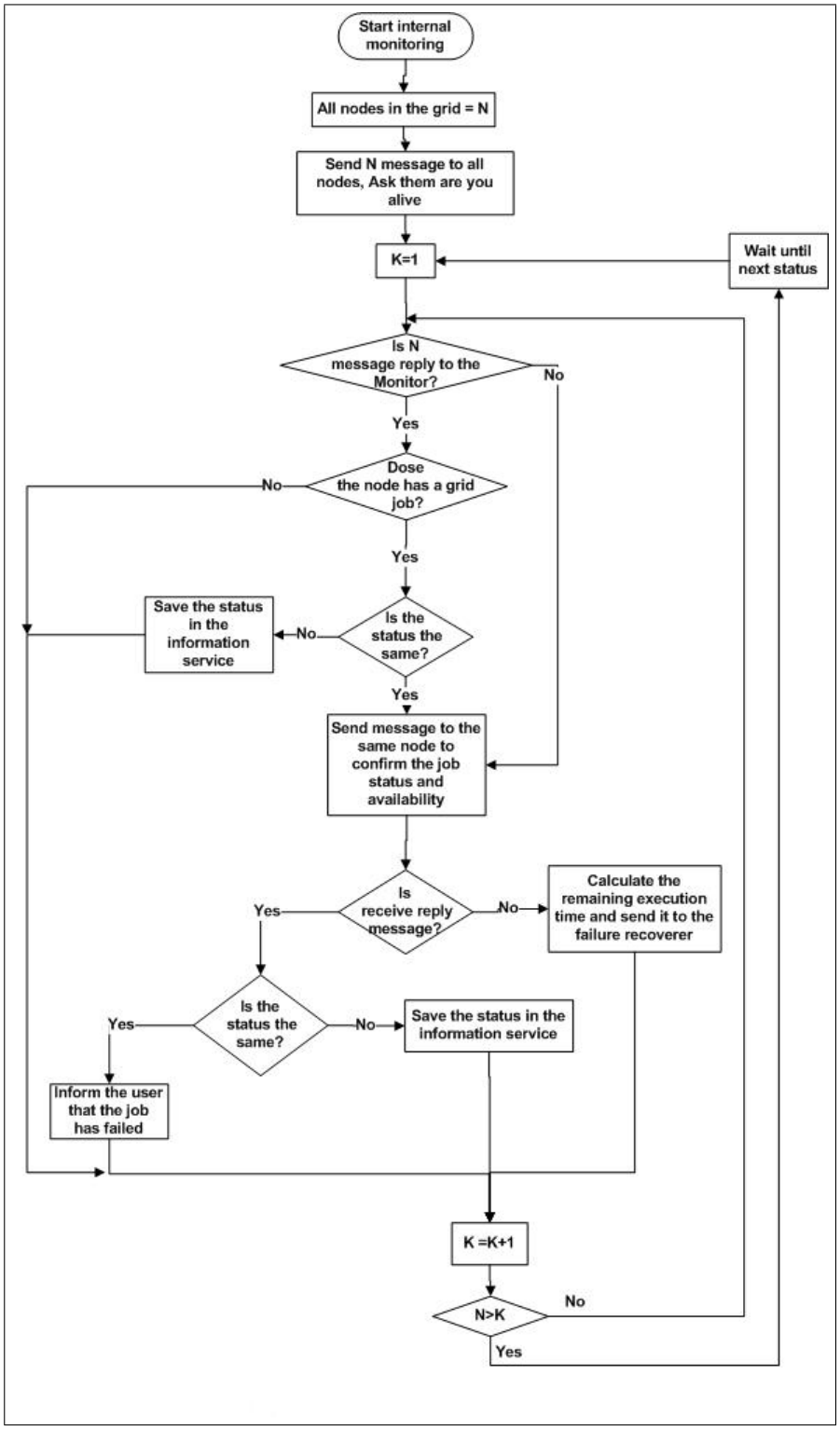

Figure 5.9: Internal Monitoring Algorithm

the resource broker sends a message to the user to say that the grid cannot complete the job because of failure.

5.9.2 External Monitoring

The monitor also tracks a job's state when it is being executed on another grid. When it notices in the information service that there is a job executed in other grid, it sends a message to this other grid asking it for the job status and saving the result in the information service. It does this for each job in this category.

Once the monitor receives the status from the other grid, it compares it with the previous status for that job. If it is the same, it sends another message to that grid to confirm the status. If the grid replies with the same status, the monitor considers that job has having failed, updates the information service and informs the user accordingly. If it is not the same, however, it saves this new status in the information service.

The monitor periodically requests job status reports until it receives one from the other grid saying that the job has finished. If the monitor receives no reply from the other grid, it calculates the remaining job execution time and sends a message containing the job name, status and remaining execution time to the failure recoverer to inform it that the grid has failed and updates information service as shown in algorithm 5.10. The failure recoverer then contacts the discoverer in order to find other nodes that can complete the job from its last state.

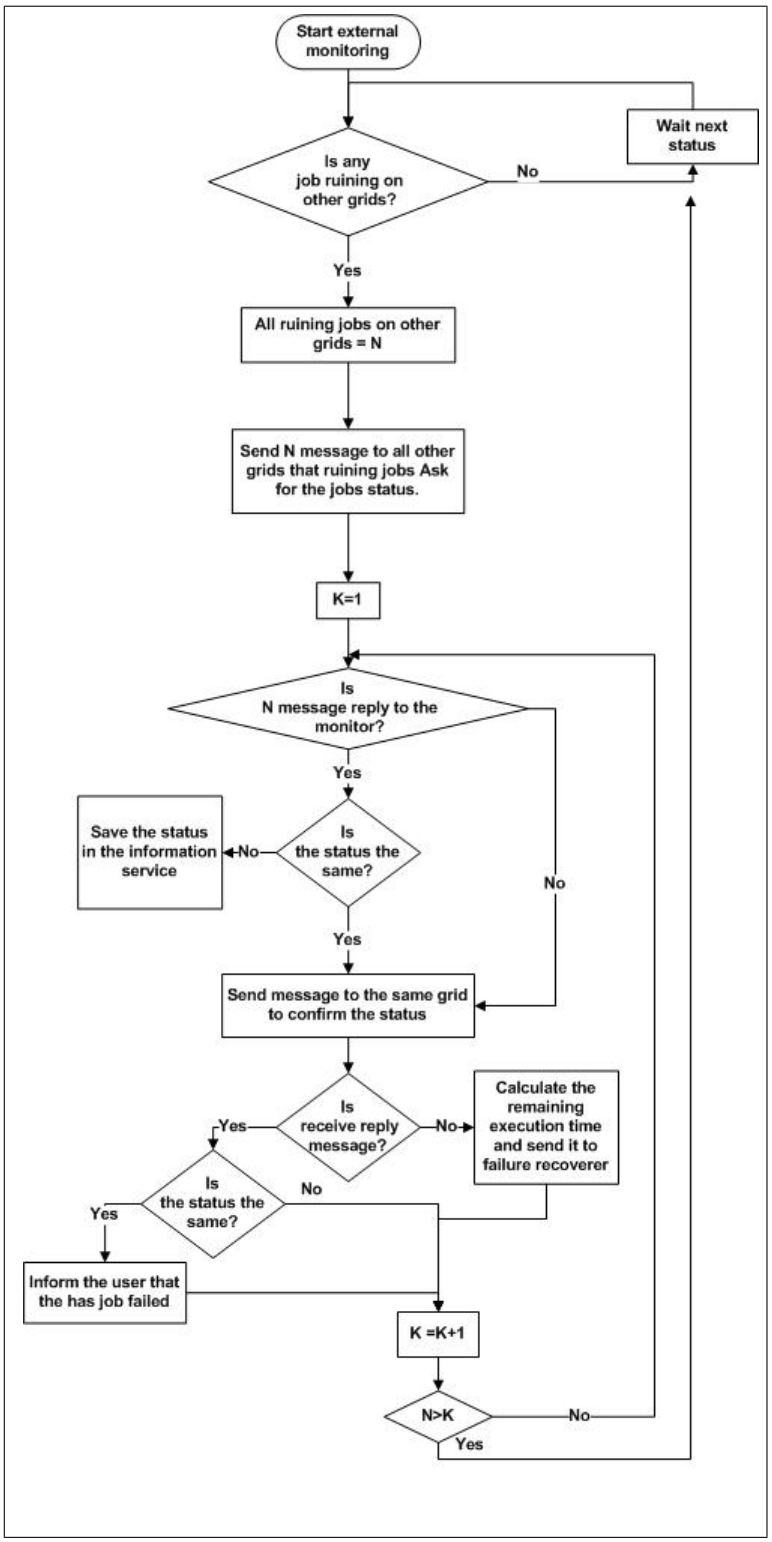

Figure 5.10: External Monitoring Algorithm

5.10 Summary

This chapter described the design and its components with their functions. It also showed how these components work together to support evolution, and explained how the portal insulates the user from the complexities of middleware, and how the resource broker performs internal and external evolution and attempts to fulfil job requirements through its powerful design by migrating the required application software and/or data to the required node or send the job to other grid to execute. It also showed how the monitor observes the grid environment and job states, both inside and outside the grid.

Chapter 6

EVOSim: A Simulation Tool for Grid Evolution

Objectives

- To present the simulation of the grid environment.
- To describe the simulation analysis and design.
- To present the simulation's implementation and operation.
- To demonstrate the simulation's validity.

6.1 Introduction

Simulation is defined as "the imitation of the operation of a real-world process or system over time" [8]. Simulation is beneficial in many ways, such as:

- A real grid infrastructure does not provide a repeatable and controllable en-

vironment for experimentation and evaluation of scheduling strategies, a situation accommodated by simulation.

- Simulation is effective in working with very large hypothetical problems that would otherwise require the involvement of a large number of active users, which is difficult to co-ordinate and construct on a real grid environment when conducting investigations [16].

There are few tools for simulating a grid computing environment. Among those that exist, there are MicroGrid [76], Simgrid [19] and Gridsim [16]. However, these tools do not support the scheduling and brokering performance evaluation, the functionalities or the evolution aspects of the system. Therefore, the simulation tool has been developed to check the results.

In the previous chapters, a framework for grid evolution was introduced. This chapter is concerned with the design and implementation of EVOSim, a simulator written in Oracle 10g based on the computational model and architecture for evaluating the performance of grid computing after evolution. This chapter gives an overview of simulation and why it is beneficial. This is followed by a description of EvoSim analysis and its design in Section 6.2; the implementation will then be presented in Section 6.3. The simulation validation will finally be outlined in Section 6.4.

6.2 EVOSim Analysis and Design

This section describe the analysis and design of EVOSim.

6.2.1 EVOSim Components

EVOSim's basic components are the system and its entities. The system is the class responsible for handling the interactions (i.e. communications) between different entities. An entity is the class that different entities are created from.

6.2.1.1 Entities

An entity is an object that can send and receive jobs. These entities can communicate with each other. Six classes of entities are created: users, grids, nodes, application software, data and jobs, as shown in Fig.6.1. All entities have certain features in common, such as unique names.

- User

  Users have jobs that need to be executed on the resources residing on the grid. The user sends a job to the grid in order to execute it and waits either for a result or for an apology that the grid cannot execute the job. However, the user must define the job's parameters by describing its requirements. Users all have names, assigned them when they register with the grid: a unique ID is generated by the system itself after registration.

