A Framework for Resource Management in A Grid Environment

A Thesis submitted

for the award of the degree of

DOCTOR OF PHILOSOPHY

by

Inderveer Chana

(9040353)

under the Guidance of

Dr. Seema Bawa

Professor & Head

Computer Science and Engineering Department

Thapar University, Patiala -147004

Computer Science and Engineering Department

Thapar University, Patiala – 147 004, INDIA

June 2009

Dedicated

to

my Spiritual MASTER

for HIS

Divine Blessings

Contents

List of Figures v

List of Tables ix

Certificate x

Acknowledgement xi

Abstract xiii

1 Introduction **1**

1.1 About Grid 2

1.1.1 Beginnings of Grid 5

1.1.2 Grid and Other Technologies 7

1.2 Grid Fundamentals 9

1.2.1 Grid Architecture 9

1.2.2 Grid Standards 11

1.3 Challenges in Grid Computing 13

1.3.1 Security Issues 13

1.3.2 Resource Management Issues 14

1.3.3 Data Management Issues 15

1.4 Grid Resource Management: The Research Motivation 16

1.5 Thesis Outline 18

1.6 Thesis Contributions 19

2 Literature Review **21**

2.1 Grid Computing 22

2.1.1 Resource Management: Core of Grid Computing 23

2.1.2 Resource Management in OGSA based Grids 24

2.2 Resource Management in Grid Middleware 26
2.2.1 Globus 27
2.2.2 UNICORE 28
2.2.3 Legion 29
2.2.4 Alchemi 31
2.2.5 SUN N1 Grid Engine 33
2.2.6 Condor 35
2.2.7 Comparative Analysis of Resource Management Techniques 36
2.3 Grid Resource Management: State of the Art 37
2.3.1 Attributes of Grid Resource Management Systems 39
2.3.2 Grid Brokers and Metaschedulers 52
2.3.3 Existing Resource Management Frameworks 53
2.4 Monitoring and Fault Management Systems 57
2.4.1 Fault Management Mechanisms in Various Grid Middleware . . . 60
2.4.2 Comparative Analysis of Fault Management in Grid Middleware . 64
2.5 Related Technologies 65
2.5.1 Semantic Web Technologies 66
2.5.2 Autonomic Computing 68
2.6 Problem Formulation 69

3 PRATHAM: Proposed Grid Resource Management Framework 72

3.1 Goals of Proposed Framework 73
3.2 Framework Requirements 73
3.2.1 UML: Tool for Analyzing Framework Requirements 74
3.2.2 Functional Requirements View 74
3.2.3 Static Structural View 75
3.2.4 Dynamic Behavior View 76
3.3 Framework Architecture 78
3.3.1 Fabric Layer 79
3.3.2 Grid Middleware Layer 80
3.3.3 Broker Layer 80

3.3.4 Information Layer 81
3.3.5 Application Layer 82
3.4 Conclusions 83

4 Design and Implementation of PRATHAM **88**
4.1 PRATHAM Taxonomy 89
4.1.1 Resource Model 89
4.1.2 Scheduling Model 90
4.2 PRATHAM Workflow 91
4.3 Interoperability in PRATHAM 92
4.3.1 Resource Repository 93
4.3.2 Implementation Details 94
4.3.3 Resource Discovery through Semantics 95
4.3.4 Implementation Details 98
4.4 Resource Provisioning in PRATHAM 103
4.4.1 Quality of Service (QoS) Requirements 104
4.4.2 Advanced Reservation of Resources 108
4.4.3 Implementation Details 109
4.5 PRATHAM Autonomic System 110
4.5.1 Autonomic Framework 113
4.5.2 Architecture of Monitoring System 116
4.5.3 Components of Monitoring System 117
4.5.4 Implementation Details 118
4.6 Incorporating Self-Management in PRATHAM 121
4.6.1 Fault Detection and Healing System (FDHS) 122
4.6.2 Components of FDHS 122
4.6.3 Implementation Details of FDHS 123
4.7 Grid Portal: Manageability Interface for PRATHAM 124
4.7.1 Portal Architecture 126
4.7.2 Grid Portlets 127
4.7.3 Implementation Details 128

4.8 Conclusions 128

5 Deployment, Testing and Validation of the Framework 130

5.1 Framework Deployment and Evaluation 131

5.1.1 Case Study: Thapar University Campus Grid 131

5.1.2 Experimental Results 134

5.2 Framework Validation 138

5.3 Conclusions 139

6 Conclusions and Future Directions 147

6.1 Conclusions 148

6.2 Future Directions 150

References 151

Publications 165

Appendix A: Information Generated by Jena 167

Appendix B: Installation Details 172

List of Figures

1.1 Examples of Internet Usage [94] 2
1.2 Scope of Grid (Source: Ian Foster) 3
1.3 IT Progress (Adapted from Globus Alliance) 5
1.4 The layered Grid architecture and its relationship to the TCP/IP protocol architecture [17] 9
1.5 Emergence of Open Grid Standards [23] 12

2.1 Scope of Grid [7] 23
2.2 Resource Management in Globus (Source- The Globus Project) 27
2.3 Components and their relationship in Alchemi.NET 32
2.4 Resource Management Architecture of SUN Grid Engine 34
2.5 Resource Management System (Abstract View)[24] 38
2.6 Resource Model Taxonomy [24] 40
2.7 Hierarchical Model 43
2.8 Actions involved in matchmaking process 45
2.9 Pulse Monitoring [49] 49

3.1 Use Case Diagram for user Authentication and Resource Discovery . . . 75
3.2 Use Case Diagram for Resource Monitoring and Job Status 76
3.3 Use Case Diagram for Advanced Reservation of Resources and Payment . 76
3.4 Class Diagram for Grid Entities (user) 77
3.5 Class Diagram for Grid Entities (administrator) 78
3.6 PRATHAM Components 79
3.7 Activity Diagram for User Authentication 84
3.8 Activity Diagram for New User Registration 84

3.9 Activity Diagram for Resource Discovery 85
3.10 Activity Diagram for Administrator 85
3.11 Sequence Diagram for Resource Provisioning 86
3.12 Collaboration Diagram for Resource Discovery and Provisioning 86
3.13 Deployment Diagram for the Framework 87

4.1 Resource Model Taxonomy 90
4.2 PRATHAM Workflow 91
4.3 Components of Resource Repository Architecture 94
4.4 Ontology based Grid Resource Discovery Approach 96
4.5 Layers in a typical Grid Information System 97
4.6 Grid Ontology Classes in Protege 99
4.7 Concept Classification Tree for PRATHAM 101
4.8 Properties of Classes 102
4.9 Steps followed in searching a Resource Semantically 103
4.10 Detailed Information Flow Diagram of Ontology based Resource Discovery 104
4.11 Steps followed in Resource Provisioning 111
4.12 Autonomic Capabilities for Grid 112
4.13 Workflow of Autonomic Manager 114
4.14 Components of Monitoring System 117
4.15 Components of Resource Monitoring System 120
4.16 Fault Detection and Healing System 122
4.17 Resource Monitoring Scenario 125
4.18 PRATHAM Portal Architecture 127

5.1 Grid setup at Thapar University 132
5.2 Successful installation of GridBus Broker v2.4 133
5.3 Thapar University Grid Portal 135
5.4 Registration of new Grid Users 136
5.5 Resource Browser Portlet 137
5.6 Resource Discovery requirements 138
5.7 Result of Query 139

5.8 Parameters for Resource Provisioning 140
5.9 Provisioned Sets 140
5.10 Comparison of Symmetric and Semantic based search 141
5.11 Snapshot of Globus Database 143
5.12 Snapshot of Alchemi Database 143
5.13 Graph generated by Ganglia showing CPU Load 144
5.14 Graph generated by Ganglia showing Network Traffic 144
5.15 Graph generated by Ganglia showing Disk Usage 145
5.16 Graph generated by Ganglia showing Memory Usage 145
5.17 Report generated by Ganglia 146
5.18 XML created by Ganglia 146

List of Tables

2.1 Comparative Analysis of Resource Management in Grid Middleware . . . 37

2.2 Comparison of Existing Resource Monitoring Systems 51

2.3 Comparative Analysis of Fault Management in Grid Middleware 65

4.1 PRATHAM Taxonomy 90

4.2 Resource Database Schema 106

4.3 Information regarding individual resources 109

4.4 Information regarding Best of Breed resources 109

4.5 Pulse Data 119

5.1 PRATHAM vs. other RMS Frameworks 141

5.2 PRATHAM vs. other RMS Frameworks (Academic-initiatives) 142

CERTIFICATE

I hereby certify that the work which is being presented in this thesis, **"A Framework for Resource Management in A Grid Environment"**, for the award of degree of **"Doctor of Philosophy"**, submitted to the Department of Computer Science and Engineering, Thapar University, Patiala, is an authentic record of my own work, carried out under the supervision of Dr.Seema Bawa. It refers to other researchers' works, which have been duly listed in the reference section.

The matter presented in this thesis has not been submitted in part or full to any other institute or university, for the award of any other degree.

Inderveer Chana

Reg.No. 9040353

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Seema Bawa (Ph.D)

Supervisor

Professor and Head

Department of Computer Science and Engineering

Thapar University

Patiala - 147 004

Punjab, INDIA

Acknowledgements

My sincere acknowledgement and deep appreciation for the entire group who contributed in this thesis endeavor.

At the outset, I would like to express my profound appreciation and gratitude to my revered supervisor Dr.Seema Bawa, a dynamic manager and an enthusiastic peer. It was an opportunity and a great pleasure to do this thesis under her direction. She not only motivated me but also provided congenial research environment. Without her advice and observations this work would have not been possible. Thank you madam!

I earnestly acknowledge Thapar fraternity for providing resourceful research environment at Thapar. My gratitude towards the Doctoral Committee, Dr.R.K.Sharma, Dean, Academic Affairs and Dr.Rajesh Kumar who persistently monitored the research progress, ensured quality work and pegged their invaluable suggestions. I also wish to pay my due regards to Dr.Susheel Mittal, Dean, Research and Sponsored Projects and Dr.K.K.Raina, Deputy Director, who had been kind enough to advise and help in their respective roles. My deep regards to Dr.Abhijeet Mukherjee, Director, Thapar University for inspiring the faculty for research and supporting research candidly.

My sincere gratitude to Computer Science and Engineering Department, Grid research group and my wonderful students for contributing significantly in the deployment and testing effort throughout the research work. I greatly appreciate Dr.Maninder Singh for his generous time and invaluable suggestions throughout the research work especially in the issues of networking and security. I extend my thanks to Mr.Anil Vashisht for encouraging and strengthening my morale during tough times. I would also like to thank my colleagues and Krishan Kumar ji for being considerate.

Last, but not the least, I would like to acknowledge my pillars of strength, my parents who had been a constant support and source of inspiration. No words of thanks can

ever match their love and affection. Mama and Papa thank you for being there always. I also appreciate the concern and support of Jasmine and Lukesh. Thank you for making life fun while working. My special acknowledgement to my little daughter Ishita for being patient and allowing me to spend most of my time in this thesis. Thanks Isha for giving me every reason to smile.

Above all, I am grateful to Almighty and my Master for conferring this opportunity and for bestowing HIS Kripa Varsha on me.

Inderveer Chana

Abstract

Grid Computing heralds the dawn of a new paradigm for next-generation high performance computing. As a descendent of distributed computing, it enables selection, sharing and aggregation of geographically distributed resources for solving intricate problems of science, engineering and commerce, which have phenomenal computational needs. It offers reasonable, consistent and pervasive access to high-end computation, mainly as a result of availability of faster hardware, more sophisticated software and escalation of Internet. The principles of Grid computing focus on large-scale resource sharing in distributed systems in a flexible, secure, and coordinated fashion. Resource management forms the core of Grid computing. Resources in a Grid are often geographically distributed, heterogeneous and dynamic in nature. Resources are owned by different organizations with different usage policies, cost models, varying loads and availability patterns. The conventional resource management schemes are based on the assumption that resources are owned by a single organization and therefore, would be available exclusively. There are relatively static models that have centralized controller that manage jobs and resources accordingly. These assumptions do not hold true for a Grid environment. Therefore, in spite of a number of advances in Grid Computing, resource management persists to be a challenging, complex, yet most significant task and has been the main focus of this research work.

Initially, a detailed review of the work done in the area of Grid Resource Management has been done. It analyzes, compares and reports existing approaches to Grid resource management, state of the art in Grid resource management and the challenges, that need to be addressed. A comparative analysis of middleware technologies has been done for Grid resource management and fault management mechanisms. The existing Grid resource management systems have also been explored for their resource discovery, resource monitoring, job execution, provisioning, brokering and scheduling scenarios. The fault management approaches in existing middleware and existing monitoring systems have been explored, compared and reported. Related technologies like Semantic Web Technologies and Autonomic Computing have also been studied and reported. Based

on the literature survey, it is apparent that issues of interoperability, provisioning and fault management are the main challenges besides numerous other issues that need to be addressed. To address these diverse Grid resource management challenges, a resource management framework, named PRATHAM has been initially proposed and further designed, developed and tested in this work.

The proposed Grid Resource Management framework, PRATHAM inspires from Semantic Web Technologies and Autonomic Computing. It has a five-layer architecture, with Fabric layer as the base layer which constitutes the resources; second layer is Grid Middleware, and it comprises of the middleware. Next is Broker layer, which renders self-management to the framework; fourth, the Information layer, realizes the interoperability and provisioning capabilities of the Grid. The top-most fifth layer is the Portal, which provides an interface to the Grid.

PRATHAM provides a hierarchical model based distributed framework and it works well for large-scale environments. It is an endeavor to provide Grid users not only the freedom of higher operability but also enhanced resource provisioning within a Grid environment, along with better monitoring and self-management facilities. Interoperability has been addressed through semantics; user requirements have been optimized and provisioned by providing facility of advanced reservation of resources to the user. Apart from this, it provides a monitoring system based on pulse monitoring which has been implemented through ganglia and helps in monitoring individual resources and status of resource queues; self-management has been realized through Autonomic Computing principles. This research work implements PRATHAM through Globus Toolkit 4 and Alchemi.NET besides various other tools. The framework has been deployed at Thapar University, Patiala. Experiments conducted clearly demonstrate its usage and various features. Finally, the framework has been compared with existing Resource Management frameworks and Resource Management systems to validate the outcomes.

The results show that PRATHAM successfully and collectively addresses the issues of interoperability; resource provisioning and self-management, the OGSA requirements for future Grids.

Chapter 1

Introduction

Numerous initiatives and architectures have been developed under the broad aegis of "Distributed Computing". Distributed computing is often used to describe a type of computing in which different computing nodes can simultaneously run an application located on different computers that are connected by a communication network. Advances in networking technology and computational infrastructure make it possible to construct large-scale high performance distributed computing environments that provide dependable, consistent and pervasive access to high-end computation.

A flavor of distributed computing where idle computing power and storage space of thousands of networked systems can be harnessed to work together on a particularly processing-intensive problem is the foundation for the current model of computation called "Grid Computing". The growth of Internet has enabled organizations to build distributed Grid environments, which are dependable and inexpensive and in future, it is professed that this will pave way for truly pervasive computing.

This chapter gives the preamble of Grid Computing. It focuses on how Grids have evolved from distributed computing, highlights fundamentals of Grid Computing and the major challenges and issues in this area. It also discusses about the motivation behind this research work, the organization of the thesis, along with its contributions.

1.1 About Grid

"Beyond the Net, lies the Grid!"

- The Net allows users everywhere, to share information.
- The Grid allows users to share raw computing power [1].

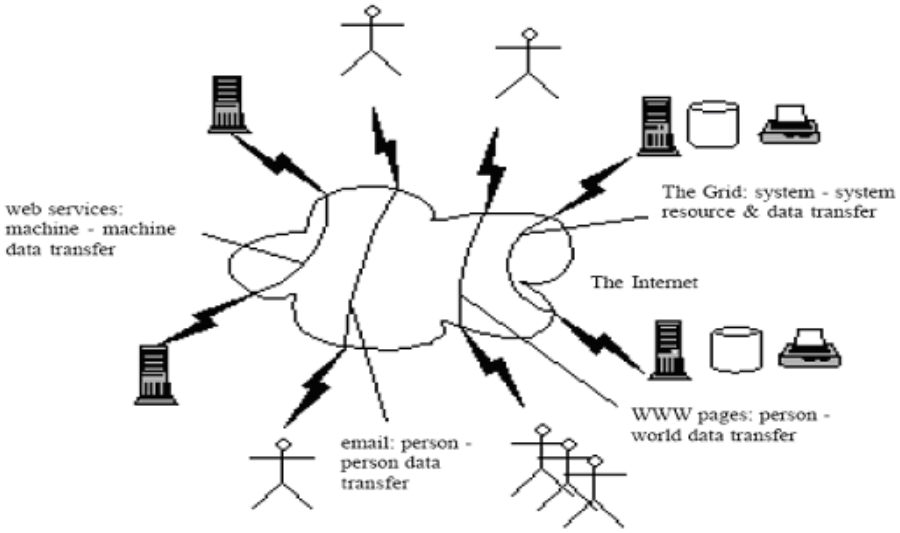

Figure 1.1: Examples of Internet Usage [94]

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level and is visualized as a large number of heterogeneous systems pooled in to act as a simple, yet large and powerful self-managing solitary virtual computer [1].

The last decade has seen a considerable increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software and escalation of Internet (as shown in Figure 1.1 [94]). Still, there are problems in the fields of science, engineering and business, which cannot be dealt effectively with the current generation of supercomputers. Due to their size and complexity, these problems are often numerically and/or data intensive and require a variety of heterogeneous resources that are not available from a single machine [2]. To address this challenge, a number of teams have conducted experimental studies on the cooperative use of geographically distributed resources conceived as a single powerful computer. This new approach is known by several names, such as, metacomputing, seamless scalable computing, global computing, and more recently Grid Computing [3],[19].

The goal of Grid Computing, which gets its name from its electric power-grid like architecture, is to link surplus computing power and other spare IT resources with clients who have periodic needs beyond the capacity of their machines. With the advent of this new technology, now it has been realized, that paralleling sequential applications could yield faster results and most of the times at lower costs. Grid Computing is technically a category of High-Performance Computing (HPC), which traditionally refers to Super Computing, but it differs in a number of ways from the conventional HPC environments [18]. Most traditional HPC technology is based on fixed and dedicated hardware, in combination with specialized operating systems and software that work together to produce a high-performance environment. Grids, on the other hand, can use commodity hardware, different operating systems, software and platforms, and even be configured to use the spare capacity in existing infrastructure. The use of idle resources of remote machines considerably increases computational power, enhances data storage, data recovery and supplies with further facilities and thus differentiates it from HPC [5]. Different types of Grids can be used to provide different types of services to the end user:

- Computational Grids for computational services
- Data Grids for data-intensive services
- Application Grids for application services

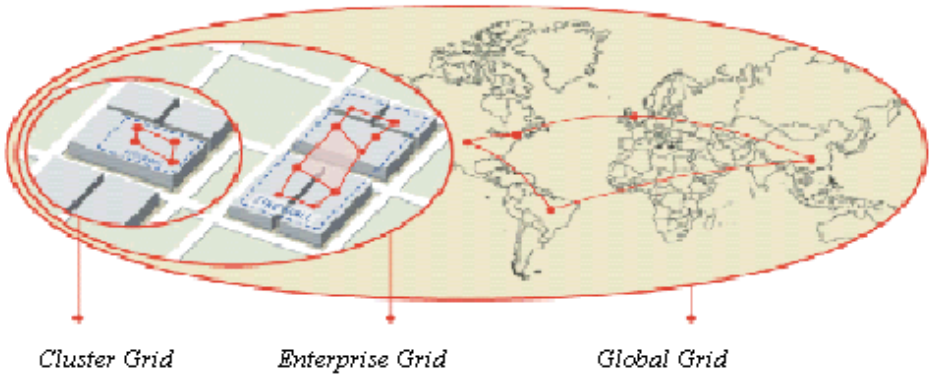

Figure 1.2: Scope of Grid (Source: Ian Foster)

Grids can further be categorized into a three stage model of Departmental or Cluster Grids, Enterprise Grids and Global Grids as shown in Figure 1.2 (Source: Ian Foster).

These correspond to a firm initially utilizing resources within a single group i.e. an engineering department connecting desktop machines, clusters and equipments. This progresses to Enterprise Grids where non-technical staff's computing resources can be used for cycle-stealing and storage. A Global Grid is a connection of Enterprise and Departmental Grids which can be used in a commercial or collaborative manner[4].
Grid Computing appears to be a promising trend for the following reasons [96]:

- In today's scenario, almost every organization possesses a large number of computers, which means that it has huge processing power, but it remains heavily under-utilized. Thus, existing set of computer resources can be put to an optimum use (when idle).

- Cost-effective solutions can be provided as instead of buying new processing power, existing, under-utilized infrastructure would be used.

- There are cross-national projects with users and resources in different domains and in future, due to IT progress, multinational projects can be perceived. Thus, compute intensive problems can now be cost effectively addressed.

- Now there is a possibility for resources of many idle computers to be organized as a single virtual collaboration. This can be cooperatively harnessed and managed towards a common objective by making resources' location irrelevant. Thus it provides a kind of "plug-n-play" access to computational resources.

Dream of Grid Computing has been realized as last decade has witnessed a sea change in the IT industry. Processor speed is increasing much more than predicted by Moore's law and advancements in networks are too rapid. Following are some of the IT progress facts that give supporting base and stronger resources to Grid Computing [52]:

- Network vs. computer performance: Computer speed doubles every 18 months; Network speed doubles every 9 months

- 1986 to 2000: Computers: 500 times faster; Networks: 340000 times faster

- 2001 to 2010 (projected): Computers: 60 times faster; Networks: 4000 times faster

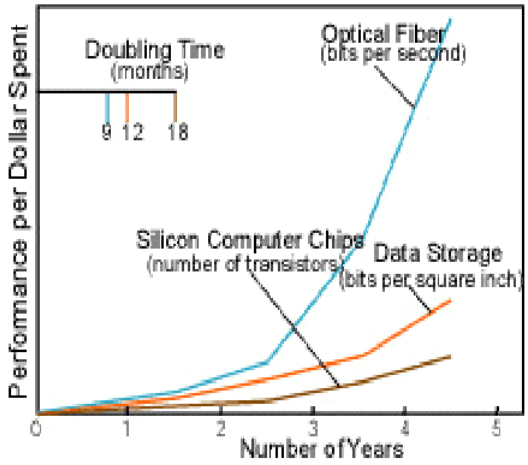

Figure 1.3: IT Progress (Adapted from Globus Alliance)

Figure 1.3 [52] shows that CPUs are fast enough; Networks are also very fast and this should be put to some effective use.

At the moment, there are a variety of Grids available, which offer a way to solve grand challenge problems like protein folding, financial modeling, earthquake simulation, and climate/weather modeling etc. Moreover, these Grids also provide a means for offering information technology as a utility for commercial and non-commercial clients. The clients pay only for what they use, as with electricity or water. In future, Grids could also provide the kind of power needed to transmit holographic images; allow instant online gaming with hundreds of thousands of players, and offer high-definition video telephony for the price of a local call [93]. The latest prediction is that the Internet may soon be made obsolete by "the Grid".

After an overture to Grid Computing, the next section would be discussing the origin of Grid Computing and how it has evolved till date.

1.1.1 Beginnings of Grid

In order to describe the current trends in Grid Computing, it would be ideal to put the concept in perspective of its origins. Grids emerged at the time when applications used by engineers and researchers were run on High Performance Computing (HPC) clusters to side-step the high costs of supercomputers [83]. The term "Grid" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering [1]. Grid computing concepts were first explored in the 1995 I-WAY

experiment, in which high-speed networks were used to connect, for a short time, high-end resources at 17 sites across North America [4]. Out of this activity grew a number of Grid research projects that developed the core technologies for "production" Grids in various communities and scientific disciplines. The SETI@home project, launched in 1999, is a widely-known example of a very simple Grid Computing project. Although it was not the first to use such techniques, and did not use all of the facilities of current Grid capabilities, but it has been followed by many others. For example, the US National Science Foundation's National Technology Grid and NASA's Information Power Grid are both creating Grid infrastructures to serve university and NASA researchers, respectively. Across Europe and the United States, the closely related European Data Grid, Particle Physics Data Grid and Grid Physics Network (GriPhyN) projects [8] plan to analyze data from frontier physics experiments. Outside the specialized world of physics, the Network for Earthquake Engineering Simulation Grid (NEESgrid) [9] aims to connect US civil engineers with the experimental facilities, data archives and computer simulation systems used to engineer better buildings. The Grid stories have been unfolding and expanding day by day as research in the related areas of networking, digital libraries, peer-to-peer computing, co-laboratories has been providing additional ideas relevant to the Grid.

In the late 1990s, Grid researchers came together in the Grid Forum, subsequently expanding to the Global Grid Forum (GGF) [10], where much of the early research evolved into the standards base for future Grids. Today, the Grid has gone global, with many worldwide collaborations and funding agencies, commercial vendors, academic researchers and numerous national centers and laboratories coming together to form a community of broad expertise with enormous commitment for building the Grid. In 2006, the Global Grid Forum merged with the Enterprise Grid Alliance and formed the Open Grid Forum (OGF) [86], [11]. At the moment, OGF is the international community dedicated to accelerating Grid adoption to enable business value and scientific discovery by providing an open forum for Grid innovation and developing open standards for Grid software interoperability.

Grid evolution can further be seen from Cern [6], the particle physics centre that created the Web; The Internet has evolved by linking together a hotchpotch of cables and routing equipment, by contrast, the Grid at Cern has been built with dedicated fibre optic ca-

bles and modern routing centres, meaning there are no outdated components to slow the deluge of data. The power of the Grid will become apparent this summer, the summer of 2008, after what scientists at Cern have termed their "red button" day – the switching on of the Large Hadron Collider, the new particle accelerator built to probe the origin of the universe. The Grid at Cern will be activated at the same time to capture the data it generates [93].

With this background and history, the next section will compare Grid with other technologies of this era.

1.1.2 Grid and Other Technologies

Many technologies have preceded Grid Computing. Evaluation of some of these is presented below:

(a) Grid vs. Web

Main objective of Grid is to solve compute or data intensive complex problems where as Web aims at e-commerce and serves generally. Grid requires authentication check, which is not required in case of Web. Grid architectures aim at reliability and security but there is no inherently reliable and secure data transmission on the Web as such. Current Internet technologies address communication and information exchange among computers but do not provide integrated approaches to the coordinated use of resources at multiple sites for computation [14].

(b) Grid vs. Distributed Computing

Grid is a core concept in modern distributed computing architectures and is aligned with other important Distributed Computing technologies such as virtualization, service orientation and data center automation [11]. Grid Computing is distinguished from conventional Distributed Computing by its focus on large-scale resource sharing, innovative applications, and in some cases, high-performance orientation.

(c) Grid vs. Cluster Computing

Both Clusters and Grids operate on the same underlying principle that a group of computers acts as one but in Clusters, a centralized resource manager performs the resource allocation. In a Cluster, all nodes work together cooperatively as a single unified re-

source whereas in a Grid, each node has its own resource manager [12]. Apart from this following are some other differences:

- Clusters primarily comprise of homogeneous computational resources whereas Grids have heterogeneous computational resources.
- Size of Virtual Organization (VO) may change as resources may join and leave the Grid but Clusters typically contain a static number of processors and resources.
- Clusters are in close physical proximity to one another, while Grids can be geographically distributed.
- Many Grids incorporate clusters among the resources they manage.

Some other current distributed computing technologies too do not address all the concerns and requirements [13], [129]. For example, Enterprise distributed computing technologies such as CORBA and Enterprise Java enable resource sharing within a single organization. The Open Group's Distributed Computing Environment (DCE) supports secure resource sharing across sites, but it is thought to be burdensome and inflexible. Storage Service Providers (SSPs) and Application Service Providers (ASPs) allow organizations to out source storage and computing requirements to other parties, but only in constrained ways: for example, SSP resources are typically linked to a customer via a Virtual Private Network (VPN). Emerging "Distributed computing" companies seek to harness idle computers on an international scale, but to date, support only highly centralized access to those resources [14], [17].

The latest incarnation of utility computing of 2007 - Cloud Computing [97] - is to enable organizations to manage global sets of resources as a virtualized cloud [176]. Again the idea is that instead of owning the computing power, simply log into an existing one, use it, get the results, pay for it and then log off. However, vendors have concentrated on offerings that make it easier for companies to set up their own clouds, as most companies still want to own their IT resources and remain unwilling to use someone else's utility to run critical workloads [95]. Moreover, Cloud-based services are too difficult to measure and justify enterprise deployment.

In summary, current technology either does not accommodate the range of resource types

or does not provide the flexibility and control on sharing relationships needed to establish Virtualizations. It is here that Grid technologies enter the picture. Grid technologies complement rather than compete with existing distributed computing technologies [15]. Grid Computing 'fabrics', combining virtualization, SOA, commodity hardware and open source software are poised to become the foundation for next-generation enterprise IT architectures. Grid technologies can be used to establish dynamic markets [16] for computing and storage resources, hence further overcoming the limitations of current static configurations.

This section discussed about origin of Grids, progress of Grid Computing and its comparison with related technologies; the next section discusses Grid architecture and its standard bodies.

1.2 Grid Fundamentals

Grid architecture and standards identify fundamental Grid system units; Architecture identifies the components and the Standards specify the rationale and utility of these components, and indicate how these components interact with one another.

1.2.1 Grid Architecture

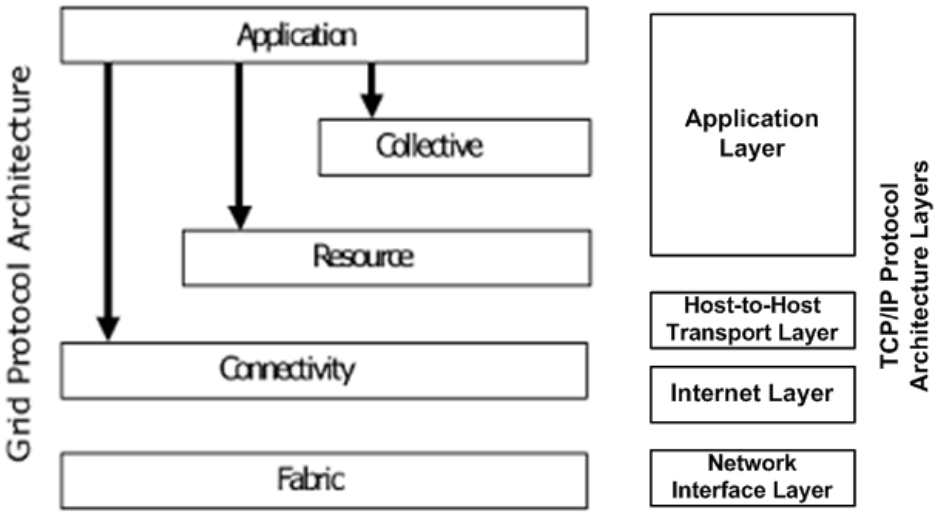

Figure 1.4: The layered Grid architecture and its relationship to the TCP/IP protocol architecture [17]

Grid architecture is often described in terms of "layers", each providing a specific

function. The higher layers are generally user-centric, whereas lower layers are more hardware-centric. Each layer shares the behavior of the underlying component layers. Figure 1.4 [17] shows the layered Grid architecture as it maps to Internet Protocol Architecture. The Internet Protocol architecture extends from network to application; this protocol architecture defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage, and share resources. Therefore, there is a mapping from Grid layers into Internet layers [17].

Fabric Layer: The base layer of the architecture is known as the Fabric and it comprises of the computational resources, storage systems, catalogs, network resources, sensors etc. The Grid protocols may access these resources on sharing basis. These components implement local resource specific operations as a result of higher-level operations.

Connectivity Layer: The core communication and authentication protocols required for Grid-specific network transactions are defined in the Connectivity layer. Communication protocols are required for communication between the fabric resources and cryptographically secure mechanisms for verifying that the identity of users and resources are ensured by Authentication protocols.

Resource Layer: Individual resource operations like monitoring, control etc. are conducted in this layer. Resource layer implementations of these protocols call Fabric layer functions to access and control local resources. These protocols are concerned with individual resource only.

Collective Layer: The Collective layer protocols are global in nature and capture interactions across collections of resources and are not associated with individual resources.

Application Layer: The top-level layer in the Grid architecture is known as Application Layer and comprises of the user applications that operate within a VO environment.

This architecture has been designed for controlled resource sharing with improved interoperability among participants [17]. In most common Grid architectures, the application layer also provides the service ware, the sort of general management functions such as measuring the amount, a particular user employs the Grid, billing for this use (assuming a commercial model), and generally keeping accounts of who is providing resources and who is using them - an important activity when sharing the resources of a variety of institutions amongst large numbers of different users.

After an introduction to Grid architecture, next section will discuss Grid Standards.

1.2.2 Grid Standards

The Global Grid Forum (GGF), now named as OGF, is an open community initiative that aims to promote and support the development, deployment, and implementation of Grid technologies and applications via the creation and documentation of 'best practices'-technical specifications, user experiences, and implementation guidelines [20]. Two main specifications that are the result of the GGF initiative for Grid computing practices are:

- Open Grid Services Architecture (OGSA)
- Open Grid Services Infrastructure (OGSI).

The success of the Grid, to date, owes to service oriented Open Grid Services Architecture that today forms the basis for both open source and commercial products. OGSA defines the technology and OGSI defines the framework. The base component of the OGSA architecture is OGSI. This is a Grid software infrastructure standard based on the emerging Web services standards. The goal of OGSI is to provide maximum interoperability among OGSA software components. The Global Grid Forum (GGF) has embraced the Open Grid Services Architecture (OGSA) as the blueprint for standards-based Grid Computing [21]. It was first announced at GGF4 in February 2002. In March 2004, at GGF10, it was declared as the GGF's flagship architecture. OGSA is an Informational specification that aims to define a common, standard and open architecture for Grid based applications [22]. It advocates building an Open Grid contributed by:

- Open Standards
- Open Source
- Open Infrastructure

The goal of OGSA is to standardize almost all the services that a Grid application may use, for example job and resource management services, communications and security etc. OGSA specifies a Service-Oriented Architecture (SOA) for the Grid that realizes a model of a computing system as a set of distributed computing patterns realized using

Web Services as the underlying technology. OGSA and the software built around it permit the resources and systems of the entire aggregation of heterogeneous architectures to be managed in an automated fashion. It will configure, optimize, heal and protect itself with minimal human intervention. In short, it will be self-managing [20]. OGSA is being developed by both the United States and European initiatives aiming to define core services for a wide variety of areas including [21]:

- Systems management and automation
- Workload/Performance management
- Security
- Availability/Service management
- Logical resource management
- Clustering services
- Connectivity management
- Physical resource management

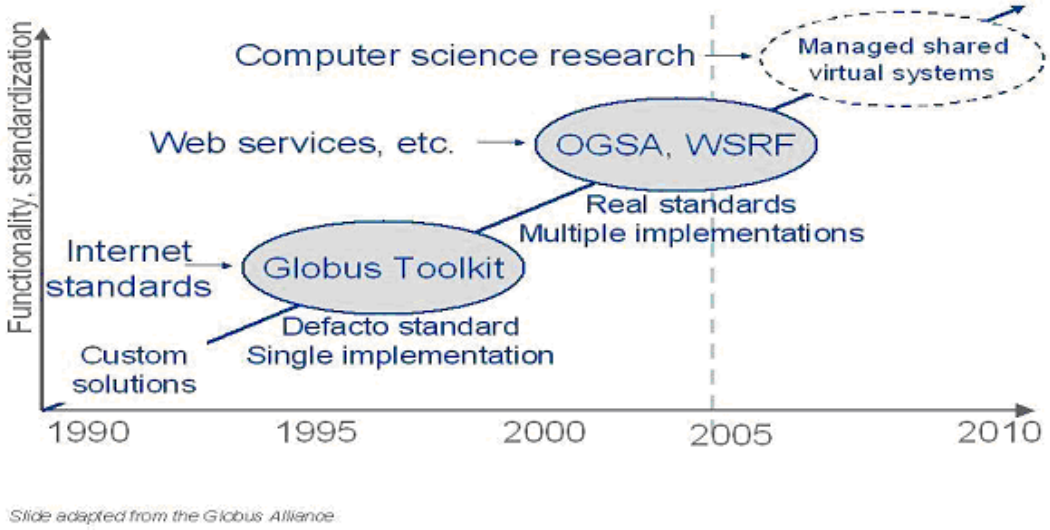

Figure 1.5: Emergence of Open Grid Standards [23]

The development of Grid has been enabled by the staged development of increasingly sophisticated and broadly used technologies. The Grid standards' evolution can be seen

from the Figure 1.5 [23]. Evolution began with Metacomputing, which kicked of in 1990. This transformed into Grid Computing with the help of introduction to middleware. Grid middleware is responsible for access to a Grid resource, for communication to the user, for scheduling, for error handling, security etc. By 1998 GT2 emerged as the de facto standard software for Grid Computing. Along with it many more user-oriented tools were also built. As interest in Grids continued, the importance of true standards increased. Thus, 2002 saw the emergence of OGSA. OGSA is still being enhanced to achieve the full Grid vision. And by 2010 expanding sets of interoperable services are perceived based on increasing degree of virtualization and sharing.
After discussing the Grid essentials, the next section would be focusing on the key challenges of this area.

1.3 Challenges in Grid Computing

Grid Computing has progressed considerably, yet, there are numerous challenges and issues, which need to be addressed before realizing the full power of the Grid. In general, a Grid Computing infrastructure must address several potentially complicated areas in many stages of the implementation [85]. Broadly these areas can be categorized as:

- Security
- Resource management
- Data management

1.3.1 Security Issues

The heterogeneous nature of resources and their differing security policies are complicated and complex in the security schemes of a Grid Computing environment [87]. These computing resources are hosted in differing security domains and heterogeneous platforms. The middleware solutions must address local security integration, secure identity mapping, secure access/authentication, secure federation, and trust management. There are issues of authentication, confidentiality and protection from risks posed by ill-formed

programs. Thus, it can be summarized that the major security issues arise due to the following reasons:

- Resources being used may be extremely valuable and the problems being solved extremely sensitive.
- Resources are often located in distinct administrative domains therefore each resource may have own policies and procedures.
- Set of resources used by a single computation may be large, dynamic and unpredictable instead of the traditional client/server arrangement.
- A standard, well-tested, well-understood set of protocols must be available for integration with wide variety of tools.

Along with this, there is also a need of developing a suitable trust model for use in global scale computer systems.

1.3.2 Resource Management Issues

The tremendously large number and the heterogeneous potential of Grid Computing resources causes the resource management challenge to be a significant effort topic in Grid Computing environments [3]. Grid Computing has evolved through an increased focus on resource sharing, co-ordination, manageability, and high performance. This sharing of resources, ranging from simple file transfers to complex and collaborative problem solving, is accomplished under controlled and well-defined conditions and policies. In this context, the critical problems are resource discovery, authentication, authorization, and access mechanisms.

Even with standardized interfaces, the fact that different organizations operate their resources under different policies is a significant impediment to being able to use Grid enabled resources as the goals of the resource user and the resource provider may be inconsistent, or even in conflict [89]. As a result, a critical aspect of Grid resource management is to define mechanisms, by which policy agreement between interested parties can be negotiated, established and enforced.

Resource sharing is further complicated when a Grid is introduced as a solution for utility computing, where commercial applications and resources become available as shareable and on-demand resources. This concept of commercial on-demand utility Grid services adds new, more difficult challenges to the already complicated Grid resource management problem, including service level features, accounting, usage metering, flexible pricing, federated security, scalability and open-ended integration [92].

Based on these facts, some of the aspects, which need to be taken care of in the resource management scenarios often include:

- resource discovery
- resource inventories
- fault isolation
- resource provisioning
- resource monitoring
- a variety of autonomic capabilities
- service-level management activities

1.3.3 Data Management Issues

Data forms the single most important asset in a Grid Computing system [88]. This data may be input into the resource, and the results from the resource on the execution of a specific task. If the infrastructure is not designed properly, the data movement in a geographically distributed system can quickly cause scalability problems. It is well understood that the data must be near to the computation where it is used. This data movement in any Grid Computing environment requires absolutely secure data transfers, both to and from the respective resources [81].

Some of the considerations developers and providers must factor into decisions are related to selecting the most appropriate data management mechanism for Grid Computing infrastructures. This includes:

- the size of the data repositories
- resource geographical distribution
- security requirements
- schemes for replication and caching facilities
- the underlying technologies utilized for storage and data access etc.

This section discussed the major concerns of Grid Computing, which need to be addressed; the next section discusses the Grid Resource Management, which has been chosen as the area of research for this thesis.

1.4 Grid Resource Management: The Research Motivation

As discussed earlier, a number of issues need to be dealt with in the area of Grid Computing. In this thesis, the focus would be on Grid Resource Management. Resource Management has been picked up due to the reasons discussed in the subsequent paragraphs.

The Grid Computing model, where resources tend to be both heterogeneous and distributed across multiple management domains, faces all the traditional IT management issues. It also brings new challenges - not only in the management of its component resources, but also of the Grid itself. Because of this reason, many issues come to the surface, which have been summarized as follows:

- In a Grid environment, shared resources must remain accessible, key infrastructure services must be available, and VO must be maintained, issues of interoperability within and outside VOs must be addressed. For e.g. resource might be stored and represented differently by different owners which impedes the resource discovery process and this needs to be dealt with. Thus a persistent problem in the IT industry is the lack of interoperability between disparate systems. Grid technologies need to tackle this head on by providing simple, consistent access to computing

resources, regardless of whether a processor is running a Windows Operating System, or Linux, or a proprietary mainframe OS. Interfaces need to be provided both for submission to the Grid and for extracting data from the Grid [94].

- The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like Grid where jobs need to be executed by computing resources, and the requirement of these resources is difficult to predict [92].

- The complex, heterogeneous and dynamic systems present new challenges in resource management such as: scalability, adaptability, reliability and fault tolerance.

- OGSA requirement of the future Grids is that it will configure, optimize, heal and protect itself with minimal human intervention. In short, it will be self-managing. Therefore, in a Grid it must also be possible to detect, report and deal with faults that may occur in any of the member domains [29].

- The resource owners and the users have different goals, objectives, strategies, and demand patterns [89]. Compared to traditional distributed systems, the resource allocation part of a Grid is confronted with the complication that, even if resources are granted to a job, these resources may be withdrawn or disappear before the job actually uses them. Moreover, it is necessary to cater to the users' needs optimally which may fluctuate from time to time and provide them provisioned services.

- Assigning users, resources, and organizations from different domains to a VO remains one of the key technical challenges in Grid Computing today [90].

- Due to its highly distributed nature and sheer size, the collection of resources forming a Grid is constantly changing. Therefore in order to maintain an up-to-date snapshot, the monitoring system must support mechanisms for failure detection and recovery. Monitoring and management of available resources on a Grid are

prerequisites for controlling a number of other basic services also such as accounting, surveillance and scheduling.

Thus, it can be construed that in such an environment, one of the key challenges is to develop a flexible, scalable, and self-adaptive resource management system which would allow users to carry out their jobs by transparently accessing autonomous, distributed, and heterogeneous resources [27]. Furthermore, Grid technologies are expected to bring improvements to provide systems that are more flexible, self-managing, or have lower management burden-attributes [29].

Therefore, the resource management issues specific to the Grid environment have been the motivation behind this work. The next section discusses the organization of this thesis.

1.5 Thesis Outline

After discussing the historical and technological aspects of Grid Computing along with its architectures and standards, the major challenges and thesis contributions in *Chapter 1*, the remainder of the thesis is structured as follows:

Chapter 2 discusses about the Grid Resource Management, existing Grid Resource Management Systems and their comparison with the management principles defined under OGSA. Attributes of resource management systems, resource brokers, metaschedulers have also been presented. Existing monitoring and fault management systems and comparison of fault management systems in different middleware has also been discussed. Along with this, it also discusses some of the related technologies. The chapter finally concludes with Problem Formulation.

Chapter 3 describes the proposed Resource Management Framework, PRATHAM and the goals which it tends to achieve. The requirements analyzed on the basis of UML have been presented with the help of different diagrams. The framework architecture and its components have also been illustrated in detail.

Chapter 4 explains the design and implementation of the proposed framework. Design and implementation of PRATHAM taxonomy, workflow, interoperability through semantics and resource provisioning aspects has been detailed. Along with this, Autonomic system, fault detection and healing system has also been described. Further, the Chapter explains the design and implementation of Grid Portal also.

Chapter 5 illustrates the deployment of the framework, the experimental results have been illustrated with the help of a case study. Further, the framework has been validated and compared with existing systems.

Chapter 6 finally concludes the thesis and discusses the future enhancements.

1.6 Thesis Contributions

The Thesis contributes in the following ways:

- Thesis presents detailed literature review of the work done in the area of Grid Resource Management and highlights its key challenges, issues and latest trends of development in this area.
- To address the challenges of Grid Resource Management, a framework named PRATHAM has been proposed, designed, developed and tested in this thesis as per the set objectives of the research work.
- Thesis addresses the issue of interoperability amongst different middleware for an efficient resource discovery incase of multiple middleware scenario.
- Thesis articulates resource provisioning feature (through PRATHAM) for optimization of user demands along with introducing advanced resource reservation.
- Thesis presents an efficient resource monitoring system for enhancing job satisfaction, better fault isolation and fault tolerance in Grid Environments.
- Thesis articulates the techniques of self-healing, self-protection and self-optimization,

the self-management issues of Autonomic Computing, in the context of Grid environments.

- Thesis presents a fault detection and healing system, that helps in achieving self-management (a requirement of OGSA based future Grids) in Grid environments.

- Thesis demonstrates Grid portal which facilitates an easy access to Grid resources.

Chapter 2

Literature Review

Grid Computing has emerged as an important field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and, high-performance orientation [17]. The motivation behind Grid Computing is coordinated resource sharing and problem solving in dynamic, multi-institutional Virtual Organizations.

At the heart of the Grid is the ability to discover, allocate and negotiate the use of resources. The process which describes all the aspects related to locating a resource, arranging for its use, utilizing it and monitoring its state is termed as Grid Resource Management. Grid resources usually span distinct administrative domains. Grid Resource Management is concerned not only with the core functionality of a resource or a service but also with the manner in which this function is performed.

