QoS based Resource Provisioning and Scheduling in Grids

A Thesis submitted for
the award of degree of
DOCTOR OF PHILOSOPHY

by

Rajni
(900903001)

Under the guidance of

Dr. Inderveer Chana
Associate Professor
Computer Science and Engineering Department
Thapar University, Patiala – 147004, INDIA

Computer Science and Engineering Department
Thapar University, Patiala – 147004, INDIA

April 2013

To

my parents

Contents

List of Figures

List of Tables

Certificate

I hereby certify that the work which is being presented in this thesis entitled "**QoS based Resource Provisioning and Scheduling in Grids**", for the award of degree of "**Doctor of Philosophy**" submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Inderveer Chana and refers other researchers works which are duly listed in the reference section.

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

Rajni 23/04/2013

(Rajni)

900903001

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

Inderveer 23/04/13

(Inderveer Chana)
Associate Professor
Computer Science and Engineering Department
Thapar University
Patiala

Acknowledgments

First of all, I express my gratitude to the Almighty, Who blessed me with the zeal and enthusiasm to complete this work successfully.

I would like to thank my supervisor, Dr. Inderveer Chana, Associate Professor, Thapar University, Patiala for her suggestions and constant support during this research. I am grateful to her for motivating and inspiring me to go deeply into the field of Grid Resource Management Systems and supporting me throughout the life cycle of PhD studies. Her wide knowledge and logical way of thinking have been of great value for me. Especially the strict and extensive comments, discussions and the interactions with Dr. Inderveer Chana had a direct impact on the final form and quality of this thesis.

I am also thankful to Dr. Maninder Singh, Head of Computer Science & Engineering Department, for his guidance through the early years of chaos and confusion. I would like to thank Dr. Seema bawa, Professor for her encouraging and strengthening my morale during tough times.

I am thankful to my Doctoral committee members Dr.Deepak Garg and Dr. R.K Sharma and Dr. P.K Bajpai, Dean of Research and Sponsored Projects for their constructive comments and regularly ensuring the progress of my research work. My deep regards to Dr. K.K. Raina, Director, Thapar University for all the facilities which have been immensely helpful for the completion of my work.

I would like to thank Dr. Rajesh Kumar, Associate Professor, School of Mathematics and Computer Applications for his support and suggestions during my PhD.

I wish to thank the faculty and staff members of Computer Science & Engineering of Thapar University, Patiala for their co-operation and support. I would also like to thank all the members of Center of Excellence in Grid Computing at Thapar University, Patiala. I would particulary thank all my lab mates for their constant support. I extend my thanks to Mr.Inderpreet Singh, Manager, Expicient Inc. for his generous time and invaluable suggestions throughout the research work.

My thesis work would have been incomplete without thanking my friend, who was always there in the hour of need. A special thanks to my best friend Mr. Parteek Gupta, Manager, Reliance Industries Ltd. for providing me with encouragement during rough times, friendship and help during difficult times.

This thesis would have been impossible without the support of my family. My deep regards to my father Mr. Arun Aggarwal and my mother Mrs. Urmila Ag-

garwal for their patience and love. Without them this work would never have come into existence. They have provided me with lessons on honesty and ethics and their humbleness and patience have always amazed me. I would also thank my brother Deepak Aggarwal, my sisters Neetu Aggarwal and Meenakshi Aggarwal and my brother-in-law Mr. Parveen Bansal for their lifetime love, companionship, and support.

Rajni

Abstract

Grid computing has emerged as a computing paradigm to solve large-scale scientific applications which require massive amount of high computation power that can be achieved by efficient utilization of heterogenous and dynamic resources. As the Grid has become a viable high-performance alternative to the traditional super-computing environment, various aspects of effective Grid resource utilization are gaining significance as resources being the base of the Grid. In order to access the appropriate resource at the right time, in the right manner, the first step should be to find out resources' features such as accessing interface, meaning of parameters, functions realized, required accessing conditions etc. Therefore resource management in Grid computing has become a key research area and due to multitude of heterogeneous resources, resource provisioning and proper scheduling in the Grid resource management is required for improving the performance of the system.