- Grid

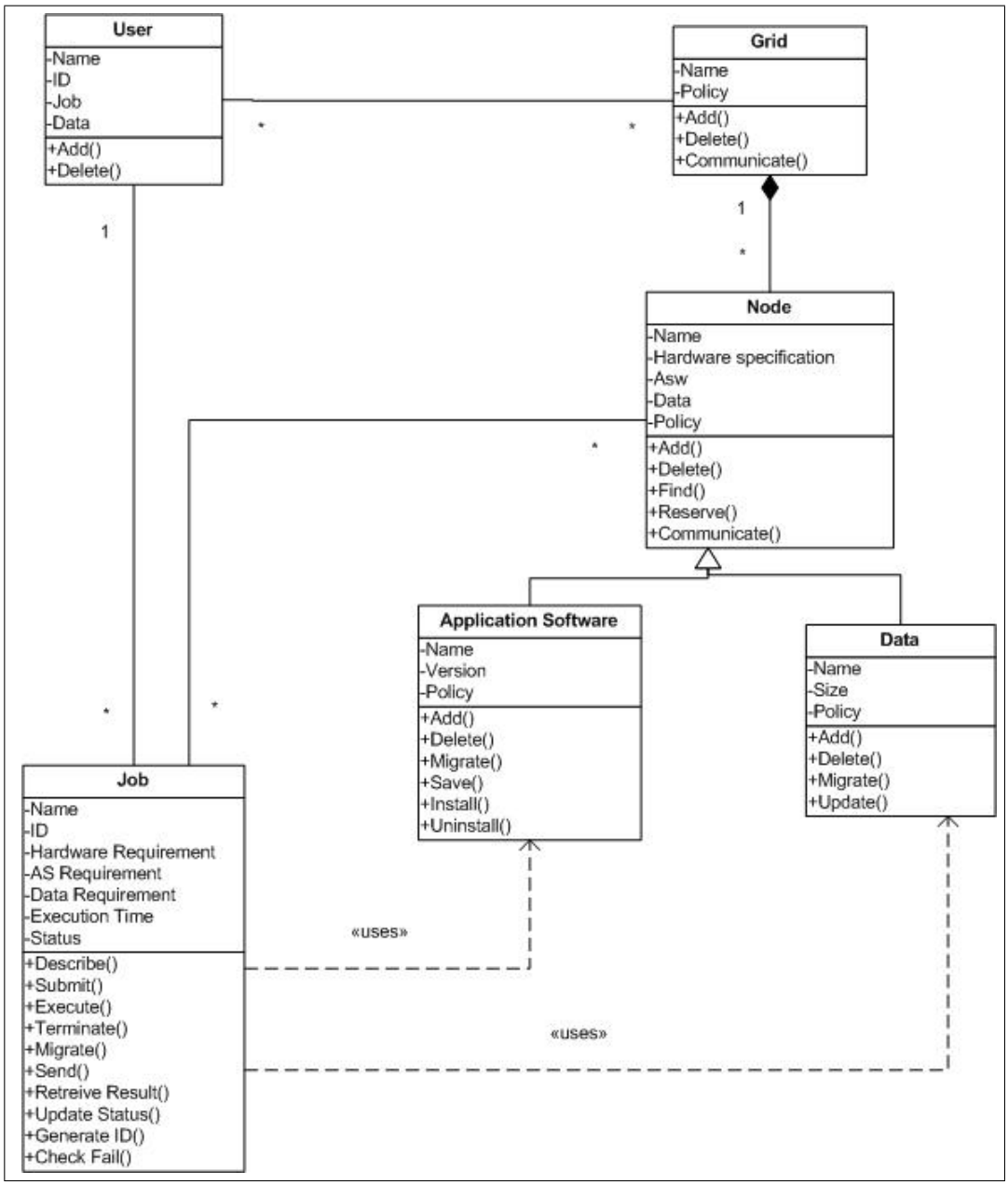

Figure 6.1: EVOSim Class Diagram

Grids are responsible for receiving jobs from and sending the results back to users, and are thus responsible for finding suitable resources on which to execute jobs. If a grid does not contain any of the job requirements, it contacts the other cooperating grids, either requesting the missing resource(s) or sending the job to other grids for execution and returning the result. Grids also contact users to inform them of any occurrences regarding their jobs, such as their inability to execute jobs because they could not find the appropriate resources. Each grid has a unique name that the user employs to submit a job to it.

- Node

  Nodes are entities responsible for receiving and executing jobs from resource brokers. Nodes can also communicate with each other in order to exchange job requirements such as application software or data. Each node has a unique name used by the resource broker to distinguish between them and to choose the appropriate node for the job. The node also has many other attributes such as hardware specification, which consists of:

  - CPU count: the number of CPUs possessed by each node.
  - CPU speed: the speed of each CPU that each node has.
  - Virtual memory: the total number of bytes of virtual memory in each node.

  - Physical memory: the total number of bytes of virtual memory in each node.
  - Disk space: the total amount of space in each node.

- Application software

  Application software is an entity placed on grid nodes responsible for executing jobs. Each application software has a name and a version by which the user can describe it in order to execute a job. The application software can migrate from node to node inside and outside the grid depending on the policy determined by its owner. It also may be uninstalled by the grid if it affects the grid's performance, while retaining the ability to save the original copy of it without installing it; when the need arises to use this application software, the grid reinstalls it again.

- Data

  Data is an entity used by jobs to accomplish their execution on the required application software. Data may either already be stored on one or more nodes or it may come with the job. Each stored data can be described by the user with its size and its unique name. The data is able to migrate from node to node inside or outside the grid depending on the policy determined by its owner.

- Job

A job is an entity described by the user in order to execute it on grid nodes. Each job has a name and a unique ID generated by the system itself. There are also a series of requirements specified for the required nodes: CPU architecture, speed, count, virtual and physical memory and disk space, required application software (application software name and version), operating system name and version, required data (name and size) and required execution time. Once the resource broker receives the job, it finds a suitable node on which to execute it until its execution time has elapsed, upon which the job terminates. If a node fails, the job can also migrate to other nodes to complete its execution.

6.2.1.2 The System

The system is responsible for the interactions occurring between various entities. It also enables the system's users to create their own system by configuring grids with their components. This will be described through the use cases and the sequence diagram below.

6.2.2 EvoSim Use Cases

The following are the EvoSim use cases, as shown in Figures 6.2, and 6.3:

1. **Use case: Submit Job.**

   Actors: user, broker, and information service.

   Purpose: to submit a job to the grid environment.

Overview: user calls the function responsible for job submission. The system submits a job to the grid environment according to the job description given by the user.

Action and response: the use case begins when the user calls the "submit job" function with job description as argument. The system responds by analysing the job description to ensure that the user has completed all the compulsory fields for submitting the job. If the job submission is successful, the job ID is generated, registered and returned to the user.

2. **Use case: Find suitable node.**

Actors: broker, information service, and other grids.

Purpose: to find a suitable resource for job execution according to the job description.

Overview: broker calls the function responsible for finding suitable nodes. The system finds such a node according to the job requirements by using the information service.

Action and response: the use case begins when the broker calls the "find suitable node" function with the job description as argument. The system responds by analysing the job requirements to find matching resources, using the information service to search in the grid or to contact other grids asking for the requirement, transfers the job for execution, informs the user and updates

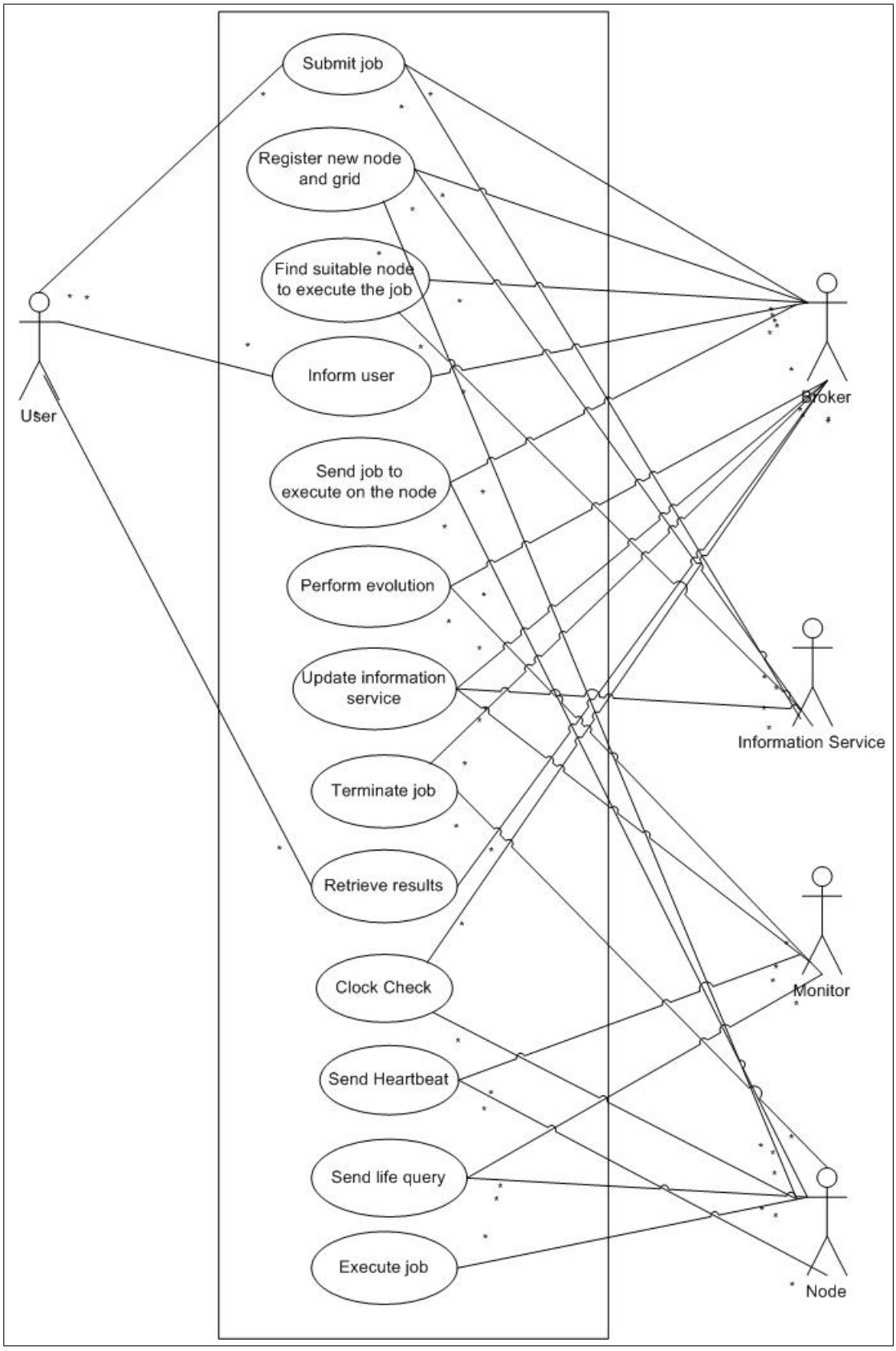

Figure 6.2: EVOSim Use Case Diagram

the information service. The pseudo code for finding suitable nodes in

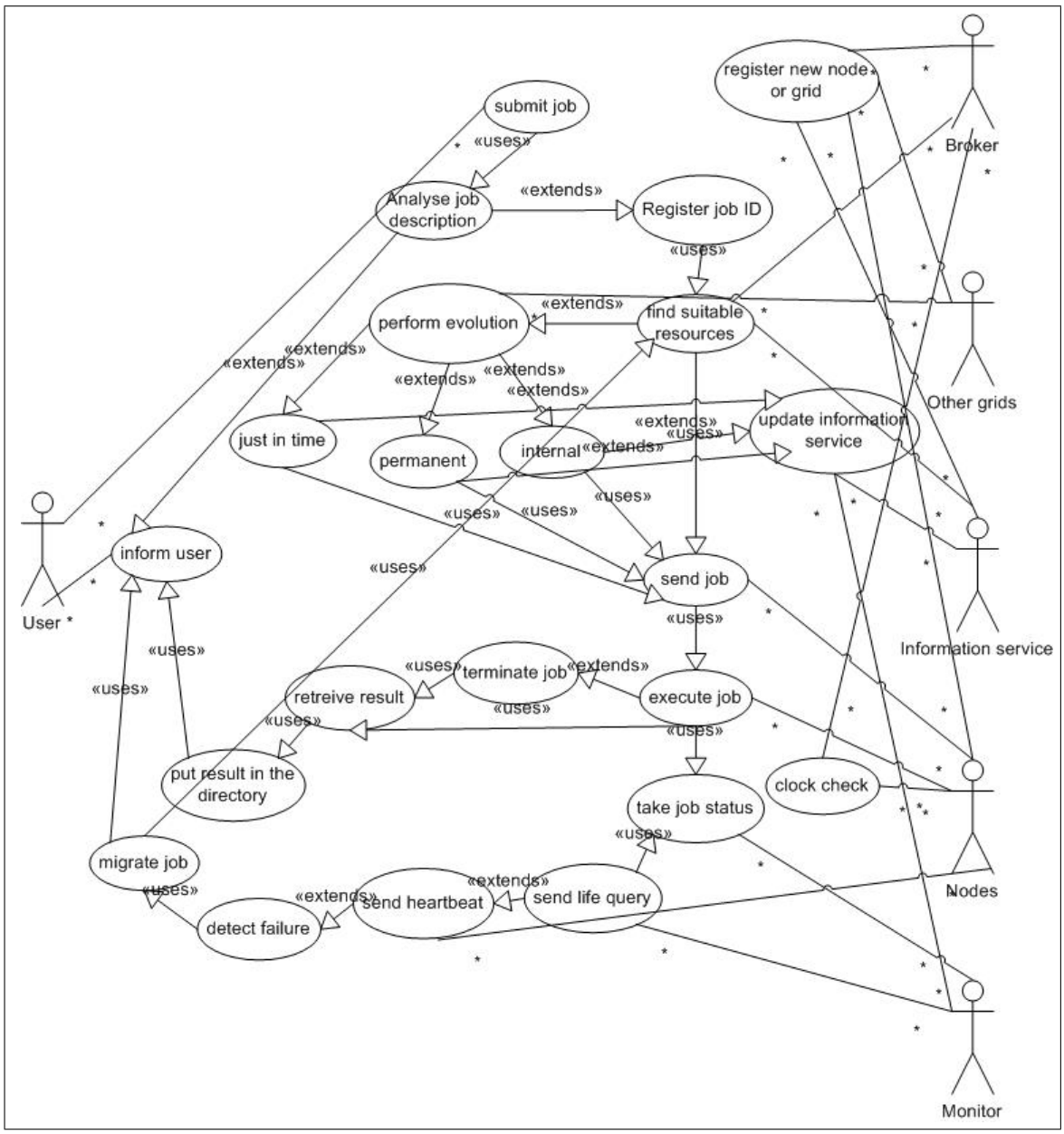

Figure 6.3: EVOSim Developer Use Case Diagram

3. **Use case: Perform evolution.**

   Actors: broker, information service, node, and other grids.

   Purpose: to perform grid evolution in order to fulfil the job requirements and

thus execute the job.

Overview: broker calls the function responsible for performing evolution. The system performs the required type of evolution.

Action and response: the use case begins when the broker calls the "evolution" function with the evolution description as argument. The system responds by analysing the evolution description and performing the required type of evolution, which may contain internal, permanent, or just-in-time evolution. This may require contact with other grids in order to complete evolution. Changes are saved in the information service.