This chapter discusses the basis of Grid Resource Management System (GRMS). It examines the existing GRMS , their attributes, Grid Brokers, Metaschedulers and existing GRMS Frameworks. Comparative study and analysis of existing Grid Middleware has been done for resource management techniques and fault management systems. Contemporary Grid monitoring tools have been discussed and compared. The related technologies and their impact on Grid Computing has also been discussed. The last section highlights the gaps in research and development work as identified during literature review. These gaps form the basis of problem formulation for this work.

2.1 Grid Computing

Grid Computing has herald the dawn of pervasive computing, sometimes also called as Ubiquitous Computing [105], [143]. Grid computing principles focus on large-scale resource sharing in distributed systems in a flexible, secure and coordinated fashion as described by J. Nabrzyski et al. in their book [23]. The real basis of Grid computing is coordinated resource sharing and problem solving in a dynamic, multi-organizational environment. This sharing is highly restricted as resource providers and consumers clearly define the sharing rules and conditions. In words of Ian Foster et al., a set of individuals and/or institutions defined by such sharing rules form a Virtual Organization (VO) [17]. Some examples of VOs are: the application service providers, storage service providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory, a crisis management team etc.

The concept of the VO is the key to Grid computing. All VOs share some characteristics and issues, including common concerns and requirements that may vary in size, scope, duration, sociology, and structure. The members of any VO negotiate the sharing of resources based upon the rules and conditions defined by the VO, and the members then share the resources in the VO's constructed resource pool [90]. Assigning users, resources, and organizations from different domains to a VO remains one of the key technical challenges in Grid Computing, even today. This task includes the determination of a definition of resource discovery mechanisms, such as identification and application of appropriate resource-sharing methods, specification and application of rules and conditions for member assignment, security federation or delegation, and access control among the participants.

A Grid environment is created to address resource needs; the use of that resource(s) (ie. CPU cycles, disk storage, data, software programs, peripherals, etc.) is usually characterized by its availability outside of the context of the local administrative domain. Grid computing works on the ideology of borrowing the idle computational power of resources instead of buying new resources. In a Grid environment, the users need to plug into a Grid in order to access additional computing on demand [84]. The user submits his/her job to the Grid resources through an interface that serves as a portal to the Grid.

Special Grid-management software (known as Grid Middleware) accepts the job, locates best suitable and available resources and submits the job to them. Finally, it collects the results and gives it back to the user. The Grid infrastructure forms the core foundation for successful Grid applications and has been depicted in Figure 2.1 [7]. Figure shows that Grid Computing is a complex combination of a number of capabilities and resources identified for the specific problems and environments being addressed.

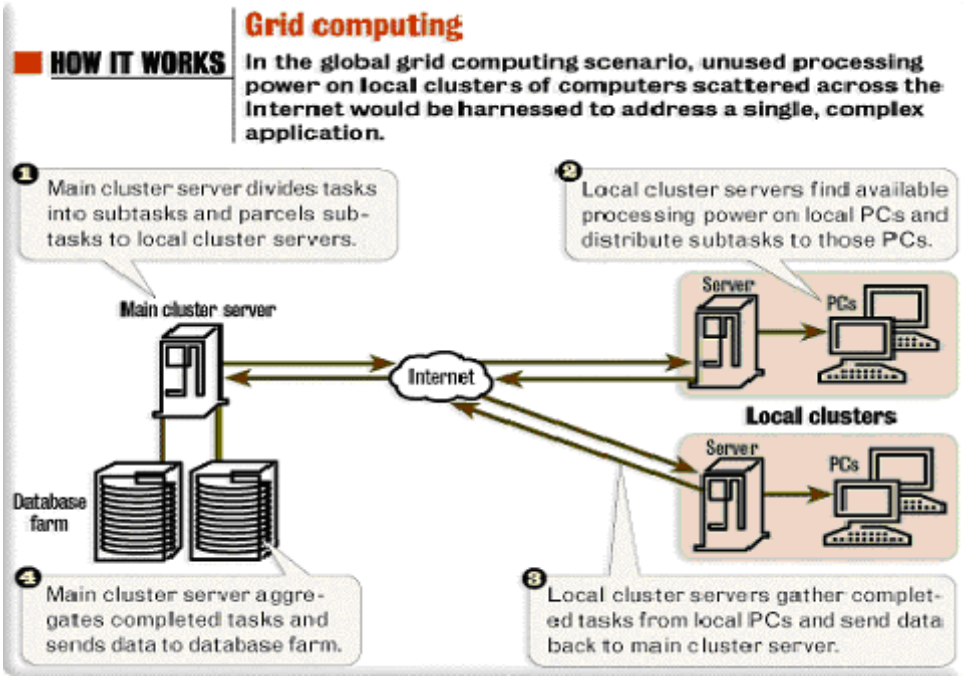

Figure 2.1: Scope of Grid [7]

The real basis of Grid is the Resource Management System, which acts like a pivot and keeps the show running.

2.1.1 Resource Management: Core of Grid Computing

Klaus Krauter et al. define Resource Management System (RMS) as central to the operation of a Grid [24]. Resource management is commonly used to describe all aspects of locating various types of capability, arranging for their use, utilizing them, and monitoring their state [23]. Resources are the entities such as processors and storage that are managed by the RMS. The basic function of a RMS is to accept requests for resources from machines within the Grid and assign specific machine resources to a request from the overall pool of Grid resources for which the user has access permission. As per CoreGrid Technical Report [25], in a general scenario, the resources belong to different

and independent resource providers, who claim administrative autonomy and usually have different usage policies. Typically, these policies are enforced by the local resource management systems. The Grid RMS has to deal with many heterogeneous resources in a highly dynamic environment while it has no exclusive control over any resource. In contrast, the local RMS typically manages only one or a few resource types in a static configuration. These resources reside within a single administrative domain in which the RMS has exclusive control over the resources.

The Grid Scheduling Architecture Research Group (GSA-RG) of the Global Grid Forum has identified following functionalities for a Grid Resource Management System [26]:

Resource Discovery: The primary step in Grid RMS is to first discover the suitable resource or set of resources.

Access to Resource Information: Static and dynamic information regarding the discovered resource needs to be accessed.

Status Monitoring: The RMS should support event notification to facilitate reactive measures while continuously monitoring the resources as well as the status of the submitted jobs.

Brokering/Scheduling: Automatic selection and scheduling of resources for job execution is required through efficient Grid functions.

SLA/Reservation Management: Some applications may require additional information, which might include reservations and precise agreements about resource-specific Quality of Service levels.

Execution Management/Provisioning: The main task of a Grid RMS system is to perform the actual job execution and job management or the required provisioning of a resource.

Accounting and Billing: This includes information about the resource consumptions as well as financial management of budgets, payments and refunding i.e. accountability for the resource consumption is required to be maintained.

2.1.2 Resource Management in OGSA based Grids

Resource management is a generic term for several forms of management as they are applied to resources. It includes various management tasks and a resource manager

is a manager that implements one or more resource management tasks. The main requirements for management in OGSA have been identified and described by Frederico Buchholz Maciel in GFD-I.045, March 2005 [29]. Here the author has taken no centralized notion of control, such as a Grid. These requirements are:

Scalability: Grid may expand dynamically. Therefore, management architecture needs to scale to potentially thousands of resources. To achieve this scalability, management needs to be done in a hierarchical and/or peer-to-peer (federated/collaborative) fashion, so OGSA should allow these forms of management.

Reliability: Resource management style must discourage a single point of failure. Resource managers must be allowed to manage multiple manageable resources, and a manageable resource must be allowed to be managed by multiple managers. This ensures better reliability.

Security: There are two security aspects in management:

- Firstly, the security infrastructure itself must be manageable; this includes the management of all types of authentication, authorization, access control, VOs and access policies.
- Secondly, management should be able to ensure its own integrity. It should pursue access control policies of the owners of resources and VOs.

Interoperability: Management architecture must be able to span the Virtual Organizations, which would be encompassing software, hardware and service boundaries. Interoperability is required:

- Between levels of a hierarchy
- At the same level of hierarchy

Policy: Enforcement of policy for authentication, transport protocol selection, QoS parameters and metrics etc. should be supported by the management architecture and it should be able to enforce such policy assertions.

Performance Monitoring: The management architecture must support the performance monitoring facilities as outlined in the Grid monitoring architecture:

- Low latency to keep performance data relevant
- Handle high data rates
- Minimal measurement overhead

Peer-to-Peer Management Requirements: Some Grid systems encompass peer- to-peer systems. In such a case following general and manageability requirements should be fulfilled [28]:

- Discovery
- Security
- Location awareness
- Group support

The next section discusses about the resource management practices followed in existing Grid middleware.

2.2 Resource Management in Grid Middleware

Grid can be seen as an infrastructure of computing resources. On top of this infrastructure there is a need for middleware that copes with the differences between the computing resources and manages requests for computational power. Grid middleware is responsible for access to a Grid resource, for communication to the user, for scheduling, for error handling, and for security. Thus, middleware is the most important part of a Grid, as it manages all communication and hides the differences between all connected entities [91]. A. S. Grimshaw et al. have discussed Globus and Legion in detail and compared the two with respect to different distributed systems [184]. It is evident that different parties from the Grid community have developed Grid middleware solutions for their own needs. These middleware solutions are incompatible and each uses different protocols, components and interfaces. This results in slightly dissimilar functionality provided to the end-users.

In this section, an overview of Resource management techniques used in existing Grid

middleware has been discussed. The taxonomy followed is based on the architecture of Resource management models of Globus, UNICORE, Legion, Alchemi, SUN Grid Engine and Condor. Based on this taxonomy an analysis of these middleware has also been done.

2.2.1 Globus

Globus is a software toolkit addressing key technical problems in the development of Grid enabled tools, services, and applications. The Globus resource management architecture is a layered system in which high-level global resource management services are layered on top of local resource allocation services [50].

RM in Globus

In Globus the RM package enables resource allocation through job submission, staging of executable files, job monitoring and result gathering [51],[52]. Figure 2.2 shows an overview of the various components in the Globus Resource Management architecture. The main components of Globus Resource Management model are: Resource Specification Language, Resource Management tool interface, and co-allocator.

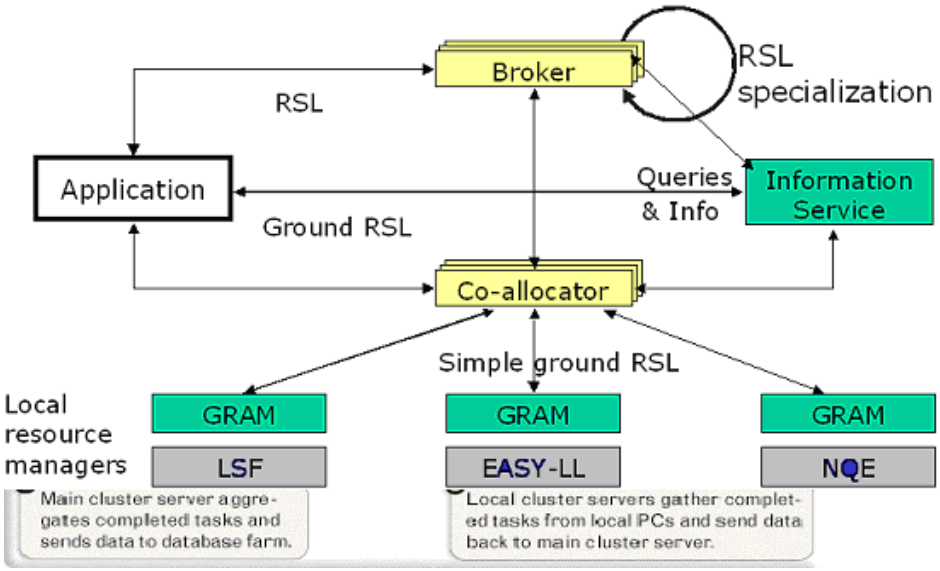

Figure 2.2: Resource Management in Globus (Source- The Globus Project)

Resource Specification Language:

Resource Specification Language (RSL) provides a common interchange language to describe resources in Globus. The RSL provides the skeletal syntax used to compose complicated resource descriptions, and the various resource management components introduce

specific attribute, value pairings into this common structure [53].

GRAM:

GRAM is the module that provides the remote execution and status management of the execution [54]. When a client submits a job, the request is sent to the remote host and is handled by the gatekeeper daemon located in the remote host. Then the gatekeeper creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates.

Co-allocator:

By using the Dynamically-Updated Request Online Co-allocator (DUROC) mechanism, users are able to submit jobs to different job managers at different hosts or to different job managers at the same host. The RSL script that contains the DUROC syntax is parsed at the GRAM client and allocated to different job managers [55].

2.2.2 UNICORE

UNIform Interface to COmputer REsources (UNICORE) is a project funded by the German Ministry of Education and Research [56]. UNICORE lets the user prepare or modify structured jobs through a graphical user interface on a local Unix workstation or a Windows PC [57].

RM in UNICORE

Jobs can be submitted to any of the platforms of a UNICORE Grid and the user can monitor and control the submitted jobs through the job monitor part of the client. The main UNICORE components used for RM are: the Job Preparation Agent (JPA), the Job Monitor Controller (JMC), the UNICORE https server, also called the Gateway, the Network Job Supervisor (NJS), and a Java applet-based GUI with an online help and assistance facility.

Jobs are constructed using JPA and the status and results of the jobs can be obtained through the JMC. The jobs or status requests and the results are formulated in an abstract form using the Abstract Job Object (AJO) Java classes [58]. The client connects to a UNICORE Usite gateway and submits the jobs through AJOs. The UNICORE Gateway is the single entry point for all UNICORE connections into a Usite. It provides

an Internet address and a port that users can use to connect to the gateway using SSL. A UNICORE Vsite is made up of two components: NJS (Network Job Supervisor) and TSI (Target System Interface). The NJS Server manages all submitted UNICORE jobs and performs user authorization by looking for a mapping of the user certificate to a valid login in the UUDB (UNICORE User Data Base). NJS also deals with the incarnation of jobs from the AJO definition into the appropriate command sequences for a given target execution system, based on specifications in the Incarnation Data Base (IDB). UNICORE TSI accepts incarnated job components from the NJS, and passes them to the local batch systems for execution [56].

2.2.3 Legion

Legion [59] is an object-based 'metasystem', developed at the University of Virginia. Legion provides the software infrastructure, so that a system of heterogeneous, geographically distributed, high-performance machines can interact seamlessly. Legion attempts to provide users, at their workstations, with a single integrated infrastructure, regardless of scale, physical location, language and underlying operating system. Legion encapsulates all its components as objects [60].

RM in Legion

Legion is structured as the system of distributed "objects" active processes that communicate using remote method invocation service [60]. Legion objects represent all hardware and software resources in Grid system. Legion defines a set of core object types that support basic system services. The components of the Legion resource management framework are: class objects, resource objects (hosts and vaults), information database objects (collections), scheduler objects, schedule implementation objects (enactors) and implementation objects.

Class Objects:

Class objects manage particular instances. Class objects are managers and policy makers and have system like responsibility for creating new instances, activating and deactivating them, and providing bindings for clients. Legion encourages users to define and build their own class objects. These two features - class object management of the class' in-

stances and applications programmers' ability to construct new classes - provide flexibility in determining how an application behaves and further support the Legion philosophy of enabling flexibility in the kind and level of functionality.

Resource Objects (Host and Vault):

Legion host objects abstract processing resources in Legion. A host object is a machine's representative to Legion: it is responsible for executing objects on the machine, protecting the machine from Legion users, protecting user objects from each other reaping objects and reporting object exceptions. A host object is also ultimately responsible for deciding which objects can run on the machine it represents. Vault objects are responsible for managing other Legion objects' persistent representations (OPRs).

Information Database Objects (Collections):

Collections act as the repository of the data, which contains all the information that form the system. Each object is stored as set of Legion attributes and also provides security for the access and updating of the system. Another important use of the collections is to structure the resources. Collections may receive or send data to other collections.

Scheduler Objects:

Scheduler objects map the request to the resource. The Scheduler also knows the number of instances of the particular class, which is running. It obtains resource description information by querying the Collection, and then computes a mapping of object instances to resources. This mapping is passed on to the Enactor for implementation. Each schedule has at least one master version and a list of variant versions. The master version of a schedule contains the scheduler's best attempt to schedule the requested objects. A variant version is similar to the master version, representing a poorer scheduling decision to which the enactor can resort if the master fails.

Schedule Implementation Objects (Enactor):

Enactor bears the responsibility of ensuring that the schedule is successful. It takes the schedule from the scheduler, and attempts to determine whether the schedule will be successful. In order to do so, it extracts the mappings from the master version, contacts each host/vault pair involved and inquires whether the sub-request on it will be successful. A host/vault pair may choose to reject this sub-request based on its current situation. Such a rejection is part and parcel of the negotiation philosophy. If the master version

cannot be satisfied because of such rejections, the enactor resorts to the variant versions to schedule successfully. If no version can be made successful, the enactor reports an error and cancels the rest of the scheduling process.

Implementation Objects :

Implementation objects may be viewed as representations of the actual binaries required to run objects on a processor. When a class receives a viable schedule from an enactor, it communicates with the host/vault objects in order to start objects. Host/vault objects in turn receive the names of the implementation objects for that class, and contact the implementation objects to download the associated binary for running the instance [60].

2.2.4 Alchemi

Alchemi is a .NET-based framework that provides the runtime machinery and programming environment required to construct desktop Grids and develop Grid applications [61]. It allows flexible application composition by supporting an object-oriented application-programming model in addition to a file-based job model. Cross-platform support is provided via web services interface and a flexible execution model supports dedicated and non-dedicated (voluntary) execution by Grid nodes.

RM in Alchemi

Alchemi has layered architecture for a desktop Grid-computing environment [62] and follows the master-worker parallel computing paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. In Alchemi, this unit of parallel execution is termed as 'Grid thread' and contains the instructions to be executed on a Grid node, while the central component is termed as 'Manager'. The four major distributed components used for managing the resources are: Manager, Executor, User, and Cross-platform Manager. Figure 2.3 shows the architecture of Alchemi.

Manager:

The Manager provides services associated with managing execution of Grid applications and their constituent threads. It receives the thread from various users and allots them to executors. All this scheduling is done on the basis of the thread priority using First

Come First Served (FCFS) basis. The Executors return completed threads to the Manager, which are subsequently collected by the respective users. It is also the responsibility of Manager to employ a role-based security model for authentication and authorization.

Executor:

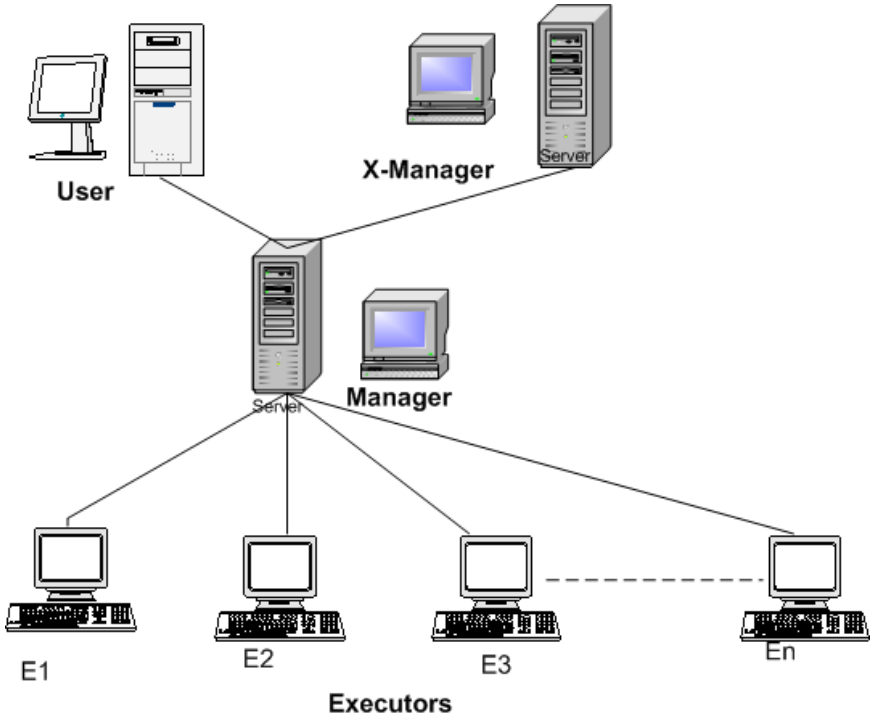

Figure 2.3: Components and their relationship in Alchemi.NET

The Executor accepts threads from the Manager and executes them. An Executor (shown as 'E' in Figure 2.3) can be configured to be 'dedicated', which implies, the resource is centrally managed by the Manager, or as 'non-dedicated', which implies that the user manages the resource on a volunteer basis via a screen saver or explicitly.

User:

Grid applications are executed on the User node. The API abstracts the implementation of the Grid from the user and is responsible for performing a variety of services on the user's behalf.

Cross-Platform Manager:

Jobs submitted to the Cross-Platform Manager (shown as X-Manager) are translated into a form that is accepted by the Manager (i.e. Grid threads), which are then scheduled and executed as described above. It helps to run Alchemi on any platform that supports web applications [62].

2.2.5 SUN N1 Grid Engine

The Sun N1 Grid Engine (SGE) system is an advanced resource management tool for heterogeneous, distributed computing environments [63]. Sun N1 Grid Engine software provides advance resource management and policy administration for UNIX environments that are composed of multiple shared resources. It can integrate into the Grid - permanently or temporarily (at night, weekends, vacations) or dedicated to specific usage.

RM in SUN N1 Grid Engine

Hosts (machines or nodes) in SGE are classified into four categories - master, submission, execution, administration and shadow. Figure 2.4 shows the SGE architecture [64],[65],[66]:

Master host (M-H):

A single host is selected to be the SGE master host. This host handles all requests from users, makes job-scheduling decisions and dispatches jobs to execution hosts.

Submit host (S-H):

Submit hosts are machines configured to submit, monitor and administer jobs, and to manage the entire cluster.

Execution host (E-H):

Execution hosts have the permission to run SGE jobs.

Administration host (A-H):

SGE administrators use administration hosts to make changes to the cluster's configuration, such as changing distributed resource management parameters, configuring new nodes or adding or changing users.

Shadow Master host (S-M-H):

While there is only one master host, other machines in the cluster can be designated as shadow master hosts to provide greater availability. A shadow master host continually monitors the master host, and automatically and transparently assumes control in the event when the master host fails. Jobs already in the cluster are not affected by a master host failure.

Job management in SGE:

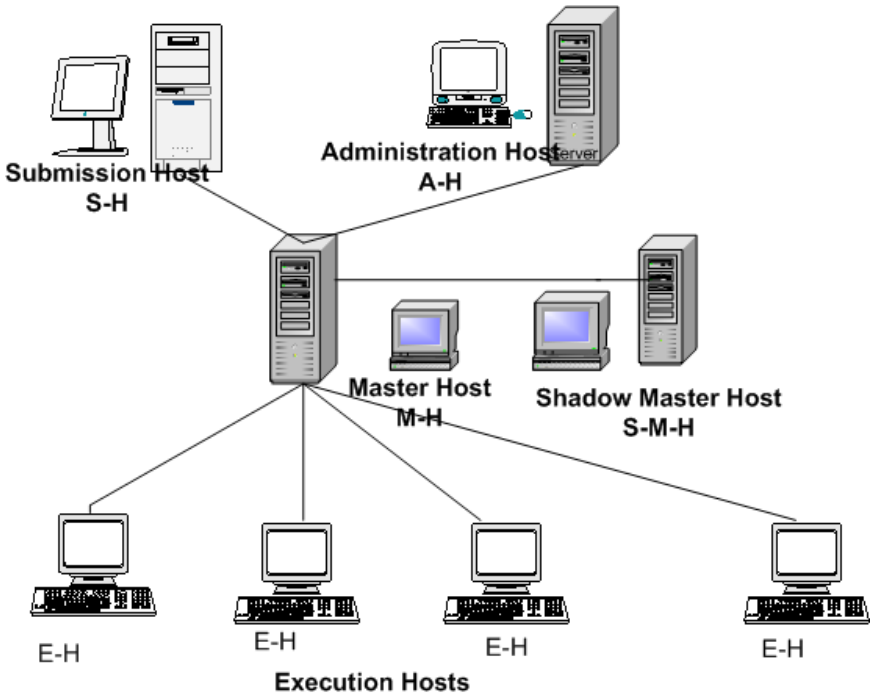

Figure 2.4: Resource Management Architecture of SUN Grid Engine

SGE supports four job types - batch, interactive, parallel and array [65]. The first three have obvious meanings, the fourth type - array job - is where a single job can be replicated a specified number of times, each differing only by its input data set, which is useful for parameter studies. Submitted jobs are put into job queues. An SGE queue is a container for a class of jobs allowed to execute on a particular host concurrently. A queue determines certain job attributes; for example, whether it may be migrated etc. Throughout their lifetimes, running jobs are associated with their queues. Association with a queue affects some of the actions that can happen to a job. For example, if a queue is suspended, all the jobs associated with that queue will also be suspended.

In SGE, there is no need to submit jobs directly to a queue [66]. A user only needs to specify the requirement profile of the job (such as memory, operating system and available software) and SGE will dispatch the job to a suitable queue on a lightly loaded host automatically. If a job is submitted to a particular queue, the job will be bound to this queue and to its host, and thus SGE daemons will be unable to select a lightly loaded or better-suited resource.

2.2.6 Condor

Condor is a batch queuing system to manage compute-intensive jobs [67]. It does this by providing a High Throughput Computing (HTC) environment [68, [69], [70]. It provides traditional queuing and scheduling functionality, along with innovative technology, such as resource classifications. In a typical usage scenario, a user submits a job to Condor, which queues and monitors the job, then presents the results when the job completes.

RM in Condor

Batch systems providing this functionality historically use dedicated machines owned by a single organization. Condor takes advantage of spare cycles when those resources are idle. This is sometimes referred to as cycle scavenging.

ClassAds allow machines to advertise resources available for others to use and allows jobs to advertise for the resources they want to use. Machine ClassAds advertise the machine attributes and list the conditions under which it is willing to accept jobs. A job ClassAd is a job-specific advertisement that would include the type of machine it needs to run the job and the minimum or maximum attribute values acceptable. Certain types of jobs are suitable for Condor. A Condor job must be able to run as an unattended background task. The job should use algorithms that can generate a large number of independent tasks that are self-contained, making them portable (able to run anywhere). Of course, the job should also be able to detect failure(s) and react in a graceful manner. Condor refers to the environment as the universe. The most common type of environment is the standard or the vanilla universe. The universe type and other details of job submission are specified in a submit description file.

When a job begins, Condor starts a process, known as the condor shadow, on the machine where the job was submitted. This shadow process responds to requests from the remote executing job to access the environment where it was submitted. Condor has the concept of a pool of resources. A pool is a collection of machines and jobs. A Condor Pool has a single central manager and a number of other machines. The central manager collects information about resources and jobs, and then negotiates matches between resources and job resource requests.

The Condor master process runs on every machine in the pool. It reads the configuration

file and determines which condor processes should be running on the machine. The Condor startd runs on machines configured to execute jobs. The condor schedd runs on machines configured to submit jobs, also called submit nodes. Users submit jobs to the scheduler or schedd process on the submit node. The Condor collector collects information about the status of resources in the pool. The Condor negotiator performs matchmaking. It queries the condor collector for the state of resources in the pool and queries each condor schedd that has waiting resource requests in its job queue. The negotiator tries to match available resources with those requests, while maintaining the user priorities in the system.

2.2.7 Comparative Analysis of Resource Management Techniques

Table 2.1 describes the comparison between the above discussed middleware w.r.t the management principles defined under OGSA in section 2.1.2.

A standard OGSA based Grid would achieve dynamic scalability in global size. As multiple VOs might be involved, reliability would be dependent on local managers; therefore the global middlware must ensure reliability of these managers and must take measures to incorporate this feature. Along with this it must also ensure a secure access to all the resources and secure execution of confidential data. Homogenous environment is rarely possible in Grid therefore interoperability is required to handle heterogeneity of the resources and resource management policies.

The comparison of above described middleware depicts that till date no single system exists which is able to attain the global status by satisfying all these management principles without sacrificing any of these characteristics.

Next section discusses the State of the Art in Grid Resource Management Systems.

Table 2.1: Comparative Analysis of Resource Management in Grid Middleware

Middleware / Property	*UNICORE*	*Globus*	*Legion*	*Alchemi*	*SUN N1 Grid*	*Condor*
Scalability	Limited scalability through centralized broker, brokers hierarchy	Closer to hardware hence scalability is direct result of internet	Uses LOAs to form computational hierarchy	Distributed query based service discovery; Hierarchical Architecture	N1 Grid Engine can double the productivity and it is designed to scale up to 10,000 hosts per one master.	Limited scalability as dedicated machines within an organization need to be used.
Reliability	UNICORE client will check for error and ask for correction again and again	GRAM, MDS combine with LDAP provide a reliable service	Allows users to build their own class objects	Flexible application composition by supporting OO application programming and file based Job model	SUN N1 Grid engine uses checkpointing technique	Reliability is mainly job dependent. Job should be able to detect failure(s) and react in a graceful manner
Security	UNICORE pro server for authorization X.509, SSL	GSI security layer uses SSL, PKI, X.509 No authorization model	Defined authentication,data integrity and authorization models	Security is due to powerful toolset of .NET framework	The Sun Grid Utility is designed with Solaris and network security isolation for user jobs, and with role-based access principles. Incorporates the principles of minimal privilege for users and administrators	Condor provides support for authentication, encryption, integrity assurance, as well as authorization through the use of configuration macros
Interoperability	GRIP is used for communicating with Globus	Uses Nexus communication library	Uses Interface Description Language (IDL) i.e. compatible with multiple programming languages, variation of RMI for communicating through LOIDs	Alchemi(.net) Grid Thread Model cross-platform manager interoperate with any platform which uses web services	Interoperability with Sun Control Station 2.2.1 Interoperability on all majorUnix platforms: Solaris, Linux,Mac OS X, HP-UX, AIX, Irix	Interoperability is also Job dependent.The job should use algorithms that can generate a large number of independent tasks that are self-contained, making them portable.

2.3 Grid Resource Management: State of the Art

The Grid Resource Management System (GRMS) works in combination with other Grid components and manages the allocation, scheduling, and reclamation of resources from jobs. This section discusses the existing GRMS. A Grid computing environment enables sharing of loosely coupled resources and services required by various applications in a large-scale. Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran have proposed an abstract model of Resource Management Systems [RMS] that provides a basis for a comparison between different RMS architectures [24]. The model presented in Fig-

ure 2.5 [24], shows the functional units and data stores that a general purpose RMS for a Grid would have.

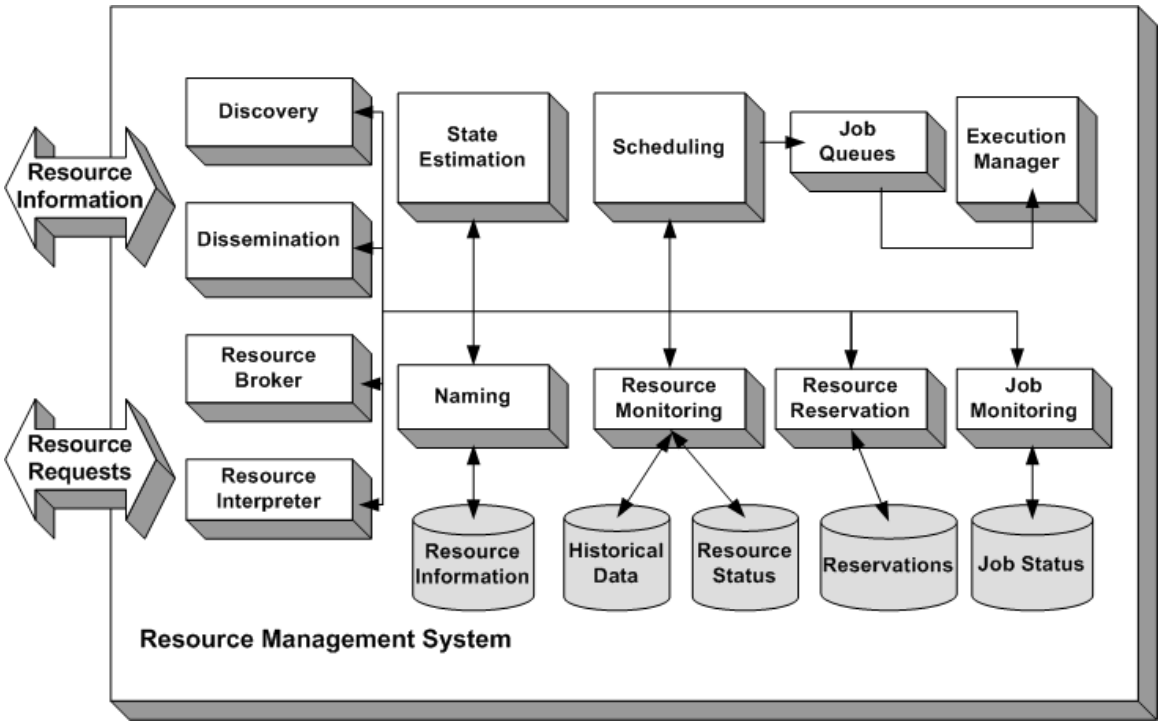

Figure 2.5: Resource Management System (Abstract View)[24]

As depicted in the model, different users of the VO may place Resource Requests and may desire Resource Information for application execution on the Grid. Firstly, the resource needs to be discovered. For this, the request may be passed on to the manager of the VO, which may be a broker and may need request interpretation and dissemination. The actual state of the resource is estimated and checked from time to time and stored in the Resource Information repository. After the resources are discovered, the job is submitted and scheduling is done. During job execution the resources are monitored. In case a new request comes in and the resource is currently busy with another job, then the new request is placed in a Job Queue and the resources are reserved in advance for their execution. All the jobs are also monitored and their status is also stored in Job status repository.

The existing resource management systems implement some of these functions but do so in very different ways as has been discussed in the next sections. Klaus Krauter et al. [24] have developed taxonomy for GRMS by which resource management systems can be classified by characterizing different attributes. There are many existing Grid computing

projects similar to Globus, Legion, NetSolve, AppLeS, and Condor, Rajkumar Buyya provides a brief description of each of these system in his thesis [16] and then classifies the resource management system attributes according to the taxonomy of [24]. The Grid resource management architecture of [82] described by Rajkumar Buyya et al. discusses three different models:

- Hierarchical Model
- Abstract Owner Model
- Computational Market/Economy Model

The hierarchical model captures the approach followed in many contemporary Grid systems. The abstract owner shows the potential of an order and delivery approach in job submission and result gathering. The (computational) market model captures the essentials of both hierarchical and abstract owner models and uses the concept of computational economy. On the basis of above discussion, following sections discuss the general attributes of GRMS.

2.3.1 Attributes of Grid Resource Management Systems

The generic resource management taxonomy and the main components of resource management, required in a Grid, discussed literature-wide has been described in this section.
(a) Grid type
Grid systems are classified as Compute, Data and Application Grids [30], [24]. In compute Grids the main resource that is being managed by the RMS is compute cycles (i.e. processors), while in data Grids the focus is to manage data distributed over geographical locations and application Grids provide different services to end users. The type of Grid system it is deployed in affects the architecture and the services provided by the resource management system. Resources could be hardware (computation cycle, network bandwidth and data stores) or software resources (applications and databases).
(b) Resource Model
The resource model determines how applications and the RMS describe and manage Grid resources [24]. A resource model is an abstract representation of manageable entities,

which defines their schema (conceptual hierarchy and inter-relationships) and characteristics (attributes, management operations, etc.). Figure 2.6 shows the resource model taxonomy. The taxonomy focuses on how the resources and operations on the resources are described. In a Schema based approach the data that comprises a resource is de-

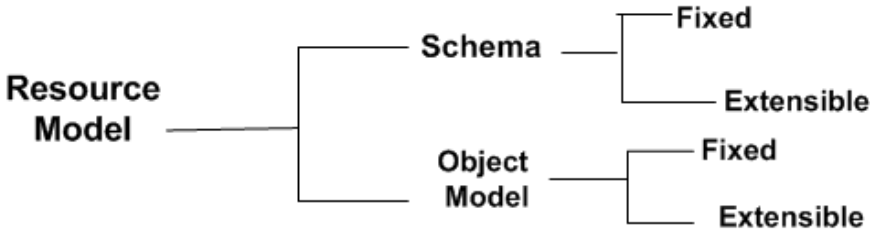

Figure 2.6: Resource Model Taxonomy [24]

scribed in a description language along with some integrity constraints (e.g., Condor ClassAd). In the object model extensible approach the resource model provides a mechanism to extend the definition of the object model managed by the RMS (e.g., Legion object model).

(c) Scheduling model

Scheduler structure determines the structure of the RMS and scalability of the system. Most contemporary GRMS can be classified as having centralized, hierarchical or decentralized structure as described by Klaus Krauter et al. in [24].

- Centralized Organization: In a centralized organization all jobs are submitted to a single scheduler, which is responsible for scheduling them on the available resources. Since all the scheduling information is available at one single position the scheduling decision are optimal but this approach is not very scalable in a Grid system.

- De-Centralized Organization: In a decentralized model there is no central scheduler, scheduling is done by the resource requestors and owners independently. This approach is scalable and suits Grid systems. But individual schedulers should cooperate with each other in making scheduling decisions and the schedule generated may not be the optimal schedule. Based on whether or not schedulers cooperate they can be further classified as cooperative or non-cooperative schedulers.

- Hierarchical Organization: In a hierarchical model the schedulers are organized in a hierarchy. High-level resource entities are scheduled at higher levels and lower

level smaller sub-entities are scheduled at lower levels of the scheduler hierarchy. This model is a combination of above two models.

D.P. Spooner et al. [122] discusses the problem of Grid workload management through the development of a multi-tiered scheduling architecture (TITAN) that employs a performance prediction system (PACE) and task distribution brokers to meet user-defined deadlines and improve resource usage efficiency. By coupling application performance data with scheduling heuristics, the architecture is able to balance the processes of minimizing run-to-completion time and processor idle time, whilst adhering to service deadlines on a per-task basis. It focuses on fully implementing operability with Globus to enable performance- based 'global' scheduling in addition to 'local' scheduling.

Savina Bansal et al have done a categorization of heterogeneity models and performance metrics to have better understanding for development and comparison of such algorithms in [110]. A generic scheduling strategy has been presented, which improves the performance of list-based heuristics with the addition of limited duplication. A comparison of state-of-the-art scheduling algorithms is next performed using an A-Cube performance model that has been suggested to study the behavior of an algorithm, application and architecture in the presence of heterogeneity, which is the basis of Grid Environment. Aram Galstyan et al. discusses about reinforcement learning [113], which can be used to improve the quality of resource allocation in large-scale heterogeneous systems. Similarly, scheduling in Real-Time applications especially distributed processing needs to be handled and has been discussed by Harpreet Singh and J.S.Bedi [144].

(d) Resource Discovery

Resources are the base components of a Grid environment. They form the low level entities that are accessed and used to fulfill a user request. Different resources can have the same functional capabilities but they may have different access policies associated, different time access, etc. The main motivation for Grid computing is the sharing of these resources and the first thing needed for efficient sharing is a well-defined resource discovery process.

The Grid resource discovery [31], [32] process can be defined as the process of matching a query for resources, described in terms of required characteristics, to a set of resources

that meet the expressed requirements. It takes as input a user request and returns a list of resources or services that can possibly fulfill the given request.

To make information available to users quickly and reliably, an effective and efficient resource discovery mechanism is crucial. However, Grid resources are potentially very large in number and variety; individual resources are not centrally controlled, and they can enter and leave the Grid systems at any time. For these reasons, resource discovery in large-scale Grids can be very challenging [126]. The steps of Resource Discovery Process have been discussed in the book, 'Grid Resource Management: State of the Art and Future Trends' [23] and have been described below:

- authorization filtering
- job requirement knowledge, and
- filtering to meet the minimal job requirements

Further the approaches to resource discovery can be broadly classified as Query Based and Agent Based.

Agent Based Resource Discovery

The (Agile Architecture and Autonomous Agents) A4 methodology [33], [120] can be used to build large-scale distributed software systems that exhibit highly dynamic behavior. An agent can represent one or more local high performance resources at the meta-level of the system. It is intended that an entire system be built of a hierarchy of identical agents with the same functionality. Agents are considered both as service providers and as service requestors. A4 system is a distributed software system based on the idea of federating agents, which can provide services to a large-scale, dynamic, multi-agent system. The A4 model comprises of a hierarchical model, a discovery model and a coordination model [33].

A large number of agents can be organized by using hierarchical model, which is the simplest model. The discovery model is the origin of the hierarchical model and solves the problem of coordinating the services of different agents, so that they locate each other. The coordination model focuses on the way that services can be organized to provide more complex services. The hierarchical model is illustrated in Figure 2.7. The single

type of component that composes the system is the agent. Every agent can act as a router between the request and the service. The broker is the agent that heads the whole hierarchy maintaining all the service information of the system. The coordinator is the agent that heads a sub-hierarchy. Only the leaf-node of the hierarchy is named as an agent. The implementation of system functions is abstracted to the processes of service

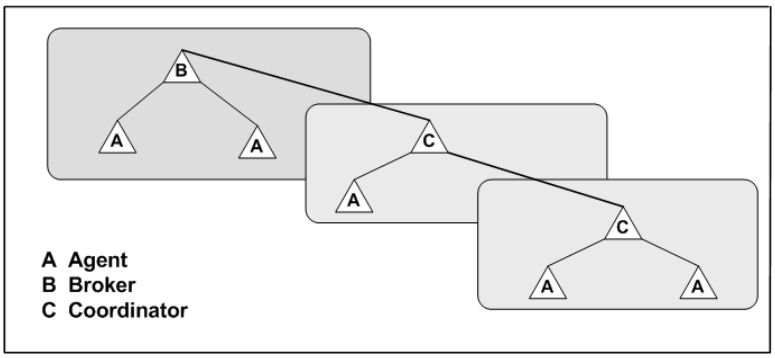

Figure 2.7: Hierarchical Model

advertisement and service discovery.

Query-based Resource Discovery

Most of the contemporary Grid systems use parameterized queries [35] that are sent across the network to the nearest directory, which uses its query engine to execute the query against the database contents. The database used in this context maybe centralized or distributed database. There are a number of query-based approaches as described below.

Directory services:

A directory service is a service that provides read-optimized access to general data about entities via a network protocol [148]. Often, directory services include mechanisms for replication and data distribution. One approach to constructing a Grid information service is to push all information into a directory service. While this approach pioneered information services for the Grid, the strategy of collecting all information into a database inevitably limited scalability and reliability. X.500, LDAP and UDDI are some of the most popular directory service standards. The Globus Toolkit's Monitoring and Discovery System (MDS) uses LDAP to publish information about the current state of resources in a Grid environment [140].

Peer to Peer Approach:

Ian Foster, Adriana Iamnitchi and Daniel C. Nurmi introduced the concept of peer-to-peer approach [34] for Grid service discovery. According to them, a resource discovery solution should consist of the following four components:

- Membership protocol: The membership protocol specifies how new nodes join the network and how nodes learn about each other.
- Overlay construction function: The overlay construction function selects the set of collaborators from the local membership list
- Preprocessing: Preprocessing refers to off-line processing used to enhance search performance prior to executing requests.
- Request processing: The request-processing function has a local and a remote component. The local component looks up a request in the local information, processes aggregated requests and/or applies local policies.

The remote component implements the request propagation rule i.e., sending the requests to other peers through various mechanisms such as flooding, forwarding, epidemic communication etc.

Parameter Based Approach:

Muthucumaru Maheswaran and Klaus Krauter introduced a concept called the "Grid potential" [35] that encapsulates the relative processing capabilities of the different machines and networks that constitute the Grid. The Grid potential concept is similar to the time-to-live idea used in the Internet. Informally, the Grid potential at a point in the Grid can be considered as the computing power that can be delivered to an application at that point on the Grid. The computing power that can be delivered to an application depends on the machines that are present in the vicinity and the networks that are used to interconnect them. Consequently, a high performance machine when connected to the Grid will induce a large Grid potential.

QoS Based Approach:

Yun Huang and Nalini Venkatasubramanian [137] addressed the problem of resource discovery in a Grid based multimedia environment, where the resources providers, i.e.

servers, are not available all the time but are only intermittently available. Multimedia applications are resource intensive, and consume significant network, storage and computational resources and have Quality-of-Service (QoS) requirements that specify the extent to which properties such as timeliness can be violated. Quality of Service (QoS) is the summarizing term for the main non-functional properties of such systems, which mainly includes reliability, security, performance and adaptability. Effective load management in such environments requires a resource discovery mechanism that will ensure the continuity of data to the user while servers are intermittently available. Huang and Venkatasubramanian proposed a generalized resource discovery algorithm based on graph-theoretic approach.

ClassAd Matchmaking:

Matchmaking is a distributed resource management mechanism developed as part of the Condor project for Grid systems. The matchmaking is based on the idea that resources providing services and clients requesting service advertise their characteristics and requirements using classified advertisements (classads) [138]. A matchmaker service that may be either centralized or distributed matches the client requests to the appropriate resources. The matchmaking framework includes several components of a resource discovery mechanism. The classad specification defines the syntax and semantic rules for specifying and evaluating the attributes associated with the characteristics and requirements. The advertising protocol lays down the rules for disseminating the advertisements. Figure 2.8 shows the actions involved in the matchmaking process. The resource

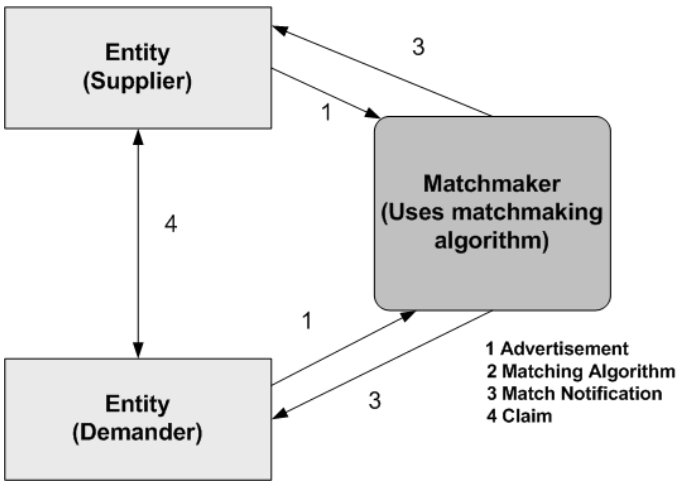

Figure 2.8: Actions involved in matchmaking process

discovery study shows that, in a query based discovery the resource information store is queried for resource availability. Most contemporary Grid systems follow this approach. In agent-based discovery, agents traverse the Grid system to gather information about resource availability. The key difference between a query based approach and an agent based approach is that agent based systems allow the agent to control the query process and make resource discovery decisions based on its own internal logic rather than rely on an a fixed function query engine. UDDI has been evaluated as a provider of Resource Discovery Services for OGSA-based Grids by Edward Benson et al. [116]. UDDI can be used for OGSA discovery, but the cost may be prohibitive for large Grids.