Resource provisioning and scheduling are key issues to handle the resource management efficiently besides other numerous issues. Unless resource provisioning is considered a fundamental capability, predictable QoS can't be delivered to Grid consumers. Therefore, it is an inherent need to design a resource provisioning policy based on QoS parameters for Grid environment. Resource provisioning and scheduling solutions strengthen the management of Grid resources in an efficient and effective way. To achieve the set objectives of addressing QoS based resource provisioning and scheduling challenges laid for this thesis, a comprehensive literature review on Grid resource provisioning and scheduling has been done. A thorough study of resource provisioning with QoS and without QoS has been carried out. A comparative study of Grid middleware and Grid schedulers has been done. The existing Grid scheduling heuristic approaches have also been studied and analyzed. Based on the literature survey, it is apparent that issues of provisioning and scheduling are the main challenges besides numerous other issues that need to be addressed. To address diverse Grid resource provisioning and scheduling challenges, a Resource Provisioning and Scheduling Framework has been proposed in this work.

The proposed Resource Provisioning and Scheduling Framework offers resource provisioning policies and resource scheduling algorithm that caters to provisioned

resource allocation and resource scheduling. The policy rules have been specified in XML schema. QoS parameter(s) based Resource Provisioning Policies provide provisioning of the resources according to user's requirements. The policies have been validated by Z Formal specification language. Further, the QoS based resource provisioned approach has been implemented in GridSim toolkit. The results demonstrate that QoS based provisioned approach is effective in minimizing cost and submission burst time of applications in comparison to non-QoS based resource provisioned approaches. The implementation of this policy enables the users to analyze customer requirements and define processes that contribute to the achievement of a product or service that is acceptable to their consumers.

A hyper-heuristic approach for resource provisioning based scheduling can be used to effectively schedule the jobs on available resources in a Grid environment as it applies a low-level heuristic that associates the best mapping of the resources to the corresponding jobs. Bacterial Foraging Optimization (BFO) is a technique which is able to attain optimal scheduling decision by satisfying QoS services and can thus be applied to a Grid environment. Therefore, a novel Bacterial Foraging Optimization (BFO) based hyper-heuristic resource scheduling algorithm has been designed, proposed and implemented for scheduling of jobs in Grid environment so as to minimize the cost and time by minimizing the makespan and maximizing the security and reliability. The comparison of the proposed algorithm with existing scheduling heuristic based algorithms has also been done. The proposed algorithm not only minimizes the time and cost but also maximizes security and reliability. The performance of the proposed algorithm is evaluated through the GridSim toolkit using Ali's simulation model. The experimental results show that hyper-heuristic based Grid resource scheduling algorithm outperforms in comparison to hybrid-heuristic in all cases.

Finally, the framework has been compared with existing Resource Provisioning and Scheduling frameworks to validate the outcomes. The results show that Resource Provisioning and Scheduling Framework successfully and collectively addresses the issues of resource provisioning and scheduling to establish an efficient Grid.

Chapter 1

Introduction

Distributed computing is often used to describe a type of computing, where computers are not only networked, but also smartly distribute their workload across each computer so that they can provide dependable, consistent and pervasive access to high-end computation.

The new inventions and developments in the distributed system design, collaborative environments, high performance computing and high throughput computing made Grid the logical step thereafter. This led to the emergence of Grids in the 1990s, which collaborate resources from multiple organizations to fulfill the computing needs of applications with varying resource requirements. The participating organizations together form Virtual Organization (VO) and pool their resources into a common shared resource pool. The resources in this shared Grid infrastructure are used for applications which would have been otherwise impossible without massive computing power.

This chapter provides a high level view of the thesis. It discusses the fundamental concept behind the Grid technology, its evolution, Grid architecture, its key areas along with the major issues of this area. It further provides the motivation to propose resource provisioning and scheduling framework for Grid systems. It culminates with discussion of the organization of the rest of the thesis along with its contributions.

1.1 Grid Computing : An Overview

The performance and reliability of IT networks has led to the idea of extending the concept of remote computing to a significantly large set of users, similar to the provisioning of electric power to every user even in the most remote places with the help of power nets. Borrowing the name from the electrical power infrastructure, Grid computing was born in the mid 1990s [1]. Grid computing originated as a distributed paradigm for scientific high performance computing (Distributed Supercomputing, High-Throughput Applications, Data-Intensive Applications, etc.) and as an alternative to expensive supercomputers by virtually joining a large number of interconnected computers.

The term Grid was coined to describe technologies that would allow consumers to obtain computing power on demand. Grid computing can be described as a distributed computing paradigm in which virtualized applications, softwares, platforms, computation and storage can be provisioned, scaled and released instantly through the use of self-manageable services [2][3]. The technology flourished as it allowed consumers to obtain computing power on-demand by offering computing as a utility to the consumers.