4. **Use case: Send job for execution.**

   Actors: broker, and node or other grids.

   Purpose: to send the job to the selected node for execution.

   Overview: broker calls the function responsible for sending jobs for execution. The system sends the job to the selected node for this purpose.

   Action and response: the use case begins when the broker calls the "send job" function with the job execution description as argument. The system responds by sending the job to the selected node or grid in order to execute it until the job execution time expires.

5. **Use case: Inform Users.**

   Actors: broker, information service, and user.

Purpose: to inform users of their jobs' statuses.

Overview: broker calls the function responsible for informing users. The system sends a message to the user to inform them of the situation.

Action and response: the use case begins when the broker calls the "inform user" function with the user and situation description as argument. The system responds by sending a message to the user describing the situation.

6. **Use case: Execute Job.**

   Actors: node, and broker.

   Purpose: to execute a job on the selected node.

   Overview: node calls the function responsible for executing jobs.

   Action and response: the use case begins when the node calls the "execute job" function with the job execution description as argument. The system responds by commencing job execution according to the job execution description.

7. **Use case: Terminate Job**.

   Actors: broker, information service, and node.

   Purpose: to terminate job execution.

   Overview: broker calls the function responsible for terminating job execution.

   Action and response: the use case begins when the broker calls the "ter-

minate job" function after the expiry of the job execution time, with the job name and node name as argument. The system responds by terminating the job's execution.

8. **Use case: Retrieve Results.**

   Actors: broker, and node.

   Purpose: to retrieve the executed job's results.

   Overview: the broker calls the function responsible for retrieving job results.

   Action and response: the use case begins when the broker calls the "retrieve results" function after the job has been terminated, with the job name and node name as argument. The system responds by retrieving the job results, placing them in the certain directory that came with the job description and informing the user.

9. **Use case: Send life query.**

   Actors: monitor, and node.

   Purpose: to ask for node life and job status.

   Overview: the monitor calls the function responsible for sending life queries.

   Action and response: the use case begins when the monitor calls the "send life query" function. The system responds by sending this query to all nodes in the grid environment and waiting for the response, upon which it takes the appropriate action.

10. **Use case: Send heartbeat.**

    Actors: monitor, and node.

    Purpose: to send a node heartbeat to the monitor.

    Overview: the node calls the function responsible for sending heartbeats.

    Action and response: the use case begins when the node calls the "send heartbeat" function. The system responds by sending this heartbeat with the job status to the monitor. If monitor didn't receive a heartbeat from any node, it classes this node as failed. And detect job failure if its status is the same.

11. **Use case: Clock check.**

    Actors: node, and broker.

    Purpose: to check the node clock with that in the broker.

    Overview: the node calls the function responsible for clock checking.

    Action and response: the use case begins when a node calls the "clock check" function. The system responds by synchronising the node's clock with that in the broker.

12. **Use case: Register new node or grid**.

    Actors: node, grid, information service and broker.

    Purpose: to register a new node or grid with the broker.

Overview: the node or grid calls the function responsible for registration with the grid.

Action and response: the use case begins when the node or grid calls the "register" function. The system responds by registering this node or grid with its grid and saving its specification in the information service.

6.2.3 Sequence Diagram

This diagram as shown in Fig. 6.4 and Fig 6.5, begins when the user sends a job to the grid. At this point the grid ensures that the user has completed all the compulsory fields required to describe the job. It then passes this description to the resource broker, which searches the information service in order to find all the job requirements on one node. If found, it sends the job to this node for execution and waits until the execution time has elapsed, upon which it terminates the job, retrieves the result, informs user and saves the result in the directory specified by the user when they described the job. If the broker does not find all the job requirements in one node, it attempts to perform one of the evolution types depending on which job requirement has not been met. If the broker finds that the grid has all job requirements scattered across different nodes and that their policies support evolution, it performs internal evolution by migrating the application software and the data to the required node and sending the job to execute on this node. If the broker finds that the grid has the node only, and that its policy supports evolution,

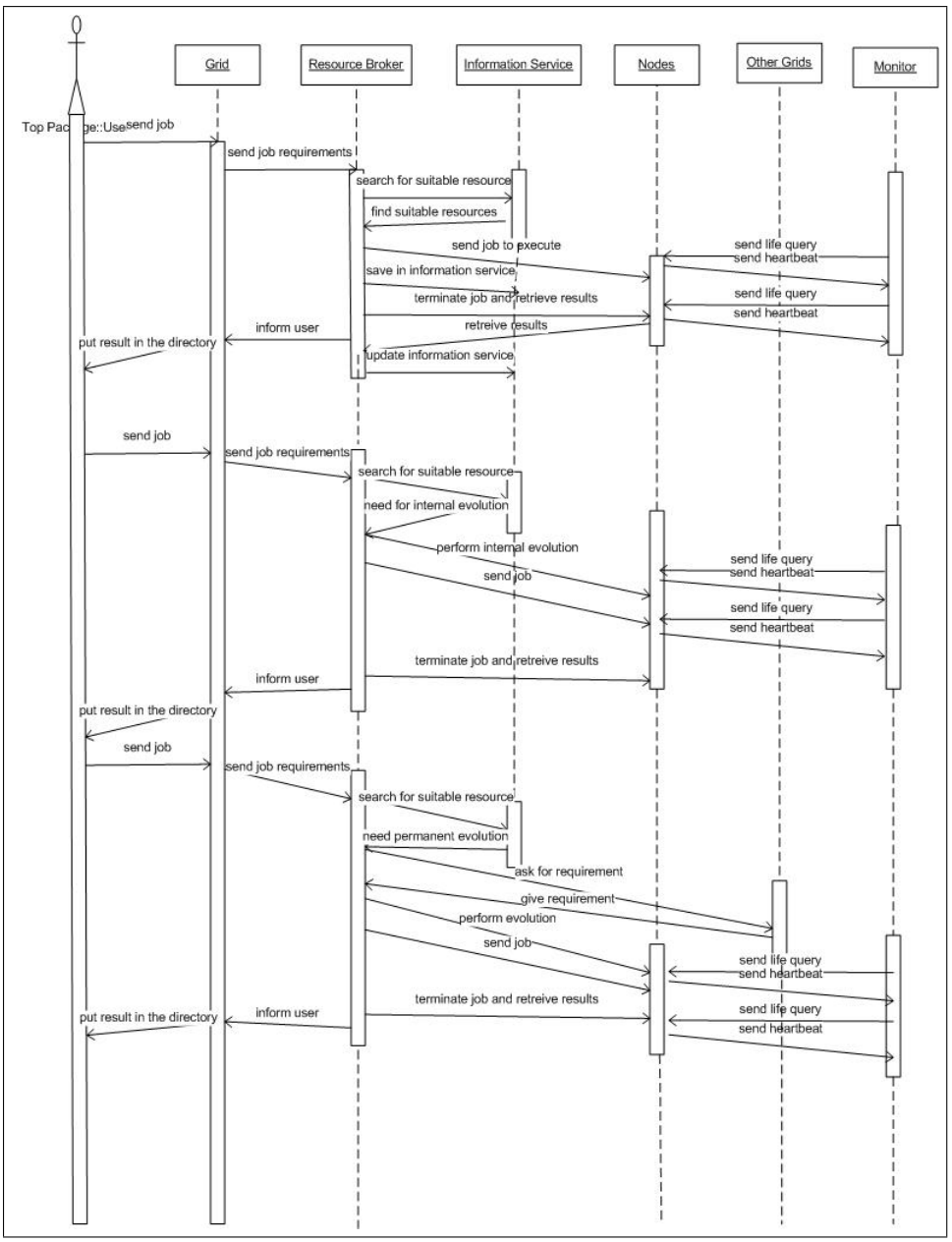

Figure 6.4: Sequence Diagram 1

the broker contacts other grids and asks them if they have the required resources. If these are found, the grid performs permanent evolution. Once the broker performs the evolution, it sends the job to the evolved node and waits until the execution time has elapsed, upon which it terminates the job, retrieves the results and informs the user. The grid then saves the results in the directory specified by the user when they described the job.

At the same time, the monitor periodically sends life queries to all the nodes and waits for the heartbeat reply. If it replied, the monitor compares the job status with the previous one; if it is the same, it determines that the job has failed and informs the resource broker accordingly. If the monitor does not receive the expected heartbeat from any node, it puts that node into a failed state in the information service, also informing the broker that this node has fail with sending that node's last job state. It also looks for other suitable nodes on which to execute the job. If it finds one, it migrates the job to it in order to complete the job execution from the last state, and informs the user that the grid has recovered the job from failure. If it does not find another suitable node, it attempts to perform one of the evolution types, and if this proves unsuccessful, it informs the user that the job has failed and that the grid cannot recover it. If the grid does not have the job requirements, the broker performs just-in-time evolution by asking other grids for all the job requirements. If any grid has them, the broker sends the job to it for execution, waiting until this is completed and asking the grid to terminate the job, retrieve the results and

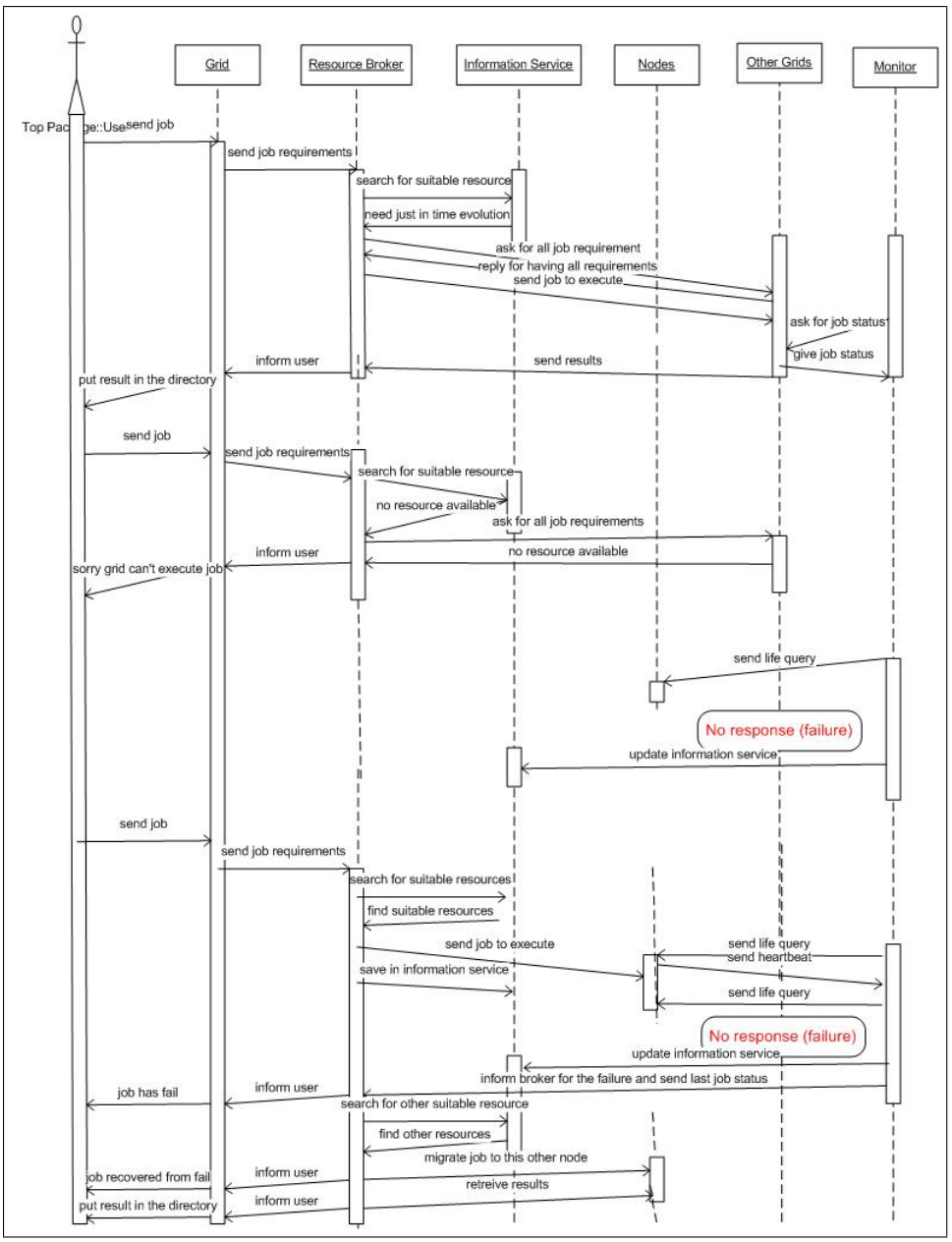

Figure 6.5: Sequence Diagram 2

inform the user. It then places the results in the directory specified by the user when describing the job. While this is happening, the monitor periodically sends queries regarding the job status to the executing grid in order to provide for failure, which it deems to have occurred if it does not receive the expected state from the grid. In this case the monitor updates the information service and informs the broker, who deals with this as a new job in order to find a suitable node on which to complete its execution from its last state.

6.3 EVOSim Implementation

The simulation tool has been developed and used for checking theresults from scratch. This tool is written in Oracle 10g and was built using the computational and architecture model. This simulation assume a heterogeneous grid computing environment which has an infinite number of nodes in a fully connected topology.