Existing resource description and resource selection in the Grid is highly constrained. Traditional service discovery is done based upon symmetric attribute-based keyword matching with the support of simple query languages. However, exact keyword matching and coordination between providers and consumers make such system inflexible and difficult to extend to new characteristics or concepts. Moreover, in a heterogeneous multi-institutional environment such as the Grid, it is difficult to enforce the syntax of node descriptions since a Grid is setup using a number of middleware, all of which describe the services in their own way [15], [139]. Thus there is a need for a system that is independent of the syntax used for resource descriptions and provides a semantic approach for finding services based on their concept similarity rather than exact matching of their attribute value. It is necessary to allow matchmaking engines to perform flexible matches, those that recognize the degree of similarity between advertisements and requests in order to provide a softer definition of "sufficiently" similar. By increasing the flexibility of a match, they increase the likelihood of finding services that match their requirements by reducing false negatives.

(e) Resource Provisioning

Provisioning is the automation of all the steps required to manage (setup, amend and revoke) user or system access and entitlement rights to electronic services as described by Thomas Lehman in [39] and by Chunfeng wang et al. [136]. A facility or capability exists within a network to carry out provisioning actions on a given set of resources [36], [38]. Gurmeet Singh et al. [40], [128] suggests that the objective of provisioning is to allocate sufficient resources in order to avoid violations of service level guarantees. As per Ronald

P. Doyle et al. [37], Provisioning scenarios can be classified into three categories:

- Dedicated resource provisioning: The simplest case of provisioning is that of dedicated resources allocated to a single consumer. When a subscription request is received, resources are provisioned after considering the service specifications versus the anticipated load. In this case, the SLA system monitors for service level compliance, report violations when they occur. In the absence of the capability to add resources dynamically, the dedicated environments are typically over-provisioned anticipating the worst-case workload scenario.

- Per-SLA virtualized resource provisioning: In an environment in which consumed resources are virtualized (and underlying physical resources are shared in support of multiple SLAs), the provisioning system evaluates whether an additional SLA can be supported with the resources available to this virtualized system. The system may also add new resources to this shared pool based on the aggregate anticipated load of all supported SLAs. As before, the SLA system monitors for compliance and reports violations.

- Dynamic resource provisioning: In this, the most sophisticated on demand environment, resources are allocated to services, as needed. An initial set of resources is provisioned for a service, and the SLA system monitors the performance of that service. When the load changes, new resources are allocated or de-allocated dynamically, in order to minimize service delivery costs while meeting SLA objectives. Dynamic provisioning can be used in conjunction with the previous two scenarios for initial resource allocation. Another provisioning category can be on the basis of static or dynamic i.e. Resource provisioning can be done statically using advance reservations or dynamically using mechanisms such as Condor Glidein [41].

For resource provisioning, QoS parameters need to be defined and provided to the user. Guaranteeing QoS in a non-deterministically shared heterogeneous environment is the main challenge of the matching task. Resource reservation is also considered to be fundamental QoS attribute. A Grid that provides the ability to specify QoS at job submission time but cannot reserve resources in advance is considered to provide only a partial solution to QoS. Yang-suk Kee, Carl Kesselman [187] have presented a resource provisioning

technique named guided redundant submission (GRS), which probabilistically guarantees a timely resource slot allocation. They have introduced a HPDC resource management paradigm named resource slot which defines a network of logical machines across time and space.

In Grid-JQA [109] solution to guarantee the quality of service in Grid computing environment by using Active Database has been provided. It accommodates some actions to be down automatically by insert, delete or update transactions. As a result the best matching is obtained by using the most up to date information. The report in [112] examines the aspect of QoS in the Fabric layer and Application layer. I-Charak, a project that seeks to install a Grid catering to critical health services in rural areas has been the motivating factor in seeking out methods of ensuring QoS for multimedia applications. Xuan Lin et al. [101], [106] discuss the problem of providing performance guarantees to divisible load applications and investigates such algorithms for a cluster environment. Here the design parameters that affect the performance of these algorithms and scenarios when the choice of these parameters has significant effects have been studied. A novel algorithmic approach integrating DLT (divisible load theory) and EDF (earliest deadline first) scheduling has been proposed. Application of DLT to real-time cluster-based scheduling leads to significantly better scheduling approaches as compared to the traditional approaches.

(f) Monitoring

Another important attribute in GRMS is monitoring which involves monitoring of the resource, network and the status of the Job itself. Monitoring is the act of collecting information concerning the characteristics and status of resources of interest [45]. In order for information services to address the mentioned user needs, they must systematically collect information regarding the current and, sometimes, past status of Grid resources; a process known as monitoring. In addition to information services, monitoring is also crucial in a variety of cases such as scheduling, data replication, accounting, performance analysis and optimization of distributed systems or individual applications, self-tuning applications, and many more [42], [43], [44].

The Globus has a health-check facility referred to as the Globus Heartbeat Monitor (HBM) [45], [46]. It is designed to detect and report processes that fail to provide a

'heartbeat'. Within the Globus specification [47] the HBM consists of:

- HBM Client Library
- HBM Local Monitor and
- HBM Data Collector

Essentially a Local Monitor runs on each host, checking and reporting on the status of the monitored processes and the actual system generating "I-am-alive" messages (heartbeats). The Data Collector receives heartbeat messages and identifies failed components from missing heartbeats. There may be one or more data collector per application.

Beacon Monitoring:

Beacon monitoring was first used in NASA missions, particularly those to deep space [48]. In high-level concept terms, the beacon monitor is similar to the heartbeat monitor in Grid Computing, with the addition of a tone to indicate the degree of urgency involved.

Pulse Monitoring:

A hybrid approach for the Autonomic environment [49] is to use the urgency concept of the beacon monitor to turn the heartbeat monitor into a pulse monitor-so instead of just checking the presence of a beat; the rate is also measured as shown in Figure 2.9. The

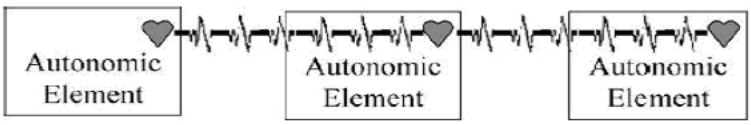

Figure 2.9: Pulse Monitoring [49]

concept of the pulse monitor is based on extending the HBM construct. The HBM itself provides a means to ensure a vital process may be safeguarded. The lack of a heartbeat will alert the designated remote HBM that the process has died (or indeed the communications themselves have failed). This relative instant alert to the fact a process is no longer functioning enables immediate actions such as restarting the process and as such minimizing disruption.

Yet, vital as it is, essentially the HBM only informs if a process is alive or dead (assuming communications are working) - not the processes actual health or state of being. Taking

the biological analogy the rate of the heartbeat indicates the current conditions within which the biological 'system' is operating and determines strategies for 'components' within the system.

An important point to note from the HBM, and also from the Beacon Monitor, is the minimization of data sent - essentially only a 'signal' is transmitted. Any move towards sending more information must not compromise this reflex reaction. As such the tone or the beat must contain within it the urgency level.

B. Tierney et al. [46] discuss about the Grid Monitoring Architecture (GMA) provided by GGF. The GMA is a producer/consumer-based architecture that is driven by a standard and extensible set of events. The different components of the GMA are sensors, producers, consumers, and the directory service. The GMA was devised as a framework for monitoring Grid systems. However, it goes beyond that and provides a framework for combining monitoring and information systems. The model is not constrained by any protocol or data models. Therefore, implementers are free to choose their own data models for querying over the monitoring system.

Hawkeye [154] is another distributed monitoring system, coming from the University of Wisconsin Madison. It provides mechanisms for monitoring distributed collections of computers by gathering computer-based resource information. Hawkeye's design goals include retrieval of host resource information in a consistent and extensible manner and the ability to automatically execute tasks in response to observed conditions on the monitored hosts. Resource information, is packaged as a self-contained system. Data collected by Hawkeye is made available for applications and users to manage monitored resources. Condor uses Hawkeye [45] to monitor characteristics and status of resources.

Ganglia [46], [155] is an open-source project that grew out of the University of California, Berkeley. Ganglia is a distributed monitoring system for high performance computing systems such as clusters and the Grid. It is based on a hierarchical design targeted at federations of clusters. Ganglia uses XML for data representation, XDR [157] for data transport, and RRD tool [158] for data storage and visualization. Ganglia attempts to minimize per-node system overheads by using specialized data structures and algorithms [155], [156].

The Relational Grid Monitoring Architecture (R-GMA) [160] monitoring system is an

Table 2.2: Comparison of Existing Resource Monitoring Systems

Parameters/Systems	*Ganglia*	*Hawkeye*	*RGMA*	*MDS4*
Hierarchical Design	Yes	No	Yes	Yes
Visualization	Through RRD-tools	No	Simple like Pulse	WebMDS
Scheduler Supported	Independent	Mostly Condor	Independent	Mostly Globus
Implementation Language	C	C++	Java Servlets	Java
Scalability	High	Low	High	High
Extensibility	High	Medium	High	High
Robustness	High	Low (single point of failure)	Medium (single point of failure in registry and schema)	High
Security	Not explicitly	X.509 based,Not explicitly	GSI (Grid security Infrastructure)	GSI (Grid Security Infrastructure)
Data Representation	XML	XML based ClassAd	XML	XSLT based visual interface
Portability	MS Window, Linux, Solaris	Limited to Solaris and Linux	Linux only	Limited to Solaris and Linux

implementation of the Grid Monitoring Architecture (GMA) defined within the Global Grid Forum (GGF). It is based on the relational data model and Java servlet technologies. Its main use is notification of events that is, a user can subscribe to a flow of data with specific properties directly from a data source. For example, a user can subscribe to a load-data data stream and create a new producer/consumer pairing to allow notification when the load reaches some maximum or minimum level [159], [160].

The Monitoring and Discovery System (MDS4) [179] of the Globus Toolkit 4 is a suite of components for monitoring and discovering Grid resources and services. MDS4 is compliant with WSRF and WS-Notification specifications. The Globus Toolkit 4's Monitoring and Discovery System (MDS4) implements a standard Web Services interface to a variety of local monitoring tools and other information sources. Grid computing resources and services can advertise a large amount of data for many different use cases. MDS4 was specifically designed to address the needs of a Grid monitoring system - one that publishes data that is available to multiple people at multiple sites. As such, it is not an event handling system, like NetLogger [180], [181] or a cluster monitor on its own, but can interface to more detailed monitoring systems and archives and can publish summary data using standard interfaces.

Table 2.2 presents a comparison of above explained Resource Monitoring Systems on the basis of the architecture, characteristics, implementation model and several other features.

Comparison shows that no Resource Monitoring system is complete. Each is having certain limitations; Ganglia is compatible with all Operating Systems but it does not have any security feature available explicitly; Hawkeye has a single point of failure and is not very much scalable, also it does not provide any visual user interface; RGMA is compatible with Linux only and has single point of failure in registry and schema; MDS4 is better than rest of the monitoring systems discussed above but it too is not compatible with MS Windows. So there is need to develop a monitoring system for Grid environment that overcomes the limitations of existing resource monitoring systems.

2.3.2 Grid Brokers and Metaschedulers

The Gridwise-Tech Report [149] discusses the features and use cases for the Grid Brokers and Metaschedulers. It discusses Community Scheduler Framework (CSF), CSF Plus, Grid Service Broker, GridWay, Moab Scheduler, EGEE Workload Manager Service, Nimrod/G, MP Synergy and Condor-G. Grid Brokers bridge the gap between user-level applications, like web portals and low-level Grid fabric. One of the popular brokers, GridBus Broker [182] supports both computational and data Grids. Failures of Jobs can be handled through Job wrapper and Job monitor code. Gridbus provides a service-oriented cluster and Grid middleware for e-Business and e- Science applications. Its design and development is based around the notion of 'utility computing' and uses various economic models for efficient management and usage of shared resources.

Sathish S. Vadhiyar and Jack J. Dongarra [118] describe implementation of a metascheduler for the Grid that takes into account both the application level and system level considerations. These components provide valuable scheduling services that play important roles in providing scalability of the Grid system.

Resource Brokering has also been discussed by Johan Tordsson in his Master's Thesis submitted in 2004 [123]. This thesis describes the development of a broker that makes resource selection on the basis of load and performance of the resources as well as char-

acteristics of the application, aiming at minimizing the total time to delivery for each application. The total time to delivery includes the time required for input/output transfer, batch queue waiting and application execution. The methods used to minimize the total time to delivery include the usage of advance reservations, characterization of resource performance by the use of benchmarks and estimation of data transfer times from network performance predictions. Additionally, a basic queue adaptation mechanism is used in order to adapt to changes in load on the Grid. A. Kertesz and P. Kacsuk have proposed a meta-brokering approach, which means a higher level resource management by enabling communication among existing Grid Brokers and utilizing them. They have defined a general meta-brokering architecture to be implemented as a Service Oriented Knowledge Utility (SOKU) in future generation grids [186].

GRIP Project [124] describes the Resource Discovery and Brokering system used within the UNICORE Grid System and describes how it may be used for selection of resources within a general computational Grid. The process of Resource Brokering is the matching of abstracted user tasks to concrete resources capable of executing these tasks. The EUROGRID broker works by taking the set of resources requested and testing whether those resources are available at a particular site. Once availability has been established, the broker then contacts the site to obtain a QoS offer for that particular configuration of task. Chuang Liu et al. [125] have discussed a general-purpose resource selection framework that addresses these problems by defining a resource selection service for locating Grid resources that match application requirements. At the heart of this framework is a simple, but powerful, declarative language based on a technique called set matching, which extends the Condor matchmaking framework to support both single resource and multiple-resource selection. This framework also provides an open interface for loading application-specific mapping modules to personalize the resource selector.

2.3.3 Existing Resource Management Frameworks

Karl Czajkowski et al. [104] discusses the translation of OGSI to WSRF. The WS-Resource Framework retains essentially all of the functional capabilities present in OGSI, while changing some of the syntax (to exploit WS-Addressing) and also adopting a dif-

ferent terminology in its presentation. There has also been an increasing amount of work on Grid resource management based on service agreements. An overview of such approaches can be found in LNCS 2005 by C. Molina-Jimenez [150]. A Resource Management Framework for Interactive Grids has been discussed by Raj Kumar et al. [119]. The technical report, HPL-2005 [27] discusses issues and conceptual architectural design of a dynamic resource management framework, which leverages the open-source Globus Toolkit and commercially available HP Open-View Configuration Management Solutions software (Radia). This framework is based on automated, policy-driven change and configuration management functionality that can dynamically adjust the size, configuration and allocation of various resources that will be consumed in the environment. Globus-Radia Resource Management System (GRRMS) describes a modular and extensible approach for dynamic resource allocation and management in a Grid environment. The novelty of this approach is the ability to provision resources based on the job requirements.

The GLARE system [38], developed as part of the ASKALON framework, allows automatic deployment and on-demand provisioning of components for Grid applications. In this work, the authors adopt a peer-to-peer architecture for resource management and consider 'activities' as generalized abstractions of Grid tasks/jobs, which can be deployed during run-time on different computers and executed in a coordinated manner to accomplish a particular application goal.

GrADS [151] (Grid Application Development Software) is a framework designed to solve numerical applications over the Grid. An important aspect of this framework is the ability to decide when to migrate jobs based on actual and predicted execution times. If the system detects minimal differences between the actual and predicted performance of the application, it avoids job migration. Such an approach makes those jobs, which are about to finish execution in a short period of time less susceptible to suspension and migration. Junwei Caoa et al. discuss about ARMS, an agent-based resource management system for Grid computing [120]. ARMS utilizes the performance prediction techniques of the PACE toolkit to provide quantitative data regarding the performance of complex applications running on a local Grid resource. At the meta-level, a hierarchy of homogeneous agents has been used to provide a scalable and adaptable abstraction of the system ar-

chitecture. A framework for monitoring, and managing a Grid in terms of a hierarchy of agents has been proposed in this paper. Similarly Isaac Chao, Ramon Sangesa and Oscar Oardaiz [185] have proposed Grid Resource Management using software agents. They have presented work developed by the GridCat group in order to provide an agent-based Grid Resource Management (GRM) system compliant with adopted standards both for Grid and Agent computing.

Agreement-Based Resource Management has been discussed by Karl Czajkowski, Ian Foster and Carl Kesselman [92]. The fundamental underlying concept in this framework is the representation of various resource management activities in terms of an agreement. Agreements abstract local management policy by representing an underlying resource strictly in terms of policy terms which it is willing to assert, and in doing so provides the basis for building a variety of alternative Grid resource management strategies. In this paper the concepts of agreement based resource management have been introduced. A general agreement model has been presented to examine the current resource management systems in the context of this model. The general resource management functions can be expressed as some combination of three different kinds of agreement.

- Task service-level agreements (TSLAs)
- Resource service-level agreements (RSLAs)
- Binding service-level agreements (BSLAs)

Xingchen Chu et al. presented the design, implementation and evaluation of Aneka, a new service-oriented enterprise Grid-Computing framework [102]. Aneka improves over existing desktop Grid implementations with fixed capabilities, by using a container in which services can be added to augment the capabilities of the node. While the container is lightweight in itself, there is scope for improving message handling at the container level and the response time for a large number of clients. DataGrid is a project funded by European Union [103]. The objective of this project is to build the next generation computing infrastructure providing intensive computation and analysis of shared large-scale databases, from hundreds of TeraBytes to PetaBytes, across widely distributed scientific communities.

Jia Yu, Srikumar Venugopal, and Rajkumar Buyya have discussed a web services based Grid service publication and discovery infrastructure called the Grid Market Directory (GMD) [107]. The GMD consists of two key components: the Portal Manager and Query Web Service. The GMD Portal Manager is responsible for provider registration, service publication and management, and service browsing. All these tasks are accomplished by using a standard web browser. The GMD Query Web Service provides services so that clients such as resource brokers can query the GMD and obtain the information of resources to discover those that satisfy the user QoS requirements. The GMD has been implemented as part of the open source Gridbus project.

The MS thesis by Bouwmeester [108], makes an attempt to develop GridAssist for Grid middleware interoperability with the help of wrapper code. This framework provides the possibility for multiple wrappers to be plugged into GridAssist, each wrapper providing communication to one Grid middleware environment. Globus Toolkit version 2, Globus Toolkit version 4, gLite and Uniform Interface to Computing Resources (UNICORE) are currently the best candidates for integration with GridAssist. A wrapper for the Globus Toolkit version 4 middleware environment has been implemented and tested. And a wrapper for gLite has been researched but could not be completed with the current version of this middleware.

The framework discussed by Rajkumar Buyya et al. [121] provides mechanisms for optimizing resource provider and consumer objective functions through trading and brokering services. It demonstrate the effectiveness of some of the economic models in resource trading and scheduling using the Nimrod/G resource broker with deadline and cost constrained scheduling for two different optimization strategies on the World Wide Grid (WWG) testbed that has resources distributed across five continents.

Another framework Virtuso is regarding distributed computing using virtual machines [133]. It uses VMWare GSX Server: Windows 2000 professional to create numerous virtual machines and thus creating a virtual distributed environment. Virtual Machines enable fundamentally different approach to Grid Computing. MARS , an open-source metascheduling framework [134] can be integrated into existing campus infrastructure to provide robust task scheduling and resource management capabilities, a forecasting algorithm to predict resource level scheduling has been used and thus main focus is on

meta scheduling.

Vishwa, is a scalable reconfigurable P2P middleware for Grid Computing and is an initiative of IIT- Madras [145]. It is a two-layered P2P Grid middleware, which takes advantages of both P2P and Grid systems and aims to be scalable and reconfigurable. But P2P security mechanisms need to be explored in Vishwa and currently there is no support for Data Management also.

Almost all existing Grids have their Grid Portals. A Grid Portal is problem solving environment that allows scientists to program, access and execute distributed "Grid" applications from a conventional Web Browser and other desktop tools [168]. A number of Grid portals exist or are currently under construction including the NEESGrid Portal [9], the Alliance Portal [172], the GENIUS Portal for the European Data Grid [169], and the IeSE (Integrated e-Science Environment) [170]. Also, several large collaborative efforts are underway to build portal construction toolkits based on OGSA Grid services, including the GridSphere project [173] in Germany and the NSF NMI Open Grid Computing Environment project in the US. The GridSphere portal framework [175] developed as part of Gridlab project provides a framework that offers external developers a model for easily adding new functionality and hence increasing community collaboration. M. P. Thomas et al. [174] presents an overview of the current state-of-the-art in Grid portals, based on a component approach that utilizes portlet frameworks and the most recent Grid standards, the Web Services Resource Framework and a summary of current DOE portal efforts.

This section reported about the resource management attributes, Grid brokers, metaschedulers and the existing resource management frameworks. The next section would present the monitoring techniques and fault management practices of existing middleware.

2.4 Monitoring and Fault Management Systems

The inherent instability of Grid environments arises the need to address fault-detection and recovery. Till date, no system can be said to be absolutely fault tolerant. The need for fault-tolerance is especially acute for large parallel applications since the failure rate grows with the number of processors and the duration of the computation. Grid is a

dynamic system and the nodes can join and leave voluntarily. For making fault tolerance system a success, how new nodes join the system, how computing resources are shared, how the resources are managed and distributed, must be considered. Fault tolerance is the survival attribute of all future computer systems [130].

Paul Stelling et al. proposes a fault detection service designed to be incorporated [131], in a modular fashion, into distributed computing systems, tools, or applications. The service utilizes unreliable fault detectors to detect and report component failure, while allowing the user to tradeoff timeliness of reporting against false positive rates. Specific fault detector service, namely the Globus Heartbeat Monitor (HBM), which provides a fault detection service for applications developed with the Globus toolkit has been implemented.

The need for Web-based portals that hide low-level details of accessing Grid services for deployment and execution management of applications is increasing. G-Monitor [141] allows the user to monitor, control, and steer execution of application jobs on Global Grids. Unlike related systems, the users of G-Monitor delegate the responsibility of selection of appropriate resources for execution of their application jobs to the broker. Another system, Grid Enactor and Management Service (GEMS) [161] is used for submission, monitoring and and management of jobs to Grid resources. It supports the detection of individual job failures; it requires that a local resource manager support certain fault detection and reporting capabilities. Another issue of Grid Faults is regarding Load Balancing [132]. Xuan Lin et al. also discuses about the issues of Load Balancing in a distributed environment [101], [106].

Different middleware in the market use different approaches for handling faults. Raissa Medeiros et al. [152] have categorized the Grid Faults as:

- Configuration faults
- Middleware faults
- Application faults
- Hardware faults

Configuration faults can occur because of the mismatch of the resources in the Grid. For example, configuring queues to execute the jobs may result in configuration fault. The second reason of fault occurrence in the Grids is due to the deficiencies left while designing the middleware. For example, these types of failures may include suspension of job in waiting state due to poor scheduling approach used by the middleware. Applications faults can occur due to some exception or error occurrence in job while executing it. Failure of hardware many times lets the execution nodes to die in between the job execution. This leads to failure of the job itself.

Existing Grid middleware still lacks mature fault tolerant features and next generation Grids [153] need to solve this problem. This can be overcome by providing a more dependable infrastructure to execute large-scale computations, by using remote clusters and high performance computing system etc. The fault tolerance system must satisfy following properties in order to be more successful and absolute [131].

- Completeness: is the state of being complete and entire; having everything that is needed. Any failure (process or machine crash) is eventually detected by all normal processes.

- Accuracy: The fault detector must identify faults accurately, with both false positives and false negatives being rare. Here false positives are responses that incorrectly identify items as being from the original list when they were not on that list and false negatives are vice-versa.

- Consistency: All processes that are not declared as failed obtain consistent failure information including false positives.

- Detection Latency: Problems must be identified in a timely fashion, so that responses and corrective actions can be taken as soon as possible. The latency between a failure and its detection must be low.

- Scalability: The design of the fault detector must be capable of scaling to large numbers of processes and computers. CPU and network loads should be low for a large number of participating processes.

- Flexibility: Various network settings should be tolerated. In particular, firewalls or NATs may restrict connectivity between some live node pairs. Finally, they must be accomplished without tedious and error-prone manual configuration.

- Adaptiveness: Systems should bring up with a small amount of manually supplied information about network settings.

- Low overhead: Monitoring should not have a significant impact on the performance of application processes, computers, or networks

Based on the above-defined taxonomy, the fault tolerance in existing Grid middleware has been surveyed and compared in the next section.

2.4.1 Fault Management Mechanisms in Various Grid Middleware

Fault Tolerance in Globus

Globus provides a heartbeat service to monitor running processes to detect faults [131]. The application is notified of the failure and expected to take appropriate recovery action. Many tools are available in the market to handle faults in Globus. The basic entities used by each system are: local monitor and data collector.

- A local monitor is responsible for observing the state of both the computer on which it is located and any monitored processes on that computer. It generates periodic "i-am-alive" messages or heartbeats, summarizing this status information.

- A data collector receives heartbeat messages generated by local monitors and actually identities failed components based on missing heartbeats.

- Scalability and performance concerns require that the heartbeats communicated by the local monitors be transmitted via a connectionless, typically unreliable protocol. Hence, the data collector must take into account network delays and possible packet loss when interpreting a lack of heartbeat from a particular local monitor. Globus Heartbeat Monitor (HBM), which provides a fault detection service for applications developed with the Globus toolkit [161].

In brief, an application that wants to use the HBM service needs to do just two things:
(i) Register the processes for which failure detection is required, either by calling the registration API directly, or by having the registration function called externally on behalf of the application.
(ii) Use the data collector API to construct a data collector that implements the desired application-specific fault behavior, whether this is global termination, rescheduling of a failed component, further testing to verify that failure has occurred, etc.

Fault Tolerance in CONDOR-G

Condor [69] provides system-level checkpoints of distributed applications but it is not geared to high-performance parallel programs. Condor-G [41] is built to tolerate four types of failure:

- crash of the JobManager
- crash of the machine that manages the remote resource (the machine that hosts the GateKeeper and JobManager),
- crash of the machine on which the GridManager is executing (or crash of the GridManager alone), and
- failures in the network connecting the two machines

The GridManager detects remote failures by periodically probing the JobManagers of all the jobs it manages. If a JobManager fails to respond, the GridManager then probes the GateKeeper for that machine. If the GateKeeper responds, then the GridManager knows that the individual JobManager crashed. Otherwise, either the whole resource management machine crashed or there is a network failure (but the GridManager cannot distinguish between these two cases). If only the JobManager crashes, the GridManager attempts to start a new JobManager to resume watching the job. Otherwise, the GridManager waits until it can reestablish contact with the remote machine. When it does, it attempts to reconnect to the JobManager. This can fail for two reasons:
(i) the JobManager crashed (because the whole machine crashed), or
(ii) the JobManager exited normally (because the job completed during a network failure).

In either case, the GridManager starts a new JobManager, which will resume watching the job or tell the GridManager that the job has completed. To protect against local failure, all relevant state for each submitted job is stored persistently in the scheduler's job queue. This persistent information allows the GridManager to recover from a local crash. When restarted, the GridManager reads the information and reconnects to any of the JobManagers that were running at the time of the crash. If a JobManager fails to respond, the GridManager starts a new JobManager to watch that job.

Fault Tolerance in Legion

Legion [131] provides fault tolerance mechanisms such as Checkpointing, but the implementation policy is left to the application. A component-based reflexive architecture allows fault tolerance methods [60] to be encapsulated in reusable components that user applications may choose from .

Two properties characterize reflective systems: introspection and causal connection.

- Introspection allows a computational process to have access to its own internal structures.
- Causal connection enables the process to modify its behavior directly by modifying its internal data structures, there is a cause-and-effect relationship between changing the values of the data structures and the behavior of the process.

The implementation of the reflexive architecture relies on common basic primitives such as:

- Intercepting the message stream
- Piggybacking information on the message stream
- Acting upon the information contained in the message stream
- Saving and restoring state
- Detecting failure
- Exchanging protocol information between participants of an algorithm

Fault Tolerance in SUN N1 Grid Engine

N1GE6 uses Checkpointing as the main method to handle the faults. Checkpointing in N1GE6 can be classified at two different levels [63], [64]:

Kernel-level

- Tools are built into the kernel of the operating system. During a checkpoint, the entire process space (which tends to be huge) is written to physical storage.
- The user does not need to recompile/re-link their applications.
- Checkpointing and restarting of application is usually done through OS commands.
- Checkpointed application is usually unable to be restarted on a different host

User-level

- Tools are built into the application which will periodically write their status information into physical storage.
- Checkpointing of such applications is usually done by sending a specific signal to the application.
- Restarting of such applications is usually done by calling the application with additional parameters pointing to the location of restart files.

N1GE6 has built-in support for the integration of 3rd party checkpointing tools. Certain checkpointing tools (mostly user-level) allow the restart of applications on different hosts. These tools, coupled with the migration support on the N1GE6 and with proper configuration of the queue threshold levels, allow the administrator to finely load balance the N1GE6 cluster. One such tool is Berkeley Lab Checkpoint/Restart (BLCR). BLCR is a kernel module that allows you to save a process to a file and restore the process from the file. This file is called a context file. A context file is similar to a core file, but a context file holds enough information to continue running the process. A context file can be created at any point in a process's execution [63]. The process may be resumed from that point at a later time, or even on a different workstation.

Fault Tolerance in Alchemi.NET

Alchemi [62] is a .NET-based Grid computing framework that provides the runtime machinery and programming environment required to construct desktop Grids and develop Grid applications. Till date not much work has been done on Alchemi Fault tolerance. The basic technique used by Alchemi for Fault Tolerance is Heartbeating. The Executors in the Alchemi send the heartbeat signals to the manager at regular interval of time. Manager after receiving signals from executors guesses that the executor node is still working. The time interval at which the executor will report their status can be adjusted. The procedure of fault tolerance is not foolproof. For restart of the jobs (which may be caused due to many reasons), Alchemi maintains the logs rather than checkpoints.

The main problems identified in Alchemi include:

- Whole Grid fails as the manager node fails. An immediate problem here is that once a Manager goes offline all Executors that are running threads will probably fail as well.

- Executing Thread 'Hangs' and manager fails to detect it. This means that the executing thread is not responding but the heartbeat thread keeps working. Currently the Executor remains in hang state until it is restarted.

The next section compares the above-discussed middleware on the basis of taxonomy defined in section 2.4.

2.4.2 Comparative Analysis of Fault Management in Grid Middleware

Among all the Grid middleware discussed in previous sub-section, Sun N1GE6 provides a comparatively efficient and sturdy method of handling faults. It provides the Shadow manager host to handle the central authority failure has standard monitoring system to monitor the Grid and also provides facility for Checkpointing, to restart the job using third party software. Globus and Alchemi use the heartbeat techniques. On the other hand both Condor/G and Legion use the Checkpointing technique. Condor/G uses the System-level Checkpointing whereas Legion uses the Application-level Checkpointing.

Table 2.3: Comparative Analysis of Fault Management in Grid Middleware

Feature/Middleware	*Globus*	*Condor/G*	*Legion*	*N1GE6*	*Alchemi*
Technique Used	Uses HBM	Automatic system level checkpointing to handle single process jobs	Application level checkpointing	Supports Kernel or user level checkpointing	Heartbeating
Completeness	Facilitates fault detection only, no provision for fault management	Fails many times incase of individual parallel processes	Depends on the reusable code inserted in application for fault handling	Depends on third party software like BLCR for fault detection and management	Only detects the failure of executors, failure of manager and manual switch off of executors is undetected
Accuracy	Faults are detected correctly at different nodes	Mostly accurate, occasionally, local resource manager fails to detect fault, thus user intervention is required	Vary from application to application	Context file is created to save the current state, which helps in restart	Is accurate but can handle a limited faults only
Detection Latency	Depends upon heartbeat interval	Uses automatic checkpointing, thus fault detection time is very less	Application performance degrades slightly as fault tolerance code has to be compiled	for maintaining the checkpoints, the latency to log the failure increases	Executor nodes must specify heartbeat interval, while it connects to manager
Overhead	Not much overhead involved	Overhead increases with the increase in number of new nodes in VO	More checks for fault detection, more is overhead	Depends upon the checkpointing package used in the system	Uses UDP, so no overhead
Adaptiveness	Highly flexible, therefore, easily adapts itself to Grid environment	Less adaptive, due to system level checkpoints	Code must be rearranged to fit it into the application	Built-in support for integrating third party checkpointing system	Integrated in Alchemi package, requires .NET(Window) platform

Further comparison between all these middleware has been described in Table 2.3.

This section discussed about the existing Grid monitoring and fault management systems. Next section discusses about two techniques related closely to Grid Computing.

2.5 Related Technologies

This section discusses two popular technologies very closely related to Grid Computing namely Semantic Technologies and Autonomic Computing. Both these technologies have impacted the Grid research in numerous ways.

2.5.1 Semantic Web Technologies

Semantic Grid [71], [147] is an extension of the current Grid in which information and services are given well defined and explicitly represented meaning, so that it can be shared and used by humans and machines, better enabling them to work in cooperation. Semantics need to be applied to a Grid as universal resource description language, common to all Grid systems, because it does not yet exist [114].

A Grid depends on understanding the available resources, their capabilities, how to assemble them and how to best exploit them. Grid middleware, and the Grid applications they support, thrive on the metadata that describes resources in all their forms, the VOs, the policies that drive them and so on, together with the knowledge to apply that metadata intelligently. Due to these reasons, resource discovery in large-scale Grids can be very challenging. Semantic Web Technologies [72], [98] can be applied as a solution to this problem and as an alternative for bridging the gap that exists between infrastructures with incompatible information schemas [77], [78], [127].

In this direction, the Semantic Grid Research Group (SEM-RG) is working to realize the added value of emerging Web technologies and approaches [164], in particular Semantic Web and Web 2.0, for Grid users and developers. Resource Description Framework (RDF) [165] and Web Ontology Language (OWL)[166], [117] are key technologies of Semantic Web extended to the Grid. RDF is a web-enabled language of Subject, Verb, Object triples to represent relationship between data. Ontology is the specification of conceptualization- defines terms and relationships between them. RDF and OWL are developed under the vision of semantic web and W3C recommendations, the Grid Ontology is thus open, and compatible with other systems and solves the problem of interoperability.

Oscar Corcho et al. [99] discuss a reference architecture for the Semantic Grid. The Open Grid Service Architecture (OGSA) aims to define a core set of capabilities and behaviors for Grid systems. In this paper a Reference Architecture that extends OGSA to support the explicit handling of semantics, and defines the associated knowledge services to support a spectrum of service capabilities has been discussed. Guided by a set of design principles, Semantic-OGSA (S-OGSA) defines a model, the capabilities and the

mechanisms for the Semantic Grid.

The myGrid [163] project is a pioneering Semantic Grid effort, which has developed a suite of tools and services to enable workflow based composition of diverse biological data and computational resources. Within the project SemanticWeb technologies have been applied to the problems of resource discovery and workflow results management. A characterizing aspect of these solutions is their focus on user-orientation, which has resulted in

(a) semantic aware decision-support components instead of decision making components and

(b) user-facing tools that keep the human in the loop during metadata generation and querying.

InteliGrid [162] proposes architecture based on three layers: conceptual, software and basic resource. At the conceptual layer descriptions of knowledge entities such ontologies, notions, graphs, etc. has been done, while the software layer consists of software that consumes the knowledge entities found in the conceptual layer (this is equivalent to the Semantic Aware Grid Services from our architecture). The basic resource layer includes the low level infrastructure and resembles the notion of the Grid fabric. Ontology services, situated in the software layer, play a central role in this architecture as they are considered as interoperability services that support multiple functionalities such as data consistency, service discovery, VO set-up management, etc.

Thamarai Selvi et al. [115] proposes semantic Grid architecture by introducing a knowledge layer at the top of Gridbus broker architecture and thereby enabling broker to discover resources semantically. The semantic component in the knowledge layer enables semantic description of Grid resources with the help of ontology template. P. Wieder et al. [146] propose to make scheduling-specific parts of such knowledge exploitable by introducing scheduling domain ontology in the Core Grid Technical report. This ontology provides a common semantic understanding to be shared between the components involved in the scheduling process. By agreeing upon and integrating such ontology the automation level can be increased and can further make usage and administration of Grids easier.

2.5.2 Autonomic Computing

"Autonomic Computing" is IBM's term for an approach and blueprint including a set of products, tools and services that add self-managing capabilities to Information Technology systems [73], [74]. IBM introduced the Autonomic Computing in 2001, with the aim to develop self-managing systems [75]. The most direct inspiration for these capabilities that exist today is the autonomic function of the human central nervous system. Autonomic Computing advocates the use of Technology itself to manage, configure and make technology self-reliant. Autonomic Computing includes capabilities unknown in traditional products and also includes the capacity not just to take automated action, but to do so based on an innate ability to sense and respond to change and thus addresses the issues of prohibitively expensive and too time consuming needs of today where near perfect service uptimes are desired. By automatically tuning and balancing workloads, dynamically shifting resources to areas of differentiation and growth, and routinely and automatically handling mundane manual processes, self-managing systems help to optimize resources for greater efficiency and productivity. An autonomic system senses its operating environment, models its behavior in that environment, and takes action to change the environment on its own [76]. It has the following four properties: Self-healing, Self-optimization, Self-protecting, Self-configuring [79], [80].

Self-healing: Self-healing systems can detect improper operations (either proactively through predictions or otherwise) and then initiate corrective action without disrupting system applications. The complete system as a whole becomes more resilient as changes are made to reduce or help eliminate the business impact of failing components.

Self-optimization: Self-optimization refers to the ability of the system to efficiently maximize resource allocation and utilization to meet end-users' needs with minimal human intervention. Self-optimization verifies optimum quality of service for both system users and customers.

Self-protecting: A self-protecting system can detect hostile or intrusive behavior as it occurs and take autonomous actions to make itself less vulnerable to unauthorized access and use, viruses, denial-of-service attacks and general failures. The self-protecting capability allows, to consistently enforce security and privacy policies, reduce overall security

administration costs and ultimately increase productivity. Self-protection is also about recognizing and dealing with overload conditions that could jeopardize the integrity of the system.

Self-configuring: With the ability to dynamically configure itself a system can adapt immediately-and with minimal human intervention-to the deployment of new components or changes in environment. Dynamic adaptation helps verify continuous strength and productivity of any system.

Tianfield Huaglory and Unland Rainer present an overview on some of the exciting interdisciplinary research areas that have emerged in recent years in multi-agent systems and Grid computing, e.g., semantic Grids, agent-based Grid computing, service-oriented architectures, autonomic computing, self-organizing software applications, and large-scale open multi-agent stems [100]. It is a common trend that large-scale complex computing systems and software applications are required to be endowed with self-organizing capabilities. Ya-Ping Zhang et al. propose a self-management model based on multi-level distributed intelligent agents and worm techniques [135]. As per this model, the computing system comprises of Autonomic Elements. Every autonomic element involves two parts: a functional unit and a management unit. Management units use agent and worm techniques. The major feature of this model is that autonomic elements can manage themselves given high-level objectives from administrators. Several key technologies involved in resource discovery, relationship among agents, autonomic element upgrading have been discussed too.

2.6 Problem Formulation

Based on the literature survey, it is apparent that Grid is a heterogeneous and dynamic environment and in such an environment, there are a number of resource management challenges [142]. One of the key challenge is to develop a flexible, scalable, and self-adaptive resource management system which allows users to carry out jobs by transparently accessing autonomous, distributed, and heterogeneous resources. A number of middleware exist today but still, it is very difficult to find out an interoperable resource discovery approach as the resources may lie under different administrative domains and

access and design policies may also differ. Another issue is that it is necessary to cater to the user needs optimally which may fluctuate from time to time and provide the user set of provisioned services. Apart from this, the failures in Grid can be due to many reasons like configuration problems, independent failures of each element, failures resulting from interactions between components etc. The existing solutions for these failures are mainly application dependent. Thus, the main patterns, which surface from the study, are as follows [29]:

First, currently there is not enough manageability functionality in OGSA. This functionality needs to be defined, since it is essential to provide systems that are more flexible, self-managing, or have lower management burden-attributes in which Grid technologies are expected to bring improvements.

Second, OGSA is defining new entities, such as the jobs and containers of EMS (Execution Management Services) or the VOs (Virtual Organizations), and these entities will need resource models. These models should be defined based on the re-use of existing models.

In the proposed work, the Grid resource management has been explored to

- describe a more flexible, self-managing resource model, having lower management burden and which allows higher interoperability.
- A resource management framework, with these parameters has been proposed. The proposed framework has been inspired from Semantic Grid and Autonomic Computing.

The main objectives of the proposed work are:

a) To study and explore resource management techniques in a Grid environment.

b) To propose a self-managing model having higher interoperability for resource management in OGSA by analyzing and re-using existing models along with its interface specification.

c) To propose manageability interface framework for job management to monitor the status of individual resource queues, to be able to manage and control them.

d) To test and demonstrate the use of b) and c) above.

Next chapter proposes a resource management framework to address the issues identified in problem formulation and to accomplish the objectives of this research work.

Chapter 3

PRATHAM: Proposed Grid Resource Management Framework

The previous chapter discussed the research work accomplished in the area of Grid Resource Management. The study depicted that challenges of interoperability, self-management and better user provisioning have not been fully addressed. To provide a solution to these problems, a Grid Resource Management Framework, named PRATHAM has been proposed and designed.

This chapter discusses the goals, requirements, architecture and components of the proposed framework. The requirements have been analyzed with the help of UML and different perspectives have been modeled through various diagrams. On the basis of these requirements a five-layer architecture for the framework has been proposed which is based on OGSA. The architecture shows all the components in detail, which help in accomplishing the goals of the Framework.

3.1 Goals of Proposed Framework

In this work, the outcome of the research objectives is a Grid Resource Management Framework (GRMF), PRATHAM. In general the Grid Resource Management scenarios often include:

- Resource discovery
- Resource inventories
- Resource provisioning
- Resource monitoring
- Variety of autonomic capabilities

PRATHAM should also accomplish these tasks besides catering to the needs of interoperability, resource provisioning and self-management in Grid environment. It is an endeavor to provide Grid users not only the freedom of higher operability but also enhanced resource provisioning within a Grid environment, along with better self-management facilities. Major Goals perceived by PRATHAM are:

- Providing efficient and interoperable resource discovery service.
- Providing resource provisioning of resources with advanced reservation facility.
- Providing an efficient monitoring system.
- Providing autonomic capabilities to the Grid.

3.2 Framework Requirements

The first step in designing a framework is to collate its requirements and examine the basic necessities and functions, which the Grid Resource Management System (GRMS) should accomplish. The detailed requirements have to be gathered for designing a system, which furnish the solution for the desired problems and fulfills its goals. As these requirements need to be studied and analyzed from different perspectives, Unified Modeling Language (UML) has been used.

3.2.1 UML: Tool for Analyzing Framework Requirements

UML is designed to let developers and customers view a system from different perspectives and in varying degrees of abstraction [183]. It helps to express and explore potential designs, and further to validate the architectural design of the systems. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model.

Because of these reasons UML has been used in this work to represent three different views of the proposed framework:

- *Functional requirements view*
- *Static structural view*
- *Dynamic behavior view*

3.2.2 Functional Requirements View

The Key functions of the proposed GRMF are:

- New User Registration
- User Authentication
- Discovery of desired resources
- Specification of desired resources
- Searching the repository
- Execution Services for Job Execution

These functional requirements of the Framework have been analyzed with the help of Use Case Diagrams. Use Case diagrams use two symbols, the actor, which is the entity and the oval which is the use-case. Figure 3.1 shows the Use Case diagram for authentication of the users. Once the user credentials have been verified, then they may discover the resources.

After resource discovery, next step is to monitor the status of the resources and Job submitted to the resources. Users may desire resource optimization/provisioning also. The

Administrator would desire monitoring status information for further using it for fault handling purposes. This is shown in Figure 3.2.

Incase immediate Job submission is not required; the resources may be reserved in advance for future use. If job has been successfully executed, the results would be submitted to the user and payment would be made as shown in Figure 3.3. (Payment can be made through bank, but it has not been included as part of this work).

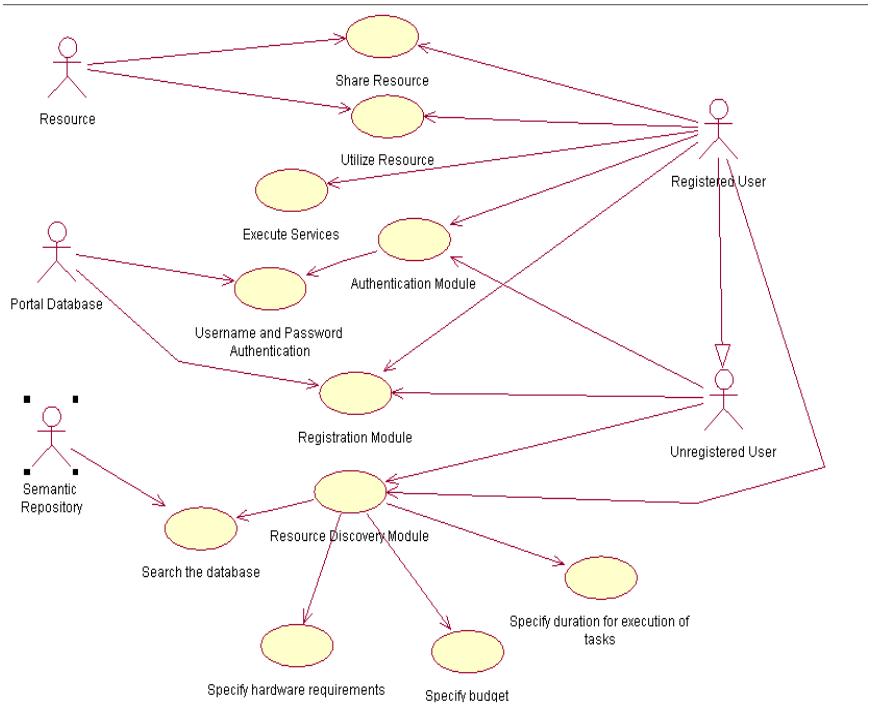

Figure 3.1: Use Case Diagram for user Authentication and Resource Discovery

3.2.3 Static Structural View

The Structural Requirements of the Framework have been analyzed with the help of Class Diagrams, which have been shown in Figure 3.4 and Figure 3.5. Different entities need to be probed from User as well as Administrator perception. These entities are shown as classes (e.g. user, Resource, Job etc.) and the collaborations along with their cardinality has also been shown.