Figure 1.1: Simple View of Grid[2]

Grid has emerged as a computing paradigm for solving grand challenge applications in science, engineering and economics through the sharing and collaboration of numerous heterogeneous resources [4]. Grid computing connects computers that are scattered over a wide geographic area, allowing their computing power to be shared, which implies coordinated resource sharing and problem solving

by multi-institutional VOs as shown in Figure 1.1 [5][6]. Grid computing is a promising technology for providing a uniform and transparent access to geographically dispersed computing resources, such as computers, databases, experimental equipments and observational equipments [7]. Although it has been used within the academic and scientific community; its standards, enabling technologies, toolkits and products are readily available that allow businesses to use and reap the advantages of Grid computing.

The concept of Grid computing started as a project to link supercomputing sites, but now it has grown far beyond its original intent [8][9]. Grid computing not only provides the resources that allow our scientists to manage the infinite collection of data but it also allows this data to be distributed all over the world, which means that the scientific teams can work on international projects together and simultaneously.

On the basis of the utilities, Grid can be categorized as follows [10]:

i. Computational Grid: A Computational Grid is a hardware and software infrastructure that allows components of information technology infrastructure, computational capabilities, databases, sensors, and people to be shared flexibly as a true collaborative tool[11]. Computational Grid coordinates resources that are not subject to centralized control using standard, open general purpose protocols and interface to deliver nontrivial Quality of Service (QoS).

ii. Data Grid: A Data Grid is utilized for large-scale data storage and management of data-intensive services. It also provides the main infrastructure to store and handle large amount of data that is required by many scientific and engineering applications. The main aim of data Grid is the management and controlled sharing of large amounts of distributed data; however, different data can have their own formats. The European Data Grid is one of the Grids which provides the facility to large projects like Large Hadron Collider Computing Grid (LCG) [12]. The Grid PhyN [13], Tera Grid [14] [15] and PPDG [16] data Grid are other well known examples of data Grids.

iii. e-Science Grid: e-Science Grids are known for providing solutions to problems arising in fields like medicine, finance, weather forecast and engineering, etc. Such Grids give support to the computational infrastructure (access to computational and data resources) needed to solve many complex problems arising in the areas of science and engineering. Representative examples are

EGEE Grid Computing [17], UKe-ScienceGrid [18], German D-Grid [19], BIGGRID (the Dutch e-Science Grid) [20] and French Grid'5000 [21].

iv. Enterprise Grid: An Enterprise Grid can be loosely defined as a distributed system that aims to dynamically aggregate and co-ordinate various resources across the enterprise and improve their utilization such that there is an overall increase in productivity. Enterprise Grids enable running several projects within one large enterprise to share resources in a transparent way. Examples of enterprise Grids are Sun Grid Engine (SGE) [22], IBM Grid [23], Oracle Grid [24] and HP Grid [25].

v. Knowledge Grid: These are Grid-based environments that enable interoperation among users, applications, and resources to effectively manage knowledge resources used in virtual organizations, e-learning, online collaboration, etc.

vi. Desktop Grids: Desktop Grids are a new form of enterprise Grids emerging in institutions, which use the idle cycles of desktops. Small enterprises and institutions are usually equipped with hundreds or thousands of desktops, mainly used for office tasks. These PCs are thus a good source for setting up a Grid system for the institution. In this case, the particularity of the Grid system is its unique administrative domain, which makes it easier to manage due to the low heterogeneity and volatility of resources. Of course, the desktop Grid can cross many administrative domains and in this case the heterogeneity and volatility of the resources is not an issue, as in a general Grid system setting.

vii. Application Grid: Application Grid provides application server level infrastructure for the processing of applications thus allowing them to meet their goals and increase performance by resource sharing.

After an overview of Grid computing, the next section would be discussing its evolution till date.

1.1.1 Evolution of Grid Computing

The popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we use computers today [26][9]. Distributed computing is transforming the IT landscape towards an increasingly global and knowledge-based economy. Distributed computing is critical to the development of new, innovative and scalable applications

and infrastructures that helps in the advancement of commerce and science. Grid computing emerged in the second half of the 1990s as a new computing paradigm for advanced science and engineering [6][27]. The term Grid, however, may mean different things to different people. To some users, a Grid is any network of machines, including personal or desktop computers within an organization. To others, Grids are networks that include computer clusters, cluster of clusters, or special data sources. As defined by Foster & Kesselman in 1999, "The word "Grid" is chosen by analogy with the electric power Grid, which provides pervasive access to power and thus having a dramatic impact on human capabilities and society". Four distinct phases of this evolution have been illustrated in Figure 1.2.