In the simulation, there are many steps that should be followed in order to simulate grid environments; these steps start after running the simulation. Firstly the main user interface appears. This allows the user to choose one of the following options. The first is to configure the new grid, the second is to open the existing grid, and the third is to submit a job. If the user chooses the first option, other interfaces open sequentially as shown in Fig. 6.6. Through these interfaces the user can configure new grids by determining both their names and the other grids it need to communicate with which gave communication permission, and then determine its

nodes by determine the following for each node (as shown in Fig. 6.7 and Fig. 6.8.

Figure 6.6: Grid Configuration

- Node type, which can be PC, mainframe, or data storage.

- specified availability time (i.e. the first and last times at which the node will be available for the grid to use).

- Node policy, which will be either static or dynamic. Static means that the node does not allow the addition of new application software or data; dynamic allows such additions.

- Node hardware specification, application software and data contained in each node.

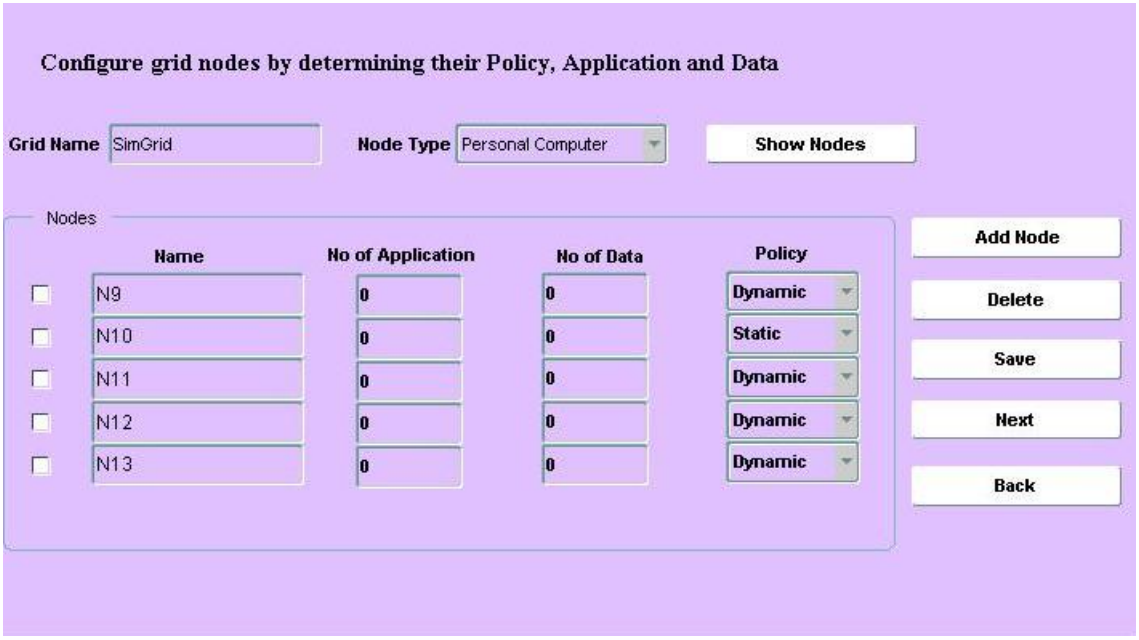

Figure 6.7: Node Configuration

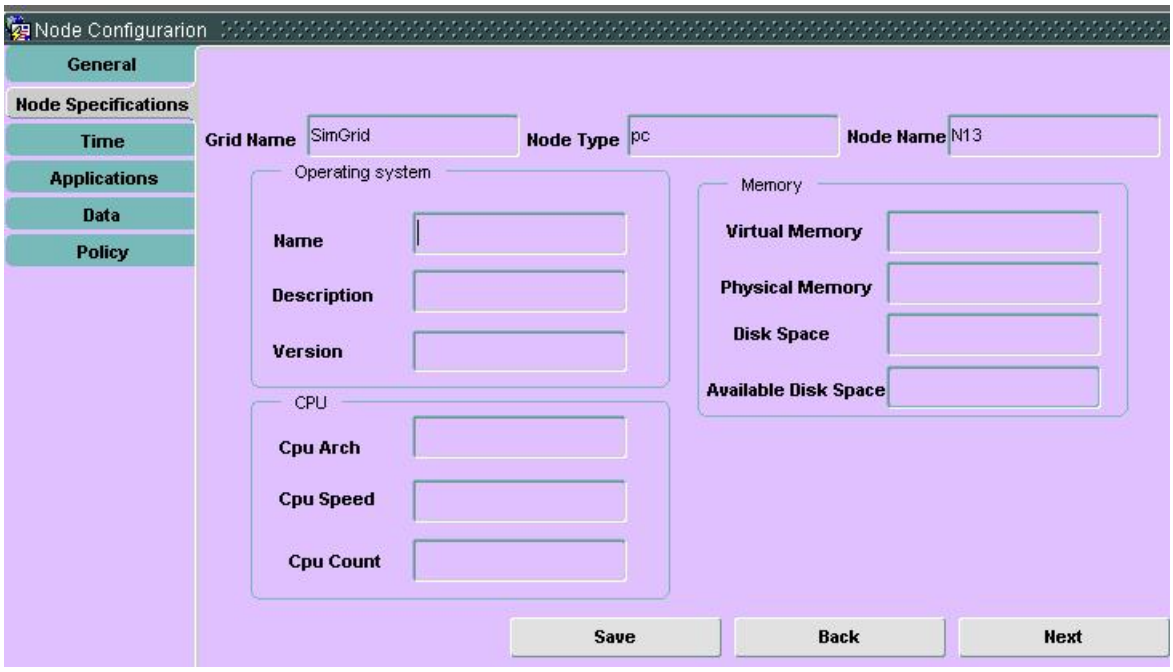

Figure 6.8: Node Specification Configuration

– To determine hardware specification, the user should specify the operating system (OS) (name, description, version), memory (virtual and physical memory, disk space and available disk space), and CPU (CPU speed, architecture and count).To determine application software, the user should specify the name, type, version, size, speed, RAM and applicable OS, and to determine data, the user should specify the name, format, and size. After the configuration has been completed, all of the information will be saved in the information service.

If the user chooses to open an existing grid, they can choose one of the existing grids that will appear as a list, and then can edit any of them and change its configuration, saving these changes in the information service. The user can also choose a job submission option that allows description of its job through the job submission portal that appears when this option is chosen. This portal allows the user to describe the job by completing fields using by the resource broker in order to find a suitable resource for that job, as shown in Fig.6.9.

Through this portal, the user should determine the grid which the job should be submitted and should then describe the job by name, required application software (name and version), required OS (name and version), required node specification (i.e. CPU architecture, speed and count, virtual memory, physical memory, disk space, required execution time and required data name (if the data exists in the grid). The user can also upload the data necessary to execute the job by determining the data

name, source and target.

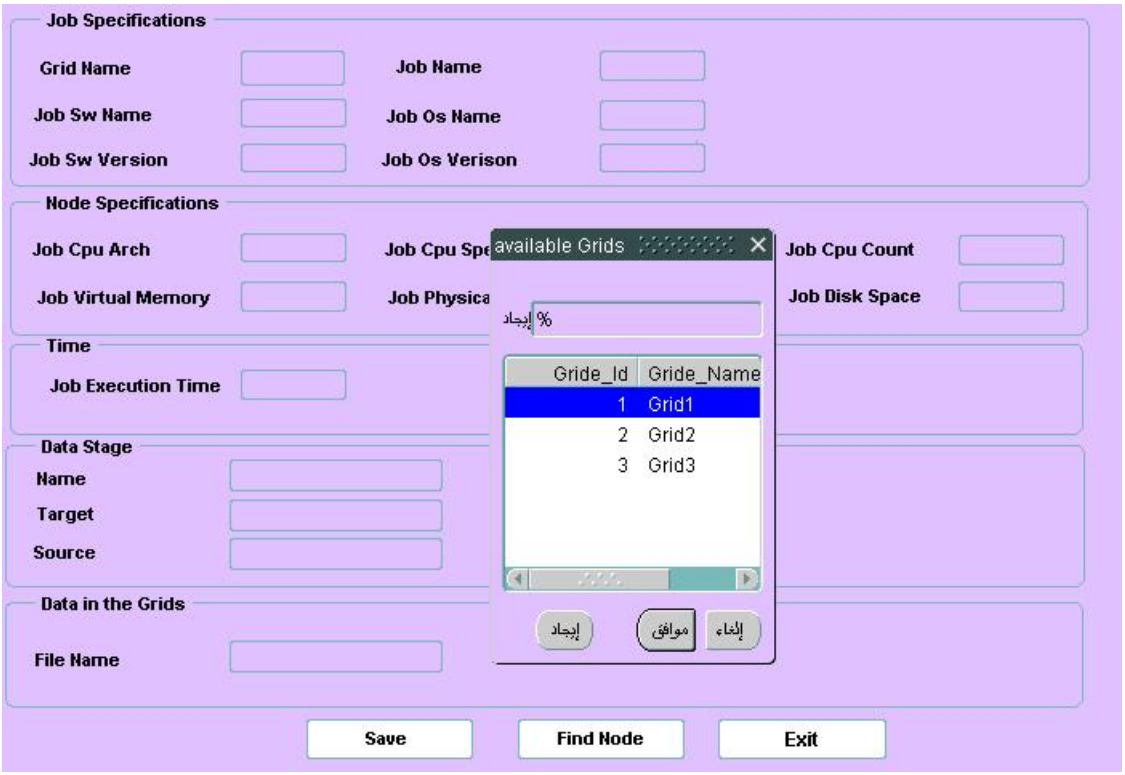

Figure 6.9: Grid Portal

After the user has completed all the required fields for the job description and has clicked on the "send" button, the system translates all the inputs from this interface to JSDL by generating them as an XML file and sending them to the resource broker. The latter searches for suitable resources that meet these requirements, so that the job can be executed according to the resource information held in the information service. The broker then migrates application software, and/or data, or contacts other grids if required, reserves required resources, transfers job input files to them and waits for the job's execution time to elapse, at which point it terminates the job and sends the job outputs back to the user. If the resource broker does not find

the required resources on its own or on other grids, it informs the user by sending a message to the effect that the grid cannot execute the job. While this is happening, the monitor observes the grid environment and saves the job's status periodically, keeping any changes in the status of any node in the information service

6.4 Simulation Validation

In this section, the simulation is validated by showing that the output matches the results obtained by manual calculation, as described in the following scenario.

Four grids called EVO grid, Grid1, Grid2 and Grid3 has been configured by using the simulation interfaces, each of which has own specification (policy, number of nodes and specification for each node). The specification of these grids is explained in the tables below. Then performed two steps, first, checked the information service, and found the same interface information that entered, then submitted five jobs to the EVO grid to see how it worked and what the changes were. The requirements for those jobs are described in Table 6.5.

From these requirements, and according to the node specifications in all five grids, the following becomes clear:

- Job1 required hardware specifications that could be met by all the grid nodes, application software B version 1 which resided on node N1 in the EVO grid, and data D3 which resided on node N3 in the EVO grid. N3's policy was dynamic, which means that it could be copied or the data could be moved.

Node Name	Hardware				Application				Data		Policy(StaticorDynamic)
	CPU		Memory				Requirement				
	Speed (gigahertz)	Count	RAM (Mega)	Disk Space(giga)	Name	Version	CPU Speed (gigahertz)	Disk Space (Mega)	DataName	Size(Megga)	
N1	1	1	1024	100	A	1	0.5	300	D1	700	Dynamic
N2	1	1	1024	120	C	1	1.0	800	D2	900	Static
					B	1	0.5	700			
N3	1	1	2048	120	D	2	1.0	500	D3	700	Dynamic
N4	2	2	2048	140	E	3	1.0	900	D2	570	Dynamic

Table 6.1: EVO Grid Nodes Specification

Node Name	Hardware				Application				Data		Policy(StaticorDynamic)
	CPU		Memory				Requirement				
	Speed (gigahertz)	Count	RAM (Mega)	Disk Space(giga)	Name	Version	CPU Speed (gigahertz)	Disk Space (Mega)	DataName	Size(Megga)	
N5	1	2	1024	140	A	2	1.0	700	D5	900	Dynamic
					F	1	1.0	900	D6	800	
N6	2	2	2048	160	A	2	1.0	900	D7	700	Static
					G	2	1.0	800	D8	1200	

Table 6.2: Grid1 Nodes Specification

Node Name	Hardware				Application				Data		Policy(StaticorDynamic)
	CPU		Memory				Requirement				
	Speed (gigahertz)	Count	RAM (Mega)	Disk Space(giga)	Name	Version	CPU Speed (gigahertz)	Disk Space (Mega)	DataName	Size(Megga)	
N7	2	4	2084	200	L	1	1.0	900	D13	900	Dynamic
N8	2	2	2048	180	N	3	1.0	900	D15	700	Dynamic

Table 6.3: Grid2 Nodes Specification

Node Name	Hardware				Application				Data		Policy(StaticorDynamic)
	CPU		Memory				Requirement				
	Speed (gigahertz)	Count	RAM (Mega)	Disk Space(giga)	Name	Version	CPU Speed (gigahertz)	Disk Space (Mega)	DataName	Size(Megga)	
N9	4	2	2084	300	H	5	1.0	900	D11	900	Dynamic
N10	8	2	2048	200	M	6	1.0	700	D12	1000	Dynamic

Table 6.4: Grid3 Nodes Specification

Job	Node Specification				Application Software		Data
	CPU		Memory		Name	Version	
	Speed	count	RAM	DiskSpace			
Job 1	1	1	1024	300	B	1	D3
Job 2	1	1	1024	200	A	2	D2
Job 3	1	1	1024	400	F	1	D11
Job 4	8	2	2048	500	M	6	D12
Job 5	2	2	2048	500	N	3	D2

Table 6.5: Jobs Requirement

After this job was submitted to the EVO grid a message saying that the job had started running on the node was received. The information service was then checked to see what had transpired, and it was found that the grid had evolved internally by migrating D3 from N3 to N1; this means that D3 had come under N1's specification in the information service and was deleted from N1's specification. The job was sent to N1.