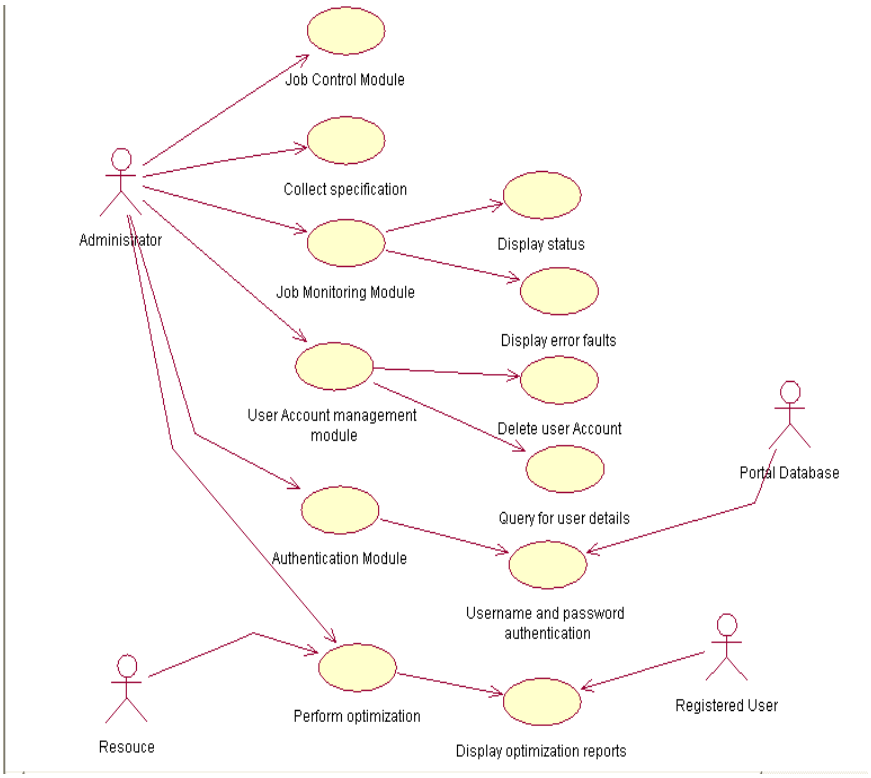

Figure 3.2: Use Case Diagram for Resource Monitoring and Job Status

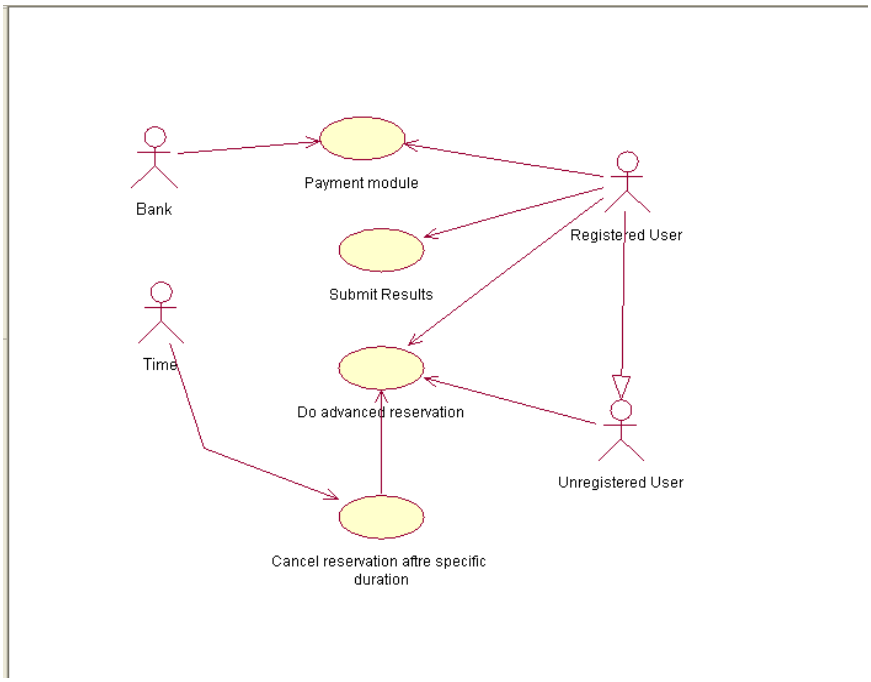

Figure 3.3: Use Case Diagram for Advanced Reservation of Resources and Payment

3.2.4 Dynamic Behavior View

The Dynamic behavior of the Framework has been analyzed with the help of Sequence Diagrams, Activity Diagrams and Collaboration Diagrams, which have been shown as follows:

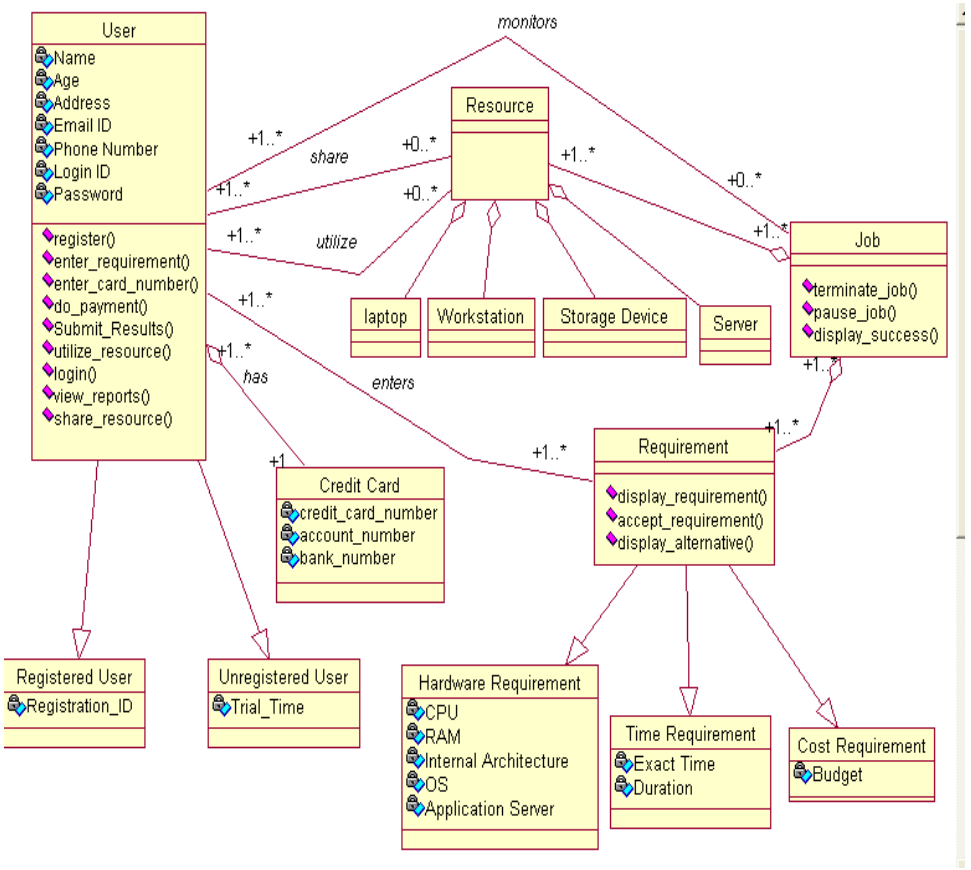

Figure 3.4: Class Diagram for Grid Entities (user)

Activity Diagrams

Figure 3.7 shows the details of Authentication Activity for an existing user. Activity diagrams show the activities in a flow with the standard symbols of flowcharts. Figure 3.8 shows the activities required for registering a new user to the Grid.

Figure 3.9 shows the steps that should be followed for discovering a resource. These steps should be possible only if the user still holds a valid registration. In the Administrator mode, user should follow steps for administrating login and passwords and manage accounts of registered users. After Job submission administrator should be able to monitor the individual resources and the Job status for isolating the faults at the earliest. Figure 3.10 shows the Activity diagram for the activities to be performed by an Administrator.

Sequence Diagram

Sequence diagrams help in analyzing the sequence of all the events as per the timing. The entities are shown at the top. Figure 3.11 shows the sequence of events performed by a user for resource provisioning.

Collaboration Diagram

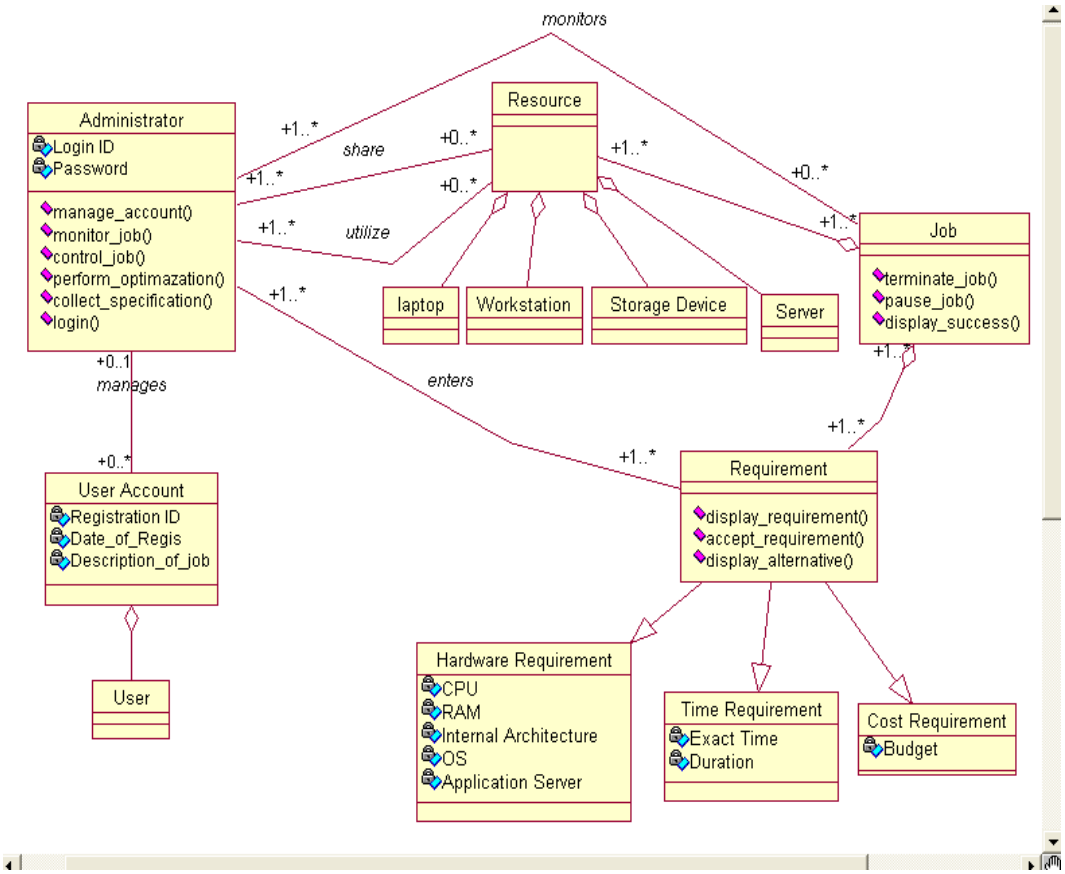

Figure 3.5: Class Diagram for Grid Entities (administrator)

Figure 3.12 shows how different entities would collaborate to perform Resource Discovery and Resource Provisioning.

Deployment Diagram

Deployment Diagram displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units. Finally, the Deployment diagram of the Framework is shown in Figure 3.13 (Rest of the UML diagrams are at the end of the chapter).

On the basis of these UML diagrams the requirements of the Framework have been analyzed and the next step is to design the Framework architecture.

3.3 Framework Architecture

The Framework must cater to the basic needs of the Resource Management in Grid Environment as has been analyzed with the help of UML diagrams. To realize these scenarios

a five-layer OGSA based architecture (OGSA advocates layered Grid architecture) has been proposed for the Framework as shown in Figure 3.6. This section discusses the detailed components of the Framework, layer-wise.

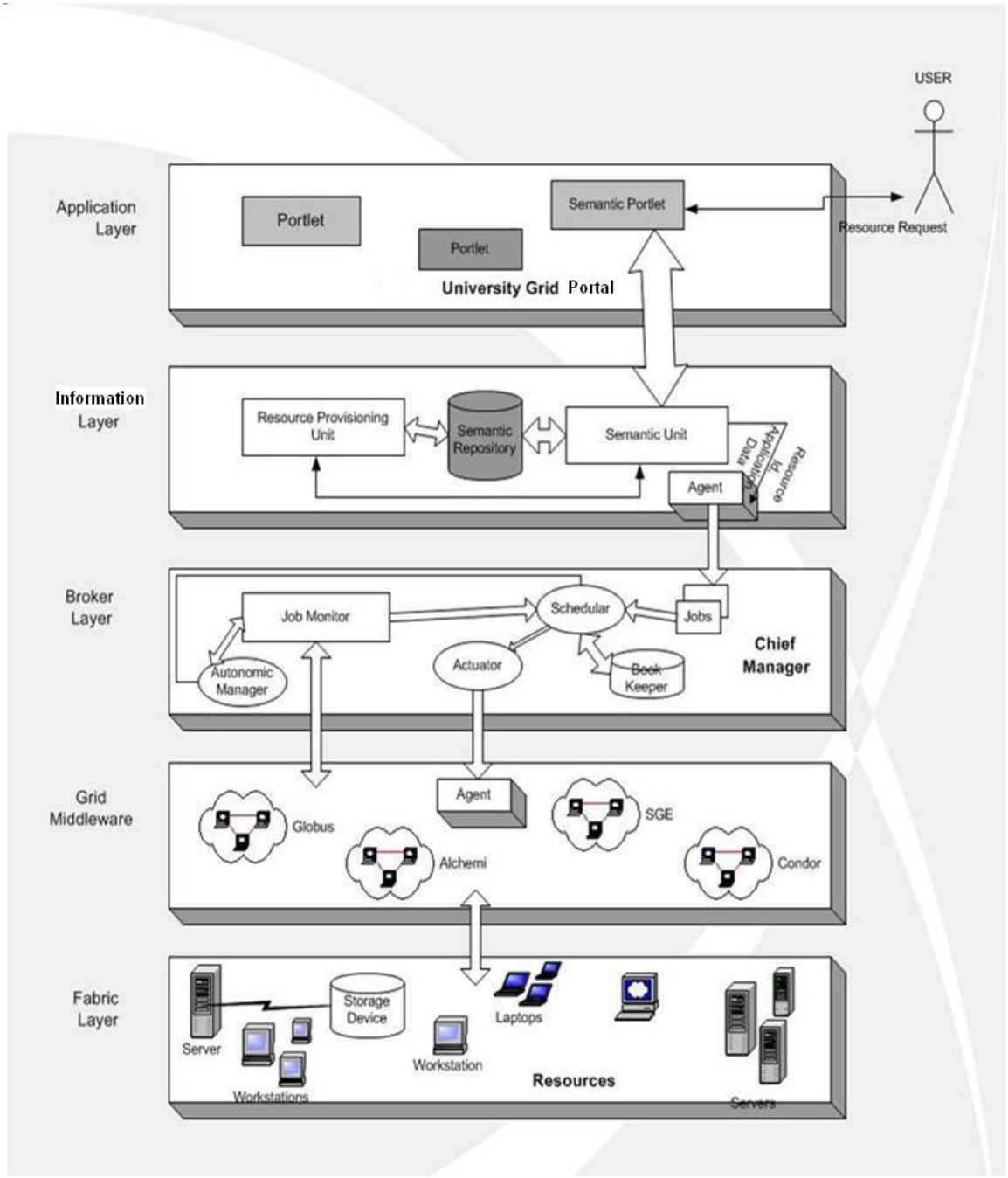

Figure 3.6: PRATHAM Components

3.3.1 Fabric Layer

The base layer forms the fabric of the Grid and is based on the hierarchical model of scheduling and constitutes all the resources, which are present in the form of distributed

clusters. Every cluster would be having its own manager and may have homogenous or heterogeneous resource configurations and may run different middleware. The main components comprising Fabric are -Individual computers, file systems, archives, metadata catalogs, networks, sensors, etc. Thus the individual components forming this layer are the resources.

3.3.2 Grid Middleware Layer

This Layer consists of the middleware being deployed on different clusters. Middleware layer provides a safe and integrated access to the underlying resources. For PRATHAM, Globus, Alchemi.NET, Condor and Sun N1 have been considered. The Managers of different middleware form a part of this layer and perform the local resource management tasks. The resources comprising the Fabric can be accessed through the middleware only. Details of these middleware and their integration with PRATHAM have been discussed in the next chapter.

3.3.3 Broker Layer

Broker layer is required for the interoperability amongst the middleware. Grid Bus Broker has been chosen for interoperability and for scheduling jobs on different middleware. It follows a service-oriented architecture and utilizes the semantic discovery services presented by the Information Layer. The Broker has been designated as the Chief Manager in the framework and the highest controller for scheduling in hierarchy. This layer executes and manages the whole Grid and comprises of the following components:

- Scheduler
- Book Keeper
- Actuator
- Job Monitor
- Autonomic Manager

The Information layer (Second layer) above the Broker layer sends the Job/Application details along with the Resource details to the Scheduler. The scheduler checks the current status of the resources from the Book Keeper. If the desired resources are available, the information is passed on to the Actuator, which creates an agent with the resource and data information and passes it on to the Manager(s) of the middleware (Third Layer) under whom the resources (First Layer) work. On the completion of the Job, the result is passed back to the scheduler. Incase the desired resources are currently busy, then the scheduler keeps the details in its queue and initiates the process as soon as the desired resources become available again.

The Job Monitor keeps track of the individual resources and Jobs. Autonomic Manager utilizes this monitoring data. With the introduction of this Autonomic Manager, all the resources become managed resources as it keeps track of their state or health and takes necessary action itself or informs the administrator in the event of crisis or failure and thus helps in fault management.

3.3.4 Information Layer

Information Layer provides an interface to the Broker layer and incorporates the Semantic and Resource Provisioning units. The main aim of introduction of this layer is to use semantics for interoperability amongst the underlying middleware of the Grid while providing optimum resource allocation for current application execution as well as advanced reservation of resources, if user desires. This layer is domain oriented and usually uses the domain knowledge built with domain ontology. This layer consists of the following components:

- Semantic Unit
- Semantic Repository
- Resource Provisioning Unit

The user requests resources for Job execution. But different middleware store and represent their resources differently. To make resource discovery process flexible and user friendly and for better interoperability of the resources and middleware, Semantics has

been used. The resources forming part of Grid are represented using Ontology. These resource descriptions are stored in the Semantic Repository. The inferences drawn on the basis of Inference Engine are also stored in Semantic Repository. Whenever a user requests for a resource, the request, converted into query, searches the Semantic Repository and availability of resources is checked. Apart from this, Resource provisioning facility is also provided to the user for optimum resource utilization. The Quality of Service (QoS) parameters have been laid down for the Framework. On the basis of these QoS factors, the user further refines his resource preferences and queries the database and correspondingly provisioned resource sets are found. The user may then finalize the set of resources on the basis of Time and Cost Optimization. Incase the user does not want the resources for immediate execution; advanced reservation of resources may also be provided. For Advanced Reservation user may need to enter the time and date of execution. Incase, the user does not confirm his request till two days before the execution time, the reservation stands canceled and the reserved resources are released and entered to the available pool of resources.

3.3.5 Application Layer

The topmost layer is the Application layer, which provides User/Grid integration for the Grid environment and facilitates the use of resources through various collaborations and resource access protocols. In PRATHAM, this interface is the Grid Portal (design and development has been discussed in next chapter), which provides different services to the user through portlets, namely:

- Resource Browser
- Resource Discovery
- Resource Provisioning
- File Transfer
- Resource Monitoring

The User can access the Portal in two modes, as a Guest User or as Grid User. For Guest user, general information regarding the Grid is provided along with the services,

which the Grid offers. Incase the Guest User, wants to access Grid services, then the user needs to authenticate himself by registering with the Grid Portal. Once the credentials are verified, user becomes the Grid User and gets access to the Portal services. The Grid User may request for resources, for resource provisioning and advanced reservation of resources. User may also see the status of submitted job(s).
The Grid Administrator is solely responsible for the successful completion of the Job. Administrator may need to check the status of individual resources, Job status and incase of any failure administrator may need to take appropriate action also.

3.4 Conclusions

This chapter discusses the proposed Grid Resource Management Framework, PRATHAM as an outcome of the research work. Initially, the goals of the framework have been outlined in this chapter. The detailed requirements of resource management framework have been analyzed through UML. Further, the layered architecture and major components of the Framework have been discussed. PRATHAM is based on Open Grid Services Architecture (OGSA) and hence has a layered architecture. In addition to the traditional layers of OGSA, another layer has been introduced i.e. Information Layer, which houses semantic unit for interoperability and also helps in advanced resource reservations and resource provisioning. The Broker Layer also includes Autonomic Manager in addition to the other components and thus renders self-healing through fault and detection system (discussed in detail in next chapter).Thus, PRATHAM provides interoperability, resource provisioning and self-management facilities to the end-user.

In the next chapter the detailed design and implementation of these components is discussed.

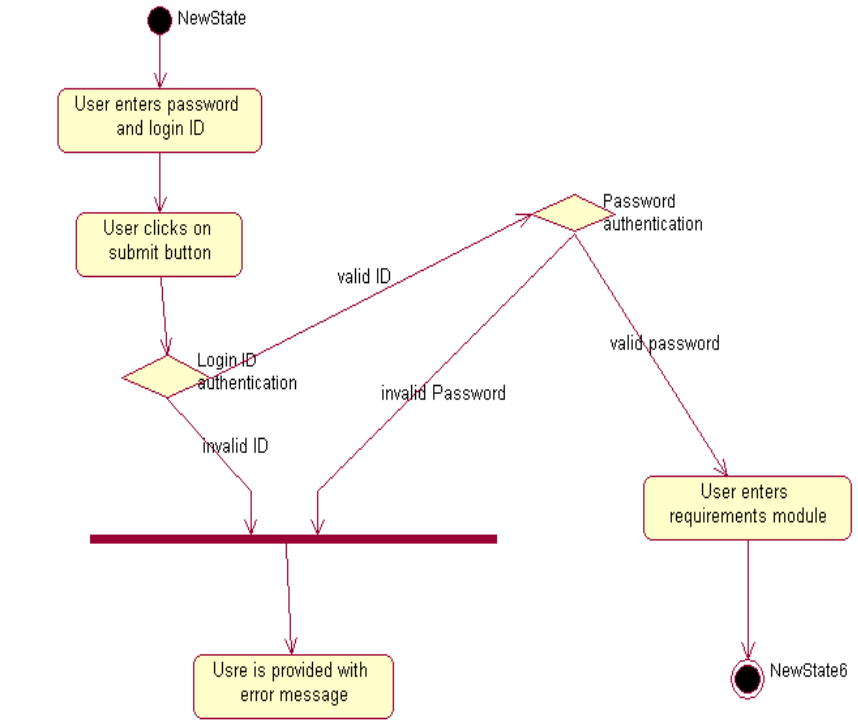

Figure 3.7: Activity Diagram for User Authentication

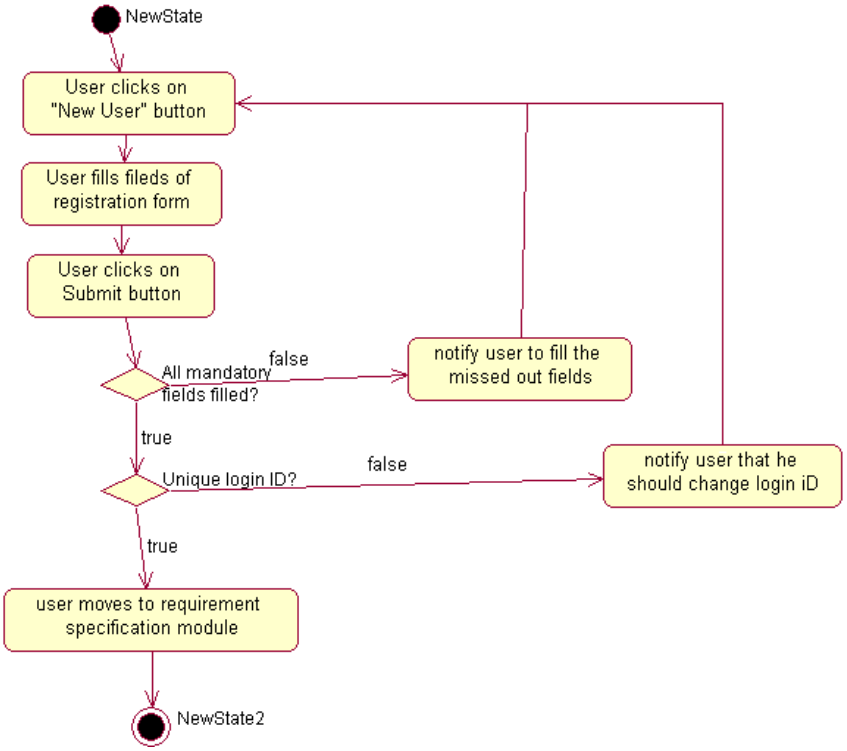

Figure 3.8: Activity Diagram for New User Registration

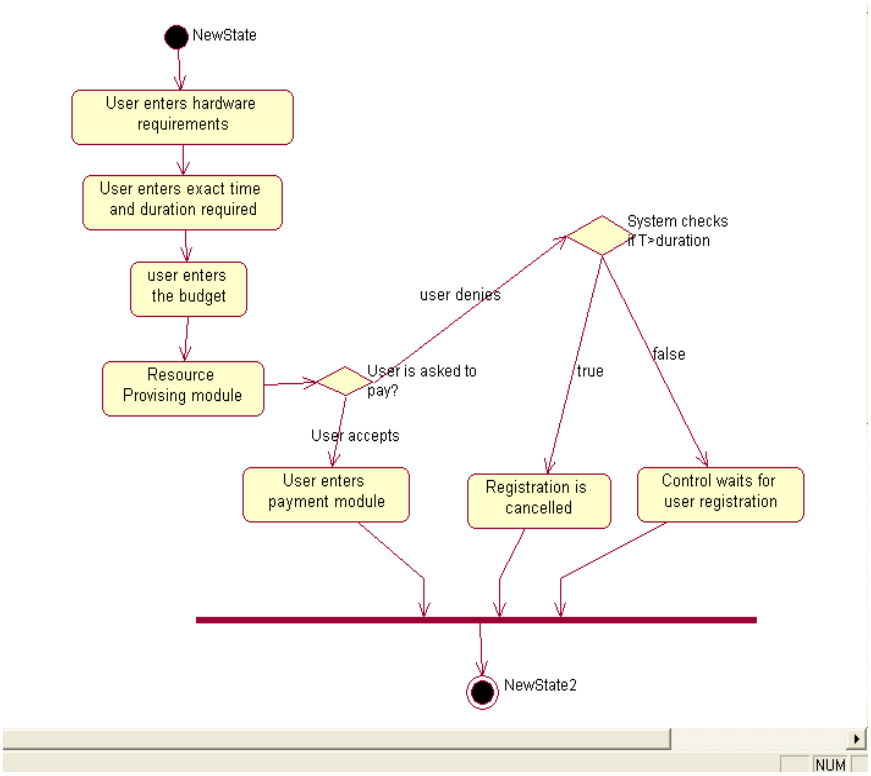

Figure 3.9: Activity Diagram for Resource Discovery

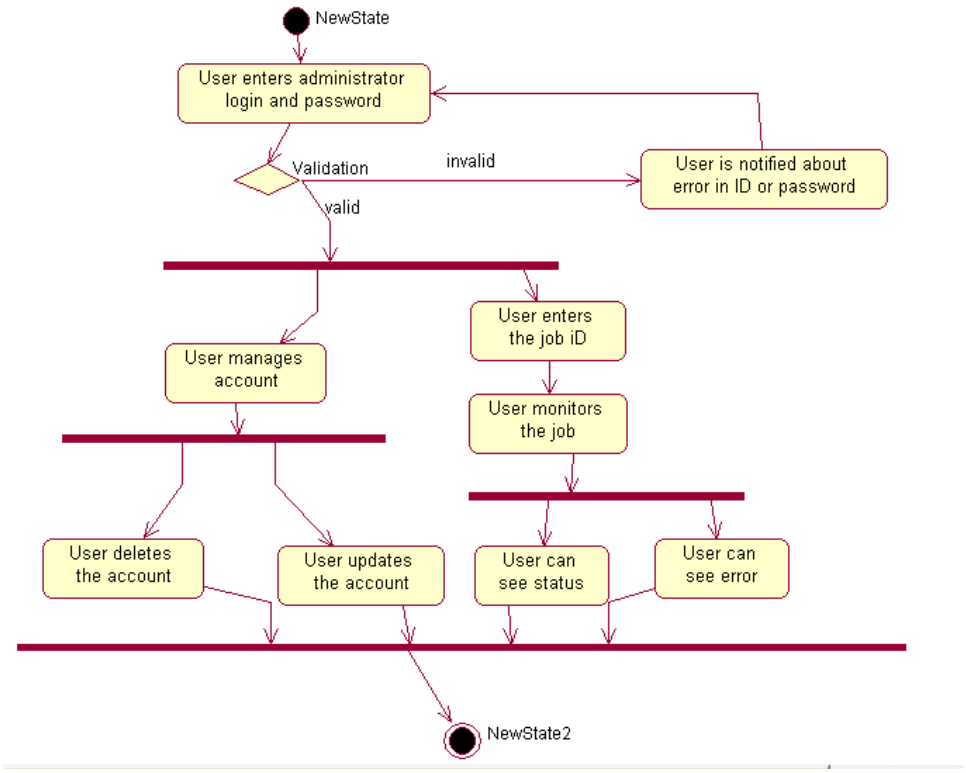

Figure 3.10: Activity Diagram for Administrator

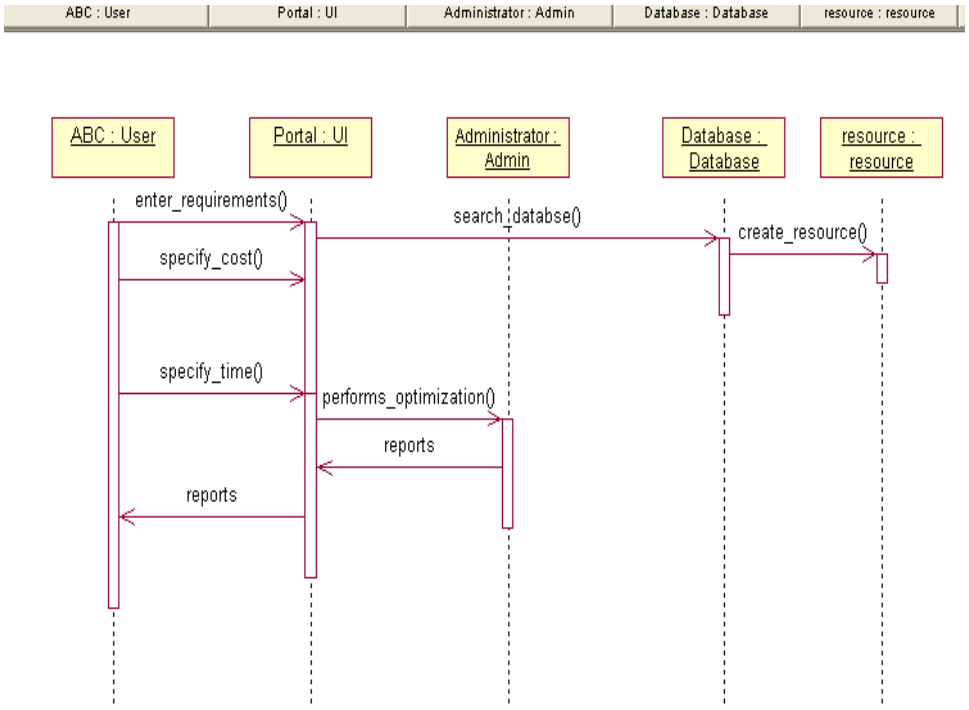

Figure 3.11: Sequence Diagram for Resource Provisioning

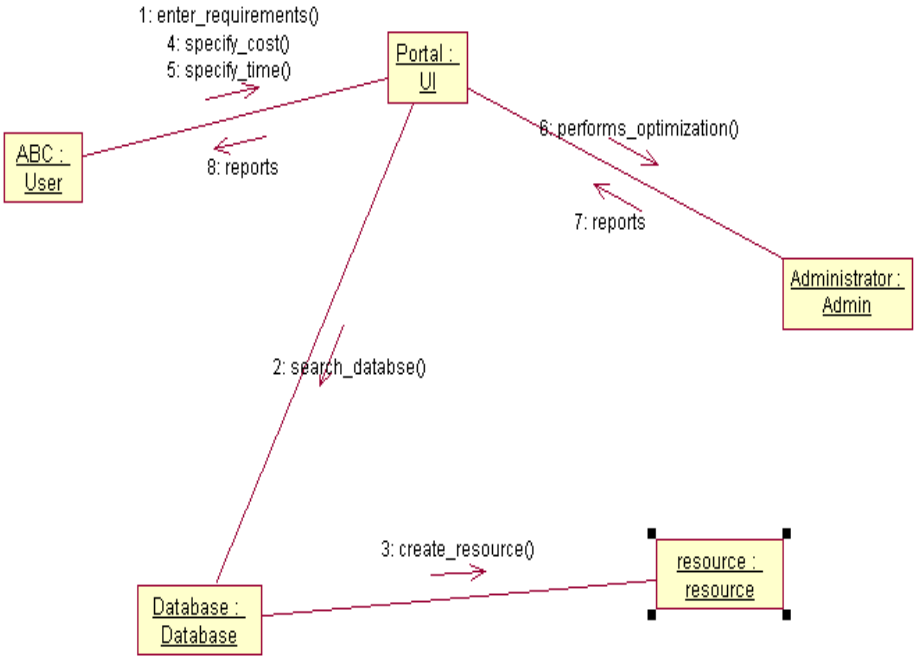

Figure 3.12: Collaboration Diagram for Resource Discovery and Provisioning

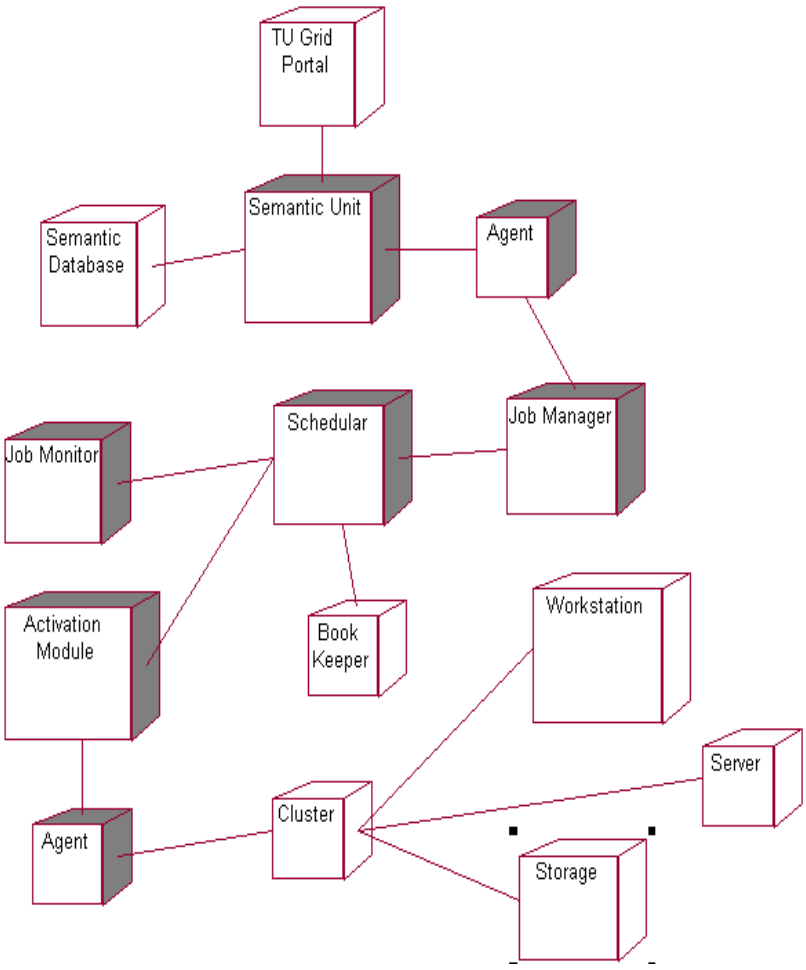

Figure 3.13: Deployment Diagram for the Framework

Chapter 4

Design and Implementation of PRATHAM

This chapter discusses the design and implementation of PRATHAM, the proposed resource management framework. The chapter starts with PRATHAM Taxonomy and describes the framework Workflow. It then moves to the design of components of Information Layer, which implements interoperability and resource provisioning aspects of PRATHAM. For common information storage of resources amongst the middleware, a resource repository has been designed and discussed. This information has been converted into semantic repository for interoperability in resource discovery. After that the Resource provisioning aspect of PRATHAM has been discussed, which is based on Quality of Service (QoS) parameters. Next, details of advanced reservation for better resource provisioning, have been presented.

Further the components of Broker Layer have been detailed which implements self-management aspect of PRATHAM. For providing self-management an efficient monitoring system is required. Based on this system Fault Detection and Healing System (FDHS) has been designed and implemented. Finally, the chapter details about the Application Layer, which has been designed through Grid Portal, which forms an interface for PRATHAM.

4.1 PRATHAM Taxonomy

The taxonomy classifies Resource Management Systems (RMS) by characterizing its different attributes. The taxonomy in this section describes the different aspects of the RMS that provide the interface to the resources managed by the Grid RMS in PRATHAM. Thus, the RMS has been classified according to machine organization within the Grid, resource model, namespace organization, data store organization, resource discovery, QoS support, scheduler organization, scheduling policy, state estimation and rescheduling.

PRATHAM Framework has been proposed for Computational Grids. Namespace organization used for PRATHAM is hierarchical; the naming of resources is done via one or more namespaces that uniquely identify managed resources within the Grid. Jobs use the namespaces to identify the resources that are required to carry out the computation. Quality of Service (QoS) support is soft, which implies it is flexible and tries to accommodate maximum user preferences. The resource and job status information may change frequently which could result in a periodic push of information through the Grid. The addition of new resources to the Grid is infrequent and could be performed either on demand or periodically pulled. Extensible scheduling policy schemes allow external entities the ability to change the scheduling policy. In the ad hoc extensible scheme the resource manager implements a fixed scheduling policy but provides an interface whereby an external agent can change the resulting schedule. The State Estimation of individual resources is heuristics based. The complete PRATHAM Taxonomy has been shown in Table 4.1. The details have been discussed in the subsequent paragraphs.

4.1.1 Resource Model

The resource model determines how applications and the RMS describe Grid resources. Figure 4.1 shows the resource model taxonomy. The taxonomy is focused on how the resources and operations on the resources are described.

PRATHAM has schema based resource model. In a Schema based approach the data concerning the resource is described in a description language along with some integrity constraints. The choice of data model typically results in using a network directory information store organization, as there can be an efficient mapping of the logical content

Table 4.1: PRATHAM Taxonomy

Attributes of Resource Management Framework	***Taxonomy***
Grid Type (Service focus)	*Computational Grids*
Machine Organization	*Hierarchical*
Resource Model	*Schema based*
Namespace Organization	*Hierarchical*
QoS	*Soft*
Resource Information Store	*Network Directory*
Resource Discovery	*Query*
Resource Info Dissemination	*Batch/Period (push or pull)*
Scheduler Organization	*Hierarchical*
Scheduling Policy	*System-Centric*
State Estimation	*Predictive (Heuristics, machine learning)*
Rescheduling	*Event Driven*

of the resource information store to the physical implementation.

In Operational model scheme the operations on the resources are defined as part of the resource model. As with Data model the Operational model can be predetermined and fixed as part of the definition of the RMS. In the Operational model extensible approach the resource model provides a mechanism to extend the definition of the Operational model managed by the RMS.

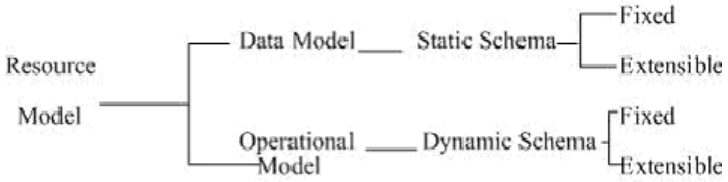

Figure 4.1: Resource Model Taxonomy

4.1.2 Scheduling Model

PRATHAM follows a hierarchical scheduling organization; the scheduling controllers are organized in a hierarchy. The higher-level controllers manage larger sets of resources and lower level controllers manage smaller sets of resources. Compared with the centralized scheduling this mode of hierarchical scheduling addresses the scalability and fault-tolerance issues in a better way for Grid environments.

4.2 PRATHAM Workflow

Figure 4.2 shows the workflow of PRATHAM. PRATHAM considers a multi-Middleware scenario. The origin of resource data is the Grid Middleware. This data is collected in Resource Information Collector and placed at a central location. This resource data is further transformed into Web Ontology Language (OWL) and stored in the Semantic Repository. With the semantic translation of the complete data a uniform description of the resources is possible, which enhances the interoperability amongst different middleware.

In PRATHAM, the resource is being discovered semantically. The end user simply inputs his requirements and the RMS takes the responsibility of searching the resources from the entire Grid. The user is facilitated with resource provisioning and with the features of advanced reservation also. Once the resources have been decided the resource information along with the job details are passed on to the broker, which takes the responsibility of scheduling the resources. The Broker contacts Middleware managers and they in turn get the job done (as shown in Figure 3.14 of chapter 3). Once the job has

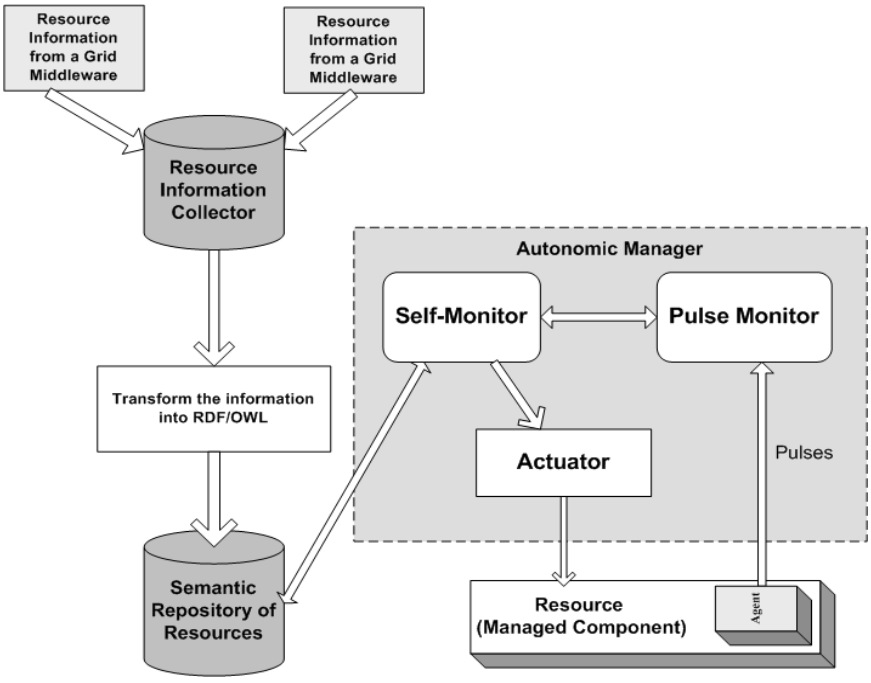

Figure 4.2: PRATHAM Workflow

been submitted the next step is to monitor the status of both, the resource as well as the job. In PRATHAM, Autonomic Manager monitors and performs self-management of the Grid. The Autonomic Manager utilizes the data in semantic repository for this

purposes and it also utilizes the metadata of each resource for predictions. Every new resource, which enters the Grid, needs to register with the Virtual Organization (VO). At the time of registration, an agent is loaded on to the resource to make it a managed component. This agent keeps on sending pulses to a Pulse Monitor in the Autonomic Manager. The Self-Monitor reads and analyzes the pulse data and if any action needs to be taken the Actuator is activated and acts accordingly. Details of Semantic Unit and Autonomic Manager would be covered in the subsequent sections.

These sections discussed about the overall taxonomy and workflow of the proposed framework. Forthcoming sections will explain the components of top three layers, namely, Application, Information and Broker Layers. Bottom two layers, Fabric and Middleware are the standard layers of OGSA architecture, hence their design and implementation details have not been discussed.

4.3 Interoperability in PRATHAM

This section describes the design and implementation of Semantic Unit and Semantic Repository along with other components of Information Layer, which renders interoperabilty to PRATHAM. For interoperability, the resources must be stored and accessed on a uniform basis. In PRATHAM, a resource repository has been created which collects data from different middleware sources and then it is mapped to semantic repository. GridBus Broker provides interoperability among different middleware but there is no support for semantics in the broker. It has a directory structure called Grid Market Directory (GMD) for discovery of services, which uses a keyword-based serial search technique to locate a service or resource. For better interoperability, a semantic-based search technique is required for discovering grid services semantically. Therefore, in PRATHAM resource discovery component of Grid Bus Broker has been over-ridden by the semantic unit. Thus, the second layer of the Framework i.e. the Information Layer renders interoperability to PRATHAM and helps in resource discovery. This section would be discussing the design and implementation of the components of this layer in detail.

4.3.1 Resource Repository

The primary step in resource discovery is to have resource information in hand i.e. to establish Resource Information Collector. As Figure 4.2 depicts, the resource information from different middleware is collated and then transformed and stored in the semantic repository. So the first step is to assemble the resource data from different middleware. In PRATHAM, for implementation purposes, Globus and Alchemi.Net have been chosen. But the approach can be extended to other middleware like Sun Grid Engine also.