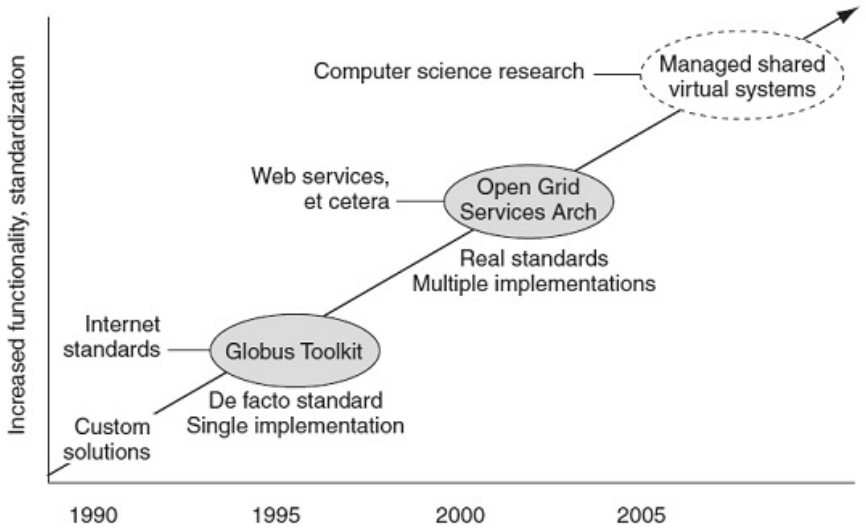

Figure 1.2: The Evolution of Grid Technologies [27]

Starting in the early 1990s, work in "metacomputing" and related fields involved the development of custom solutions to Grid computing problems. From 1997 onwards, the open source Globus Toolkit version 2 (GT2) emerged as the defacto standard for Grid computing. Focusing on usability and interoperability, GT2 defined implemented protocols, APIs and services used in thousands of Grid deployments worldwide. The year 2002 saw the emergence of the Open Grid Services Architecture (OGSA), a true community standard with multiple implementations, including the OGSA-based GT 3.0, in particular, which was released in 2003. The definition of the initial OGSA is an important step forward, but much more remains to be done before the full Grid vision is realized [27]. For more than ten years, Grid computing has been promoted as the global computing infrastructure of the future. Today advances in distributed computing have enabled the creation of national and international Grids such as the TeraGrid [14]

[15], Open Science Grid [28], Enabling Grids for E-sciencE (EGEE) [17] , APAC-Grid in Australia [29], K*Grid in Korea [30], NAREGI in Japan [31], Garuda in India [32], E-Science Grid in the UK [18], Our Grid in Brazil [33], Grid'5000 [21] and Auver Grid in France [34] [35] and DAS in the Netherlands [36] [37].

With this background, the next section discusses the Grid architecture.

1.1.2 Grid Architecture

Grid architecture [2] is often described in terms of layers, where each layer has a specific function. The higher layers are generally user-centric, whereas the lower layers are more hardware-centric. Figure 1.3 shows the layered Grid architecture and its relationship to the Internet Protocol architecture [2]. Grid architecture comprises of five layers: Fabric, Connectivity, Resource, Collective and Applications as shown in Figure 1.3.