- Job2 required hardware specifications that could be met by all the grid nodes, application software A version 2 which resides on node N5 in Grid1 (which in turn gave communication permission to the EVO grid) and data D2 which resided on nodes N2 and N4 in the EVO grid. N2's policy was static, however, which means that this data could not be copied or migrated, as could N4, whose policy was dynamic.

  After the job was submitted to the EVO grid, a message was received from the latter saying that the job had started running on the node. The information service was then checked to see what had transpired, and it was found that the grid had permanently evolved by copying application software A version 2 from N5 in Grid1 to N4, after which it had sent the job to N4.

- Job3 required hardware specifications that could be met by all the grid nodes, application software F version 1 which resided on node N5 in Grid1 (which in turn gave communication permission to the EVO grid) and data D11 which resided on node N9 in Grid3 which likewise gave communication permission

to the EVO grid. The policy of both nodes was dynamic. After the job was submitted to EVO grid a message was received from the latter saying that the job had started running on the node. The information service was then checked to see what had transpired, and it was found that the grid had permanently evolved by migrating application software F version 1 from N5 in Grid 1 and D11 from N9 in Grid3 to N4 in the EVO grid. From this it can be seen that, because N1 and N3 were busy with other jobs and N2's policy was static, the grid chose N4 to execute the job because it was idle and its policy was dynamic, which allowed the new additions.

- Job4 required hardware specifications that could be met by node N10 in Grid3 (which gave communication permission to the EVO grid), application software M version 6 (which also resided on node N10 in Grid3) and data D12 (which also resided on the same node). After the job's submission a message was received from the grid saying that the job had started running on the node. The information service was checked to see what had transpired and it was found that the grid had performed Just-in-Time evolution by sending the job to Grid3 for execution.

- Job5 required hardware specifications that could be met by node N4 in the EVO grid, application software N version 3 which resided on node N8 in Grid2 (but which did not give communication permission to the EVO grid) and data D2 which resided on node N4 in the EVO grid. After the job's submission a

message was received from the grid saying "Sorry, the grid cannot execute the job". This rejection occured because the job needed application software N residing on Grid2, but this grid did not give communication permission.

6.5 Summary

This chapter gives a brief overview of the definition of simulation and its benefits. It then described the simulation's analysis and design by detailing the simulation classes and use cases, and by providing a sequence diagram which describes the simulation scenario in a simplified manner. The simulation is then described through its interfaces and how its works. Finally, the simulation's validation is discussed.

Chapter 7

Evaluation

Objectives

- To evaluate theconcepts.
- To discuss the results obtained from grid evolution.

7.1 Introduction

In Chapters 3, 4 and 5, a grid computing framework was presented, while a simulation that models this framework was described in Chapter 6. In the present chapter, the simulation is used to evaluate the performance of the grid by comparing the performance that done on the same grid environment both with and without the use of the evolution technique presented in Chapter 3.

This chapter will describe the experiments performed and results achieved using the simulation without evolution techniques. It also describes the evaluation of the proposed research carried out on the simulation that was designed to analyse the

impact of internal evolution in Section 7.4, permanent evolution in Section 7.5, and just in time evolution in the grid environment in Section 7.6.

7.2 Experimental Environment

The simulation developed so that each node had the ability to receive and execute only one job at the same time; this meant that if the node executed a job, it was fully utilised (100%) during the period of execution time. As a result, the node would be either busy or idle when it was available. The nodes configured with seven hours' availability time for each node. This section describe the simulation experiments for a grid environment without evolution. The simulation has been built without the ability to evolve, which means that the policies for all resources were static and prohibited communication with other grids. Then configured the grid environment to comprise 45 nodes, each of which had different hardware specifications, operating systems, application software and data. After the grid was configured, then sequentially submitted four batches of 16 jobs. The result from the first batch was 10 jobs were accepted. Eight jobs were accepted from the second batch while two jobs were accepted from the third batch and only one job was accepted from the final batch as shown in Fig. 7.1.

These results clearly show that the highest job acceptance rate was around 32, out of which 18 were accepted. while the other intervals increase the acceptance in 3 jobs. From the analysis of these results, the following can deducted:

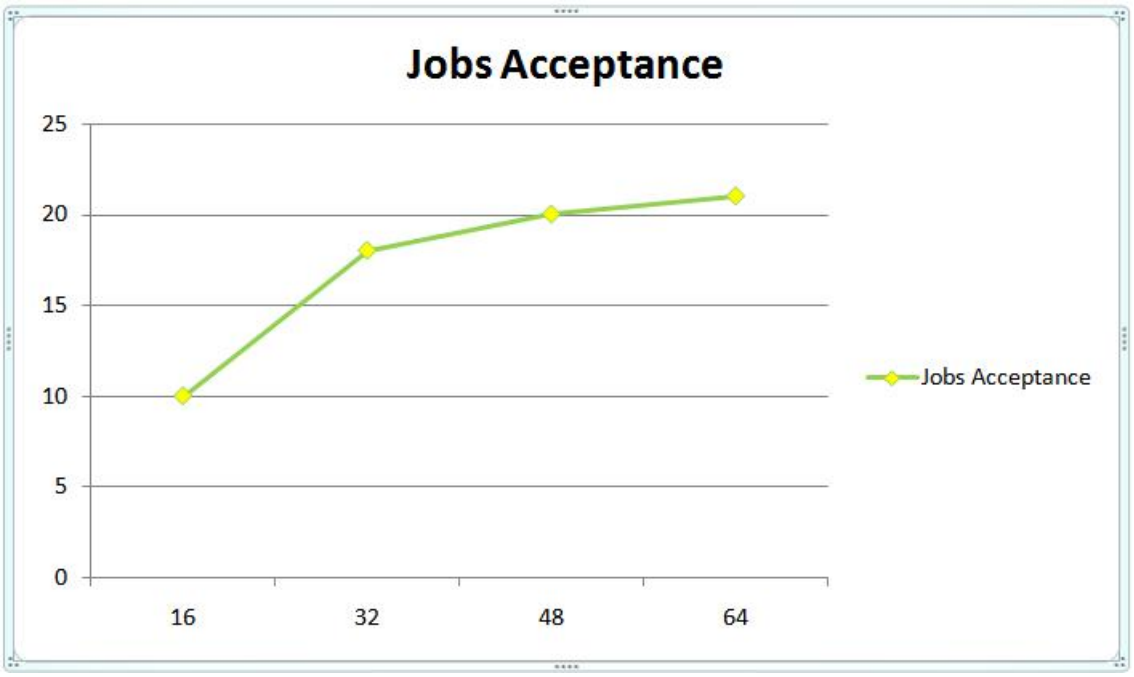

Figure 7.1: Job Acceptance

- 35 jobs was the optimum number that could be submitted to perform the evaluation in order to achieve the most realistic results.

- It is clear that the grid could reach the point of saturation, at which point more jobs may not be accepted because of the lack of resources which triggered the evolution technique.

- Because the grid comprised 45 nodes to which 64 jobs were submitted, this lead to that the grid did not have enough nodes to accept more jobs; this triggered what called permanent evolution and Just in Time evolution.

- When the number of jobs increased from 32 to 48, the number of accepted jobs increased by just two jobs, although the total number of nodes was 45. This triggered both internal and permanent evolution in order to add new resources

to those nodes to enable them to accept more jobs.

7.3 Grid Experiments without Evolution

In order to carry out the experiments and perform the evaluation, the grid environment has been created. This environment consisted of 45 nodes, each available for seven hours. Each node had different hardware specifications, operating systems, application software and data. Then submitted 35 different jobs sequentially to the grid with execution times of three hours for each job. This meant that no jobs were rejected because all the nodes were available to execute all the jobs at their execution times. In order for the results to be as realistic as possible, those same 35 jobs were used to perform all the evaluation experiments. The following experiments were performed and the results analysed.

1. Rejected Jobs.

   The results were that 16 of the 35 jobs submitted were rejected by messages saying that the grid could not execute them. This means that 45.7% of those jobs were rejected a huge number absolutely, as well as a substantial ratio, as shown in Fig. 7.3. There are many reason for the rejected job such as: the node that met the job requirements is busy at the required time, the node that met the job hardware requirement does not has the required application software and/or data, or the grid does not has the required node. This result of huge number of rejected jobs means that huge user's jobs were rejected which

will effect on grid's user by avoid using the grid and this conflict with grid aims which create to serve as much as it can from users. All this reasons and this result of huge number of rejected jobs supports the need for evolution.

2. Employment of resources.

   From the point at which each node was executing only one job at the same time, it was fully occupied with executing that job until it was complete. This resulted in only 19 nodes working (i.e. busy) while the other nodes were idle, which meant that the employment of resources was 42.2% as shown in Fig. 7.2. This result conflict with grid aims which is exploitation of underutilised resources which trigger the evolution.

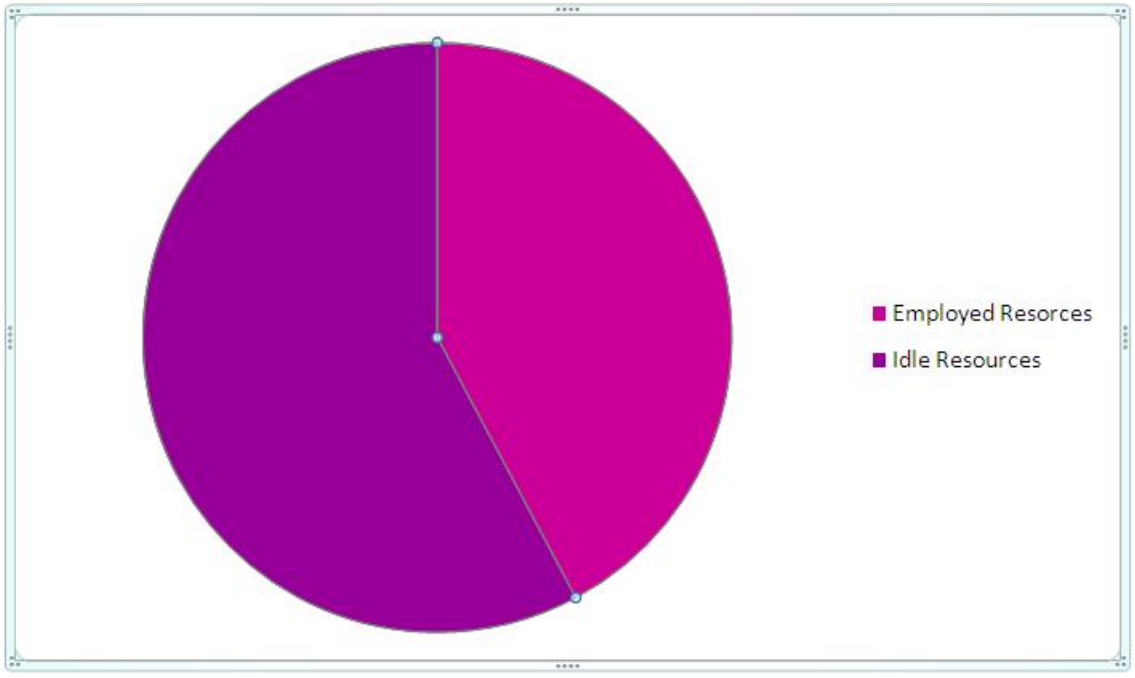

Figure 7.2: Resources Employment without Evolution

3. CPU Utilisation.

In the simulation configuration there were 45 nodes, each node available for seven hours, which meant that collectively they would be available for 315 hours. There were also 35 jobs submitted to the grid, each of which needed three hours to execute. Those jobs therefore needed 105 hours for execution, which meant that they would utilise 33.3% from the CPU if executed all on the grid. In the simulation, the 35 jobs submitted to the grid without evolution resulted in the execution of 19 jobs, whose execution therefore took a total of 57 hours; the CPU utilisation was thus 18%, as shown in Fig. 7.5. This ratio of CPU utilisation is very small when compared with 33.3%, and resulted in a demand for new resources to increase this utilisation. This is why internal evolution was created.

7.4 Grid Experiments with Internal Evolution

The simulation has been configured to support internal evolution by making many resources policies dynamic. This allowed application software and data to migrate from node to node inside the single grid, depending on policies. In this case, the goal of the simulation was to show the impact on the number of rejected jobs, resource employment and CPU utlisation of the grid environment, as well as to demonstrate how this feature enables the capability for change.

1. Rejected Jobs.

   After this change, resubmitted the same 35 jobs as before. Only nine jobs

were now rejected as opposed to the previous 16, which means that the internal evolution reduced the rejected jobs from 16 to 9 as shown in Fig. 7.3. In this experiment, the advantage of internal evolution appeared by decreasing the number of rejected jobs. This occur because internal evolution allow application software and data to migrate from node to other inside the grid. This means that if the grid has the required hardware, application software, and data scaterred on its node, this trigger the internal evolution which will allow for those requirements to migrate and gathering in the required hardware in order to execute the job and then reducing the rejected job as a result.