Individual Grid middleware have their own repository to store the valuable information about their cluster, like Globus uses MyProxy, which is an online credential repository. Storing Grid credentials in a MyProxy repository allows administrators to retrieve a credential whenever and wherever they need one, without the need to manage private key and certificate files. Where as, Alchemi.NET uses SQL server to store information about its cluster. Because of these differences, interoperability in a VO is a major issue especially in multi-middleware scenario.

Globus displays the information of its resources through web MDS. An individual XML file is created for each client connected to server. Further more this file can be fetched either from server or from client itself. By using WSRF query a XML can be generated and this XML file can be parsed by a XML parser program. This XML parser program reads the generated XML file and produces the content of selected tags. Tags can be selected as per requirement/priority. Alchemi.NET manager has all the information of its resources, which is stored in SQL Server. Using SQL query, data can be fetched from Alchemi.NET's database and then by update query this fetched data can be stored in designated repository. PRATHAM uses the following design approach:

(Resource repository has been designed on Windows platform.)

(a) Gather XML files of all clients on server

(b) Parse all the XML files on the server itself

(c) Store the data in the resource repository.

4.3.2 Implementation Details

System architecture for development of Resource Repository for PRATHAM is shown in Figure 4.3 Architecture shows various components involved in development of Resource Repository and the relation between these components. It consists of set of clusters, XML Converter and Transporter, Repository Update Module and Resource Repository. The repository architecture has been implemented through Globus and Alchemi.NET. XML Converter and Transporter module fetch information of Globus cluster. This module transfers retrieved XML file to Repository Update Module. Repository Update Module parses and stores the information about Globus into Resource Repository. Alchemi.NET uses SQL server for storing cluster information. Repository Update Module stores the information about Alchemi.NET's cluster into Resource Repository by running SQL queries. The main components of resource repository architecture are:

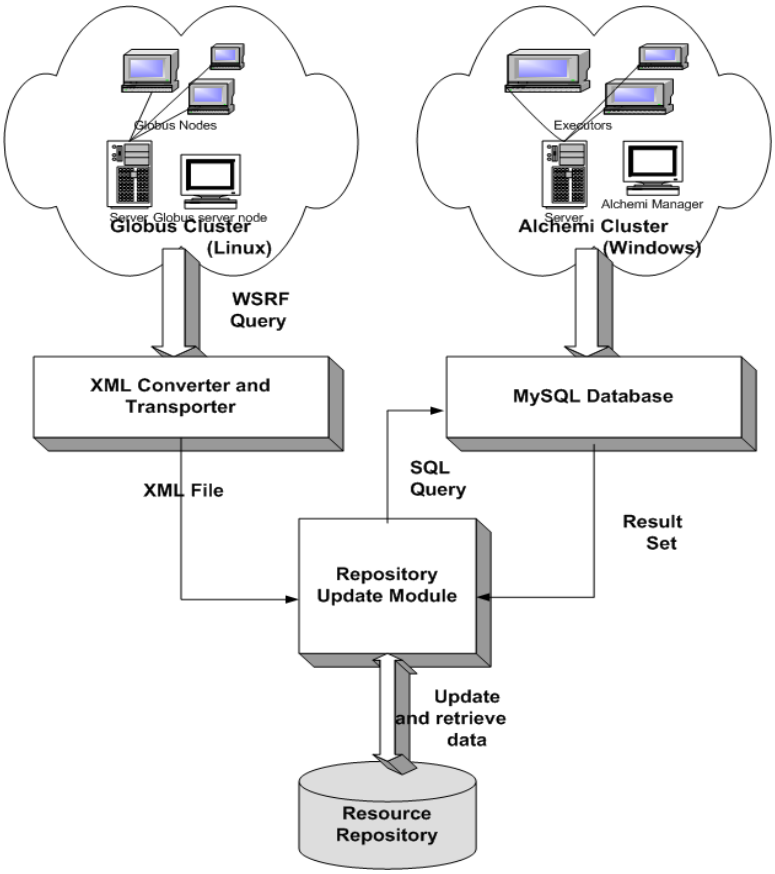

Figure 4.3: Components of Resource Repository Architecture

(a) Middleware

Two middleware have been considered for the implementation of this architecture. One set is of Globus, which has a Globus Server associated with its clients. Server has valu-

able information about its clients. Globus uses a web-based tool Web MDS for browsing XML-based information (i.e. resource properties) about Globus cluster. Other cluster is of Alchemi.NET. Alchemi Manager is connected to its executors. Alchemi Manager uses SQL Server to store the information about its resources.

(b) XML Converter and Transporter

This Module works in two steps:

In step 1 module runs a WSRF Query, which gives a corresponding XML that has the valuable information about Globus cluster (i.e. resource information). This step repeats itself after frequent intervals.

In step 2 module transfers the XML file to a Repository Update Module; this step repeats itself after predefined intervals.

(c) Repository Update Module

Repository Update Module performs following tasks: Parsing XML file, Querying on SQL Server to fetch information about Alchemi.NET's cluster, and populate the relevant information into proposed repository. This module can be used to update the Repository by entering details of resources manually.

(d) Resource Repository

This is the proposed repository, which has various attributes. It has separate data tables for Globus and Alchemi.NET. It automatically stores and fetches the data on the basis of middleware selected.

Thus the resource information of the entire Grid is present in this repository centrally and becomes the metadata for the resource discovery process and it can be further utilized for creating semantic repository.

This section described the design and implementation of resource Information Collector (Resource Repository). The next section discusses about Semantic Repository and Semantic Unit, the major components of Information Layer.

4.3.3 Resource Discovery through Semantics

In order to improve the search of resources and its selection, ontology [166] has been chosen as an alternative approach and for enhancing interoperability. This paradigm can

provide a description of available resources in a grid configuration, leading users to desire operations and describing resources' syntax and semantics that can form common domain vocabulary. Traditional resource discovery methods strive for symmetric syntactic description of resource and request properties, so it is difficult to introduce new concepts or characteristics into the system. Solution to this problem is provided by Ontology i.e. adding semantic annotation to Grid resources. Ontology provides unambiguous, asymmetric resource description that can be easily shared across various VO. In this framework Ontology has been used for resource discovery and has been embedded in Information Layer on top of Broker Layer. Thus, the Information layer is handling resource discovery and match making and the Resource Broker is handling Job scheduling. Broker is the Chief Manager in this framework and it assigns the jobs to the underlying clusters managed by individual Middleware managers (as shown in Figure 3.14 of Chapter 3).

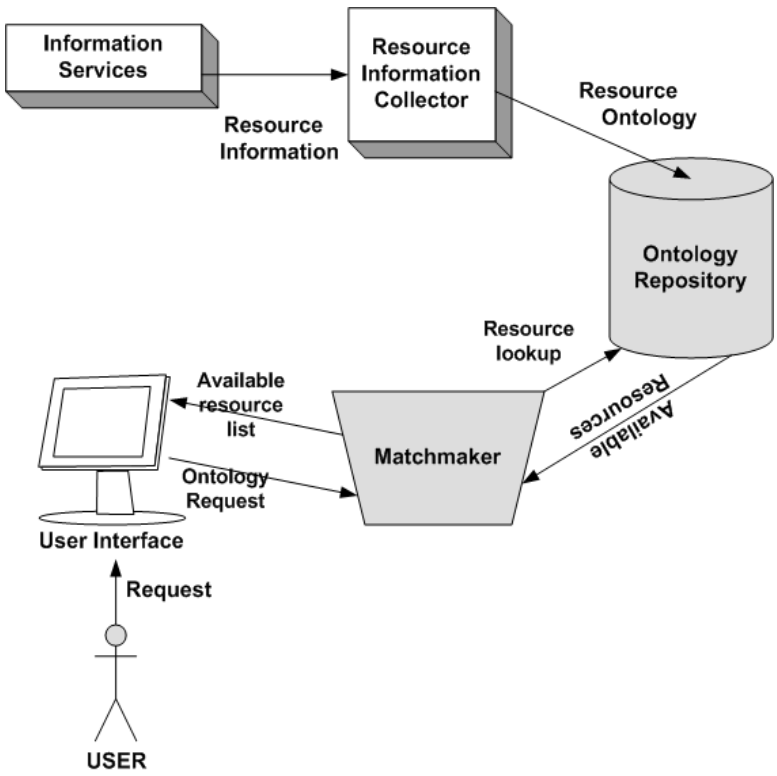

Figure 4.4: Ontology based Grid Resource Discovery Approach

The workflow of resource discovery approach has been shown in Figure 4.4. The basic components of resource discovery approach are:

- Information services of Grid Systems (Provides information of resources, part of middleware)

- Information Collector (Common repository discussed in previous section)
- Ontology Repository (Resource Information is transformed and stored as OWL)
- User Interface (For user inputs)

Resource discovery process starts with submission of user requirements. User enters requirements like CPU Architecture, CPU Availability, memory requirement, Operating System of target machine and other relevant information at the User Interface (Grid Portal). Query is formulated on the basis of above information and is send to the Semantic Unit. The Semantic Unit finds out classes of instances provided in user query e.g. 'Pentium 4' is instance of Intel Class. The second step is to find out all the equivalent classes of given class, if any. In this step, all respective instance class and equivalent classes are retrieved; the third step is to find all the instances of given class and equivalent class. Result of this step provides a list of instances against a single instance. This list of instances is called as Inferred instance or Result Set. Then the list of instances is searched in dynamic Ontology Repository (semantic repository) which is updated at desired frequency by Information Collector.

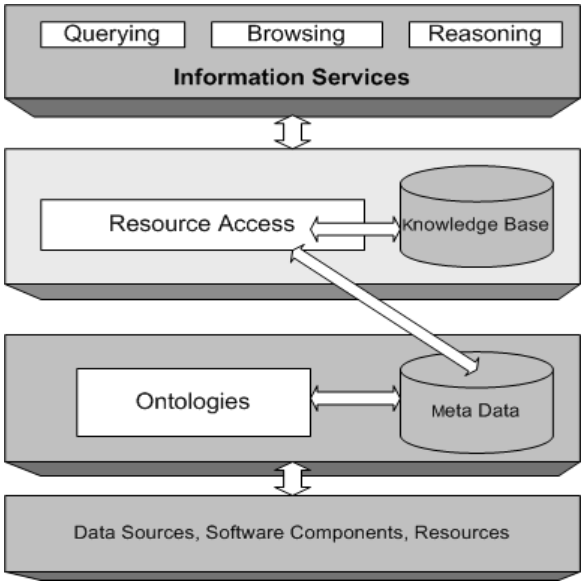

Figure 4.5: Layers in a typical Grid Information System

Reference architecture for such Grid information system where Ontologies are used to describe resources and refer to their metadata to allow resource access is depicted in Figure. 4.5. In particular there would be following layers:

Resource and Repository Layers: These layers (bottom two layers) contain concrete Grid resources, such as data sources and software components along with metadata of resources, describing details of each concrete resource on each Grid node (This information was discussed in section 4.3.1 and 4.3.2), and Ontologies that describe semantic properties of resources and semantic relations.

Resource Access Layer: This layer allows general primitives to access heterogeneous resources i.e. the same functions are used to access a text document or a relational database. The Resource access can be directly implemented by applications using metadata.

Information Services Layer: This layer offers a set of functions to query and browse resources, approach to describe resources and to publish such information in both metadata and ontologies, and finally functions to infer new knowledge by reasoning or mining.

A vocabulary using the component axiom, which represents a key element to ensure all the ontology requirements has been designed for PRATHAM. In addition to this component, other two structures help the ontology in the description of resources:

- Metadata: Reflects the information related to a data. In PRATHAM, the metadata stores information about the computational resources, for examples : resource id, group id, capacity of storage, available memory, existing operating system etc. and the details of this have been discussed in section 4.3.1 and 4.3.2.

- Semantics view: This structure stores information related to the present state of a computational resource and their relationships. Thus, every time that a request comes to a semantic view, the structure returns information about that moment. In other words, a semantic view replies if the resource is available for use, out-of-work or heavily loaded between others status.

Metadata and semantics view are both used as additional references to ontology. Even not commonly used in other ontologies, these structures can improve the ontology action, returning answers more quickly.

4.3.4 Implementation Details

For implementing Semantic Unit and Semantic Repository, the ontology is implemented using the OWL Language (Web Ontology Language) and was edited using the Protege-

2000 software package (This editor was the obvious choice as it is freely available and supports several operations of Ontology Language). OWL is used as a standard by the W3C. It has been designed for use by applications that need to process the content of information instead of only presenting information to humans. In this environment, concepts, attributes and relationships of the ontology systems have been described. Protege facilitates the creation of ontology and to build knowledgebase. Jena inference engine has been used to interact with the knowledgebase for semantic retrieval of information as it is having direct compatibility with Protege-OWL. It also maps the central repository to semantic repository

Design of Grid Ontology

Firstly, set of Grid Ontology classes have been defined. Subclasses have then been added accordingly. After that, the relationships and constraints among the Ontology classes using Object and Data type properties have been defined. These properties form links among the defined classes. Finally, the classes, properties, and instances together form a Knowledge base (shown in Layer 3 of Figure 4.5) that captures the configuration and state of specific Grid infrastructures. Such a Knowledge base can be used for various inquiries and for decision-support of end-users, application developers, Grid administrators, etc.

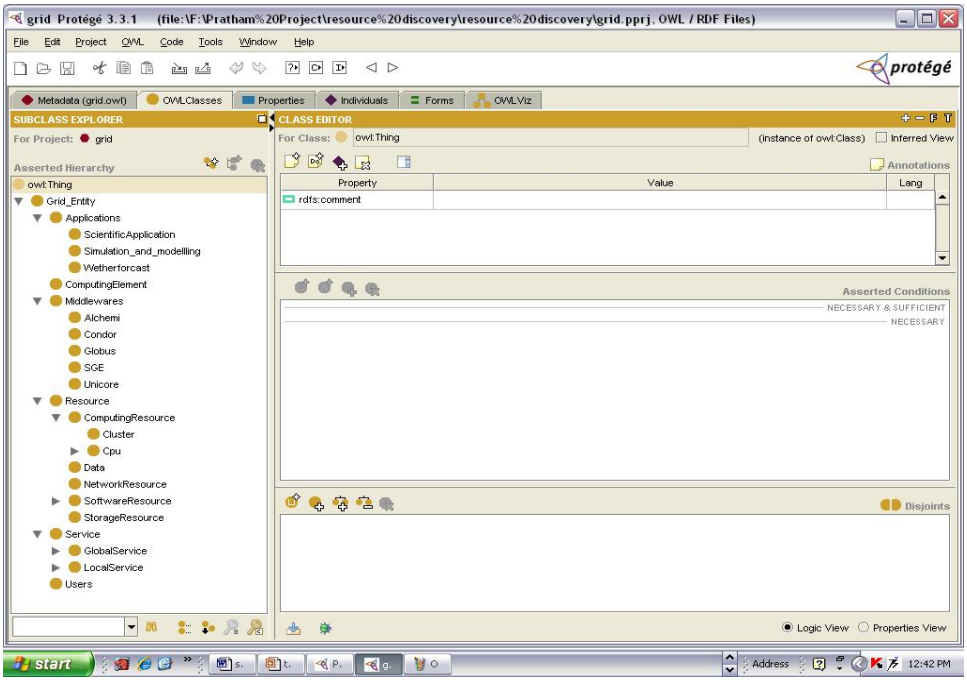

Figure 4.6: Grid Ontology Classes in Protege

(a) The Grid Ontology Classes

In Grid Ontology some core classes have been defined, and its subclasses and description of Top-level classes (Figure. 4.6) have been given below:

Resource: This class describes Hardware, Software, computing resource, storage resource and network resource within a Grid. This class is further classified into subclasses. Its subclasses are SoftwareResource that includes classes of Operating Systems, ComputingResource that contains cluster, CPU as subclass etc.

Middleware: This class encapsulates Software that glues Grid services together following a specific Grid architecture. This class is further categorized into subclasses of different Middleware Condor, Globus, Alchemi and SGE.

Users: This class describes types of users who use the Grid system.

Applications: This class encapsulates Applications that can run on Grids.

Service: This class encapsulates services that can provide one or more Grid functionalities. This class is further categorized in LocalService class and GlobalService class. LocalService has one subclass JobSchedular and GlobalService has subclasses broker and InformationService.

Computing Element: This class encapsulates Grid entity and has several kinds of computing resources, such as CPU, Memory etc. The resources are organized by Grid Middleware, user and Grid services to provide computing power to respective domains.

The documentation required to build the ontology, consists of all concepts, described in a formal way that will form the ontology. The following components characterize the proposal:

- Data Dictionary: gathers all the classes and instances from the ontology, together with their meanings. At the end of the ontology creation, the Data Dictionary presented a whole of 15 classes. The first class created is Grid_Entity that was the main class of the ontology, all other classes were created as its subclasses;

- Concept Classification Tree: this component comprises of all the classes and subclasses of the ontology. The tree presents all ontology classes and subclasses. Figure 4.7 shows upper part of the Concept Classification Tree. Some more subclasses/instances (though designed) have not been shown in the figure.

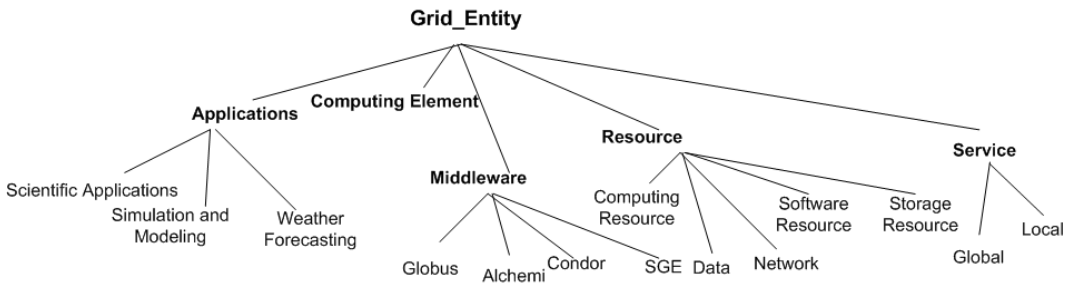

Figure 4.7: Concept Classification Tree for PRATHAM

(b) The Grid Ontology Properties

In order to represent the relationships and constrains among the Ontology classes, properties have been defined that provide the semantic meaning for the Grid Ontology entities. In Grid Ontology two types of properties are defined:

- Object Property: Object Property is relationship between instances of two Classes.
- Datatype Property: Datatype Property is relationship between instances of Classes and RDF literals and XML Schema Datatypes.

In PRATHAM these properties have been designed as hasCpu, hasOs etc. corresponding to the resources:

Class Properties

For each class and its instances the property, cardinality and type has been defined. An example is illustrated in Figure 4.8, where the structure of attributes related to Computing Element; subclass from Grid_Entity class can be seen; The complete detail of the OWL file created by Protege is stored as URI. The URI (generated by Jena) for PRATHAM is www.owl-ontologies.com/grid.owl#Grid_Entity .

Thus the first stage of creating ontology pertaining to the Grid is complete. Next step is to create a semantic repository with this ontology as its schema. For creating semantic repository, a tool Jena [177] has been used, which is a Java API for RDF [165]. The URI for PRATHAM can be accessed by JENA. RDF has XML syntax, which should be understood in terms of its data model. Resource can be anything whatever can be identified. Resources have properties. Each property has a value known as literal. Jena has object classes to represent graphs, resources, properties and literals. In Jena a graph is called a model and is represented by Model Interface.

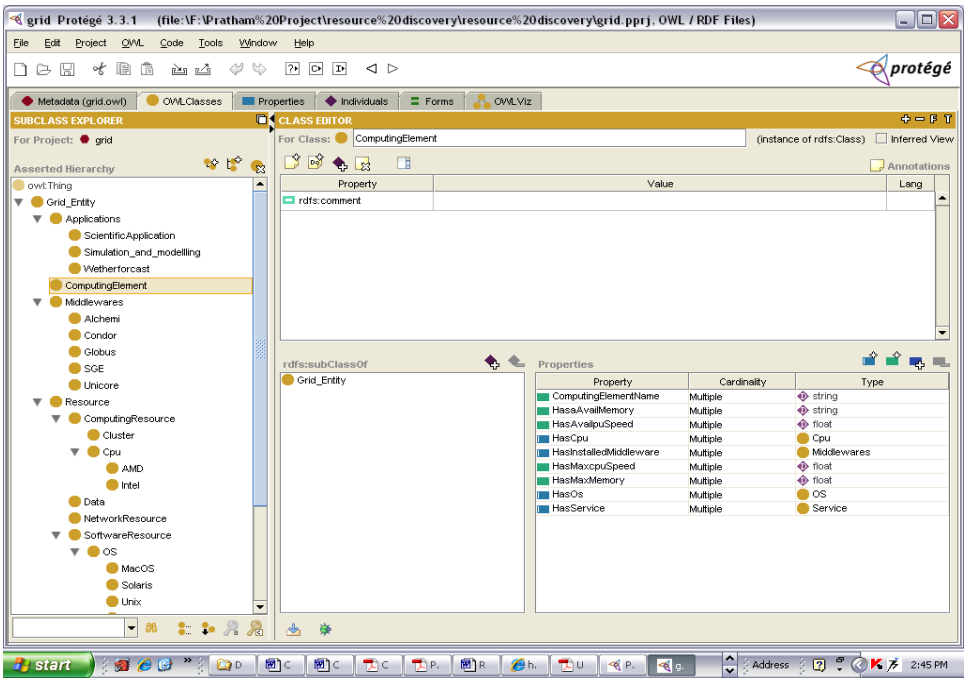

Figure 4.8: Properties of Classes

Jena loads ontologies and instance data in MySQL database. Then inference model is created to store Class properties. These properties then configure search interface. The steps followed have been depicted in Figure 4.9. PRATHAM RDF, List of Statements and Inferences generated by Jena have been appended in Appendix A.

Jena supports SPARQL [178], for querying the semantic repository, as shown in following example: Syntax of SPARQL Query in Jena:

String sparql = “PREFIX rdf:” + “PREFIX rdfs: ” + “SELECT ?class ?label ” +

“WHERE ?class rdf:type rdfs:Class . ” +“?class rdfs:label ?label ”;

ResultSet results =

QueryExecutionFactory.create(QueryFactory.create(sparql),ds).execSelect();

new ResultSetFormatter(results).printAll(System.out);

Thus the search results can be obtained by writing direct SPARQL queries or through Jena programs. The complete set of steps has been shown in Figure 4.10.

This section discussed the Semantic Unit and Semantic Repository, the two components of Information Layer. Next section discusses about the third component, Resource Provisioning, for PRATHAM.

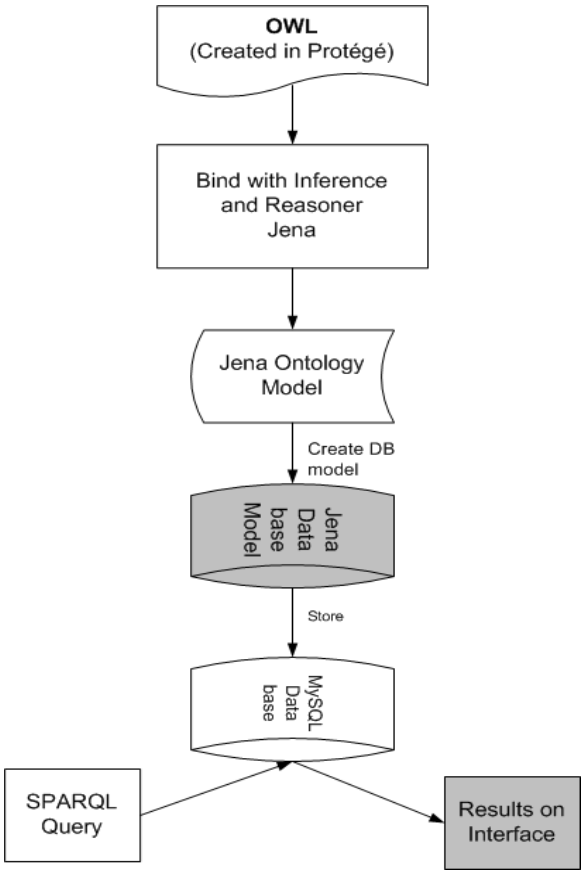

Figure 4.9: Steps followed in searching a Resource Semantically

4.4 Resource Provisioning in PRATHAM

Resource Discovery is the primary step of Resource Management System. Further, a user can be given the facility of provisioning for optimum results and better services i.e. Service level guarantees can be offered through provisioning. PRATHAM Framework offers Resource Provisioning services to the end user as per the demand environment, resources are allocated to services, as needed. An initial set of resources is provisioned for a service, and static resource provisioning has also been offered through advanced reservations. One aspect of QoS is the ability for a Grid application to negotiate a Service Level Agreement (SLA) with the RMS. In some cases an application can accept lower overall performance or an inexact match between a specified resource and the actual resource supplied. This idea of general QoS or SLA can be extended to placing bounds on resource discovery or other grid provided services such as data migration. Thus, for provisioning firstly, QoS parameters need to be defined and provided to the end user. Next section discusses about QoS requirements of PRATHAM.

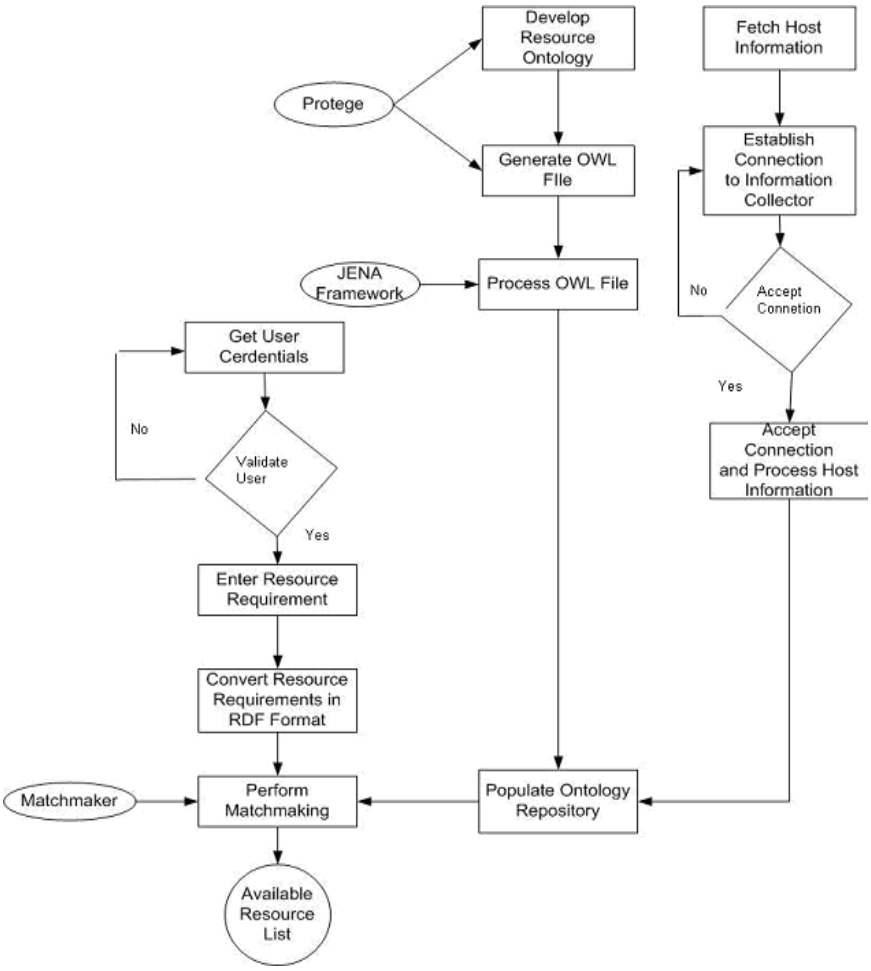

Figure 4.10: Detailed Information Flow Diagram of Ontology based Resource Discovery

4.4.1 Quality of Service (QoS) Requirements

The Resource Management System (RMS) should predict the impact that an application's request will have on the overall resource pool and QoS guarantees already given to other applications. This prediction can be done either by using past behavior of the application on similar input values or by using heuristics. In order to provide QoS, a RMS must be able to perform admission control and policing. Admission control is the process of determining if a resource request can be honored. Policing is ensuring the contract for resource utilization agreed upon between the RMS and the requesting application has not exceeded. PRATHAM has been designed based on past data prediction i.e. heuristics. This past data is created and stored in the resource repository based on the following methodology.

Data Generation

Following information is generated during the lifetime of the application. This informa-

tion needs to be stored either in Data model or Job model (shown as Operational Model in Resource Model) for better management and provisioning:
Static - Information is maintained by logging and bookkeeping services which needs:

- Identification of user and Job: For unique identification of Jobs and Users, Identification codes are required, Job Id, User Id, and User Password for complete Authentication and Authorization checks.
- Job/Application Size: Job/Application size is calculated in Kilo Lines of Code (KLOC) or Function Points (FP). It can be input before Job submission or can be calculated approximately on the basis of historical data of similar jobs.
- Time: Approximate duration along with start and end date of Job.
- List: Initially, list of vital resources is required for allocation.

Dynamic - Following information needs to be stored and updated dynamically:

- Total number of running jobs with percentage progress.
- Input/Output data required and produced
- Job start time
- Progress in total (%) and resource wise job completion.
- Resource Prediction for the leftover part (esp. useful for lengthy jobs which may take days).

For a resource model, monitoring information is of utmost importance. A monitoring service is required at the resource level to report about:

- Status of specific type of Grid resource or application
- To produce and process, frequently changing information, dynamic information collected from Grid subsystems or applications.
- Provide filtering and statistical processing module that produce and publish "Summary" information about monitored resources.

Table 4.2: Resource Database Schema

Information	***Units/Value***
Resource ID	***ID code R_ID 1...R_ID n***
Resource Name	***Hardware, Software, Operating System etc***
Type of Fault	***Error detail***
Date & Time of Fault	***mm-dd-yyyy and hh:mm***
Downtime	***Duration in hh:mm***
Remedy	***Action taken***

Usage Scenarios of Information

There are many uses for monitoring data generated and stored as part of Job Model within a Grid environment, which can be effectively used for better management and resource provisioning. These include:

a) Prediction and detection of faults: At the resource level, monitoring is required to be done to check whether the resource is live or not. Sometimes the resources may be live but the network may be having problem. For this a Knowledgebase has been created with the following schema shown in Table 4.2:

Now this Knowledgebase is being queried every time the monitoring system reports resource is not alive for predicting the possible faults and their remedies. This considerably reduces the time of manual detection of faults, as past history becomes handy. This process is being further utilized for better management.

b) Reliability Index (RI): A Reliability Index (RI) is calculated for each individual resource on the basis of data provided by Table 4.2. Reliability [111] value lies between 0 and 1. Grid normally comprises of several clusters, RI for the cluster can also be calculated similarly. This data can be further categorized on the basis of Hardware configuration and/or Software configuration. The Index provides valuable information on the expected costs of individual resource, how frequently it might go wrong and which components can be expected to go wrong. MTBF (mean time between failures) is a measure of how reliable a hardware product or component is and has been used for calculating Reliability Index based on statistical data. For most components, the measure is typically in thousands or even tens of thousands of hours between failures. For example, a hard disk drive may have a mean time between failures of 300,000 hours. It gives an idea of how much service to plan for. For a computer system, a simple measure of

reliability is MTBF, and it can be calculated as [167]:

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

Where MTBF stands for Mean Time Between Failures

MTTF stands for Mean Time To Failure

MTTR stands for Mean Time To Repair

Further, based on this data, Reliability Index, RI, is calculated.

c) Health check of resources: A resource or a cluster can be checked for performance based on data collected in Table 4.2. If a resource is frequently down then its performance might be degrading. Obsolete components with limited capacity, which might need change, might be generating unsatisfactory throughput. This data can be queried and analyzed to find out the performance levels of individual resource/ services for optimum, consistent, degrading performance levels and correspondingly Performance Index, PI has also been calculated.

d) Future resource-use predictions: In future, how many resources can be put to use is predicted on the basis of this data. If optimum usage is required the set of resources, which is giving optimum performance, can be chosen. In case of very sensitive data, only highly reliable systems need to be chosen. This Knowledgebase can be used for such predictions and correspondingly better sets of resources can be matched to the user requests.

e) Report for Self-management: With the Knowledgebase, maintained as said above, each individual resource and the corresponding clusters, of which it forms a part, is being managed in a better way. This information can be utilized, along with the other traditional techniques, which have been used for self-management. Two basic requirements for implementing this information for better management are, efficient monitoring technique and a Knowledgebase. The initial cost would be the overhead, but the results will considerably reduce the service downtime, preventive management is possible, optimal matching of resource and requests has also been achieved.

Based on these inputs, following QoS parameters have been identified for resource provisioning in PRATHAM:

- Required Configuration

- Point of time of availability of resources.
- Duration of availability
- Desired level of performance and performance level offered by the individual resource and set as a whole.
- Desired level of reliability and level of reliability offered by the individual resource and set as a whole.
- Cost of each resource for the duration it is being used

These QoS parameters need to be optimized for resource provisioning and advanced reservation, which has been discussed in the next section.

4.4.2 Advanced Reservation of Resources

Historical data produced, maintained and published by monitoring information (as discussed in previous section) can thus be used for optimization, which helps in better resource provisioning, advanced reservation, prediction of availability time and duration for future applications. For advanced reservation and best matching following strategy has been used:
Consider a base set of resources in a Grid, R={r1, r2, r3...rn}. Corresponding to each resource ri, Reliability Index, RI ri and Performance Index, PI pi has been created based on past experimental data and stored in the Knowledgebase. When a user inputs a query for the availability of resources, the user defines the type of resources he requires, the type of application and data set (Confidential, sensitive etc.) to be input. This request is matched with the data stored in the Knowledgebase and the information is used for generating a Best of Breed Set, (BoB_R) based on the user requirements,

BoB_R = {r4, r7, r8. . .rx} where BoB_R ϵR,
This set can further be negotiated for allocation and negotiations can be done on price/usage basis. For advanced reservation and resource provisioning optimization of above QoS parameters is essential. As this requires simultaneous optimization of more than one objective function hence Multi-Objective Optimization (MOP) has been used. Using the

Table 4.3: Information regarding individual resources

Information	*Units/Value*
Resource ID	*ID code R_ID 1...R_ID n*
Resource Name	*Hardware, Software, Operating System etc*
Availability of Resource	*Start Date and duration*
Performance Level	*On the Scale of 1-5*
Reliability Level	*On the Scale of 1-5*
Cost	*Per hour of usage in Rs.*

Table 4.4: Information regarding Best of Breed resources

Information	*Units/Value*
Resource Configuration	*Actual Values of Candidate Set 1...Candidate Set n*
Time	*-do-*
Duration of Availability	*-do-*
Performance Level	*-do-*
Reliability Level	*-do-*
Cost	*-do-*
Best of Breed Set of Resources	*Sum Total*

standard technique for MOP minimize a positively weighted convex sum of the objectives, that is

$$\sum_{i=1}^{n} \alpha_i f_i(x) \qquad , \alpha_i > 0 \, , i=1,2,\ldots,n$$

where α_i is the weight assigned to each parameter by the user.
$f_i(x)$ is the Optimization parameter

Minimize of this function is Pareto Optimal. It is up to the user to choose the appropriate weights depending upon his resource requirements. Table 4.3 shows the schema of the database, which stores the information regarding each individual resource, which can be used for optimization data. This is static schema of the Data model. It is initially fixed and can be extended as more resources are added to the Virtual Organization or if performance or reliability levels change.

4.4.3 Implementation Details

Table 4.4 shows the schema of the candidate sets and the values of the sets based on the optimization parameters. Depending upon the importance of parameters, user can choose

weights and the sum total is calculated. The candidate resource set, having highest sum total would be the Best of Breed Set. For example, the user gives a requirement and on the basis of that, three Best of Breed Sets can be picked up, Set S1, S2 and S3. Now the user can further add upon his weights to these sets, lets say the user desires:

- Configuration: 'Intel'
- Date: 30 August
- Duration:2 days
- Performance level 2 (On a scale of 5)
- Reliability level 1 (On a scale of 5) (Reliability lies between 0 to 1 but has been mapped on a scale from 1-5)
- Cost: minimum

User may rank these requirements as: Reliability is no.1, cost no.2, time no.3, performance no 4 and duration no. 5 and then weighted values can be generated (Actual Value X Weight). Finally, the sum total of set S1, S2 and S3 can be calculated and checked and the set with minimum value would be picked up. This set can be then reserved in advance for the user on a minimum cost/time (or any other user preference). The user needs to check in two days before the actual scheduling of application and if the user does not turn up the resource set is free and can be used for some other request. These steps are shown in detail in Figure 4.11.

This section discussed the design and implementation of the main components of the Information Layer. Next section discusses about the Autonomic system of PRATHAM framework, along with other components of the Broker Layer.

4.5 PRATHAM Autonomic System

An Autonomic system has the following characteristics:

- Self-managing capabilities in a system accomplish their functions by taking an appropriate action based on one or more situations that they sense in the environment and can result in a significant improvement in system management efficiency.

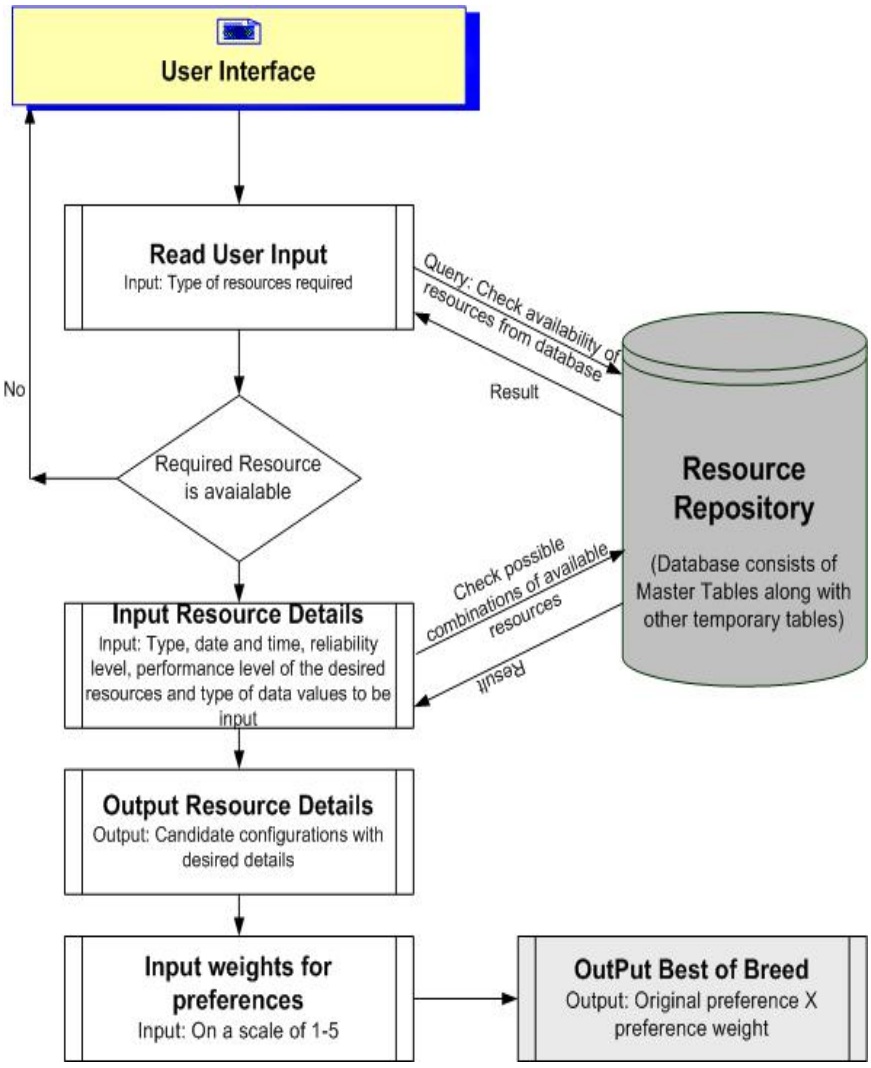

Figure 4.11: Steps followed in Resource Provisioning

- The function of any autonomic capability is a control loop that collects details from the system and acts accordingly.
- The goal of autonomic computing is to dramatically reduce the cost and perceived complexity of these systems by enabling them to manage themselves in accordance with high-level guidance from humans.

Grid should be free from the burden of manually maintaining resources, thereby allowing them to concentrate on the higher-level organizational tasks in the Grid. Amongst the four self-management issues of Autonomic Computing, presented by Kephart and Chess [74], focus of PRATHAM is on Self-healing, Self-optimization and Self-protection features.

Self-Healing Goals

(a) To predict problems and take necessary actions.

(b) To make the system more fault tolerant.

Self-Protection Goals

(a) Protection against accidental human errors.

(b) Protection against malicious intentional actions.

Self-Optimizing Goals

(a) Self-Tuning the systems dynamically by relocation of resources based on load balancing functions.

(b) Storing and exhibiting systems runtime information for overall resource utilization and system performance by doing resource provisioning. Figure 4.12 shows the autonomic

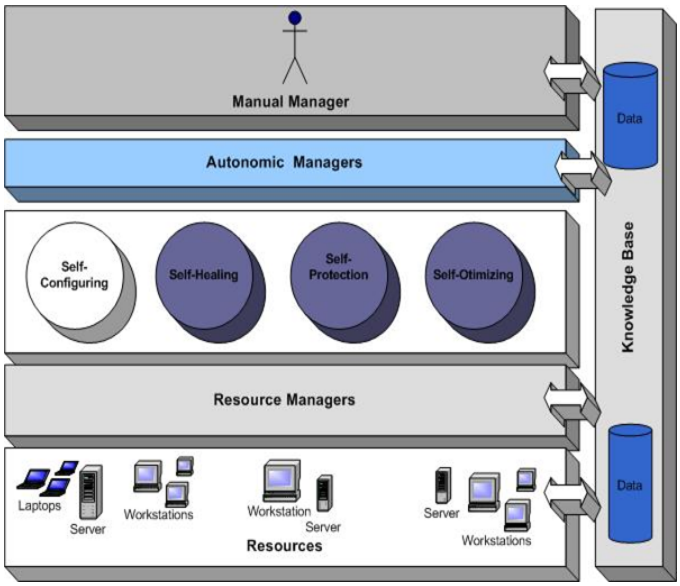

Figure 4.12: Autonomic Capabilities for Grid

capabilities of PRATHAM. The various manual and autonomic manager layers can obtain and share knowledge via knowledge sources i.e. Knowledgebase. Managers are responsible for determining which tasks can best benefit from autonomic self-managing technologies and they develop the policies that direct the autonomic function. A policy often starts as a technical implementation of a defined manual process. A self-managing autonomic system requires:

- A collection of self-managing resources that implements the desired function.
- Autonomic and manual managers that implement system functions that enable the required system-level behaviors.

- Knowledge Base that contain management data such as faults and remedies information.
- Design patterns that offer models for the structure and arrangement of components for system self-management.

4.5.1 Autonomic Framework

PRATHAM is an automated agent-based architecture, which helps in monitoring the Grid resources based on Pulse Monitoring technique.

A. Self-Healing

To achieve self-healing goals the first step is to predict errors and faults. Failures in Grid can be due to many reasons like configuration problem, independent failures of each element, failures resulting from interactions between components etc. The existing solutions for these failures are mainly application dependent. To facilitate the self-healing of the PRATHAM, a monitoring system is required which would be based on Pulse Monitoring. Pulse Monitoring is a hybrid approach for the autonomic environment. It employs the necessity concept of the beacon monitor to turn the heartbeat monitor into a pulse monitor. In Pulse Monitor instead of just checking the presence of a beat, the rate is also measured. The Pulse Monitor provides an indicator of the health of the system instead of just checking presence or absence of liveliness of a component.

Monitoring data about different components is gathered and stored in a repository as shown in Figure 4.12. The Knowledge Base stores the details of the resources, their uptime, current state, last down time, downtime duration etc. At finer level, clusters are being used as computing nodes in Grid environment, which are integrated into the Grid. Here, the problem of management of Grid becomes even more critical, because single clusters should (ideally) function autonomously without human intervention. But in a large cluster or fabric one faulty node can cause serious problems for the whole Grid. To address this problem, self-healing framework is required. For making the Grid autonomic, the self-healing framework involves:

(a) Local Self-Healing: A monitoring part to monitor the elements of the clusters and fault detection and recovery part for detecting the faults and automatically recovering

from them making the cluster self healing. In PRATHAM, local healing is based on individual clusters like Globus uses Heartbeat monitoring for monitoring nodes in cluster managed by Globus middleware.

(b) Grid Self-Healing: A monitoring part to monitor the Grid in totality and fault detection and recovery part for detecting the faults and automatically recovering from them making the Grid self healing. In PRATHAM, the Autonomic manager embedded in the Broker Layer takes care of self-healing at broker level (shown in Figure 4.2).

Autonomic Manager

Self-healing has been rendered to PRATHAM through Autonomic Manager. By using Autonomic Manager the time to collect problem data (like erroneous nodes, network connections etc.) can be reduced considerably, which ultimately reduces problem resolution. As a result On-Demand Computing can be provided to users in a cost-efficient way. Autonomic Manager for PRATHAM (shown in Figure 4.13) resides in Broker layer and collects data from the Monitoring Service for faults, overloading etc. It then passes

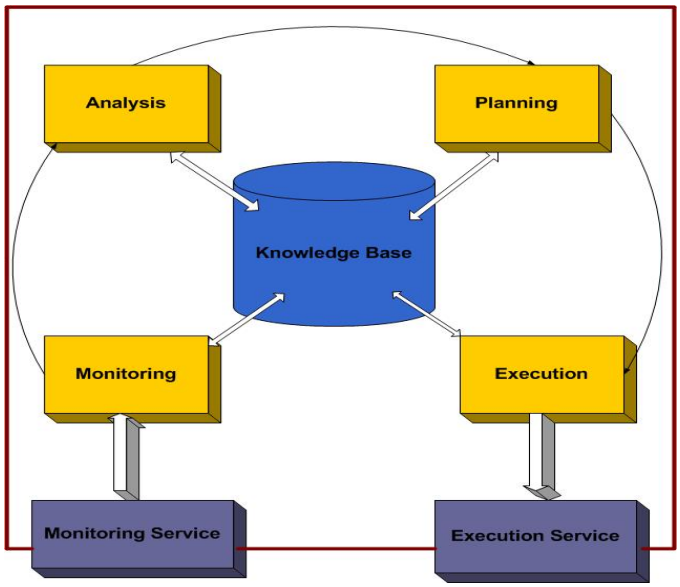

Figure 4.13: Workflow of Autonomic Manager

it to Monitoring unit, which keeps on collecting data and stores in Knowledgebase and further passes this data for analysis. Depending upon the Analysis Report, action Plan is identified. After the line of action is finalized the information is passed to Execution unit, which further passes the necessary information to Execution Service for actual execution.