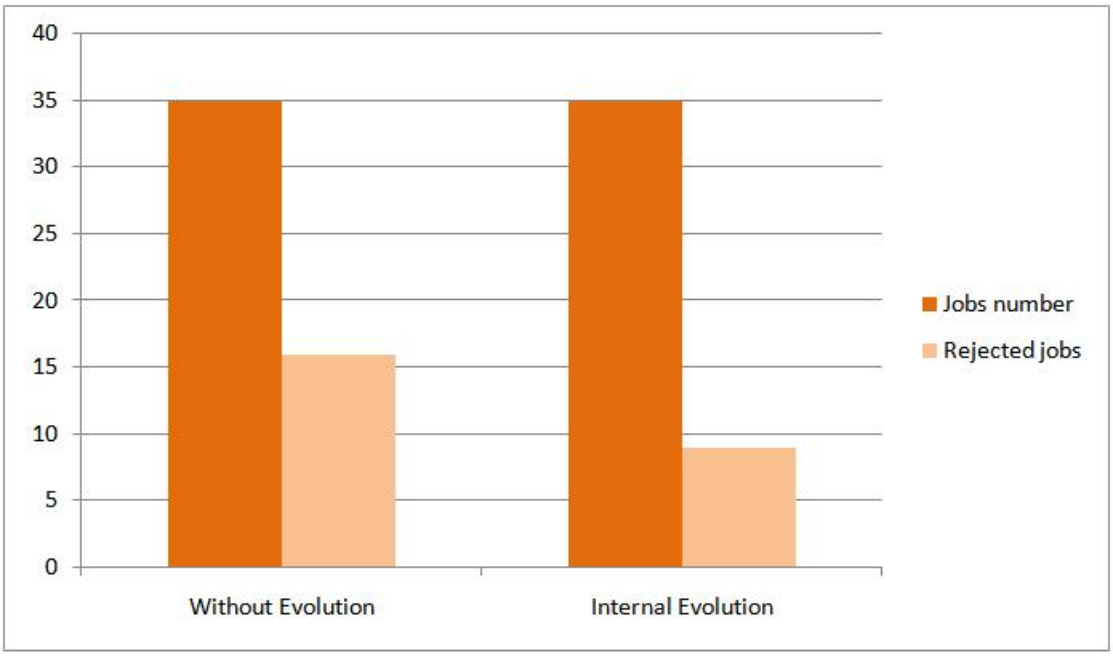

Figure 7.3: Rejected Jobs with Internal Evolution

2. Resource Employments.

26 jobs accepted for execution on the grid entails 26 busy nodes. Internal

evolution added more features and resources to the nodes, enabling them to execute more jobs, thereby increasing the resource employment rate to 80%, as shown in Fig. 7.4. This was a direct result of internal evolution, and led us to investigate the results of other types of evolution.

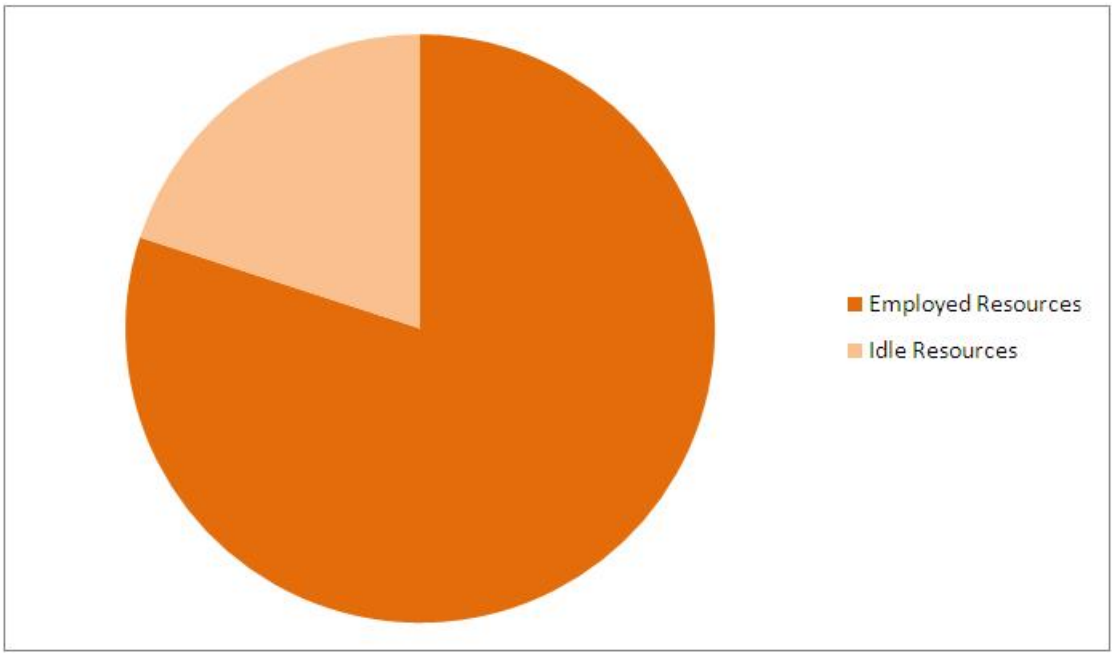

Figure 7.4: Resources Employment with Internal Evolution

3. CPU Utilisation.

   The 45 nodes were available for a total of 315 hours but the 26 jobs only needed 78 hours for execution, which meant that CPU utilisation was just 25%, as shown in Fig. 7.5.This ratio was also much less than the 33.3% that triggered the need for more new resources to increase the CPU utilisation to be close to that ratio. This in turn gave rise to permanent evolution, which would hopefully increase CPU utilisation.

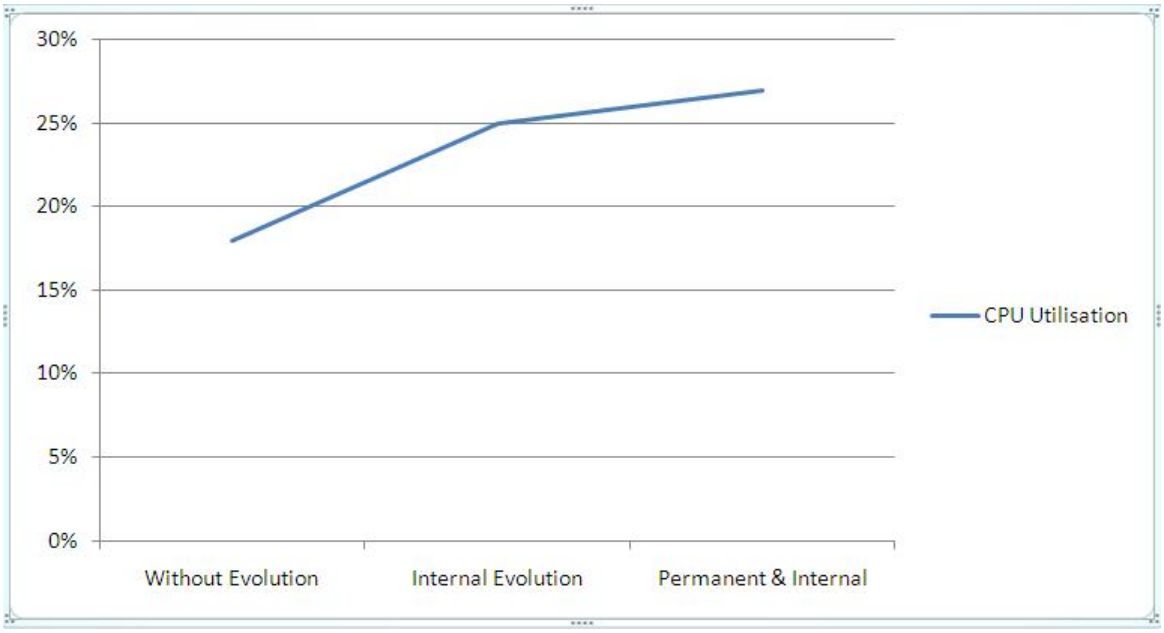

Figure 7.5: CPU Utilisation

7.5 Grid Experiments with Permanent Evolution

The results of the internal evolution caused us to develop the simulation in order to perform permanent evolution. This done by configuring three more grids, each with 20 nodes. Each node had different hardware specifications, operating systems, application software, data and policies. Also the original grid has been modified by adding much application software and data to the nodes. At this stage, the application software and data could migrate from one node to another on a different grid, depending on their policies.

1. Rejected Jobs.

   Permanent evolution gave the grid the ability to bring in new resources from other grids.In this case,if the grid does not has the required application soft-

ware and/or data while other grid has; then this application software and data will migrate from this grid to the original grid in order to satisfy the job requirements. At this stage, the same 35 jobs as before the extension of evolution have been resubmitted, resulting in the rejection of seven jobs rather than the previous nine, which means that the permanent evolution reduced the rejected jobs from 9 to 7 as shown in Fig. 7.6. The desire to eliminate the number rejection jobs entirely now triggered Just in time evolution, which hoped would achieve this outcome.

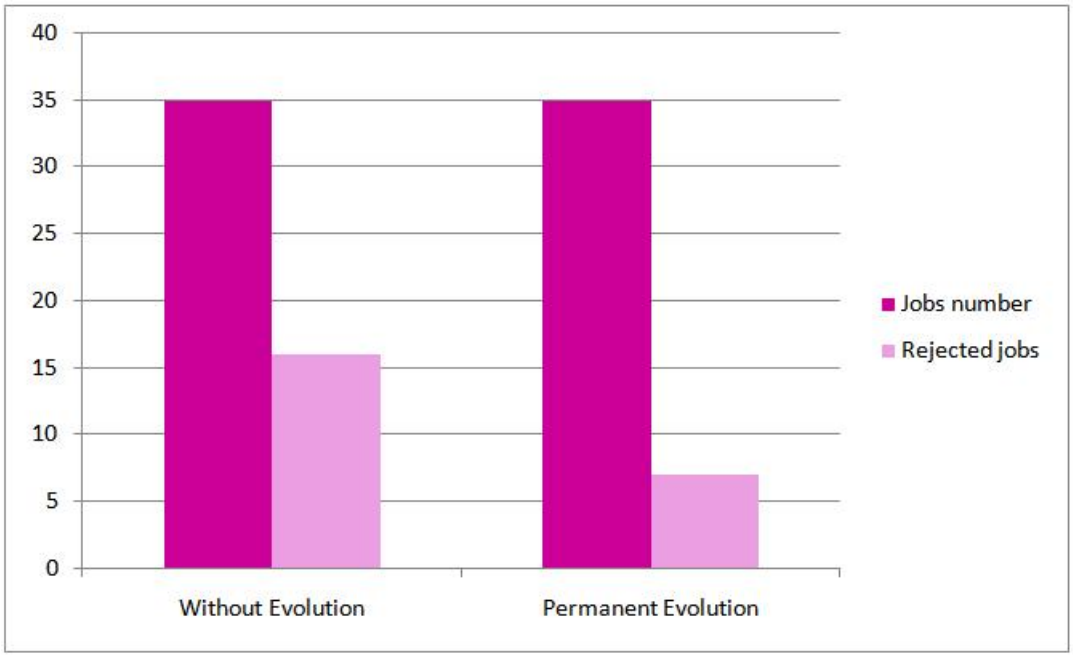

Figure 7.6: Rejected Jobs with Permanent Evolution

2. Employment of resources.

As a result of permanent evolution, 28 nodes were utilised in the execution of 28 jobs, which meant that the resources employed increased to 85% as shown

in Fig. 7.7.

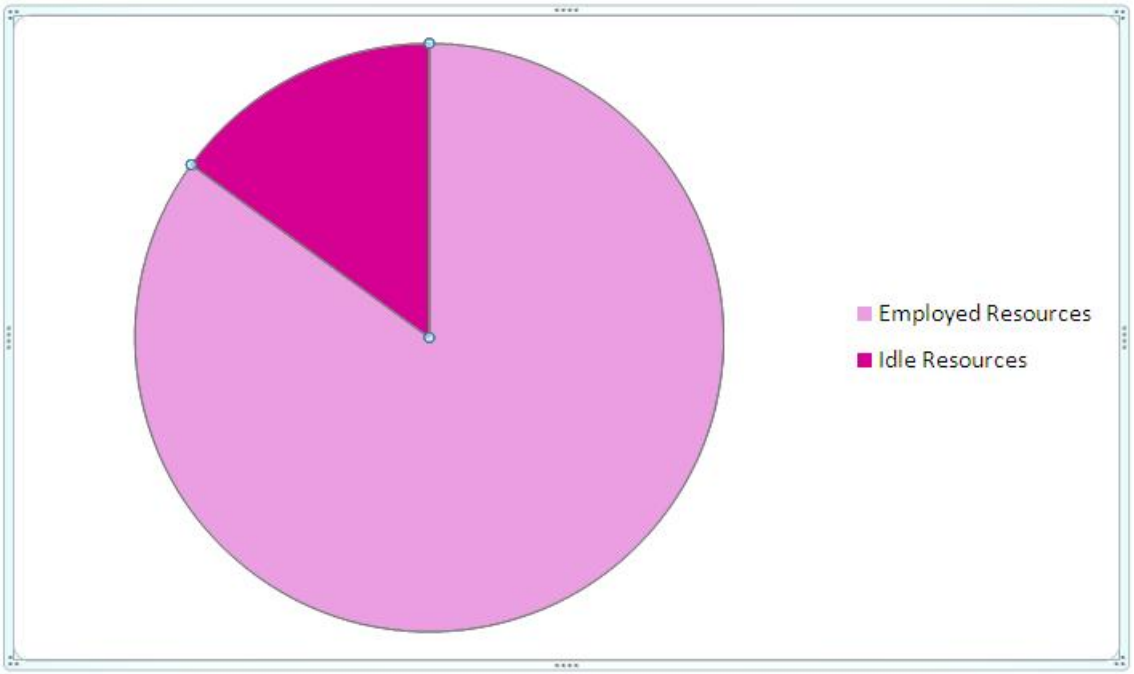

Figure 7.7: Resources Employment with Permanent Evolution

3. CPU Utilisation.

   Permanent evolution increased the number of jobs executed to 28, which requiring 84 hours for execution. As a result, CPU utilisation increased to 27% as shown in Fig. 7.5.

7.6 Grid Experiments with Just-in-Time Evolution

After evaluated the results of the experiment with permanent evolution, the simulation has been developed to enable Just in Time evolution. This allows jobs to be

executed on other grids if the original grid does not have all its requirements and if the other evolution mechanisms (internal and permanent) have failed. In these cases, the job would be sent to other grids for execution. To perform this, the same 35 jobs have been resubmitted to the grid, in the same environment configured to perform permanent evolution.

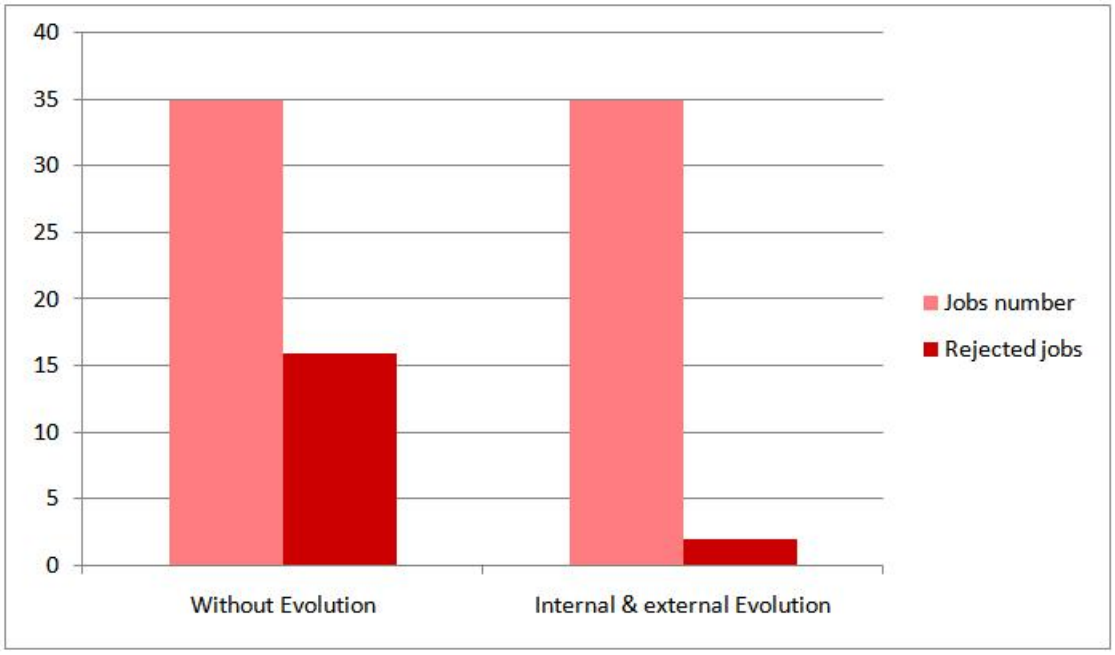

Figure 7.8: Rejected Jobs with Just in Time Evolution

The results show rejection of three jobs. Then the information service has been checked to see if there was any job scheduled on nodes on other grids and found that many jobs were scheduled to other grid nodes. It now transpired that just in time evolution had reduced the number of jobs rejected from seven to three, as shown in Fig. 7.8.

From Fig. 7.9 and Fig. 7.10, it is clear how the evolution technique helps the grid to reduce its number of rejected jobs and increased those executed. The

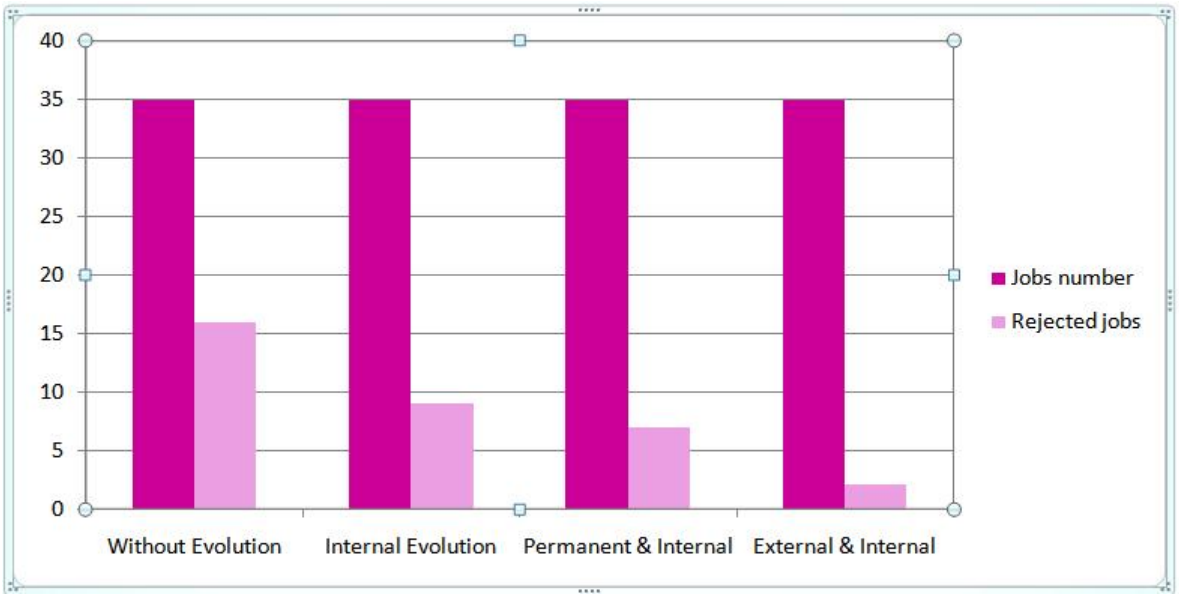

Figure 7.9: Rejected Jobs with both External and Internal Evolution

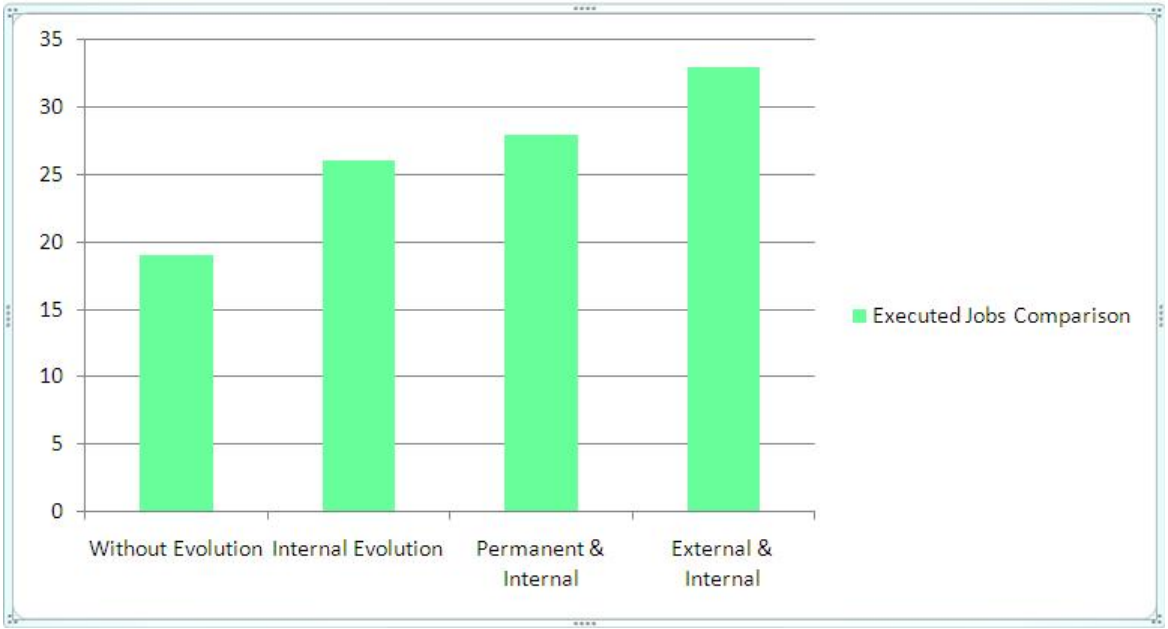

Figure 7.10: Executed Jobs Comparison

contribution of both types of evolution acting in concert in achieving this result is also quite obvious. These results clearly show the advantage that the approach has.

7.7 Summary

This chapter assessed the technique for grid evolution by using a simulation. It described the experiments done on this simulation after configuring four grids, each one has its own resources and by applying the evolution, the application software can migrate between those grids after checking policy, CPU speed, disk space and operating system. Jobs also can send to one of those grids to execute. This chapter also showed the results obtained from the use of grid evolution, which is clearly shown to help the grid reduce the number of rejected jobs and increase those executed, thereby increasing the employment of their resources as well as CPU utilisation.

Chapter 8

Conclusions and Future Work

8.1 Summary

Grid computing systems are the latest computing environments and have been gaining popularity for the past few years. They can be considered as extensions of distributed computing systems in which their number and heterogeneity are much higher.

The literature chapter's research on grid computing shows that there are hundreds of grids around the world, each one created to serve a specific group of researchers or users. The dream of grid computing is to create a single grid composed of many conjoined smaller grids that together form a global network of computers that can operate as one vast computational resource.

The present work found grid evolution to be a good mechanism with which to tackle this problem. A framework for grid evolution have been proposed, designed

and implemented. This framework allows grids to be flexible and to evolve dynamically at run time, and gives them the ability to acquire extra properties and attributes, thus executing more jobs and serving more users.

Evolution in grid computing means the addition of new functions and/or equipment or the deletion of unusable resources that affect performance in some nodes. The concept of internal and external evolution has been introduced . Internal evolution concerns evolution within a grid, which is done by migrating special resources such as application software and data from node to node in order to fulfil job requirements. External evolution extends the boundary of the grid to include other grids. It can be either just in time or permanent. The latter occurs if any addition or deletion occurs permanently, while just in time evolution occurs only when it is needed at run time, by enabling the job to be executed on other cooperation grids. A novel computational model has also constructed to support evolution. This model comprises objects (grids, nodes, application software, data, jobs and applications) that are managed and controlled by models (communication, termination, timing, failure and migration) and strategies (policies).

This research also developed a grid design based on this computational model that depends on a migration-based policy approach allowing the migration of application software, data and jobs within a grid environment to support evolution. This design allows grids to communicate with each other in order to exchange data, application software and jobs, which gives the grids the ability to enhance their

performance, execute more jobs and thus serve more users. The design consists of users, portals, resource brokers, information services, a number of nodes, monitors and other grids.

The resource broker developed in this design differs from existing ones because it is based on the migration framework, and enables a grid to communicate with others. It thereby controls external and internal evolution. To perform its functions, it consists of several components: discoverer, communicator (including reserver, user informer and dispatcher and receiver), evolution maker (including internal, permanent and just-in-time evolution) and failure recoverer. Resources are subject to change due to performance degradation and node failure. One of the main properties of these resources is that their characteristics change even during the execution of jobs.

Resource usage by jobs cannot therefore be static during run-time; neither can changes in resources be considered as faults. Grid job designers must therefore bear in mind that resources and resource management are highly dynamic within grid designs. This thesis therefore develops a new technique for grid monitoring which observes both inside and outside the grid environment. This is because fault tolerance is an essential characteristic and a necessary function of such environments in order to avoid the loss of computational time. The monitor in the design tracks job execution states by sending messages periodically to the nodes and asking for the status of each of their jobs, which it then saves. This helps the monitor to detect

any node or communication failure, which in turn enables the resource broker to recover this failure. This technique allows grids to detect and handle, and thereby recover from, failure.

A simulation tool called EVOSim has been developed to check the results. This simulation written in Oracle 10g is based on the computational model and design for evaluating the performance of grid computing after evolution. Many experiments have been performed on the simulation, and the results decisively confirmed the hypothesis that grid evolution reduces the number of rejected jobs and increases those that are executed, thereby increasing resource employment and CPU utilisation.

8.2 Contributions

This thesis has constructed a framework for the evolvable grid that insulates users from the complexities of grids. To this end, the framework has developed resource brokers together with grid monitors to cope with internal and external evolution, advance reservation, fault tolerance, the monitoring of grid environments, increased resource utilisation and the high availability of grid resources.