B. Self-Protection

The goal of self-protecting environment is to provide the right information to the right

users at the right time through actions that grant access based on the user's role and pre-established policies. A self-protected Grid should be less vulnerable to unauthorized access and use, viruses, denial-of-service attacks and general failures. Self-Protection framework exists at the following levels:

(a) Firstly, at the Application interface where the user submits application for execution. Jobs from the Grid community pass an authentication check before they can be submitted to a local queuing system. Authentication may depend on multiple factors, e.g. user's affiliation, resource requirements, current state of the resources, and may change dynamically. If a job request has passed the authentication step it is provided with the necessary credentials for their execution.

(b) At the cluster-level, so that only authorized nodes enter the Virtual Organization, Grid being a dynamic environment.

(c) At the individual node level so that only authorized users are able to switch On and Shutdown the nodes forming part of clusters and Grid.

To stop malicious intrusions onto Grid, authorization checks are desired which have been implemented in the form of Login ids and passwords. Apart from this, PRATHAM has CA based security. CA issues digital certificates, which contain a public key and the identity of the owner. The CA also attests that the public key contained in the certificate belongs to the person, organization, server or other entity noted in the certificate. A CA's obligation in such schemes is to verify an applicant's credentials, so that users and relying parties can trust the information in the CA's certificates. If the user trusts the CA and can verify the CA's signature, then they can also verify that a certain public key does indeed belong to whomsoever is identified in the certificate. If the CA can be subverted, then the security of the entire system is lost.

In PRATHAM, on the receipt of a qualified certificate request the CA will carefully check the compliance and validity of any documents presented by the subscribers. After successful authentication, the PRATHAM Certification Authority will issue a certificate. Such issuance will be notified to the subscriber at the electronic mail address specified as part of the request. If the communication fails permanently, the certificate may be revoked without further notice. No confirmation of receipt of electronic mail notification is done. A request for certification is normally handled within one week.

C. Self-Optimization

As per the resource management requirement, available resources should be optimally used in order to balance the distribution of the tasks over the different computing nodes. Self-optimization is required for the Grid as a whole. Cluster-level optimization is handled by the local resource managers and by incorporating local load balancing policies. At the Grid level the role of the scheduler has to be as Autonomic Manager incorporating the Self-optimization framework. In PRATHAM, Self-optimization has been implemented by Resource provisioning and advanced reservation which needs to be handled by the Chief Manager at Grid level and has been discussed earlier (discussed in Section 4.4.1 , QoS parameters have been optimized). Self-optimization also needs to be achieved at the cluster level by load balancing. With the implementation of these two features, Grid becomes self-optimized. In PRATHAM, as scheduling is taken care of by the local schedulers (at cluster level), the load-balancing features of local resource managers have been used. Thus, with the implementation of Resource Provisioning Unit and local load balancing features, PRATHAM achieves Self-optimization to a large extent.

4.5.2 Architecture of Monitoring System

Self-management requires an efficient monitoring system. Architecture of the PRATHAM monitoring system is shown in Figure 4.14. Assuming that a Grid environment may have different middleware installed, resource information needs to be gathered from them and stored in a Resource Information Collector. This information is transformed and sent to the semantic repository. This resource information is being used by the Autonomic Manager, which manages all the managed components (resources) inside a Grid environment. Every Managed component has an agent, which sends out pulses to the Autonomic Manager at defined frequency. Autonomic Manager consists of a self-monitor, pulse monitor and an actuator. The pulses are intercepted by the pulse monitor and sent to self-monitor which analyzes the data received and executes the corresponding action by the Actuator.

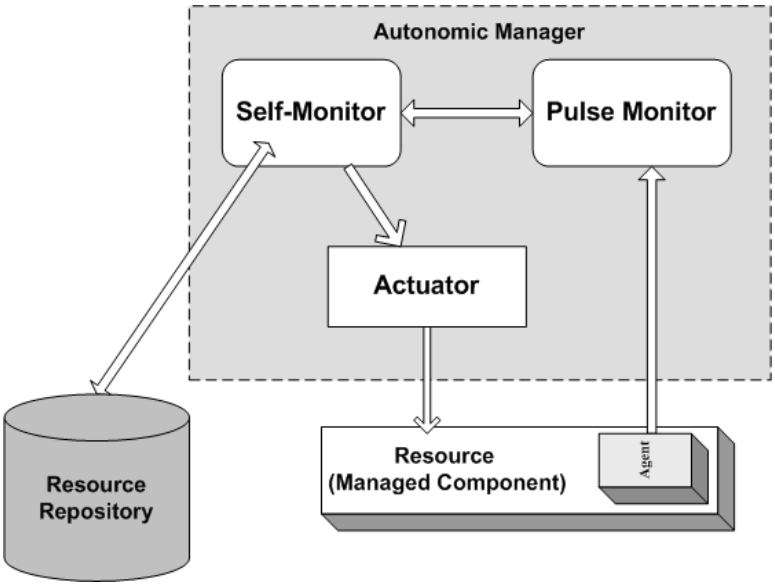

Figure 4.14: Components of Monitoring System

4.5.3 Components of Monitoring System

Main components of Monitoring System are:

(a) Resource Repository

This component collects information from the repository or index service of the Grid middleware, assuming that there can be more than one middleware running in the Grid environment. This repository stores the information of resources collected from different middleware inside MySQL. Different middleware represent and store their resources differently, therefore it becomes difficult to share the common data in a VO. To avoid this and for better interoperability the information is converted into RDF using OWL and stored along with their ontologies in a Semantic Repository. This repository can be queried for the states of different resources, their performance data etc. on uniform basis (This has already been discussed in section 4.3.1)

(b) Self-Monitor

Self-monitor gathers the information about the resources and for each resource checks the corresponding pulse data generated by the pulse monitor on the basis of rules. If some deviation from normal behavior is assessed, the actuator is activated. The alterations in components for self-healing of the Grid are processed in this component.

(c) Pulse Monitor

Pulse monitor intercepts pulses send out by each resource. As per the pulse rules, the

meaning is passed on to self-monitor which utilizes it for adjustment information.

(d) Actuator

Actuator takes the corresponding action as directed by the self-monitor. Input is given by self-monitor and corresponding to this input, it generates the output for the managed component.

(e) Managed Component

Each individual resource in VO gets registered with the VO Manager and the Manager places an agent on the resource to make it a Managed Component (MC). Managed Component is created using agent based technology as agents can perform a number of tasks. They can securely start monitoring programs on any hosts and manage their resulting data. They can provide a standard interface to host monitoring sensors for CPU load, interrupt rate, and so on. Agents can also independently perform various administrative tasks, such as restarting servers or monitoring processes. In case of PRATHAM Monitoring System, the residing agent takes input of resource information from Ganglia and passes it on to the pulse monitor.

4.5.4 Implementation Details

Pulse Monitor has been implemented as an agent. A sending-agent is loaded on each resource at the time of its registration with the Middleware manager. One listening-agent would be residing on the manager. The pulses are sent in the form of pulse data, which would be deciphered at the listening-agent for deducing the meaning regarding the health of each resource. Pulse Data, being sent by sending-agent has been designed as shown in Table 4.5. The above shown matrix shows the level of the Pulse and the corresponding rules. The listening-agent would be assessing the packet details, bit by bit. For example, data 000 (0) means, all components are working normal and no action needs to be taken, data 010 (2) means, the resource would be approaching its load limit and the corresponding action would be to inform the Manager so that no more jobs are directed towards this resource. Similarly, if complete failure occurs, job migration might be required.

A standardized pulse monitor can provide a convenient design of fault handling and pro-

Table 4.5: Pulse Data

Bit (2)	***Bit (1)***	***Bit (0)***	***Meaning of Pulse***
0	*0*	*0*	***All components are working normal***
0	*0*	*1*	***Memory approaching max limit***
0	*1*	*0*	***CPU Load approaching max limit***
0	*1*	*1*	***Disk Usage approaching max limit***
1	*0*	*0*	***Major component has failed***
1	*0*	*1*	***Network traffic exceeding the limit***
1	*1*	*0*	***Reserved (Can be used in future)***
1	*1*	*1*	***Complete failure, needs immediate attention***

vides efficient information for self-healing. Pulse Monitor sends out a steady beat at defined frequency. When detected, this indicates that the autonomic element is alive and when missing indicates an operational problem, like network failure or complete failure of the element. PRATHAM utilizes the resource-monitoring agent (gmond) of Ganglia to provide quantitative data regarding the status of local Grid resource. At the metalevel, the monitoring system utilizes the agent-based methodology, where each agent acts as a representative for a local Grid resource and provides information about the resource. First of all Client agents retrieve resource info from client machines and Server Agent fetches that information from Client Agents via TCP connection, processes the received info to generate an XML file. XML file is transferred to the Resource Monitoring Module within monitoring system. Resource Monitoring Module parses the XML file and updates the central database repository; finally it analyzes the resource information (metrics) stored in central database repository and provides the monitoring information to Grid Administrator who can then take the action accordingly.

Components of Agent based Resource Monitoring System

Agent based monitoring System consists of the following basic components that coordinate together to perform Grid resource monitoring as shown in Figure 4.15.

(a) Client Agent

It is a multi threaded software module that runs on each client machine that is to be monitored. It has four main responsibilities: monitor changes in host state; multicast relevant changes; listen to the state of all other Client nodes via a multicast channel; answer requests for an XML description of the Grid state. Each Client agent transmits

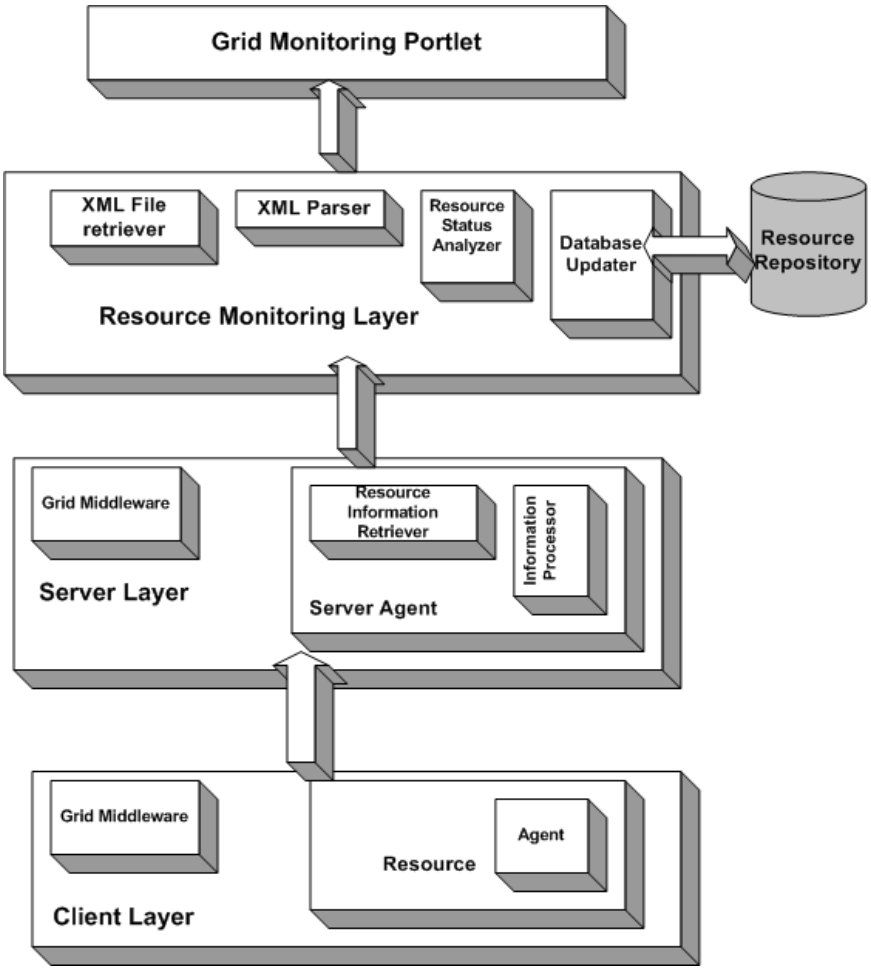

Figure 4.15: Components of Resource Monitoring System

information in two different ways: multicasting host state in XDR format or sending XML over a TCP connection.

(b) Server Agent

Server Agent Module (in Java), includes a resource information retriever and a resource information processor. Resource information retrieves the resource information from the client Agents repeatedly in the form of XML over a TCP connection. This module continuously telnet the server to retrieve the resource information from all the nodes in the Grid and stores it in a file which is then processed by a resource information processor. Finally, the processed XML file having resource information about all the nodes connected in the Grid is transferred to the resource-monitoring module for further processing and information retrieval. Processed XML file has one host tag for each node in the Grid environment.

(c) Resource Monitoring Module

It is the software module written in Java that retrieves the resource information from

the processed XML file delivered by the Server Agent. It fetches the processed XML file from Server Agent via ftp server, XML file is parsed by the XML parsing module within the Resource monitoring module and resource information extracted from the XML file is used by the database updater module to update the central database repository. This process is repeated frequently to update the central database repository.
If any of the resource parameter shows an abnormal behavior or is overloaded or under loaded the resource status analyzer informs the administrator through a pop up message, administrator can take the action accordingly: it may be migrating the job from overloaded resource to under loaded resource, removing the abnormal resource from the Grid infrastructure and so on.
(d) Central database repository
Central database has been created in MySQL. The complete resource information is stored in it and it is updated regularly. It has 2 parts:

- Node_Info
- Feature_Info

Node_Info stores the information about the different nodes connected to the Grid. Feature_Info stores the value of various important resource parameters that need to be monitored.
This section explained the Autonomic system, design and implementation of monitoring system for PRATHAM, the major components of Broker Layer. Next section elucidates the self-management aspect of PRATHAM (also addressed by this Layer).

4.6 Incorporating Self-Management in PRATHAM

To achieve self-healing goals, the first step is to predict errors and faults. Failures in Grid can be due to many reasons like configuration problem, independent failures of each element, failures resulting from interactions between components etc. The existing solutions for these failures are mainly application dependent (as already discussed). To facilitate the Self-Healing of the Grid, the fault discovery and healing system uses monitoring data that has been generated by the Monitoring System.

4.6.1 Fault Detection and Healing System (FDHS)

The Fault Detection and Healing System (FDHS) is rule based. Each rule compares the data received from the monitoring system against the permissible limits. If the condition of a rule holds, the corresponding action is performed. Figure 4.16 shows the components of FDHS.

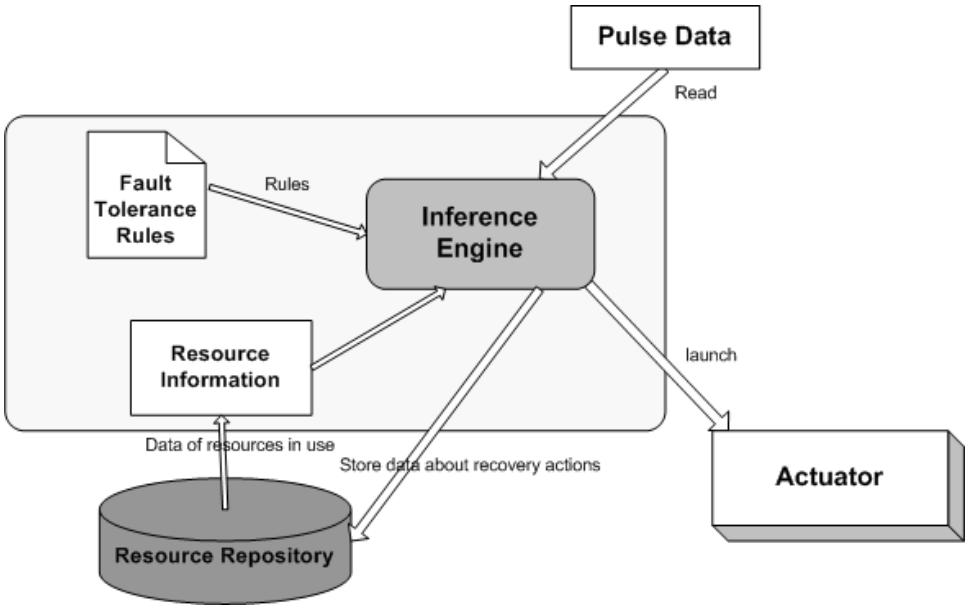

Figure 4.16: Fault Detection and Healing System

4.6.2 Components of FDHS

The main components of the FDHS are:

(a) Inference Engine

Inference Engine is the heart of FDHS. It reads the Resource information, which is stored in the Resource Repository along with the pulse data, which is being transmitted at defined frequency by the sending agent residing on each individual resource. This data is then compared with the Fault Tolerance Rules, which have been predefined. If resource is performing within the configured limits no action is taken else if adjustments are required the corresponding action is executed by launching the Actuator. This process works in a control loop and the corrections are made till the resource performs again within the permissible limits.

(b) Fault Tolerance Rules

Rules have been predefined and shown in Table 4.5. These rules have been kept simple

to avoid situations where different rules conflict with each other. Here, in total, we have defined eight rules as per our requirement. To reduce the network traffic, only three bits are being sent. In future, if some more inferences have to be drawn then more bits can be introduced but this will increase the network traffic especially in a large Virtual Organization.

(c) Resource Repository

Repository stores the details of the resources (discussed in section 4.3.1) along with the current details of Jobs, which are running on it. It stores the details of the resources' uptime, current state, last down time, downtime duration etc. It also stores the past performance data of the resource so that future performance predictions can be made and past actions which have resulted successfully may be used again by the inference engine.

(d) Actuator

Actuator executes automatic recovery actions. For example, resource or service restart may be required in case of some fault, Automatic backup may be required or alert messages may be mandatory to be passed on to the middleware managers etc. The action can be executed by the Actuator on the individual resource or on the manager of the middleware as per the requirement.

4.6.3 Implementation Details of FDHS

Most of the implementation details have been covered in section 4.5.4. To summarize:

Local Self-Healing: For local Self-Healing the individual middleware use their own inbuilt monitoring techniques for the whole cluster. For example, in the large clusters supported by GT and Alchemi.Net, their existing Heartbeat Monitoring Technique has been utilized.

Grid Self Healing: Grid Self-healing has been done by incorporating the Monitoring system and corresponding FDHS into the PRATHAM Framework.

In PRATHAM, the Portal would send the jobs to the Chief Manager, (which is the broker in this case and handles the Job scheduling part). At the time of registration of a new Resource in the VO, the Grid Middleware Manager would register the resource and then this information would be passed on to the Chief Manager (broker). The Chief Manager

would then send the residing-agent to the Resource for installation and for becoming Managed Component. The Job Monitor collects the information of different resources through the managers of the Grid Middleware (currently GT4 and Alchemi.NET) and stores it into the repository. The Autonomic Manager (shown in Figure 4.2) stores the Self-monitor and the Pulse monitor, which perform their respective roles. If some adjustment is required the information is passed on to the Actuator. The Actuator sends an agent for executing the adjustment on the actual Managed Component i.e. the Resource. This process is conducted in the form of a control loop and thus helps in self-healing of the components individually and the Grid in totality. The implementation details have been covered in section 4.5.4.

In this case there are two fold advantages of incorporating a dual monitoring technique. Firstly the local resources can do their local self-healing autonomously. Secondly, if in the whole Virtual Organization some adjustment would be required the Grid self-healing would help. The resources would be monitored in a dual mode and would ensure better management and Self-Healing. One typical scenario of resource monitoring (Job migration due to processor overloading) utilized for self-healing purposes has been shown in Figure 4.17.

This section discussed the self-managed aspect of PRATHAM, handled by Broker Layer. Next section discusses the Application layer i.e. Grid Portal.

4.7 Grid Portal: Manageability Interface for PRATHAM

A Grid portal is a user's point of access to a Grid system. It provides an environment where the user can access Grid resources and services, execute and monitor Grid applications, and collaborate with other users (For example, Yahoo Portal- email, personal groups, personalized news channels, stocks, weather etc.) The portal must provide a way to easily integrate new application interfaces and expose them to the user and hide the details of Grid middleware.

For PRATHAM, the interface for its manageability functions is Grid Portal, which is a combination of many portlets. The PRATHAM architecture presents a uniform appearance to users and provides for a Grid made up of diverse computing environments (hard-

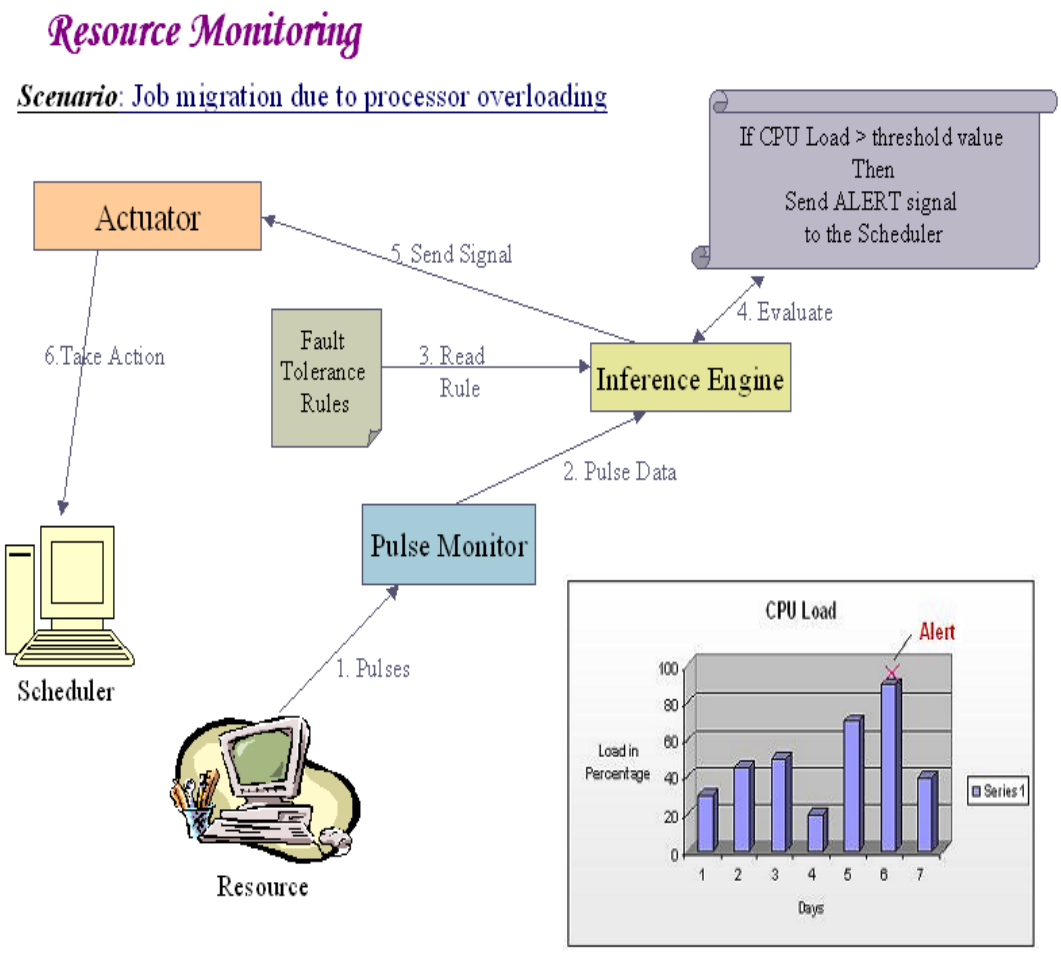

Figure 4.17: Resource Monitoring Scenario

ware, operating systems, job schedulers) and autonomous administrative domains. Local organizations own and maintain control of the resources involved, and local administrative procedures and security solutions take precedence. For each individual function like resource discovery, resource provisioning, Grid portlets have been created. PRATHAM Grid Portal:

- Provides a single through-the-web interface to all the clusters in a Grid. This interface hides user interface and scheduler differences among clusters and it makes it easy to work with multiple clusters at once.

- Provides a single login for users. Users log onto a portal using a web browser and authenticate by means of a user-id and password. A user logs into the Grid Portal, not into each of the individual clusters that the user will use.

- Provides resources to users who have login ids on individual clusters (Cluster Users). Any new user who registers with PRATHAM and gets affiliation, can easily gain

access to resources throughout the Grid by becoming a Grid User.

- Is secure to the extent possible by up to date technology. Proxy certificates are used for authentication at every step of the way.

- Provides Remote File management. Tools to access to file metadata directories and remote file archives is a central requirement for the portal. Simple tools for Grid FTP are essential and are available.

- Provides Remote Job management. The ability to submit jobs to the Grid for execution and monitoring is a classic requirement for portals. Users with allocation on specific resources want to be able to see job queues on those resources and consult scheduling assistants. They need to be able to keep track of job execution and understand when things fail by reading logs.

- Provides access to information services. Access to directories and index tools is an essential role of the portal. Each user should have a private, persistent store of references to important information they have stored on the Grid. As soon as a Grid User logins, user lands up in his own access area only.

4.7.1 Portal Architecture

The Grid Portal architecture is being defined in terms of a collection of remote Grid services. The portal architecture that has been developed is based on the idea that the portal server is a container for "user facing" Grid service clients which are designed according to the portlet model. A portlet is a server component that controls a small, user-configured window in a "pane" on the user's web browser. The advantage of this architecture for Grid Portals is that it provides a natural way to incorporate "user-facing" Grid services into the portal environment. That is, if a Grid service has been designed to be used through an interactive client, that client can be written as a portlet and provided to the user in the portal.

When the user logs into the portal, portal should bring up user's current "Grid Context". This implies, for the user there exists, persistent directory of objects, annotations, references and notification logs. Portal should also provide the set of tools (portlets)

which user uses to access remote services, configured into groups of tabbed panes and organized the way user last left them. There are many advantage of portlet architecture. Each Grid service can be associated with a unique portlet. It is very easy to add new services and many different Groups can contribute portlets, which can be plugged into a portal. PRATHAM Portal architecture has been shown in Figure 4.18.

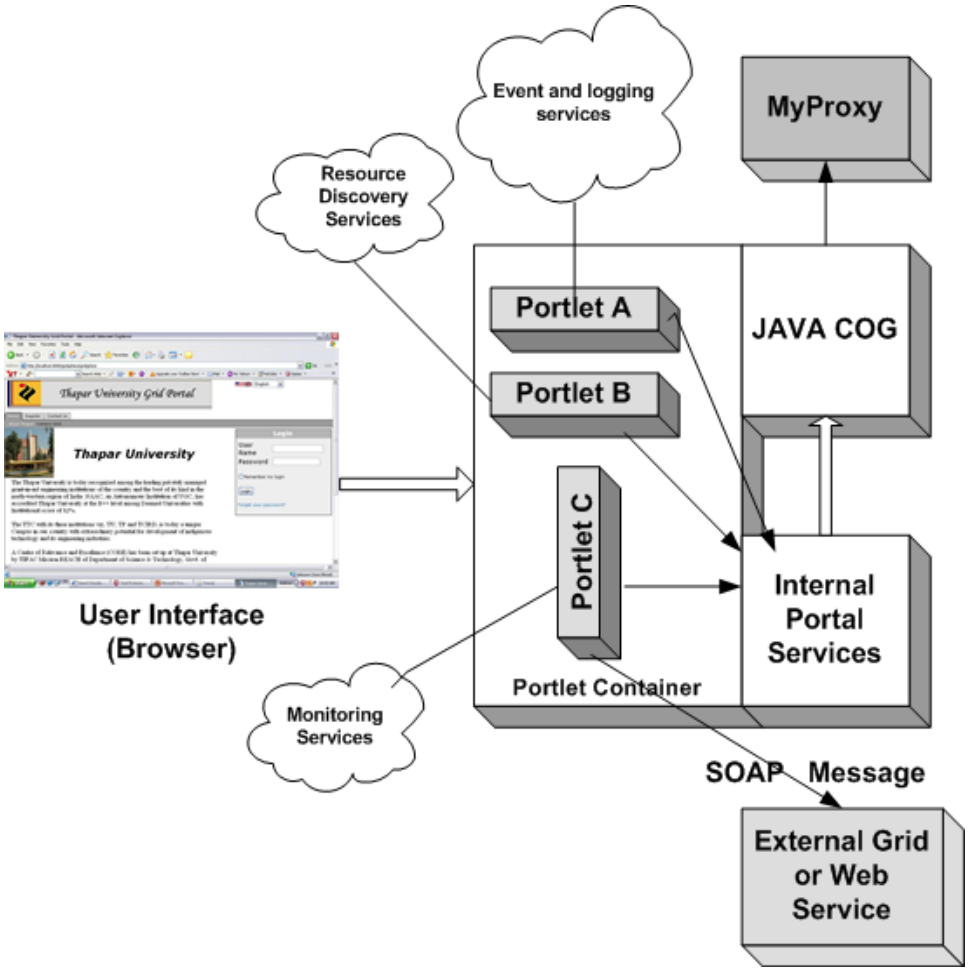

Figure 4.18: PRATHAM Portal Architecture

4.7.2 Grid Portlets

Following main portlets have been created on the Grid Portal:

(a) User Registration

(b) Resource Browser

(c) Resource Discovery

(d) Resource Provisioning

(e) Monitoring

4.7.3 Implementation Details

The Grid Portal has been implemented using Gridsphere 2.2.10 [173]. It is part of GridLab projects funded by EU, 100% JSR168 compliant and open source. Apart from this following software have also been used:

- JSR 168 Portlets [171]
- Grid Portlets
- JSP
- HTML
- Apache Tomcat (Application Server)

The GridSphere portlet container is implemented as a web application and requires a hosting environment such as the Apache Tomcat container. Initially when the portal is started, the portlets are all initialized and the service method is invoked for every client request until the portal is shutdown at which point the portlets are all destroyed to allow portlets to perform any shutdown operations. The portal snapshots have been detailed in the next chapter.

4.8 Conclusions

This chapter describes in detail the components of the PRATHAM resource management framework. PRATHAM Taxonomy and Workflow have been explained along with the components of top three layers of the framework. Design and implementation of Information Layer for resource discovery and resource provisioning, along with Broker Layer for monitoring and autonomic system and Application Layer for Grid portal have been described in detail.

Next chapter illustrates experimental results and validation details of the framework.

Chapter 5

Deployment, Testing and Validation of the Framework

Previous chapter discussed in detail the design and implementation of the proposed framework PRATHAM. This chapter focuses on the deployment, testing and validation of the proposed framework. For testing purposes, framework PRATHAM has been deployed at Thapar University, Patiala. It has been tested as a campus wide Grid by using different middleware. These middleware were installed on the computer systems of different departments, laboratories and hostels of the university. The snapshots of the experimental results exhibit user authentication, resource discovery, resource repositories, resource provisioning and monitoring results. The framework has also been validated from various other perspectives similar to existing frameworks and Grids like UK e-Science, White Rose, Nordu Grid etc.

In this chapter initially the details of deployment of the framework at Thapar University have been given. Further, the Grid setup and experimental results obtained have been illustrated. Next the framework has been validated and finally the last section concludes the chapter.

5.1 Framework Deployment and Evaluation

The Grid Resource Management framework PRATHAM has been deployed at Thapar University Campus, Patiala and mainly managed from Centre of Excellence in Grid Computing. Before deploying the framework, a Grid has been established. The details of the Grid setup are being presented as a case study for university campus in the subsequent sub-section.

5.1.1 Case Study: Thapar University Campus Grid

For evaluation purposes, PRATHAM has been discussed in the context of Thapar University Grid (as shown in Figure 5.1). Different departments, such as Computer Science and Engineering, Electronics and Communication Engineering, Instrumentation Engineering, Applied Sciences, etc. have individual computational resources at the University. Due to the lack of resource co-operation, a compute-intensive research project, may be restricted by the limited resources available within the local administrative domain, or occupy and overload all the local resources, while those in other departments may be lying idle. Therefore, this campus Grid setup is motivated by the need to allow easy access to researchers to computational resources across the departments, anywhere within the University. There is a need to make good use of not only dedicated resources, like servers, but also workstations, desktops and computers available in the labs and offices, which are under-utilized, especially at night and at weekends and can be a huge resource if aggregated. Therefore, a Grid environment has been setup in the Centre of Excellence (CoE) for Grid Computing, computer laboratories in different departments, hostels etc. by installing various middleware on different machines.

The installation of middleware has been done as per the following detail:

- N1 Grid Engine 6 has been installed on three systems in CoE that included one master cum execution host and two execution/submission hosts.
- Globus Toolkit 4.0 has been installed on six systems having Fedora core 6 operating system at Centre of Excellence for Grid Computing at Computer Science and Engineering Department.

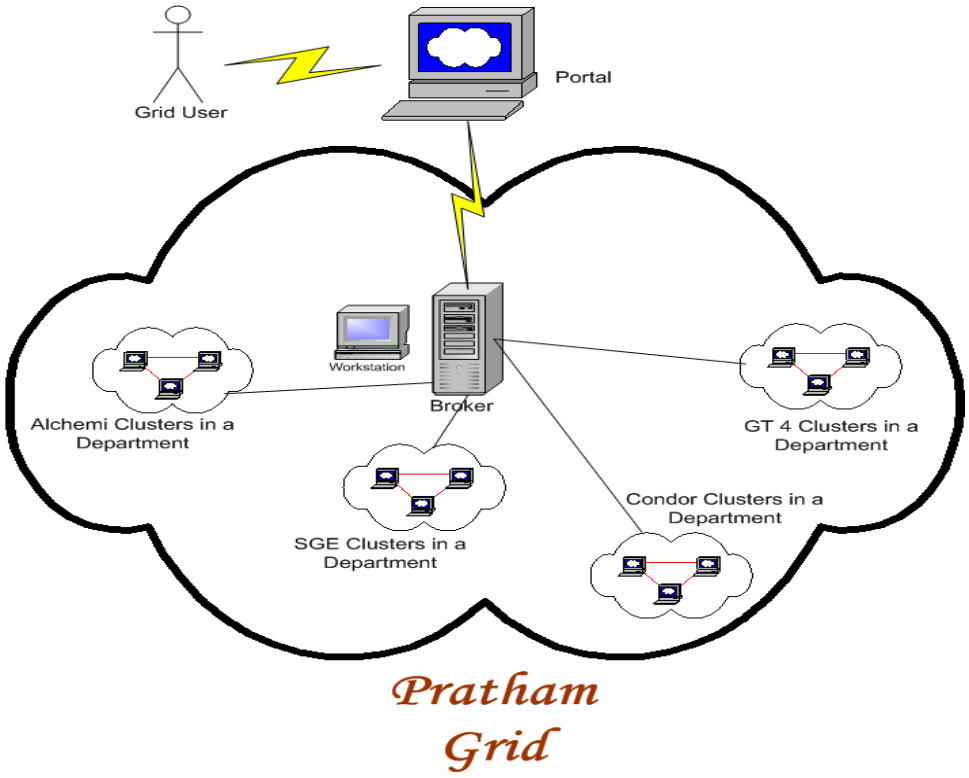

Figure 5.1: Grid setup at Thapar University

- Windows based version of Condor 6.6.10 has been also installed on five systems in CoE, CC-CT, CC-M.

- Alchemi 1.06 has been installed on twenty systems within the Centre of Excellence for Grid Computing, CCCT lab, Software Engineering and hostel.

For experimental results, initially Globus and Alchemi middleware have been considered. Seamless job scheduling has been assumed due to constraints of Alchemi.NET and Grid bus broker. The step-by-step installation procedure is given in Appendix B.

GridBus Broker

As mentioned above, different middleware have been used to setup the Grid environment in the Centre of Excellence for Grid computing and other locations campus-wide. Since these middleware are based on different standards and are not interoperable with each other, a common platform through which these middleware could communicate (and for job scheduling) was required. As mentioned in Chapter 3, GridBus Broker 2.4 has been installed for this purpose. The installation details of GridBus Broker have also been given in Appendix B. Figure 5.2 gives the snapshot of successful installation of GridBus Broker.

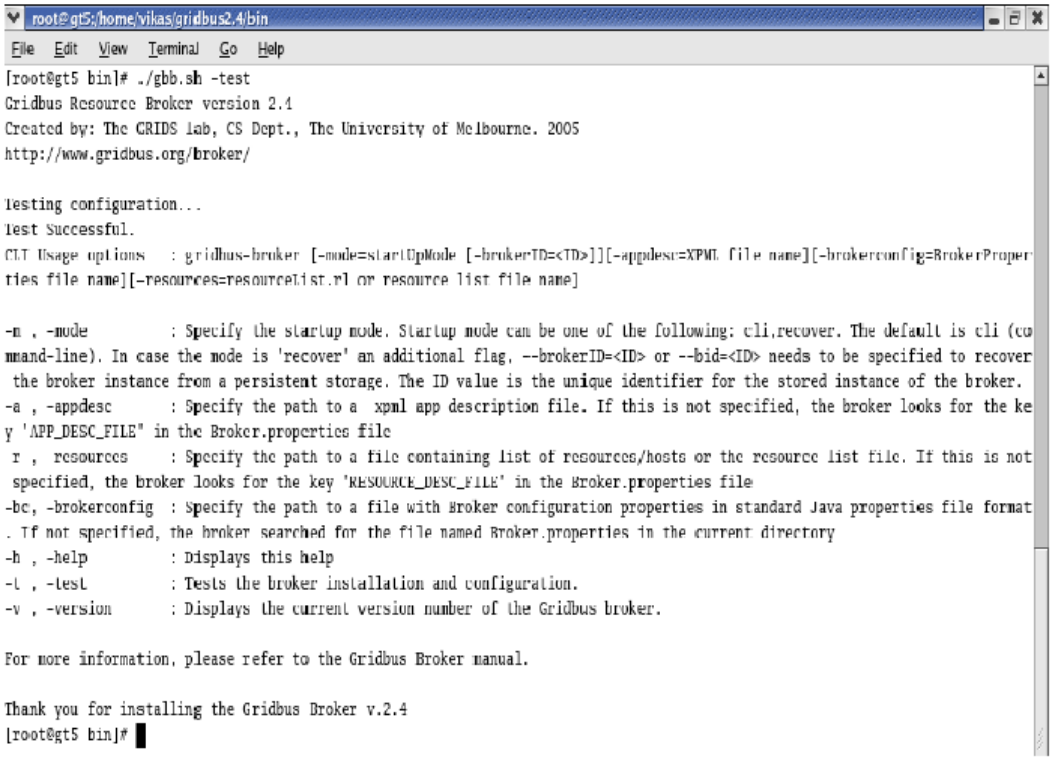

Figure 5.2: Successful installation of GridBus Broker v2.4

Along with broker, Grid-monitoring tool Ganglia has also been installed. Ganglia [46], [155] is an open-source project that grew out of the University of California, Berkeley. Ganglia is a distributed monitoring system for high performance computing systems such as clusters and the Grid. It is based on a hierarchical design targeted at federations of clusters. Ganglia uses XML for data representation, XDR [157] for data transport, and RRD tool [158] for data storage and visualization. It attempts to minimize per-node system overheads by using specialized data structures and algorithms. Ganglia consists of two components, Gmond and Gmetad:

Gmond: The Ganglia Monitoring Daemon (Gmond) is a multi-threaded daemon, which runs on each cluster node to be monitored; it has four main responsibilities:

- monitor changes in host state;
- multicast relevant changes;
- listen to the state of all other Ganglia nodes via a multicast channel;
- answer requests for an XML description of the cluster state.

Each daemon transmits information in two different ways: multicasting host state in XDR format or sending XML over a TCP connection.

Gmetad: Ganglia Meta Daemons (Gmetad) are used to provide a federated view. At each node in the tree, a Gmetad periodically polls a collection of child data sources, parses the collected XML, saves all numeric, volatile metrics to round-robin databases, and exports the aggregated XML over a TCP socket to clients. It thus helps in monitoring the status of individual resource queues as the node on which the Gmetad has been installed is the manager node and it gathers the information of the nodes in its cluster. The RRDtools package is required on every client machine running the Gmetad since it supports the functions of the Gmetad by providing a way to efficiently store the data collected. Client RRDtools have been installed on each client machine running Globus toolkit 4 on fedora 6 platform. Along with this, Client agent Gmond has also been installed on each client machine.

Grid Portal which has been designed and developed in GridSphere 2.2.10 (as discussed in chapter 4) has also been installed on the server along with the broker.

5.1.2 Experimental Results

Figure 5.3 shows the Portal of PRATHAM. Grid user would be interacting with the Grid through portal, which is Thapar University Grid Portal for this case study. Existing user will simply login through his authentic user name and password credentials.

New users, who want to utilize the services of the Grid, will first need to register themselves with the Grid (as shown in Figure 5.4). After verifying the credentials, the administrator will issue a username and password through e-mail to the new user and then the user may login. After logging, the user may view the available resources through the resource browser portlet as shown in Figure 5.5. Figure 5.5 shows the list of resources available at different locations, like CC-CT laboratory, Research laboratory etc. Details of these resources can be seen by selecting the location name. The Authentic user may need to discover resources for his job execution. Figure 5.6 shows the resource discovery portlet. User may enter his resource requirement in the portlet. These resource requirements can be type of processor, processor speed, amount of memory to be uti-

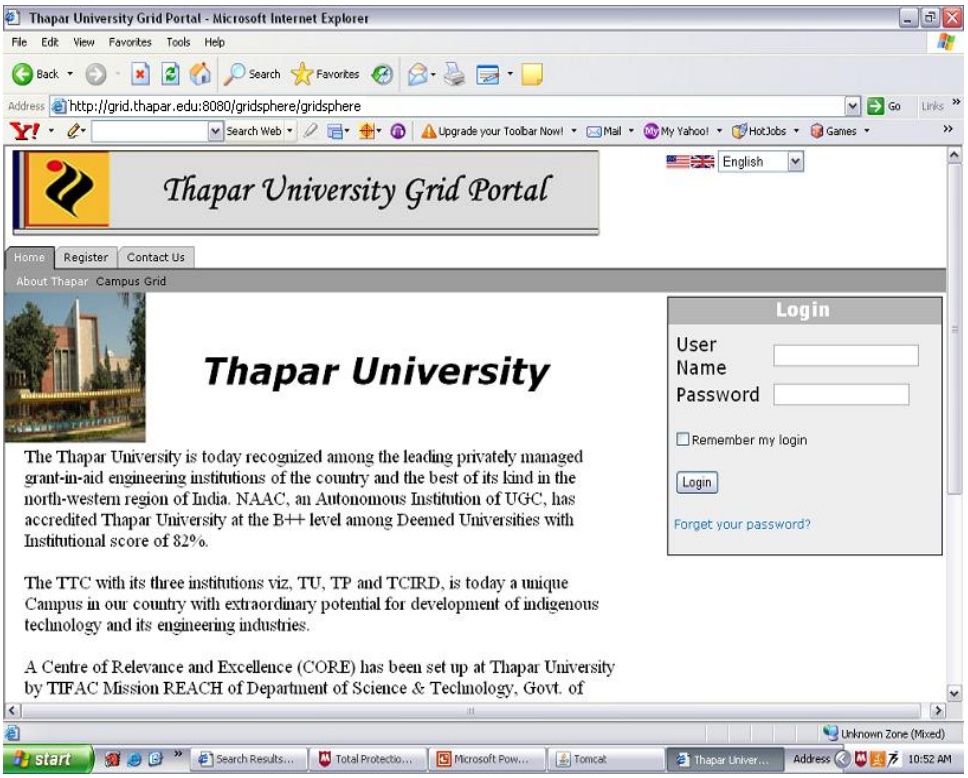

Figure 5.3: Thapar University Grid Portal

lized, secondary memory required and network speed (if required). For experimentation purposes only these parameters have been considered. Further, new parameters can be added as per the application requirement. The results of the available resources are shown in this format (Figure 5.7). Resources with similar configuration have been grouped and given a group id. The details of all the available resources are shown in the figure. The user enters the resource data irrespective of the resource representation and storage. This input is transformed into SPARQL query and the result is obtained. Further, if the user desires resource provisioning, he may press 'yes' on the interface depicting list of available resources and then enter QoS requirements (shown in Figure 5.8). An example result set generated for a set of user preferences shown above is depicted in Figure 5.9. This figure shows the resource sets which have been provisioned as per the user requirement. This provisioning is made on the basis of user preferences of specific configuration and the corresponding time and cost of that configuration. For e.g. the chart in Figure 5.9 shows that for a specific preference there are three sets of resources which are available for job execution, resource set 1 and 2 would be taking same execution time and cost, but resource set 3 would be taking much less time and correspondingly the cost would also be high for a given size 's' of user application in Function Point (FP).If advanced

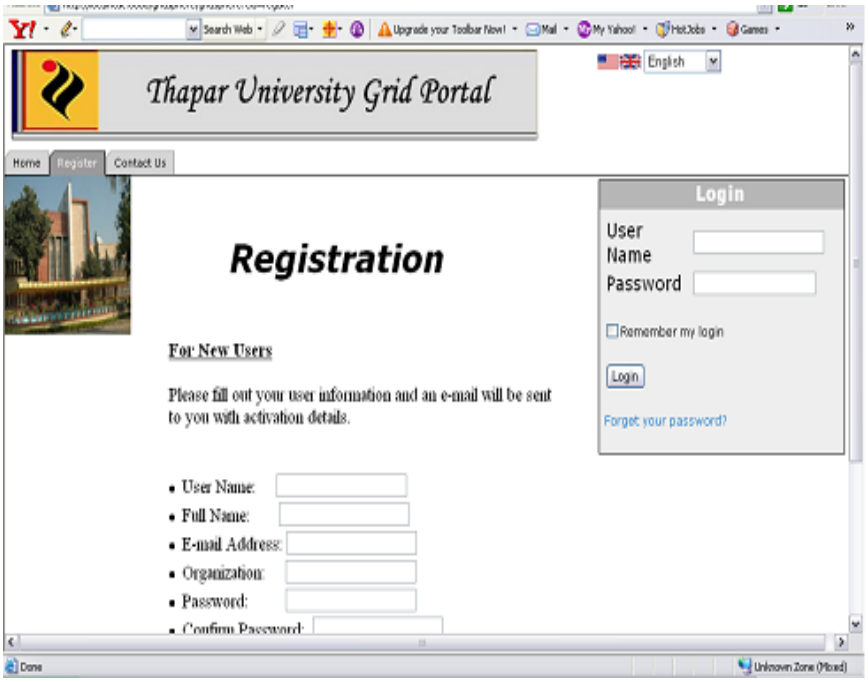

Figure 5.4: Registration of new Grid Users

reservation is required they can be booked for specific date and time. This date and time should be confirmed 24 hours before actual execution, else the resources are released and added in the resource pool.