The research has presented the development of a mixture of user-centric and system-centric resource brokers based on the evolution framework, which uses a modular technique to handle user requirements as well as system throughput, in order to find, select and submit jobs. It is distinguished from others in that it insulates the user from grid middleware by automating the process from the point at

which the user's job requirements are received through to submission for execution at the appropriate resources. The resource broker can adapt a resource to meet a user's requirements using evolution: if the required resource does not have the required application software or data, the resource broker can migrate this application software or this data, depending on policies, from other resources. If the grid does not have the required resources at the required time, the resource broker can send the job to other grids for execution, and to retrieve the results after the execution time has elapsed. In addition, the monitor has been developed to establish checkpoints for the running jobs inside and outside the grid environment by taking a snapshot of their current states in order to allow them to be restarted from their previous states. It does this because fault tolerance is an essential characteristic and a necessary function for grid environments in order to avoid the loss of computation time.

Finally a grid evolution simulation (EVOSim) has been designed, developed and tested. It is written in Oracle 10g and has been used to create more than one grid, each of which had a different infrastructure include different nodes, application software, data and polices. The experiments were performed by submitting jobs to different grids at run time, and then comparing the results and analysing the performance of the grid using the approach of evolution and comparing it with that does not use this approach. The results of these experiments have demonstrated that these features significantly improve the performance of grid environments and

provide excellent scheduling results while clearly decreasing the number of rejected jobs.

8.3 Future Work

This thesis represents a step in ongoing research efforts to realise the vision of efficient and seamless grid evolution. Several issues still require investigation. The most obvious are as follows:

- Security becomes a major challenge in open distributed systems. Security issues have not been addressed in this research. One possible solution is to leverage existing research in grid security, such as Globus security infrastructure. Future work may extend this research and combine it with Globus protocols for security.

- The framework can be applied to the support of market models by allowing the grid to use the cheapest external grid.

- The expected execution time can be handled by constructing a system to allow the grid to estimate the job execution time without this time being specified by the user.

References

[1] Grid Cafe', http://www.gridcafe.org/.

[2] A. Abbas, *Grid Computing: Practical Guide To Technology & Applications*, 1st ed. Charles River Media, 2004.

[3] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the nimrod-g resource broker," *Future Gener. Comput. Syst.*, vol. 18, no. 8, pp. 1061–1074, 2002.

[4] O. Aldabbas, "A framework for mobility and temporal dimensions of grid systems," Ph.D. dissertation, School of Computing, De Montfort University, UK, 2008.

[5] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "Gridftp: Protocol extensions to ftp for the grid," 2001, http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf.

[6] G. Alliance, globus Alliance, http://www.globus.org/.

[7] A. Anjomshoaa, F. B. andMichel Drescher, D. F. andAn Ly, S. McGough, D. Pulsipher, and A. Savva, "Job submission description language (jsdl) specification, version 1.0," http://www.gridforum.org/documents/GFD.56.pdf.

[8] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation.* Prentice Hall, 2004.

[9] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "Gass: a data movement and access service for wide area computing systems," in *IOPADS '99: Proceedings of the sixth workshop on I/O in parallel and distributed systems.* New York, NY, USA: ACM, 1999, pp. 78–88.

[10] L. Borrett, "Distributed processing past, present and future," MicroHelp Computers and Communications, Tech. Rep., 1990, http://www.borrett.id.au/computing/art-1991-03-01.htm.

[11] M. L. Bote-lorenzo, Y. A. Dimitriadis, and E. Gomez-Sanchez, "Grid characteristics and uses: a grid definition," in *Across Grids 2003, LNCS 2970*, 2003, pp. 291–298.

[12] J. Bresnahan, M. Link, R. Kettimuthu, D. Fraser, and I. Foster, "Gridftp pipelining," in *2007 TeraGrid Conference*, 2007.

[13] J. Bresnahan, M. Link, G. Khanna, Z. Imani, R. Kettimuthu, and I. Foster, "Globus gridftp: what's new in 2007," in *GridNets '07: Proceedings of the first international conference on Networks for grid applications.* ICST, Brussels,

Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, pp. 1–5.

[14] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid." IEEE Computer Society Press, 2000, pp. 283–289.

[15] ——, "A case for economy grid architecture for service oriented grid computing," in *IPDPS '01: Proceedings of the 10th Heterogeneous Computing Workshop â"" HCW 2001 (Workshop 1)*. Washington, DC, USA: IEEE Computer Society, 2001.

[16] R. Buyya 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.

[17] L. Cardelli, "Abstractions for mobile computation," in *Secure Internet Programming*, 1999, pp. 51–94.

[18] A. Carzaniga, G. P. Picco, and G. Vigna, "Designing distributed applications with mobile code paradigms," in *ICSE '97: Proceedings of the 19th international conference on Software engineering*. New York, NY, USA: ACM, 1997, pp. 22–32.

[19] H. Casanova, "Simgrid: a toolkit for the simulation of application scheduling," in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001*, 2001, pp. 430–437.

[20] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw, "The legion resource management system," in *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing.* London, UK: Springer-Verlag, 1999, pp. 162–178.

[21] L. Chunlin and L. Layuan, "An agent-based approach for grid computing," in *PDCAT'2003. Proceedings of the Fourth International Conference*, 2003, pp. 608– 611.

[22] K. Czajkowski, I. Foster, and C. Kesselman, "Resource co-allocation in computational grids," in *In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8.* IEEE Computer Society, 1999, pp. 219–228.

[23] K. Czajkowski, I. T. Foster, N. T. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing.* London, UK: Springer-Verlag, 1998, pp. 62–82.

[24] K. Czajkowskiy, S. Fitzgeraldz, I. Foster, and C. Kesselmany, "Grid information services for distributed resource sharing," in *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2001, p. 181.

[25] T. Defanti, I. Foster, M. E. Papka, T. Kuhfuss, R. Stevens, and R. Stevens, "Overview of the i-way: Wide area visual supercomputing," 1996.

[26] J. Dollimore, T. Kindberg, and G. Coulouris, *Distributed Systems: Concepts and Design*. Addison Wesley, 2005.

[27] D. Dullmann, W. Hoschek, J. Jaen-Martinez, B. Segal, H. Stockinger, K. Stockinger, and A. Samar, "Models for replica synchronisation and consistency in a data grid." Los Alamitos, CA, USA: IEEE Computer Society, 2001, p. 0067.

[28] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Dept. of Information and computer science. Univ. of California Irvine, California, USA, 2000.

[29] P. W. L. Fong, "Viewer's discretion: Host security in mobile code systems," School of Computing Science, Simon Fraser University, Tech. Rep., 1998.

[30] G. U. C. for Particle Physics, gRIDPP: UK Computing for Particle Physics, http://www.gridpp.ac.uk/.

[31] O. G. Forum, open Grid Forum. Available at: http://www.gridforum.org/.

[32] I. Foster, "What is the grid? a three point checklist," 2002.

[33] I. Foster and C. Kesselman, "The globus project: a status report," *Future Gener. Comput. Syst.*, vol. 15, no. 5-6, pp. 607–621, 1999.

[34] ——, *The Grid2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[35] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *In Proceedings of the International Workshop on Quality of Service*, 1999, pp. 27–36.

[36] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," Open Grid Service Infrastructure WG, Global Grid Forum, Tech. Rep., 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[37] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid - enabling scalable virtual organizations," *International Journal of Supercomputer Applications*, vol. 15, p. 2001, 2001.

[38] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. Volume 5, pp. 237–246, 2002.

[39] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.

[40] D. S. GRID, dOE SCIENCE GRID, http://www.doesciencegrid.org/.

[41] A. S. Grimshaw, M. A. Humphrey, and A. Natrajan, "A philosophical and technical comparison of legion and globus," *IBM J. Res. Dev.*, vol. 48, no. 2, pp. 233–254, 2004.

[42] Y. Gu and R. L. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, 2007.

[43] A. Iamnitchi and I. Foster, "On fully decentralized resource discovery in grid environments," in *GRID '01: Proceedings of the Second International Workshop on Grid Computing*. London, UK: Springer-Verlag, 2001, pp. 51–62.

[44] T. I. E. T. F. (IETF), the Internet Engineering Task Force (IETF), http://www.ietf.org/.

[45] E. S. G. II, earth System Grid II, http://www.earthsystemgrid.org/.

[46] S. Infrastructure and R. iN Europe, sharing Infrastructure and REsources iN Europe, http://www.eugrid.eu/.

[47] B. Jacob, "Grid computing: What are the key components?" iBM, 2003.

[48] B. Jacob, M. Brown, K. Fukui, and N. Trivedi, "Introduction to grid computing," IBM International Technical Support Organization (Redbooks), Tech. Rep., 2005.

[49] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J.-Y. Girard, R. Strachowski, and S. S. Yu, "Enabling applications for grid computing with globus / redbooks," IBM, Tech. Rep., 2003.

[50] B. W. Jr, *An Introduction to Scientific Research.* Dover Publications, 1991.

[51] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Softw. Pract. Exper.*, vol. 32, no. 2, pp. 135–164, 2002.

[52] S. Lim, G. Fox, S. Pallickara, and M. Pierce, "Web service robust gridftp," T. . I. C. on Parallel, D. P. Techniques, and A. (PDPTA04), Eds., 2004.

[53] S. B. Lim, G. Fox, A. Kaplan, S. Pallickara, and M. Pierce, "Gridftp and parallel tcp support in naradabrokering," *Distributed and Parallel Computing*, vol. 3719/2005, pp. 93–102, 2005.

[54] J. Linn, "Generic security service application program interface, version 2," Network Working Group/OpenVision Technologies, Tech. Rep., 1997.

[55] M. Maheswaran and K. Krauter, "A parameter-based approach to resource discovery in grid computing system," in *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing.* London, UK: Springer-Verlag, 2000, pp. 181–190.

[56] I. Mandrichenko, W. Allcock, and T.Perelmutov, "Gridftp v2 protocol description," Global Grid Forum, Tech. Rep., 2005.

[57] P. Mockapetris, "Domain names concepts and facilities," Information Sciences Institute, University of Southern California, Tech. Rep., 1987, http://www.faqs.org/rfcs/rfc1034.html.

[58] A. Natrajan, M. Humphrey, and A. S. Grimshaw, "Capacity and capability computing using legion," in *ICCS '01: Proceedings of the International Conference on Computational Sciences-Part I.* London, UK: Springer-Verlag, 2001, pp. 273–283.

[59] Z. N. Németh and V. Sunderam, "A formal framework for defining grid systems," p. 202, 2002.

[60] T. N. I. C. G. (NWICG), the Northwest Indiana Computational Grid (NWICG), http://dev.nwicgrid.org/.

[61] J. Postel, "User datagram protocol," Information Sciences Institute, University of Southern California, RFC 768, 1980, http://www.faqs.org/rfcs/rfc768.html.

[62] ——, "Internet control message protocol," Network Working Group, RFC 792, 1981, http://www.faqs.org/rfcs/rfc792.html.

[63] ——, "Internet protocol," Information Sciences Institute, University of Southern California, RFC 791, 1981, http://www.faqs.org/rfcs/rfc791.html.

[64] ——, "Transmission control protocol," Information Sciences Institute, University of Southern California, RFC: 793, 1981, http://www.faqs.org/rfcs/rfc793.html.

[65] ——, "Transmission control protocol," University of Southern California, Tech. Rep., 1981.

[66] A. project, astroGrid project,http://www.astrogrid.org/.

[67] C. A. C. project, compute Against Cancer project, http://www.computeagainstcancer.org/.

[68] E. project, ecogrid project, http://ecogrid.nchc.org.tw/.

[69] F. project, folding@home project, http://folding.stanford.edu/.

[70] T. B. Project, the BioinfoGRID Project, http://www.bioinfogrid.eu/.

[71] T. D. Project, the DataGrid Project, http://eu-datagrid.web.cern.ch/eu-datagrid/.

[72] D. D. Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt, "The evolution of the grid," in *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003, pp. 65–100.

[73] R. R. Schaller, "Moore's law: past, present, and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[74] SETI@Home, http://setiathome.ssl.berkeley.edu/.

[75] L. Smarr and C. Catlett, "Metacomputing," *Commun. ACM*, vol. 35, no. 6, pp. 44–52, 1992.

[76] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The microgrid: A scientific tool for modeling computational grids," *Sci. Program.*, vol. 8, no. 3, pp. 127–141, 2000.

[77] J. W. Stamos and D. K. Gifford, "Remote evaluation," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 4, pp. 537–564, 1990.

[78] K. Swenson, "Simple workflow access protocol (swap)," http://www.isr.uci.edu/events/twist/wisen98/presentations/Swenson/.

[79] R. Thurlow, "Rpc: Remote procedure call protocol specification version 2," Sun Microsystems, Network Working Group, Tech. Rep., 1988.

[80] S. Verma, M. Parashar, J. Gawor, and G. v. Laszewski, "Design and implementation of a corba commodity grid kit," in *GRID '01: Proceedings of the Second*

International Workshop on Grid Computing. London, UK: Springer-Verlag, 2001, pp. 2–13.

[81] F. via Network-Enabled Recursion, factoring via Network-Enabled Recursion, http://www.lehigh.edu/ bad0/fafner.html.

[82] G. von Laszewski, "Grid computing: Enabling a vision for collaborative research," in *PARA '02: Proceedings of the 6th International Conference on Applied Parallel Computing Advanced Scientific Computing*. London, UK: Springer-Verlag, 2002, pp. 37–52.

[83] T. Walsh, P. Nixon, and S. Dobson, "Review of mobility systems," TCD Computer Science Technical Report, Tech. Rep., 2000.

[84] W. Webb, "laying down the law," *IEE Communications Engineer*, February/-March 2006.

[85] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems,Elsevier*, vol. 15, pp. 757–768(12), 1999.