Monitoring Results

For Grid monitoring, the resource information is desired. This information is collated in the base tables of the middleware and has been extracted and collected in a central repository (as discussed in Chapter 4). Figure 5.11 shows the state of database which stores the information of resources stored under Globus middleware. Similarly, Figure 5.12 shows the state of resources under Alchemi middleware. As discussed in Chapter 4, section 4.4.1, for each resource its reliability and performance level is stored based on heuristics (fault data gathered by monitoring services etc.) on a scale of 1 to 5.

The resources depicted in Figure 5.11 and 5.12 are being monitored by Ganglia. On each resource Gmond is installed along with the client agent as described in Chapter 4. Following figures, Figure 5.13, 5.14, 5.15 and 5.16 show the graphs created by Ganglia for each individual node. For demonstration purposes, graphs for a single node (172.31.5.129) have been shown. Similar graphs are created for all the nodes in the Grid. The graphs generated at the Manager node help in identifying all the nodes of that cluster. Thus, resource queues can be monitored at the interface. Figure 5.13 shows graph created by

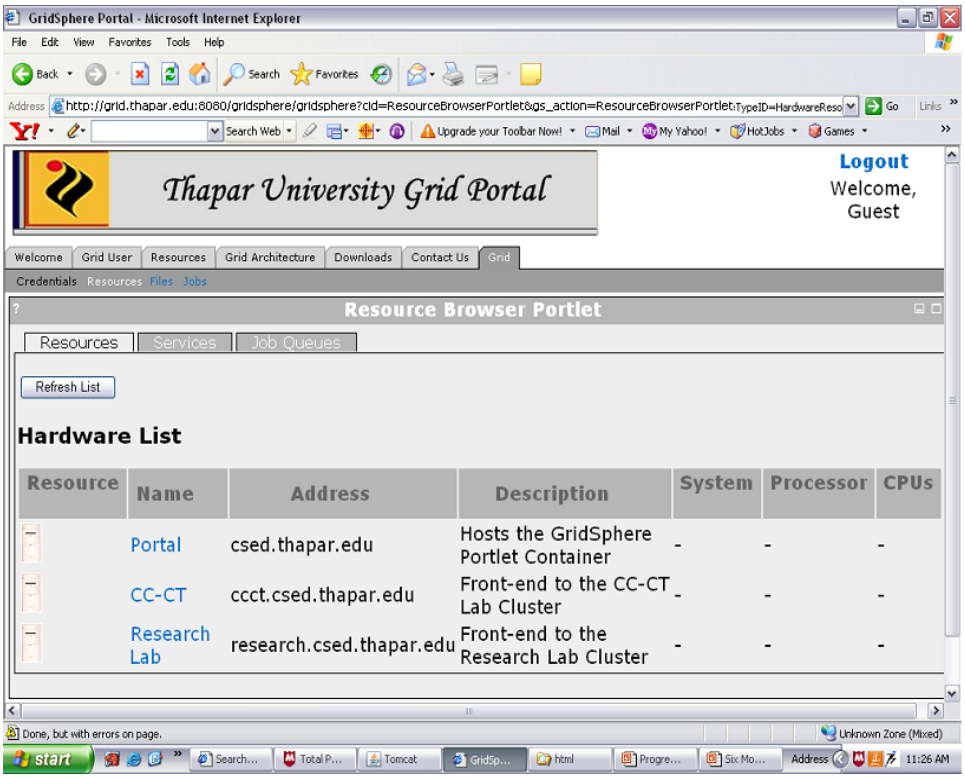

Figure 5.5: Resource Browser Portlet

Ganglia showing CPU load, Figure 5.14 shows graph created by Ganglia showing Network Traffic, Figure 5.15 shows graph created by Ganglia showing disk usage and Figure 5.16 shows graph created by Ganglia showing memory usage. Thus, each individual resource is being monitored at the interface itself. Figure 5.17 shows the details of the resource configuration in report format(Figures appended at the end of the chapter).

Gmond component of Ganglia generates resource information in XML format as shown in Figure 5.18. This information from Ganglia generated XML is parsed and deciphered and is used to generate alerts in case the values exceed the threshold. These alerts are stored in the resource repository and the administrator can directly find out not only the failure but also the reason of failure and may take further steps to fix-up the error. Thus, the debugging effort has been saved. This is particularly advantageous in large-scale Grids where identifying the reasons of faults is quite tedious. Therefore, better self-management can be accomplished.

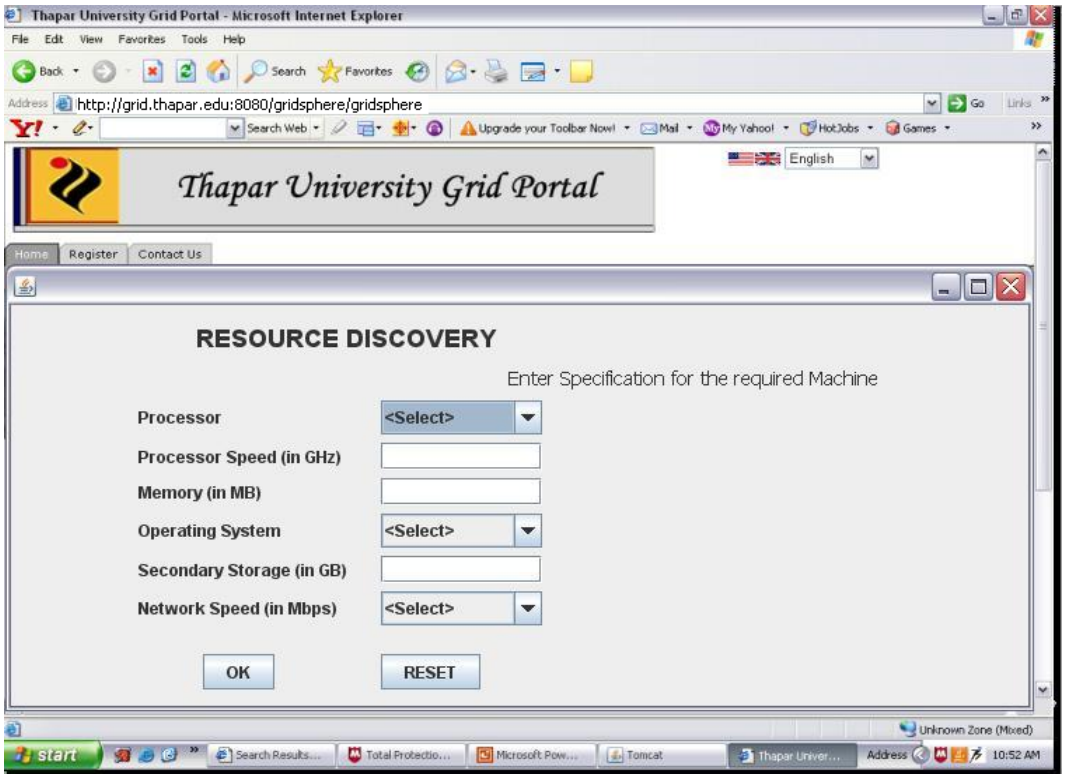

Figure 5.6: Resource Discovery requirements

5.2 Framework Validation

In PRATHAM the resource discovery is semantics based and utilizes ontology. The traditional resource discovery techniques are simple keyword based only. As the graph in Figure 5.10 shows, the different resource requests, for different set of resources; semantics generates better results as it is able to search the available resources more efficiently.

Resource Discovery Results

(a) Symmetric Key Word Search (Traditional Technique)

(b) Semantic Search (Semantic Resource Discovery)

The only overhead is initial cost of establishing central resource repository and semantic repository. But performance considerably improves (as can be seen from the graph).

PRATHAM has been validated against existing resource management frameworks. For the validation rationale, Metascheduler, MARS, OntoGrid and Garuda have been considered as these are based on recent research. The validation result has been depicted in Table 5.1. Another comparison has been done between PRATHAM and popular RMS frameworks, which are currently academic initiatives. This result has been shown in Table 5.2.

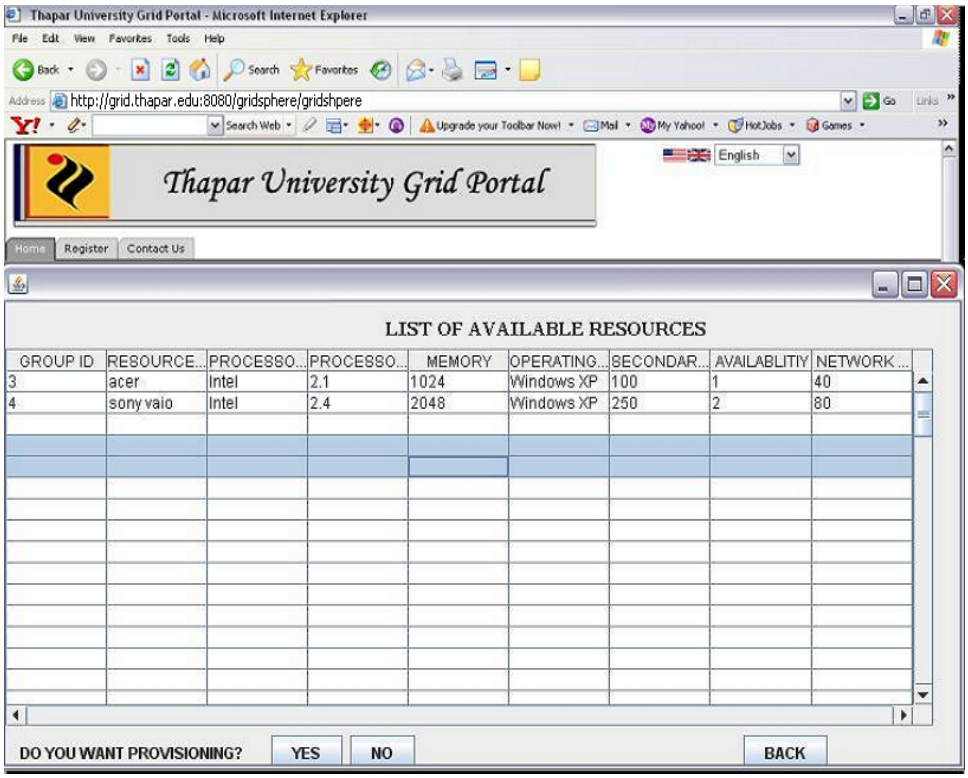

Figure 5.7: Result of Query

These comparisons have been done on the basis of clearly chosen parameters which form the basic features of the Grid Resource Management Systems and frameworks. The experimental results obtained in this work have been published also. Many Publications pertaining to Grid Resource Management and specifically to the proposed work have been published and one is under active consideration. (Details available in Publications' section)

5.3 Conclusions

This chapter describes the deployment details of the proposed framework. The case study of Thapar Campus Grid has been demonstrated. The experimental results have been illustrated for user authentication, resource discovery, resource provisioning and resource monitoring. The framework has also been validated. The validation results clearly depict that PRATHAM addresses the issues of interoperability, provisioning and self-management collectively. All these features are required for future, OGSA based Grids.

The next chapter concludes the thesis and also focuses on the future directions.

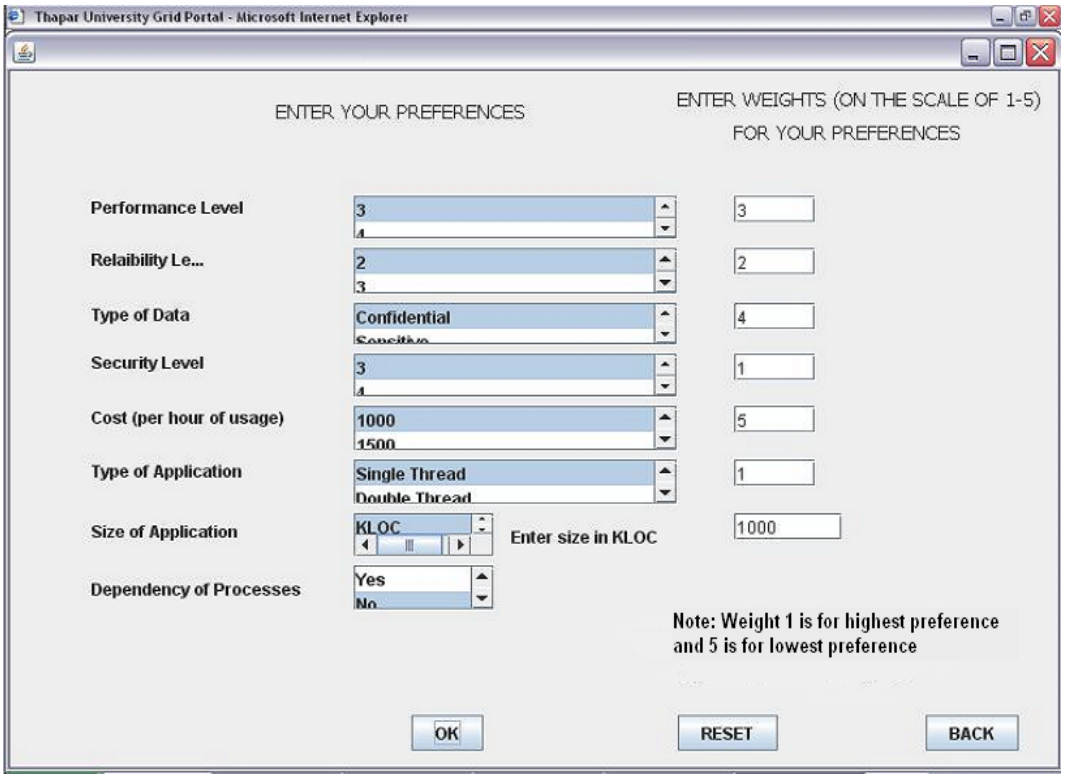

Figure 5.8: Parameters for Resource Provisioning

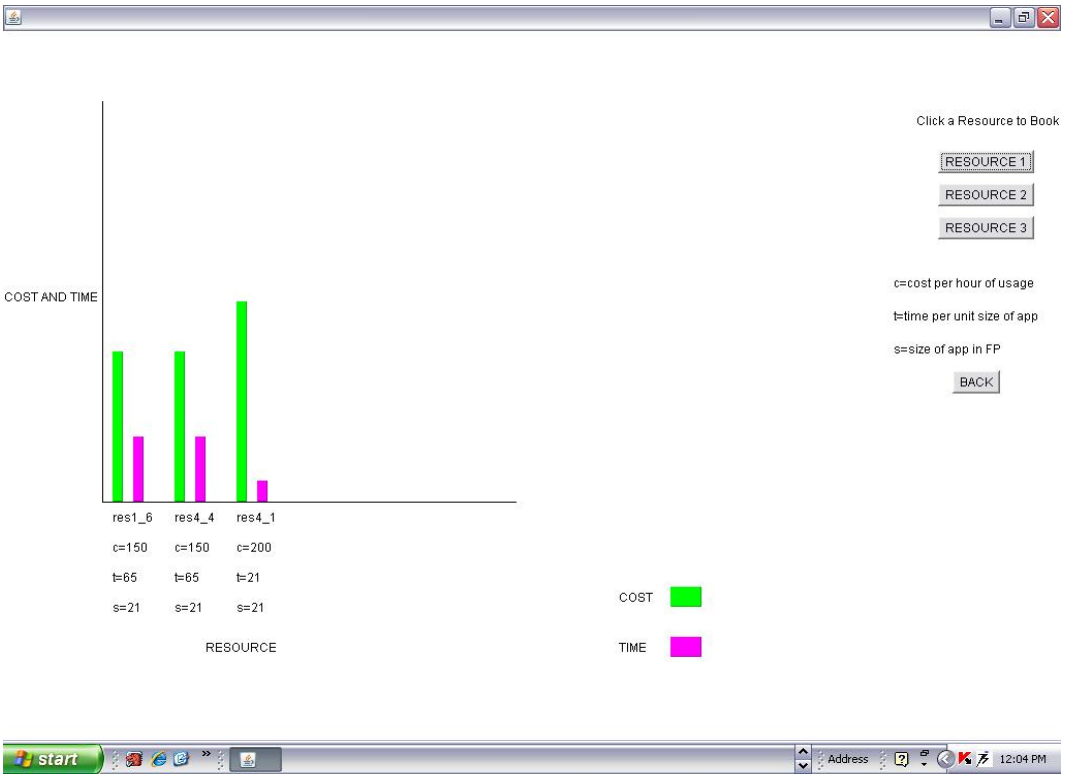

Figure 5.9: Provisioned Sets

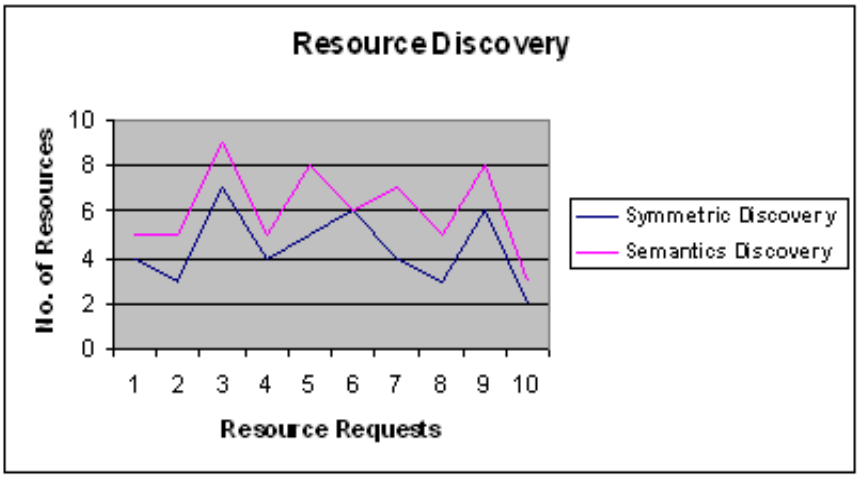

Figure 5.10: Comparison of Symmetric and Semantic based search

Table 5.1: PRATHAM vs. other RMS Frameworks

RMS/Features	***Metascheduler***	***MARS***	***OntoGrid***	***Garuda***	***PRATHAM***
Usage Description	*A metascheduling architecture for GrADS project*	*Part of MGRID initiative*	*Technological infrastructure for the Semantic Grid*	*Nation wide Grid of computational nodes*	*A Grid Resource Management Framework*
OGSA Compliant	*Yes*	*Yes*	*Semantic-OGSA: a Semantic Grid Reference Architecture*	*Yes*	*Yes*
Middleware	*Globus Toolkit*	*Globus, Condor-G*	*The model and capabilities are platform independent. Can be applied to different middleware like GT*	*GT2*	*GT4, Alchemi.NET*
Interoperability	*Not interoperable with other middleware*	*Still in progress, provided through XML interface to some extent*	*Provides interoperable services based on semantics*	*Semantics is being used*	*Semantics is being used for middleware interoperability*
Resource Provisioning	*Provides Resource performance prediction by NWS*	*Supports Resource Forecasting*	*Semantic Provisioning Services for Knowledge Entities and Semantic Bindings*	*Best match and subsumption relation*	*Provides resource QoS , performance prediction and resource optimization*
Autonomic Features	*Not supported*	*Not supported*	*Can be integrated*	*Not supported as yet*	*Supports Autonomic Features, esp. self-management through Pulse Monitoring*

Table 5.2: PRATHAM vs. other RMS Frameworks (Academic-initiatives)

RMS/Features	***White Rose Grid***	***UK e-Science Grid***	***GLOW***	***University of Houston***	***Nordu Grid***	***PRATHAM***
Usage Description	*The three Yorkshire Universities - Leeds, York and Sheffield - have formed the White Rose University Consortium (WRUC) to undertake larger-scale projects than those can be achieved by any one University*	*The UK e-Science Program began in 2001 as a coordinated initiative involving all the Research Councils and the then Department of Trade and Industry*	*The Grid Laboratory of Wisconsin -GLOW is campus wide Grid of University of Wisconsin*	*University of Houston leverages Sun Grid Computing for environmental modeling and seismic imaging*	*The core of the collaboration historically consists of several Nordic academic and research institutes*	*A Grid Resource Management Framework*
OGSA Compliant	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Middleware	*Globus Toolkit 2.4.3: Sun Grid Engine*	*Globus Toolkit 2.4.3*	*Condor*	*Sun Microsystems' Grid infrastructure*	*Advance Resource Connector (ARC middleware)*	*GT4, Alchemi 1.0*
Interoperability	*Perl modules and shell scripts perform the necessary translation of generic Globus-level requests to the equivalent resource scheduler commands understood by SGE.*	*Through Semantics*	*Not supported as yet as only Condor is being used*	*Through EZ-Grid System*	*ARC middleware provides a utility, the norduGridmap script, capable of synchronizing the local Gridmap file on the site with the user lists of requested VOs.*	*Semantics is being used for middleware interoperability*
Resource Provisioning	*Resource Optimization not fully supported*	*Not fully supported*	*Not fully supported*	*As provided by Sun N1*	*Not Supported*	*Provides resource QoS , performance prediction and resource optimization*
Autonomic Features	*Not supported*	*Not fully supported*	*Not fully supported*	*As provided by Sun N1*	*Not Supported*	*Supports Autonomic Features, esp. self-management through Pulse Monitoring*

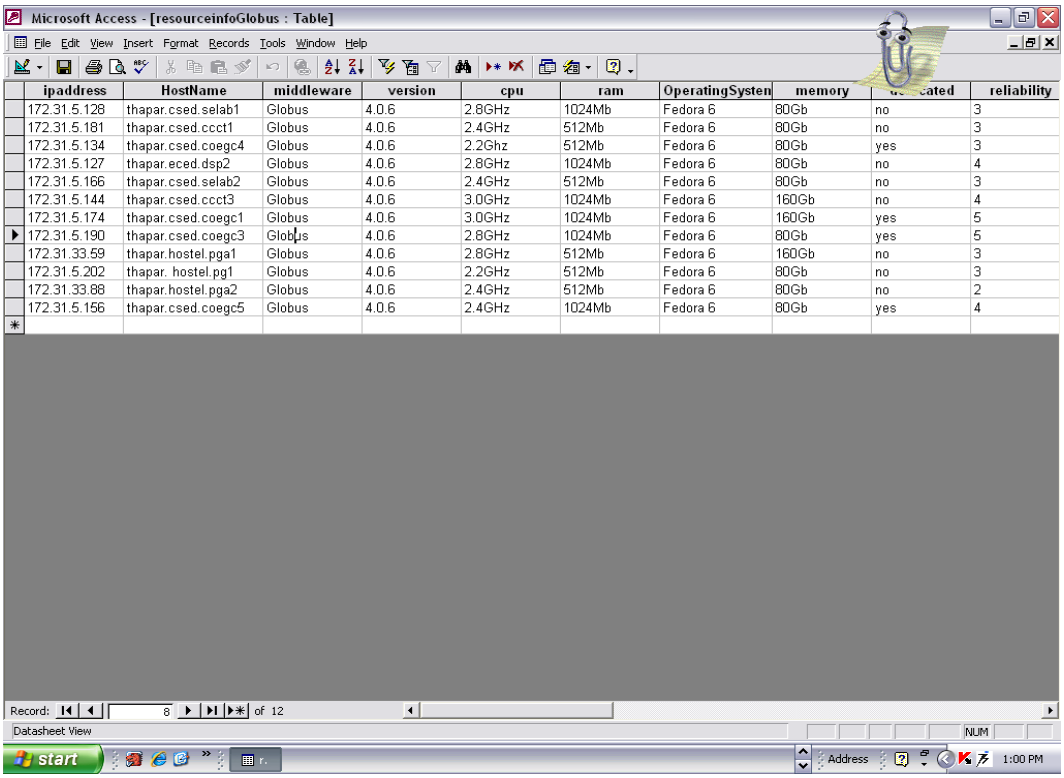

Figure 5.11: Snapshot of Globus Database

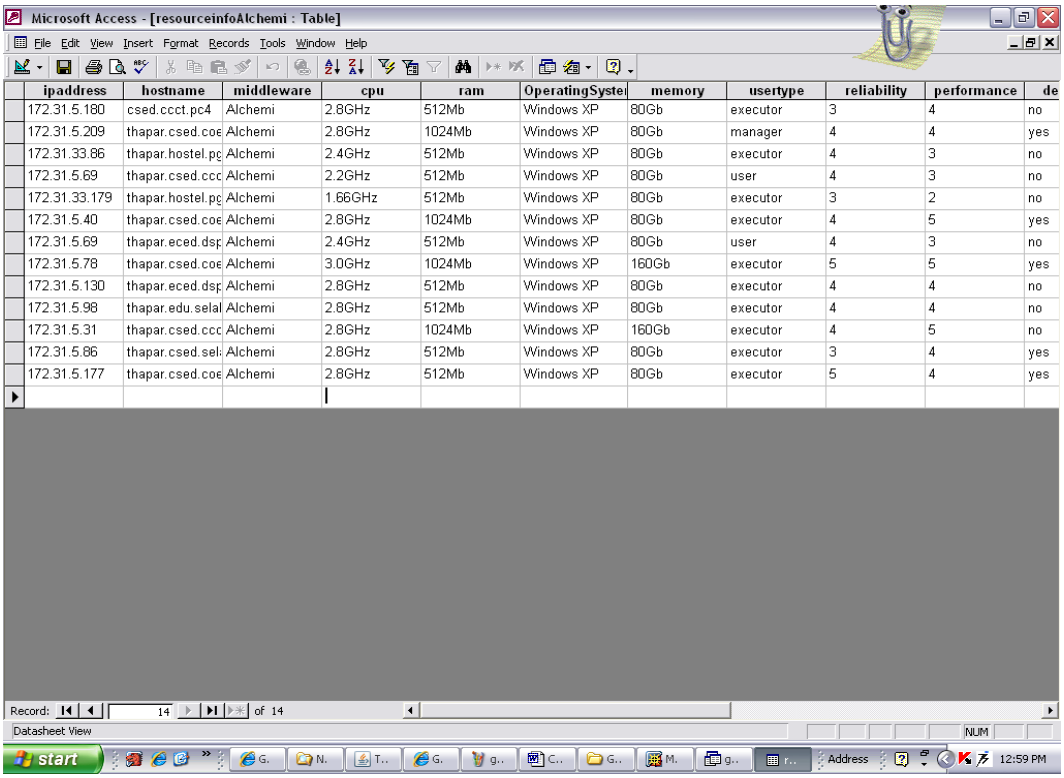

Figure 5.12: Snapshot of Alchemi Database

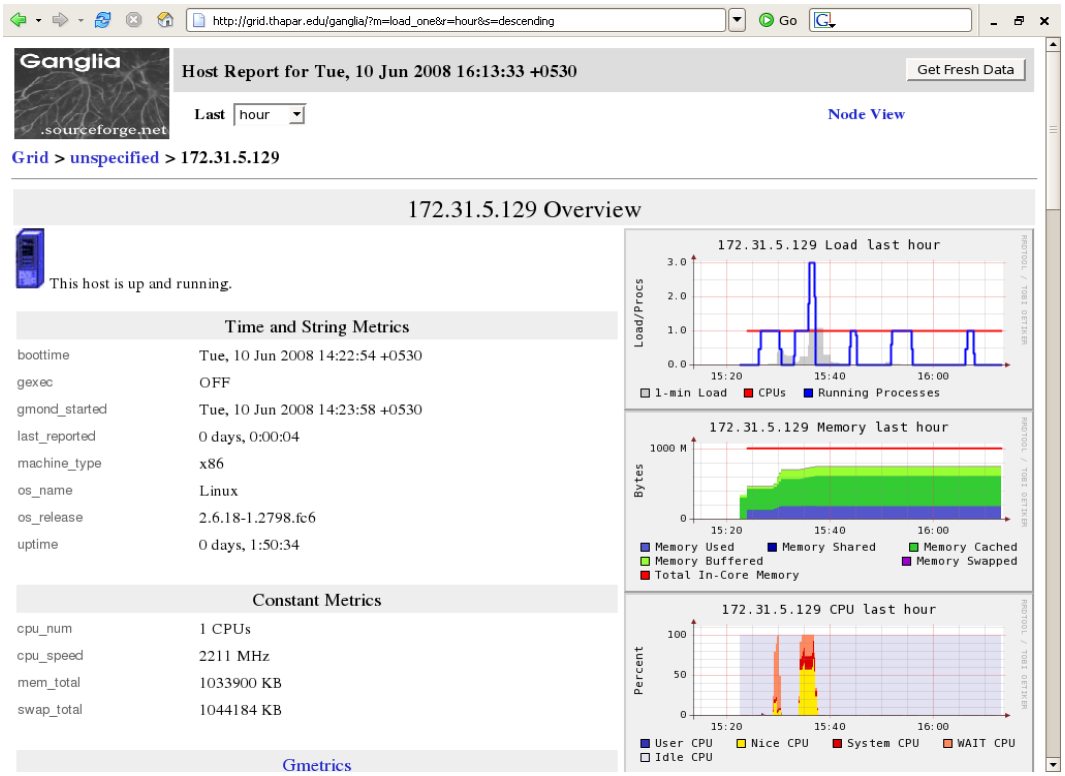

Figure 5.13: Graph generated by Ganglia showing CPU Load

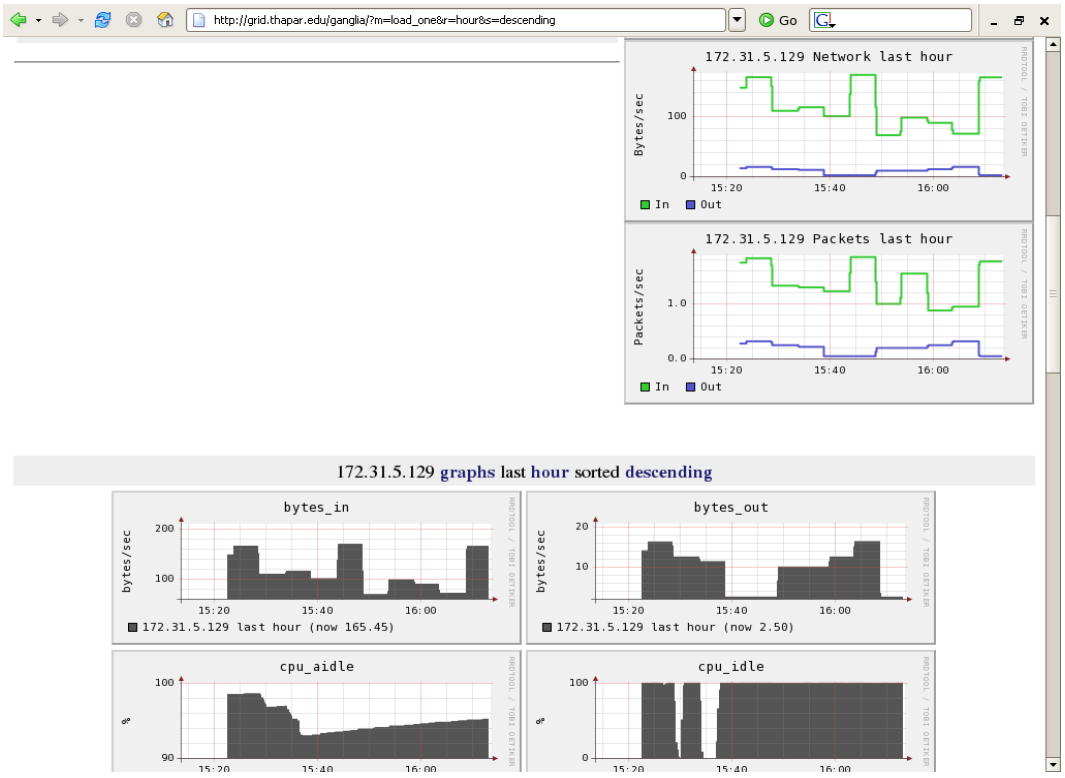

Figure 5.14: Graph generated by Ganglia showing Network Traffic

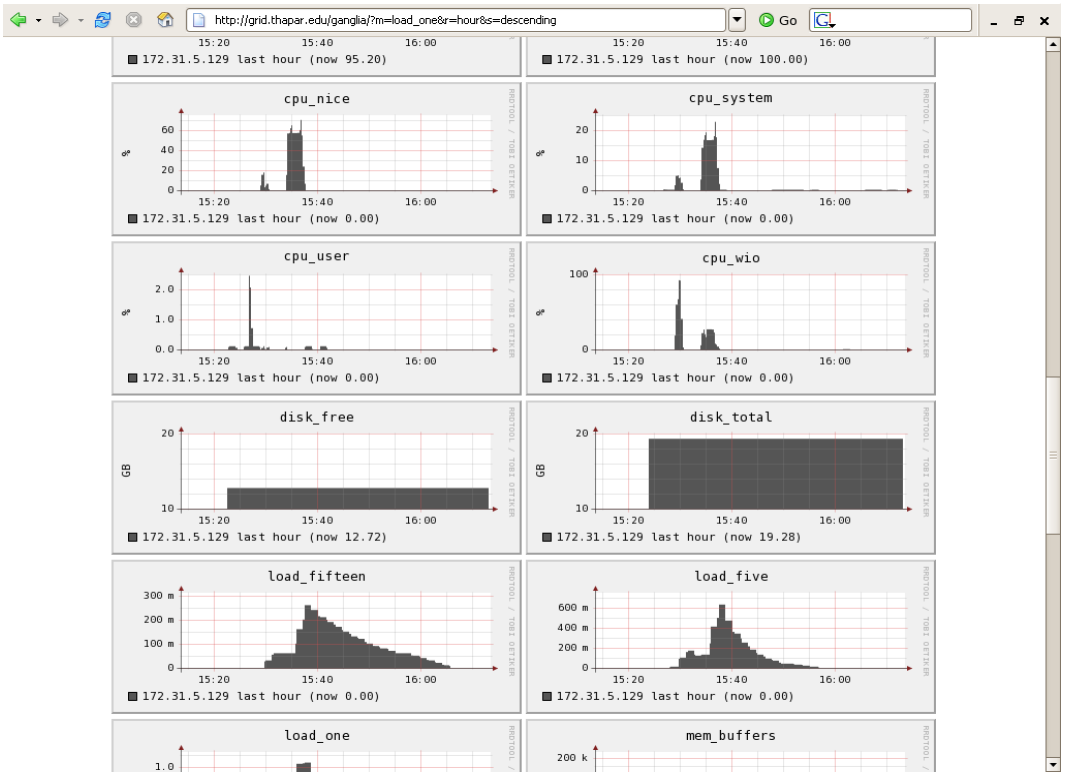

Figure 5.15: Graph generated by Ganglia showing Disk Usage

Figure 5.16: Graph generated by Ganglia showing Memory Usage

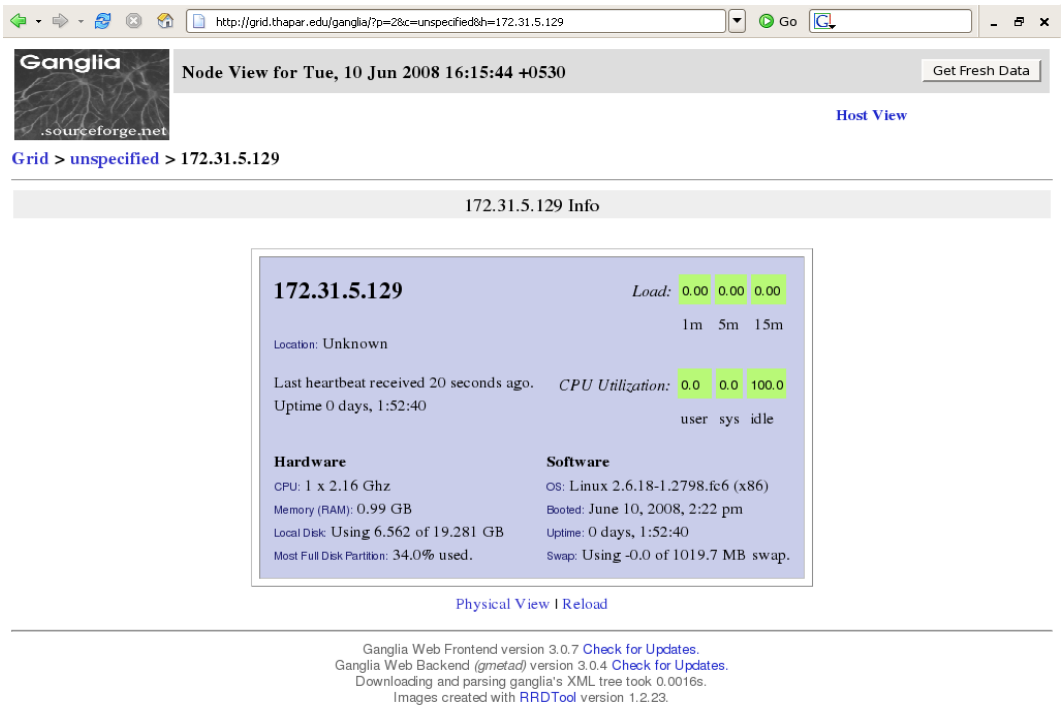

Figure 5.17: Report generated by Ganglia

```
<GANGLIA_XML VERSION="3.0.7" SOURCE="gmond">
<CLUSTER NAME="unspecified" LOCALTIME="1213087677"
OWNER="unspecified" LATLONG="unspecified" URL="unspecified">
<HOST NAME="172.31.5.129" IP="172.31.5.129" REPORTED="1213087676"
TN="0" TMAX="20" DMAX="0" LOCATION="unspecified"
GMOND_STARTED="1213085740">
<METRIC NAME="load_five" VAL="0.00" TYPE="float" UNITS=" " TN="20"
TMAX="325" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_fifteen" VAL="0.00" TYPE="float" UNITS=" " TN="20"
TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="disk_free" VAL="12.720" TYPE="double" UNITS="GB" TN="20"
TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="part_max_used" VAL="34.0" TYPE="float" UNITS="%" TN="20"
TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_wio" VAL="0.0" TYPE="float" UNITS="%" TN="30"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_user" VAL="0.1" TYPE="float" UNITS="%" TN="30"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_nice" VAL="0.0" TYPE="float" UNITS="%" TN="30"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_system" VAL="0.0" TYPE="float" UNITS="%" TN="30"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_idle" VAL="99.9" TYPE="float" UNITS="%" TN="30"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_aidle" VAL="97.0" TYPE="float" UNITS="%" TN="30"
TMAX="3800" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="load_one" VAL="0.00" TYPE="float" UNITS=" " TN="20"
TMAX="70" DMAX="0" SLOPE="both" SOURCE="gmond"/>
</HOST>
<HOST NAME="inderveer.thapar.edu" IP="172.31.5.68" REPORTED="1213087662"
TN="15" TMAX="20" DMAX="0" LOCATION="unspecified"
GMOND_STARTED="1213087621">
<METRIC NAME="disk_total" VAL="63.494" TYPE="double" UNITS="GB"
TN="54" TMAX="1200" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_speed" VAL="2200" TYPE="uint32" UNITS="MHz"
TN="54" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="part_max_used" VAL="17.1" TYPE="float" UNITS="%" TN="54"
TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="swap_total" VAL="1044216" TYPE="uint32" UNITS="KB"
TN="54" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="os_name" VAL="Linux" TYPE="string" UNITS="" TN="54"
TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond"/>
<METRIC NAME="cpu_user" VAL="0.9" TYPE="float" UNITS="%" TN="14"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
<METRIC NAME="cpu_system" VAL="0.2" TYPE="float" UNITS="%" TN="14"
TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond"/>
```

Figure 5.18: XML created by Ganglia

Chapter 6

Conclusions and Future Directions

This chapter gives concluding remarks on the thesis by highlighting the main contributions of this research work. Due to the possible involvement of extremely large number of heterogeneous resources, resource management remains to be a major challenge for Grid Computing. A number of related issues have been identified and addressed. This thesis focuses upon various issues including interoperability, resource provisioning, monitoring and self-management in Grid environments. A resource management framework named PRATHAM has been proposed, which addresses these issues. Further, the framework has been designed, developed and validated in this thesis.

To begin with, this chapter explains the outcome of each chapter and the main contributions of the framework. Later, it focuses upon the future scope of the work.

6.1 Conclusions

The thesis, "A Framework for Resource Management in A Grid Environment" addresses the resource management challenges in Grid Computing. It focuses on the issues of interoperability, resource provisioning, monitoring and self-management aspects of resource management.

Literature Review as presented in Chapter 2 highlights the resource management practices in current Grid middleware. After an exhaustive review, it becomes apparent that no existing RMS, is able to attain the global status by satisfying all the resource management principles. Moreover, the existing middleware like Globus, Alchemi, Sun N1 etc. are not directly interoperable. Only some of the existing resource management frameworks like Glare are providing limited resource-provisioning facility. Study also exhibits that no generic monitoring system exists which is able to notify the details of faults instead of simply reporting the faults. Based on these outcomes the research problem of proposing a resource management framework has been formulated.

To address the issues identified during Literature Review, Chapter 3 analyzes the complete requirements of the proposed framework with the help of UML diagrams and proposes the resource management framework, named PRATHAM. PRATHAM addresses the issues of interoperability, resource provisioning, monitoring and self-management in the Grid. PRATHAM is OGSA compliant and has a five-layer architecture. It is inspired by Semantic Web Technologies and Autonomic Computing. Both these technologies influence the framework and help in achieving the research objectives.

The components of the framework have been detailed in Chapter 4. Base layer of the framework is *Fabric Layer* and comprises of the individual resources, networks, sensors, archives etc. The second layer is the *Grid Middleware Layer*, which comprises of the middleware and its managers. Middleware directly control the resources lying in the Fabric layer. The third layer is the *Broker Layer* and is mainly responsible for the scheduling of jobs on different resources. An Autonomic Manager, comprising of self-monitor, pulse-monitor and an actuator, has been designed in the framework and is part of this layer. The Autonomic Manager helps in self-management of the Grid. The fourth layer is the *Information Layer*, which renders interoperability in resource discovery through

semantics. It thus encapsulates all the internal details of different resource representations along with the search details.

Next, the resource-provisioning unit provides resource provisioning and advance resource reservation services to the end users. Once the resources have been discovered and agreed upon by the end-user, the job information and resource information is passed on to the broker, which handles job scheduling. The top-most layer is *Application Layer,* which provides an interface to the Grid via a Grid portal that has been developed. The user interacts with the Grid through this portal and utilizes portlet services.

Chapter 5 gives the deployment details of the framework by means of a Case Study of Thapar University, Patiala. The experimental results have been shown and the framework has also been validated by comparing it with existing resource management frameworks and RMS. Thus, it can be concluded that the proposed resource management framework, PRATHAM contributes as:

- PRATHAM supports
  - Interoperability through semantics
  - Resource provisioning
  - Advance reservation of resources
  - Efficient monitoring capability
  - Self healing
- Different RMS support either or some of these features but PRATHAM supports them all collectively
- Most of the RMS support single middleware while PRATHAM supports GT4 and Alchemi.Net.
- Most of the RMS are based on heartbeat monitoring where as PRATHAM monitoring system is based on pulse monitoring which helps in self-management of the Grid.
- Efficient resource performance prediction with optimum resource matching support further differentiates PRATHAM from rest of the systems.

6.2 Future Directions

The design, development and testing of the proposed resource management framework has been presented and successfully demonstrated in this thesis. However, the work can be further enhanced and extended in future. Some of the future directions can be:

- The existing framework has been designed and tested for two Grid middleware namely GT4 and Alchemi. In future, this can be extended to some more middleware like Sun N1, Condor etc..
- Currently, PRATHAM is utilizing the load balancing services of individual middleware only. For self-optimization the load-balancing feature for the entire Grid can be designed.
- The framework can be further enhanced and made more comprehensive by introducing configurable security and dynamic load balancing techniques.
- PRATHAM informs the administrator about the exact error or reason of fault but further the administrator has to do the required changes manually. This can be automated. For example, as soon as the load limit exceeds, scheduler should automatically shift the excessive load to under-loaded resources. Similarly, jobs can be automatically migrated suitably in case of resource failures. Thus, the extent of self-management can be further enhanced.
- A tool similar to G-monitor can be integrated in the framework, thus user can be provided with the privilege of monitoring the individual resources and viewing the results in a graphical format.
- Further, economic models can be considered for accounting and billing etc. At present, PRATHAM does not take any economic model into consideration.

There is vital scope of research in this area. This work can be further enhanced to help the Grid users and scientists in achieving reasonable and expeditious solutions for their intricate problems.

References

[1] Foster, I. and Kesselman, C. (eds.), "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, 1999

[2] Luis Ferreira et al., "Grid Computing in Research and Education", ibm.com/redbooks, September 2003

[3] Joshy Joseph and Craig Fellenstein, "Grid Computing", Person Edition, 2004

[4] Ahmar Abbas, "Grid Computing: A Practical Guide to Technology and Applications",Firewall Media, New Delhi, 2004

[5] Wolfgang Leister et al, "Grid and Related Technologies", Norwegian Computing Center, Applied Research And Development Tech Report, ISBN 82-536-508-4, 16th June 2004, available at http://publications.nr.no/grid-report.pdf

[6] LHC Computing Centres at CERN, Geneva, details available at http://press.web.cern.ch/press/PressReleases/Releases2005/PR06.05E.html

[7] "About Grid Computing", article available at www.nwfusion.com/links/gridcomputing

[8] Details of Griphyn project available at www.griphyn.org

[9] Details of NeesGrid project available at www.mgrid.umich.edu/projects/neesgrid.html

[10] Details of GridForum available at www.gridforum.org

[11] Mark Linesch, "Grid - Distributed Computing at Scale, An overview of Grid and the Open Grid Forum", Open Grid Forum Document, GFD-I.112, August 28, 2007

[12] M. O'Reilly, O'Reillys Peer-to-Peer Summit Web pages, //http://www.oreillynet.com/pub/a/linux/2000/ 2000.

[13] Fran Berman, Geoffrey Fox, and Tony Hey, "The Grid: past, present, future", Copyright 2003 John Wiley Sons, Ltd.

[14] Foster, I. "Internet Computing and the Emerging Grid", Nature Web Matters, 2000 http://www.nature.com/nature/webmatters/grid/grid.html.

[15] Grid Computing Info Centre, articles available at www.gridcomputing.com

[16] Rajkumar Buyya, "Economic based distributed resource management and scheduling for Grid Computing", available at www.buyya.com/papers/

[17] Ian Foster, Carl Kesselman, Steven Tuecke, "The Anatomy of the Grid Enabling Scalable Virtual Organizations", Intl J. Supercomputer Applications, 15(3):200-222,2001.

[18] Martin Brown, "Comparing traditional grids with high-performance computing", article published by IBM developerworks, 13 Jun 2006, http://www-128.ibm.com/developerworks/grid/library/gr-tradhp/index.html

[19] Daniel F. Saffioti, "Grid Computing Fundamentals, Trends and Research", available at http://www.cse.iitb.ac.in/ joytechie/backup/seminar/papers/slides/Saffioti-GridComputingSlides.pdf

[20] Maozhen Li, Mark Baker, "The Grid Core Technologies", John Wiley Sons, 2005

[21] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, "The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[22] H. Kishimoto, J. Treadwell, "Defining the Grid: A Roadmap for OGSA Standards", GFD-I.053, Open Grid Services Architecture Working Group, September 16, 2005

[23] J.Nabrzyski, J.M. Schopf, and J. Weglarz, "Grid Resource Management: State of the Art and Future Trends", Kluwer Academic Publishers, Boston, 2003.

[24] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems", Software-Practice And Experience, Softw. Pract. Exper. 2002; 32:135-164 (DOI: 10.1002/spe.432)

[25] U. Schwiegelshohn, R. Yahyapour, "Resource Management for Future Generation Grids", CoreGRID Technical Report, TR-0005, May 19, 2005

[26] The Global Grid Forum, Grid Scheduling Architecture Research Group (GSA-RG), Web site, 2005. Online: https://forge.gridforum.org/projects/gsa-rg/.

[27] Jyotishman Pathak, Jem Treadwell, Raj Kumar, Philip Vitale, Fernando Fraticelli, "A Framework for Dynamic Resource Management on the Grid", A technical report, HPL-2005-153, Aug 22, 2005

[28] Karan Bhatia, Per Brand, Sergio Mendiola, "Peer-To-Peer Requirements on the Open Grid Services Architecture Framework," GGF document series, 2005, http://www.ggf.org/documents/final.htm.

[29] Frederico Buchholz Maciel ,"Resource Management in OGSA", http://forge.gridforum.org/projects/cmm-wg/, GFD-I.045, March 2005

[30] Arshad Ali, Ashiq Anjum, Atif Mehmood, Richard McClatchey, Ian Willers, Julian Bunn,

Harvey, “A Taxonomy and Survey of Grid Resource Planning and Reservation Systems for Grid Enabled Analysis Environment”, http://arxiv.org/ftp/cs/papers/0407/0407012.pdf

[31] Harchol-Balter, M., Leighton, T. and Lewin, D., “Resource discovery in distributed networks,” ACM Symposium on Principles of Distributed Computing, May 1999, pp. 229-237.

[32] Iamnitchi, A. and Foster, I., “On Fully Decentralized Resource Discovery in Grid Environments”, Lecture Notes in Computer Science, 2242:51-62, 2001.

[33] A4 Methodology, http://www.dcs.warwick.ac.uk/research/hpsg/A4/A4.html

[34] Iamnitchi, A., Foster, I., Nurmi, D., “A Peer-to-Peer Approach to Resource Discovery in Grid Environments,” Proc. Of the 11th Symposium on High Performance Distributed Computing, Edinburgh, UK, 2002.

[35] Maheswaran, M. and Krauter, K. “A Parameter-based approach to resource discovery in Grid computing systems”, 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), December 2000, Bangalore, India.

[36] Phil Schater, “Resource provisioning, Interoperability, and XML”, http://xml.coverpages.org/XRPM-ProvForum091001.pdf,

[37] Ronald P. Doyle, Jeffrey S. Chase,Omer M. Asad, Wei Jin,Amin M. Vahdat , “Model-Based Resource Provisioning in a Web Service Utility”, http://issg.cs.duke.edu/publications/mbrp-usits03.pdf

[38] Mumtaz Siddiqui, Alex Villazon, Jurgen Hofer and Thomas Fahringer, “GLARE: A Grid Activity Registration, Deployment and Provisioning Technique”, http://vtcpc.isi.edu/wiki/images/e/e7/Askalon3.pdf.

[39] Thomas Lehman, Jerry Sobieski, Bijan Jabbari, “DRAGON: A Technique for Service Provisioning in Heterogeneous Grid Networks,” http://ieeexplore.ieee.org/iel5/35/33764/01607870.pdf

[40] Gurmeet Singh, Carl Kesselman, Ewa Deelman., “Application-level Resource Provisioning on the Grid”, http://pegasus.isi.edu/publications/SinghAppResProvisioning.pdf

[41] James Frey, Todd Tannenbaum, Miron Livny Ian Foster, Steven Tuecke, “Condor-G: A Computation Management Agent for Multi-Institutional Grids”, http://www.cs.wisc.edu/condor/doc/condorg-hpdc10.pdf.

[42] Brian Tierney, Brian Crowley, Dan Gunter, Jason Lee And Mary Thompson, Computing Sciences Directorate, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA 94720, USA, “A Monitoring Sensor Management System for Grid Environments”, Cluster Computing 4, 19-28, 2001, Kluwer Academic Publishers. Manufactured in The Netherlands.

[43] Hee-Khiang Ng, Quoc-Thuan Ho, Bu-Sung Lee, Dudy Lim, Yew-Soon Ong, Wentong Cai1 "Nanyang Campus Inter-Organization Grid Monitoring System", http://www.ntu.edu.sg

[44] Karl Czajkowski, Steven Fitzgeraldt, Ian Foster, Carl Kesselman, "Grid Information Services for Distributed Resource Sharing" 0-7695-1296-8/01 2001 IEEE

[45] Serafeim Zanikolas, Rizos Sakellariou, "A Taxonomy of Grid Monitoring Systems", Future Generation Computer Systems 21 (2005) 163-188

[46] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, "A Grid Monitoring Architecture", GWDPerf-16-3, Global Grid Forum, August 2002. http://wwwdidc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf.

[47] The Globus Heartbeat Monitor Specification v1.0. [Online], http://www.fp.globus.org/hbm/heartbeat_spec.html

[48] D. DeCoste, S. G. Finley, H. B. Hotz, G. E. Lanyi, A. P. Schlutsmeyer, R. L. Sherwood, M. K. Sue, J. Szijjarto, and E. J.Wyatt, "Beacon monitor operation experiment DS1 technology validation report" , 2000.

[49] R. Sterritt, "Pulse monitoring: Extending the health-check for the Autonomic Grid," in the proceedings of IEEE workshop on autonomic computing principles and architectures (AUCOPA) at INDIN 2003, Banff, Alberta, Canada, 2223 August 2003 p. 43340.

[50] Tanaka, Y., Sato, M. Hirano, M., Nakada, H. and Sekiguchi, S. (1999) , "Resource Manager for Globus-based Wide-area Cluster Computing", Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing (Page: 237 Year of Publication: 1999 ISBN: 0-7695-0343-8)

[51] Ian Foster and Carl Kesselman, "Globus: A metacomputing infrastructure toolkit". Intl. J. Supercomputer Applications, 11(2):115-128, 1997.

[52] Globus Alliance, 2003, www.globus.org

[53] Resource Specification Language, http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html

[54] Globus Resource Allocation Manager (GRAM), http://www-unix.globus.org/api/c-globus 2.2/globus_gram_documentation/html/index.html

[55] "Globus-duroc, The Dynamically-Updated Request Online Coallocator (DUROC)", http://www.globus.org/toolkit/docs/2.4/duroc/

[56] UNICORE, 2001, http://www.unicore.de/. D. Erwin (Ed.), UNICORE Plus final Report - Uniform Interface to Computing Resources, Forschungszentrum Jlich, 2003

[57] UNICORE History available at http://www.unicore.eu/unicore/history.php

[58] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venu-

gopal, “Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies”, June 2005, http://www.buyya.com/

[59] Andrew S. Grimshaw, Anand Natrajan, Marty A. Humphrey, Michael J. Lewis, et.al , “From Legion to Avaki: the persistence of vision, Grid Computing: Making the Global Infrastructure a Reality”, eds. Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey, John Wiley Sons, pp.265-298, March 2003, ISBN 0-470-85319-0.

[60] Anand Natrajan, Marty A. Humphrey, and Andrew S. Grimshaw, Grid Resource Management in Legion, available at www.cs.virginia.edu/papers/GSRM03.pdf

[61] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan Srikumar Venugopal, “Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids ”, Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2003.

[62] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, “Alchemi: A .NET-Based Enterprise Grid Computing System”, Proceedings of the 6th International Conference on Internet Computing (ICOMP’05), June 27-30, 2005, Las Vegas, USA.

[63] Charu Chaubal, “Scheduler Policies For Job Prioritization In The Sun N1 Grid Engine 6 System ”, N1 Systems, Sun BluePrints OnLine - October 2005

[64]SUN Grid Engine, http://www.sun.com/service/grid/offerings.jsp

[65] GRID COMPUTING on SUN Deploying Scalable and Efficient HPC Grids in the Modern Commercial or Technical Enterprise, White Paper, March 2006 available at http://www.sun.com/servers/wp/

[66] Grid Cmputing and SUN N1 Grid Engine, Jim Gutowski www.sun.com/sungrid

[67] Condor Project at http://www.cs.wisc.edu/condor/

[68] Douglas Thain and Miron Livny, “Building Reliable Clients and Servers”, in Ian Foster and Carl Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2003, 2nd edition. ISBN: 1-55860-933-4.

[69] Jeff Mausolf, “An eagle-eye view of the Condor project”, 15 Feb 2005 available at IBM DeveloperWorks, http://www-128.ibm.com/developerworks/library/gr-condor/

[70] Maozhen Li, Mark Baker, “The Grid Core Technologies”, John Wiley Sons, 2005, pp5-6

[71] David De Roure and others, “The Semantic Grid: Past, Present and Future”, Proceedings of the 2nd Annual European Semantic Web Conference (ESWC2005),Volume 93, Issue 3, 2005.

[72] Semantic Web details available at www.SemanticWeb.org

[73] “Practical Autonomic Computing: Roadmap to Self Managing Technology”, A White Paper Prepared by Enterprise Management Associates, for IBM, January 2006

[74] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing", IEEE Computer, Vol. 36, No. 1, 41-50, 2001.

[75] "About IBM Autonomic Computing", article available at
http://www03.ibm.com/autonomic/about_get_model.html

[76] A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," IBM System Journal, vol. 42, no. 1, 2003.

[77] Chen, L, Shadbolt, N, Goble, C, Tao, F, Puleston, C, and Cox S.J., "Semantics-Assisted Problem Solving on the Semantic Grid," Computational Intelligence, Vol.21, No.2, 2005, pp.157-176, http://www.geodise.org/files/Papers/ComputationalIntelligencePaper.pdf

[78] Tangmunarunkit, H., Decker, S. and Kesselman,.C., "Ontology Based Resource Matching in the Grid - The Grid Meets the Semantic Web". In Proc. of the 2nd International Semantic Web Conference, 706-721, 2003.

[79] Roy Sterritt and David F. Bantz , "Personal Autonomic Computing Reflex Reactions and Self-Healing", IEEE Transactions On Systems, Man, And Cybernetics-Part C: Applications And Reviews, Vol. 36, No. 3, May 2006

[80] "Autonomic Management of Large Clusters and Their Integration into the Grid", Journal of Grid Computing (2004) 2: 247-260

[81] Vishal Iyengar, Sameer Tilak, Michael J. Lewis and Nael B. Abu-Ghazaleh, "Non-Uniform Information Dissemination for Dynamic Grid Resource Discovery",
www.cs.binghamton.edu/ sameer/pubs/NCA2004.pdf, 2004

[82] Rajkumar Buyya, Steve Chapin and David DiNucci "Architectural Models for Resource Management in the Grid", Lecture Notes In Computer Science; Vol. 1971 Proceedings of the First IEEE/ACM International Workshop on Grid Computing, Pages: 18 - 35, Year of Publication: 2000, ISBN: 3-540-41403-7

[83] Mark Linesch, "Grid - Distributed Computing at Scale An overview of Grid and the Open Grid Forum", GFD-I.112, August 28, 2007

[84] "How does Grid Computing Work", article available at
http://gridcafe.web.cern.ch/gridcafe/gridatwork/gridatwork.html

[85] SNF-center for Grid Computing, http://www.dcgc.dk/downloads/gridcenterapplication.pdf

[86] Open Grid Forum, OGF, http://www.ogf.org/

[87] Von Welch, Frank Siebenlist, Ian Foster, John Bresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, Steven Tuecke, "Security for Grid Services",
http://www.globus.org/alliance/publications/papers/GT3-Security-HPDC.pdf

[88] Michael Di Stefano, Distributed Data Management for Grid Computing, ISBN: 978-0-471-68719-1, Wiley Interscience, July 2005

[89] Rajkumar Buyya, "Economic-based Distributed Resource management and Scheduling for Grid Environment", Ph.D. thesis available at www.buyya.com

[90] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models", IBM Systems Journal, volume 43, Number 4, 2004

[91] P.N.J. Bouwmeester, "GridAssist Middleware Interoperability" , Master Thesis submitted in Delft University of Technology, the Netherlands , September 4, 2005, available at www.ewi.tudelft.nl

[92] Karl Czajkowski, Ian Foster, And Carl Kesselman, "Agreement-Based Resource Management", Proceedings Of The Ieee, Vol. 93, No. 3, March 2005

[93] Article in IndiaTimes Infotech, "Will Net become obsolete soon", 7 Apr 2008, 0931 hrs IST, TNN, http://infotech.indiatimes.com/articleshow/2931736.cms

[94] Keith Norman, Grid Computing, Issue V1.R1.M0 April 2003, available at http://www.tessella.com/literature/supplements/pdf/gridcomputing.pdf

[95] Dennis Szubert, Principal Analyst, Quocirca Ltd, "Grid Computing Now! - The future of grid", available at http://www.quocirca.com/pages/analysis/articles/view/dl/store251/item21170/, 01.03.2008

[96] Daniel Minoli, "A Networking Approach to Grid Computing", John Wiley Sons, 2005

[97] Varun Aggarwal, "Computing in the clouds", available at http://www.expresscomputeronline.com/20080218/technology01.shtml

[98] Chen, L, Shadbolt, N, Goble, C, Tao, F, Puleston, C, and Cox S.J., "Semantics-Assisted Problem Solving on the Semantic Grid," Computational Intelligence, Vol.21, No.2, 2005, pp.157-176, http://www.geodise.org/files/Papers/ComputationalIntelligencePaper.pdf

[99] Oscar Corcho , Pinar Alper, Ioannis Kotsiopoulos, Paolo Missier, Sean Bechhofer, Carole Goble, "An overview of S-OGSA: A Reference Semantic Grid Architecture" , Web Semantics: Science, Services and Agents on the World Wide Web 4 (2006) 102-115, Journal of Web Semantics

[100] Tianfield Huaglory, Unland Rainer ,"Towards self-organization in multi-agent systems and Grid computing", Multiagent and Grid Systems archive Volume 1 , Issue 2 (April 2005), 89 - 95 , 2005 , ISSN:1574-1702

[101] Xuan Lin, Ying Lu, Jitender Deogun, and Steve Goddard, "Multi-Round Real-Time Divisible Load Scheduling for Clusters", Proceedings of the 12th IEEE Real-Time and Embedded

Technology and Applications Symposium Work in Progress , pp. 28-31, April 2006.

[102] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, Rajkumar Buyya,"Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications", E-SCIENCE Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, 151-159 ,2007 , ISBN:0-7695-3064-8

[103] Fabrizio Gagliardi1, Bob Jones1, Mario Reale2, Stephen Burke, European DataGrid Project: Experiences of deploying a large scale Testbed for e-Science applications, http://eu-datagrid.web.cern.ch/eu-datagrid/, LNCS, 978-3-540-44252-3, 255-264,2002

[104] Karl Czajkowski et.al, "From Open Grid Services Infrastructure to WS Resource Framework: Refactoring Evolution", 3/05/2004, http://www.chinagrid.net/dvnews/show.aspx?id=983cid=40

[105] Rahul Banerjee, "Recent Advances in Computing: A Pervasive Computing Perspective", Keynote Address, Frontier-2007, Dec 17, 2007.

[106] Xuan Lin, Ying Lu, Jitender Deogun, and Steve Goddard, "Real-time Divisible Load Scheduling for Cluster Computing", published in the Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'07), 2007 IEEE

[107] Jia Yu, Srikumar Venugopal, and Rajkumar Buyya, "Grid Market Directory: A Web Services based Grid Service Publication Directory", arxiv.org/pdf/cs/0302006

[108] P.N.J. Bouwmeester, "GridAssist Middleware Interoperability", MASTER OF SCIENCE Thesis, September 4, 2005, available at swerl.tudelft.nl/twiki/pub/Main/KevinBouwmeester/kevin-bouwmeester-sept-2005.pdf

[109] L. Mohammad Khanli M. Analoui, "Grid-JQA a New Architecture for QoS-guaranteed Grid Computing System", Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06), 2006 IEEE

[110] Savina Bansal, Padam Kumar and Kuldip Singh, , "Dealing with heterogeneity through limited duplication for scheduling precedence constrained task graphs", Journal of Parallel and Distributed Computing Volume 65, Issue 4, April 2005, Pages 479-491

[111] Harpreet Singh, L. Anneberg, R. Kaushal, and N. Chamas "Determination of software reliability," 21st Pittsburgh conference on modeling and simulation, May 3-4, 1990

[112] Sudeepta Das, "Investigations into Performance Evaluation of Fabric level and Application level QoS Guarantees in Grid Environment", DISSERTATION, Birla Institute of Technology and Science Pilani, Rajasthan, India (December 2005)

[113] Aram Galstyan, Karl Czajkowski, and Kristina Lerman, "Resource Allocation in the Grid Using Reinforcement Learning", AAMAS'04, July 19-23, 2004, New York, USA, Copyright 2004

ACM

[114] Carole Goble, David De Roure, "Semantic Web and Grid Computing", available at www.semanticgrid.org/documents/swgc/swgc-final.pdf

[115] Thamarai Selvi Somasundaram1, R.A.Balachandar1, Vijayakumar Kandasamy1, Rajkumar Buyya2,Rajagopalan Raman3, N.Mohanram4 and S.Varun1, "Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker", available at http://www.buyya.com/papers/SG-Discovery-ADCOM2006.pdf

[116] Edward Benson, Glenn Wasson and Marty Humphrey, "Evaluation of UDDI as a Provider of Resource Discovery Services for OGSA-based Grids", Parallel and Distributed Processing Symposium, 2006, IPDPS 2006. 20th International Publication Date: 25-29 April 2006 On page(s): 9 pp- ISBN: 1-4244-0054-6

[117] Nickolas J. G. Falkner, Paul D. Coddington, and Andrew L. Wendelborn, "Using Ontologies to Support Customisation and Maintain Interoperability in Distributed Information Systems with Application to the Domain Name System", Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKG'06),2006 IEEE

[118] Sathish S. Vadhiyar and Jack J. Dongarra, "A Metascheduler For The Grid", available at www.netlib.org/utk/people/JackDongarra/PAPERS/hpdc_meta.pdf

[119] Raj Kumar, Vanish Talwar, Sujoy Basu, "A Resource Management Framework For Interactive Grids", 1st International Workshop on Middleware for Grid Computing, 17 June 2003, Rio de Janeiro, Brazil, Copyright Hewlett-Packard Company 2003

[120] Junwei Caoa, Stephen A. Jarvisa, Subhash Sainib, Darren J. Kerbysonc and Graham R. Nudda, "ARMS: An agent-based resource management system for grid computing", Scientific Programming 10 (2002) 135-148, ISSN 1058-9244 2002, IOS Press

[121] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing", available at www.buyya.com/papers/emodelsgrid.pdf

[122] D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini and G.R. Nudd, "Local grid scheduling techniques using performance prediction", IEEE Proc.-Comput. Digit. k h . , Vol. 1.70, No. 2, March 2003

[123] Johan Tordsson, "Resource Brokering for Grid Environments", Master's Thesis, 14th June 2004, available at www.nordugrid.org/documents/tordsson-thesis.pdf

[124] John Brooke, Donal Fellows, Jon MacLaren, "Resource Brokering: The EUROGRID/GRIP Approach", information available at www.unigrids.org/papers/brokerAHM04.pdf

[125] Chuang Liu, Lingyun Yang, Ian Foster, Dave Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications", High Performance Distributed Computing, 2002, (HPDC-11 2002 Proceedings), 11th IEEE International Symposium on Publication Date: 2002, On page(s): 63- 72, ISSN: 1082-8907 , ISBN: 0-7695-1686-6

[126] Marios D. Dikaiakos, Rizos Sakellariou, Yannis Ioannidis, "Information Services for Large-Scale Grids A Case for a Grid Search Engine", CoreGRID Technical Report Number TR-0009, May 24, 2005

[127] Manolis Koubarakis, Zoi Kaoudi, Iris Miliaraki, Matoula Magiridou and Antonios Papadakis-Pesaresi, "Semantic Grid Resource Discovery using DHTs in Atlas", available at www.ontogrid.net/ontogrid/servlet/

[128] Gurmeet Singh, Carl Kesselman, Ewa Deelman, "A provisioning model and its comparison with best-effort for performance-cost optimization in grids", High Performance Distributed Computing, Proceedings of the 16th international symposium on High performance distributed computing, ACM, Pages: 117 - 126 , Year of Publication: 2007, ISBN:978-1-59593-673-8

[129] Mark Baker, Rajkumar Buyya and Domenico Laforenza, "Grids and Grid technologies for wide-area distributed computing", Software-Practice And Experience, Softw. Pract. Exper. 2002; (DOI: 10.1002/spe.488)

[130] Paritosh Chandrapal and Padam Kumar, " A Scalable Multi-level Distributed System-Level Diagnosis", Lecture Notes in Computer Science, 192-202, Distributed Computing and Internet Technology, 2005, ISBN 978-3-540-30999-4

[131] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski , " A Fault Detection Service for Wide Area Distributed Computations", Retreat 1998 available at ftp://ftp.globus.org/pub/globus/papers/hbm.pdf

[132] B. A. Shirazi, A. R. Hurson, K. M. Kavi, "Chapter 5: Load Balancing, Scheduling and Load Balancing", Parallel and Distributed Systems, IEEE Press: Washington, 1995.

[133] "Towards Virtual Networks for Virtual Machine Grid Computing", Virtuso available at http://virtuoso.cs.northwestern.edu

[134] Abhijit Bose, Brian Wickman, Cameron Wood, "MARS: A Metascheduler for Distributed Resources in Campus Grids" , Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04) pp. 110-118

[135] Ya-Ping Zhang , Jizhou Sun, Jian-Bo Ma, "Self-Management Model Based On Multiagent And Worm Techniques", CCECE 2004- CCGEI 2004, Niagara Falls, May 2004, 2004 IEEE

[136] Chunfeng wang, Xiuzhong Chen, Ru Li, Xiaolu Huang, Zhongcheng Li, "Effective Re-

source Provision for real-time applications in Static-Priority Scheduling Networks", Proceedings of ICCT2003, http://ieeexplore.ieee.org/iel5/8586/27227/01209823.pdf

[137] Huang, Y., Venkatasubramanian, N., "QoS-based resource discovery in Intermittently available environments," Proc. of 11th IEEE International Symposium on High Performance Distributed Computing, 50-59, 2002.

[138] Raman, R., Livny, M. and Solomon, M., "Matchmaking: Distributed Resource Management for High Throughput Computing", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL

[139] Grid Interoperability project, details available at http://www.grid-interoperability.org

[140] Laszewski, G. von and Foster, I. "Usage of LDAP in Globus"
http://www.globus.org/pub/globus/papers/ldap_in_globus.pdf

[141] Martin Placek, Rajkumar Buyya, "G-Monitor: A Web Portal for Monitoring and Steering Application Execution on Global Grids," CLADE, p. 10, International Workshop on Challenges of Large Applications in Distributed Environments, 2003

[142] Geoffrey Fox, David Walker, "e-Science Gap Analysis", 30 June 2003, UK e-Science Technical Report Series, ISSN 1751-5971

[143] Rahul Banerjee, "Ubiquitous Computing Technologies and Their Future Potential", Invited Paper, Special Session organized by C-DAC on Ubiquitous Computing, ELITEX-2008, New Delhi, January 18, 2008.

[144] Harpreet Singh, J.S.Bedi, "Distributed Processing for Real Time Applications," 1984 International Conference on Industrial Electronics, Control and Instrumentation, October 22-26, 1985, Tokyo, Japan.

[145] M Venkateswara Reddy , "Vishwa : A Scalable Reconfigurable P2P Middleware for Grid Computing", information available at dos.iitm.ac.in/Vishwa/Vishwaslides.pdf

[146] P. Wieder, W. Ziegler,"Bringing Knowledge to Middleware -Grid Scheduling Ontology", CoreGRID Technical Report Number TR-0008, May 19, 2005, URL: http://www.coregrid.net

[147] David De Roure and others, "The Semantic Grid: Past, Present and Future", Proceedings of the 2nd Annual European Semantic Web Conference (ESWC2005), Volume 93, Issue 3, 2005

[148] Fitzgerald, S., Foster, I., Kesselman, C., Laszewski, G. Von, Smith W., and Tuecke S., "A Directory Service for Configuring High-Performance Distributed Computations," In Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing, pages 365-375, IEEE Computer Society, 1997

[149] "Grid Brokers and Metaschedulers Market Overview", A gridwise tech report available at

www.gridwisetech.com/metaschedulers

[150] C. Molina-Jimenez, J. Pruyne, and A. van Moorsel. "The Role of Agreements in IT Management Software In Architecting Dependable Systems III", LNCS 3549, pages 36-58, 2005.

[151] S. Vadhiyar and J. Dongarra. "Self Adaptivity in Grid Computing". Concurrency and Computation: Practiceand Experience, 17(2):235-257, 2005.

[152] Raissa Medeiros, Walfredo Cirne, Francisco Brasileiro, Jacques Sauv, "Faults in Grids: Why are they so bad and what can be done about it?" GRID'03 Proceedings of Fourth International Workshop, 17 Nov. 2003, pp18- 24, ISBN: 0-7695-2026-X

[153] Paul Townend and Jei Xu, "Fault Tolerance within a Grid Environment", in the proceedings of AHM2003, available at http://citeseer.ist.psu.edu/townend03fault.html

[154] Hawkeye Team, "Hawkeye, Monitoring and Management Tool for Distributed Systems", 2004 available at http://www.cs.wisc.edu/condor/hawkeye

[155] Information about Ganglia, available at http://ganglia.info/

[156] "Ganglia distributed monitoring and execution system", March 2004, http://ganglia.sourceforge.net/.

[157] RFC 1014 - XDR: External Data Representation Standard, June 1987, http://www.faqs.org/rfcs/rfc1014.html.

[158] RRDTool, June 2004, details available at http://www.caida.org/tools/utilities/rrdtool/.

[159] Jennifer M. Schopf and Ben Clifford, "Monitoring Clusters and Grids", Cluster World Magazine, May 2004.

[160] About R-GMA, details are available at http://www.r-gma.org

[161] Satish Tadepalli, Calvin Ribbens, Srinidhi Varadarajan, "GEMS: A Job Management System for Fault Tolerant Grid Computing", High-Performance Computing Symposium, ISBN:1-56555-278-4

[162] http://www.inteligrid.com/

[163] http://www.mygrid.org.uk/

[164] OGF Areas and Groups,http://www.ogf.org/gf/group_info/view.php?group=sem-rg

[165] Details of RDF, available at http://www.w3.org/RDF

[166]Details of OWL, available at http://www.w3.org/2004/OWL

[167] Roger S. Pressman, "Software Engineering, A Practitioner's Approach", Tata McGrwHill, 6th edition, pp. 763-764

[168] D. Gannon, G. Fox, M. Pierce, B. et al., "Grid Portals: A Scientist's Access Point for Grid Services (DRAFT 1)", can be accessed from http://www.computingportals.org/meetings/ggf9/docs/GCE-Portal-working-draft.pdf

[169] Barbera, R., "The GENIUS Grid Portal," Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.

[170] Allan, R., Integrated e-Science Environment for CLRC, http://esc.dl.ac.uk/IeSE

[171] JSR168 Portlet Specification
http://jcp.org/aboutJava/communityprocess/review/jsr168/

[172] NCSA Alliance Scientific Portal Project, http://www.extreme.indiana.edu/alliance

[173] GridLab, The GridSphere Portal http://www.gridsphere.org/gridsphere/gridsphere

[174] M. P. Thomas1, J. Burruss2, L. Cinquini et al., "Grid portal architectures for scientific applications", Journal of Physics: Conference Series 16 (2005) 596-600, doi:10.1088/1742-6596/16/1/083 SciDAC 2005

[175] Jason Novotny, Michael Russell, Oliver Wehrens, "GridSphere: A Portal Framework For Building Collaborations", available at http://mgc2003.lncc.br/cam_ready/MGC281_final.pdf

[176] Rajkumar Buyya1,2, Chee Shin Yeo1, and Srikumar Venugopal1, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", Keynote paper for HPCC 2008 Conference, The full article can be downloaded from: http://www.gridbus.org/ raj/papers/hpcc2008_keynote_cloudcomputing.pdf

[177] Prot_g_-OWL and Jena Integration, details available at protege.stanford.edu/plugins/owl/jena-integration.html

[178] SPARQL Query language for RDF, details available at www.w3.org/TR/rdf-sparql-query

[179] MDS details available at http://www.globus.org/toolkit/mds

[180] Brian Tierney, Dan Gunter, "NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging", December 10, 2002, available at
http://www-didc.lbl.gov/papers/NetLogger.overview.pdf

[181] Details of Netlogger available at http://www.netlogger.org/

[182] Details of Broker available at http://www.gridbus.org/broker

[183] James Rambaugh, Ivar Jacobson, Grady Booch, "The Unified Modeling Language Reference Manual", Pearson Education, 2006, ISBN: 81-7758-161-9

[184] A. S. Grimshaw, M. A. Humphrey, and A. Natrajan, "A philosophical and technical comparison of Legion and Globus", IBM Journal of Research and Computing, Volume 48, Number 2, 2004, Deep Computing, available at http://researchweb.watson.ibm.com/journal/rd/482/grimshaw.html

[185]Isaac Chao, Ramon Sangesa and Oscar Oardaiz,"Grid Resource Management using Software Agents", available at http://www.ercim.org/publication/Ercim_News/enw59/chao.html

[186]A. Kertesz, P. Kacsuk,"Meta-Brokering requirements and research directions in state-of-

the-art Grid Resource Management", CoreGRID Technical Report Number TR-0116, November 13, 2007, available at http://www.coregrid.net/mambo/images/stories/REP/mb_rep_tr_116.pdf

[187]Yang-suk Kee, Carl Kesselman, "Grid Resource Abstraction, Virtualization, and Provisioning for Time-Targeted Applications," ccgrid, pp.324-331, 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), 2008

List of Publications

International Journals

1. Inderveer Chana and Seema Bawa, "Grid Middleware Technologies and OGSA: A Comparative Study ", *International Review on Computers and Software (IRECOS)*, A Bimonthly Peer-reviewed Journal published by Praise Worthy Prize (PWP), ISSN 1828-6003, Vol. 1, No. 3, November 2006, pp 202-208.

2. Inderveer Chana and Seema Bawa, "Self-Managed Resource Model with Resource Provisioning Capability for a Grid Environment", *Special Issue on Advances in Computing and Optimization Techniques,* organized in *International Journal of Applied Mathematical Analysis and Applications*, January-December 2007, Vol 2, No.1-2, pp. 229-241, ISSN: 0973-3868, published by Serials Publications, India.

3. Inderveer Chana and Seema Bawa, "Incorporating Enhanced Resource Provisioning in a Grid Resource Management System", *International Transactions on Systems Science and Applications*, an International peer-reviewed Journal, published by The Systemics and Informatics World Network (SIWN), UK, ISSN:1751-1461, Vol 3, Number 4, January 2008, pp.378-386.

4. Inderveer Chana and Seema Bawa, "PRATHAM: A Framework for Resource Management in Grid Environment", *International Journal of Advanced Computer Engineering,* a peer-reviewed international scientific Journal, published bi-annually, July-December 2008, ISSN: 0974-5785, Vol.1, No.2, pp.45-55.

International Conferences

1. Inderveer Chana and Seema Bawa, "Semantic Service Discovery In A Grid Environment", in the proceedings of the *International Conference on Information Systems, Technology*

and Management held at IMT Ghaziabad, pp 62-69 New Delhi from 12-13 March 2007, organized jointly by IMT Ghaziabad and University of Florida, USA.

2. Inderveer Chana and Seema Bawa, "Fault Tolerance In Grid Middleware: Alchemi.Net, An Enhanced Approach", in the proceedings of *International Conference on Emerging Trends in High Performance Architecture Algorithms and Computing* held in SSN Campus, Chennai from 11-13 July 2007,pp 63-68, Co-sponsored by HCL, IBM, IEEE Madras Section, ISBN: 978-81-904438-1-4

3. Inderveer Chana and Seema Bawa, "Extending Autonomic Capabilities to a Campus Wide Grid", in the proceedings of the *International Conference on Embedded Systems, Mobile Communication and Computing* held in August 3-5, 2007, at PESIT, Bangalore, India, http://www.pes.edu/mcnc/icemc2

4. Inderveer Chana and Seema Bawa, "Agent based Resource Discovery for Grid Environment using Ontology", in the proceedings of the *International Conference on Data Engineering and Management* held on 9 February 2008, at Bishop Heber College, Tiruchirappalli, India, ISBN: 978-81-906267-0-5, pp 199-202

National Conferences

1. Inderveer Chana and and Seema Bawa, "A Taxonomy and Survey of Fault Tolerance System in Grid Middleware", in the proceedings of *National Conference on Recent Advances in Computer Networks NCRACN 07,* held at Karunya University, Coimbatore, March 2007

Appendix A: Information generated by Jena

Following is the detail of Resource Description Framework (RDF), List of Statements and Inference Model generated by Jena for PRATHAM. Only a part of the output has been shown (as complete detail lists are very lengthy)

Grid RDF

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
  </rdf:Description>
  < rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#Condor">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Middlewares"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#HasAvailpuSpeed">
    <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:domain rdf:nodeID="A1"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#Service">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Grid_Entity"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#HasaAvailMemory">
    <rdfs:domain rdf:nodeID="A3"/>
    <rdfs:range
    rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2002/07/owl$#DatatypeProperty"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#OS">
    <rdfs:subClassOf
```

```
    rdf:resource="http://www.owl-ontologies.com/grid.owl#SoftwareResource"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#MacOS">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#OS"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#Cluster">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#ComputingResource"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#n1ge6">
    <rdf:type
    rdf:resource="http://www.owl-ontologies.com/grid.owl#SGE"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#Data">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Resource"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#Resource">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Grid_Entity"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
.........contd.
```

List of Statements

```
Object =http://www.w3.org/2000/01/rdf-schema#Class
Subject=http://www.owl-ontologies.com/grid.owl#JobSchedular
Predicate =subClassOf
Object=http://www.owl-ontologies.com/grid.owl#LocalService
Subject=http://www.owl-ontologies.com/grid.owl#JobSchedular
Predicate = type Object = http://www.w3.org/2000/01/rdf-schema#Class
```

Subject = null Predicate = unionOf Object=25a86eca:11540294848:-7ffc

Subject = null Predicate = type

Object=http://www.w3.org/2002/07/owl#Class

Subject=http://www.owl-ontologies.com/grid.owl#HasInstalledMiddleware

Predicate = range

Object=http://www.owl-ontologies.com/grid.owl#Middlewares

Subject=http://www.owl-ontologies.com/grid.owl#HasInstalledMiddleware

Predicate = domain

Object=http://www.owl-ontologies.com/grid.owl#ComputingElement

Subject=http://www.owl-ontologies.com/grid.owl#HasInstalledMiddleware

Predicate = type Object=http://www.w3.org/2002/07/owl#ObjectProperty

Subject = http://www.owl-ontologies.com/grid.owl#Service

Predicate=subClassOf

Object=http://www.owl-ontologies.com/grid.owl#Grid_Entity

Subject=http://www.owl-ontologies.com/grid.owl#Service

Predicate=type Object = http://www.w3.org/2000/01/rdf-schema#Class

Subject = http://www.owl-ontologies.com/grid.owl#GlobalService

Predicate = subClassOf

Object=http://www.owl-ontologies.com/grid.owl#Service

Subject=http://www.owl-ontologies.com/grid.owl#GlobalService

Predicate = type Object = http://www.w3.org/2000/01/rdf-schema#Class

Subject = http://www.owl-ontologies.com/grid.owl#Alchemi

Predicate=subClassOf

Object=http://www.owl-ontologies.com/grid.owl#Middlewares

Subject=http://www.owl-ontologies.com/grid.owl#Alchemi

Predicate=type Object = http://www.w3.org/2000/01/rdf-schema#Class

Subject = http://www.owl-ontologies.com/grid.owl#n1ge6

Predicate=type Object = http://www.owl-ontologies.com/grid.owl#SGE

Subject = http://www.owl-ontologies.com/grid.owl#Globus

Predicate=subClassOf

Object=http://www.owl-ontologies.com/grid.owl#Middlewares

Subject=http://www.owl-ontologies.com/grid.owl#Globus Predicate=type

Object = http://www.w3.org/2000/01/rdf-schema#Class

Subject=http://www.owl-ontologies.com/grid.owl#Intel

Predicate=subClassOf Object =

http://www.owl-ontologies.com/grid.owl#Cpu Subject =

http://www.owl-ontologies.com/grid.owl#Intel Predicate = type

Object= http://www.w3.org/2000/01/rdf-schema#Class

Subject=http://www.owl-ontologies.com/grid.owl#Middlewares

Predicate= subClassOf

Object=http://www.owl-ontologies.com/grid.owl#Grid_Entity

Subject=http://www.owl-ontologies.com/grid.owl#Middlewares

Predicate= type Object = http://www.w3.org/2000/01/rdf-schema#Class

Subject =http://www.owl-ontologies.com/grid.owl Predicate = type

Object =http://www.w3.org/2002/07/owl#Ontology

Subject = null

```
Predicate= rest
Object=http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
Subject = null
Predicate = first
Object=http://www.owl-ontologies.com/grid.owl#ComputingElement
Subject =http://www.owl-ontologies.com/grid.owl#HasMaxcpuSpeed
Predicate =domain
.........contd.
```

Inference Model

```
<rdf:type
rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#JobSchedular">
    <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#LocalService"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#JobSchedular"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Grid_Entity"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Service"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description
  rdf:about="http://www.owl-ontologies.com/grid.owl#SGE">
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Middlewares"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#SGE"/>
    <rdfs:subClassOf
    rdf:resource="http://www.owl-ontologies.com/grid.owl#Grid_Entity"/>
    <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdf:type
    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>\\
  <rdf:Description rdf:nodeID="A1">
```

```
  <rdfs:subClassOf
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdfs:subClassOf rdf:nodeID="A1"/>
  <rdf:type
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <owl:unionOf rdf:nodeID="A5"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdf:type
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.owl-ontologies.com/grid.owl#Condor6.8">
  <rdf:type
  rdf:resource="http://www.owl-ontologies.com/grid.owl#Condor"/>
  <rdf:type rdf:resource="http://www.owl-ontologies.com/grid.owl#Middlewares"/>
  <rdf:type
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdf:type rdf:resource="http://www.owl-ontologies.com/grid.owl#Grid_Entity"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
  <rdfs:range
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdf:type
  rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>
.....................contd.
```

Appendix B: Installation Details

Installation of GT4

GT4 is downloadable from http://www.globus.org/toolkit/downloads/4.0.6/. As GT4 had been installed on Fedora Core 6 therefore gt4.0.6-all-source-installer.tar.gz has been downloaded from the above mentioned site.

1. Before Installation:

Create a user named 'globus' in the local machine. If installing in the head node of the cluster and username is shared between nodes, create a common user to all the nodes.
Create a directory 'globus-4.0.6' and set its access rights so that 'globus' user has the full control over it:

```
[ root@inderveer ~ ] # mkdir /usr/local/globus-4.0.6
[ root@inderveer ~ ] # chown globus:globus /usr/local/globus-4.0.6
Set the basic environment variable GLOBUS$\_$LOCATION
to/usr/local/globus-4.0.6
```

2. Prerequisites

The prerequisites' table at the above mentioned site provides the details of the packages that need to be installed before the installation of GT4.

3. Building the Toolkit

Build the Globus toolkit as globus user

```
[root@inderveer ~ ]  # su globus
[globus@inderveer ~ ] \$ cd /home/globus
[globus@inderveer ~ ] \$ tar xzf gt4.0.6-all-source-installer.tar.gz
[globus@inderveer ~ ] \$ cd gt4.0.6-all-source-installer
[globus@inderveer ~ ] \$ ./configure
[globus@inderveer ~ ] \$ make
[globus@inderveer ~ ] \$ make install
```

4. Setting up Security on first machine

Create the SimpleCA

```
    [globus@inderveer ~ ] \$\$GLOBUS_LOCATION/setup/globus/setup-simple-ca
    globus_simple_ca_xxxxxxxx_setup-0.18.tar.gz is created in the directory
    /home/globus/.globus/
- Make the machine trust the new CA
- Get a hostcert for the machine
- Sign the certificate using SimpleCA as globus
- Copy signed
certificate into /etc
- Get a usercert for inderveer
- Get the
certificate request to globus user so it can be signed, then send
the signed cert back to inderveer
- Copy the cert to the proper
location
- Create Grid-mapfile as root for authorization
    [root@inderveer ~ ] $\#$ cd /etc/Grid-security
    [root@inderveer ~ ] $\#$ vim /etc/Grid-security/Grid-mapfile
    [root@inderveer ~ ]$\#$ cat /etc/Grid-security/Grid-mapfile
    " / O=Grid/ OU=simpleCA-inderveer.thapar.edu/ OU=thapar.edu/ CN=inderveer"
    inderveer
5.  Set up GridFTP
6.  Starting the web service container
    Setup an /etc/init.d entry for the webservices container
7.  Configuring RFT
8.  Set up WS GRAM
9.  Run the Sample Job
```

Installation of Alchemi.Net

Following steps install various components of Alchemi.

1. Alchemi Manager Installation

The Alchemi Manager can be installed in two modes - As a normal Windows desktop applica-

tion - As a windows service. (Supported only on Windows NT/2000/XP/2003) To install the Manager as a windows application, use the Manager Setup installer.

```
The Manager can be run from the desktop or Start -> Programs ->
Alchemi -> Manager -> Alchemi Manager.
```

The database configuration settings used during installation automatically appear when the Manager is first started. Click the "Start" button to start the Manager. When closed, the Manager is minimized to the system tray.

2. Cross Platform Manager

Install the XPManager web service via the Cross Platform Manager installer. If the XPManager is installed on a different machine then the Manager, or if the default port of the Manager is changed, the web service's configuration must be modified. The XPManager is configured via he ASP.NET Web.config file located in the installation directory

```
(wwwroot\Alchemi\CrossPlatformManager by default): <appSettings>
<add key="ManagerUri" value="tcp://localhost:9000/Alchemi_Node" />
</appSettings> The XPManager web service URL is of the format
http://[host_name]/[installation_path]
The default is therefore
http://[host_name]/Alchemi/CrossPlatformManager. The web service
interfaces with the Manager. The Manager must therefore be running
and started for the web service to work.
```

3. Executor Installation

The Alchemi Executor can be installed in two modes - As a normal Windows desktop application - As a windows service. (Supported only on Windows NT/2000/XP/2003) To install the executor as a windows application, use the Executor Setup installer.

After installation, the standard Windows service control Manager can be used to control the service. The Executor service controller is a graphical interface, which looks very similar to the normal Executor application. Install the Executor via the Executor installer and follow the on-screen instructions. The Executor can be run from the desktop or

```
Start -> Programs -> Alchemi -> Executor -> Alchemi Executor.
```

The Executor is configured from the application itself.

Executor needs to be configured for: The host and port of the Manager connect to Dedicated/non-

dedicated execution.

A non-dedicated Executor executes grid threads on a voluntary basis (it requests threads to execute from the Manager), while a dedicated Executor is always executing grid threads (it is directly provided grid threads to execute by the Manager). A non-dedicated executor works behind firewalls.

Click the "Connect" button to connect the Executor to the Manager.

Installation of Grid Bus Broker 1.4

---

1. Requirements

- At Broker side (i.e. on the machine running the broker)

  - Java Virtual Machine 1.4 or higher
  - Valid grid certificates properly set up
  - Additionally, some ports on the local node should be configured to be 'open' so that the jobs can connect back to the broker. (Please refer to the Globus documentation for more details)

-At Remote Grid node side

For a compute resource Middleware installation is required.

2. Installation process

Installing the broker is a simple process. The broker is distributed as a .tar.gz (and a .zip) archive that can be downloaded from :

`http://www.gridbus.org/broker/2.4/gridbusbroker2.4.tar.gz.`

`http://www.gridbus.org/broker/2.4/gridbusbroker2.4.zip.`

The installation just involves unzipping the files to any directory and optionally setting the PATH environment variable to include the broker executable script.

Following are the steps to be followed:

- Unzip the archive to the directory in which broker has to be installed. In case of Windows, use WinZip (if you download the .zip file) or WinRar (for the .tar.gz).In case of nix, command is:

  $ tar -zxvf gridbusbroker.2.4.tar.gz

- Set the GBB_HOME variable to the directory where the broker has been installed. Additionally, it is recommended to have the directory gridbus-broker2.4/bin added to the system PATH variable.

- Set the permissions for the gbb.sh executable:
  $ chmod 755 gbb.sh

- Test the installation by running the broker from a shell:
  $ ./gbb.sh -test

The following message confirms that the configuration is ok:
"Congratulations! You have successfully installed the Gridbus broker on your machine".