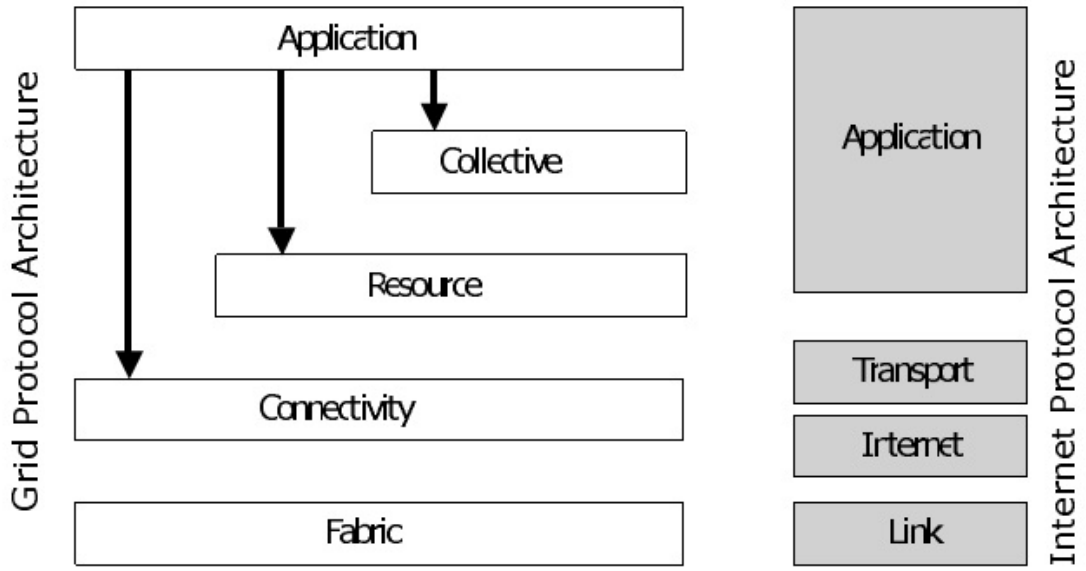

Figure 1.3: The layered Grid architecture and its relationship to the Internet protocol architecture [2]

Fabric layer: The Grid Fabric layer provides the resources to which shared access is mediated by Grid protocols, for example, computational resources, storage systems, catalogs, network resources, sensors etc.

Connectivity Layer: The Connectivity layer defines core communication and authentication protocols required for Grid-specific network transactions.

Resource layer: The role of the Resource layer is to implement core communications and authentication protocols call on fabric layer functions to access and control local resources. Resource layer protocols are concerned entirely with indi-

vidual resources.
Collective layer: Collective layer contains protocols and services that are not associated with any one specific resource but instead capture interactions across a collection of resources.
Application layer: The final layer in Grid architecture comprises of the user applications that operate within a VO environment.

The Grid architecture is based on Grid characteristics as discussed in the next section.

1.1.3 Grid Characteristics

Grids share certain common characteristics of distributed computing as enlisted below [38] [26]:

- Heterogeneity: The resources in Grid are heterogeneous in nature. The extent of heterogeneity spans across multiple institutions that are a part of Grid computing resources like processors, data storage devices, and bandwidth vary with different resource providers.

- Adaptability & Scalability: Grid adapts itself to the changing user needs. However, the distinguishing characteristic of Grid is scalability as per the user demand. Resources may scale up equally well as they scale down with the changing user's needs.

- Resource Sharing: Grid is based on virtualization to enable resource sharing. Heterogeneous resources located across multiple administrative domains are pooled to fulfill the resource needs of user applications.

- Dynamic: Grids are dynamic in nature. Resources can join or leave the computing environment any time as per the requirement.

- Security: Grid computing allows users to access a shared infrastructure. Users must be authenticated and authorized to maintain the confidentiality and integrity of data and shared resources.

After an introduction to Grid characteristics, next section will discuss the Grid standards.

1.1.4 Grid Standards

There are many standards involved in building a Grid architecture, which form the basic building block that allow applications to execute service requests. In this section, prevalent open Grid standards used for implementing Grid are discussed which are as follows:

- Web services: Grid services, defined by OGSA, is an extension of web services. So, Grid service can leverage the available web service specifications[39]. Four basic types of web services used in Grid systems are described as follows:

  a. Extensible Markup Language (XML): XML is a markup language that forms the basis of web services. It is used to store and transport the data.

  b. Simple Object Access Protocol (SOAP): It is an XML based, platform independent protocol providing simple and relatively light weight mechanism for exchanging structured information in the implementation of web services in computer networks like Grid.

  c. Web Service Description Language (WSDL): WSDL[40] is an XML based language used for describing the model of web services.

  d. Universal Description, Discovery, and Integration (UDDI): UDDI [41] protocol is approved by the Organization for the Advancement of Structured Information Standard (OASIS) as a standard for service registries in the context of SOA. It is a key member of the web services stack and defines the ways in which web services are published and discovered across the network.

- Open Grid Service Architecture (OGSA): OGSA represents an evolution towards a Grid system architecture based on web services concepts and technologies. The Global Grid Forum (GGF) has embraced the OGSA as the blueprint for standards-based Grid Computing [3]. OGSA is a Service-Oriented Architecture (SOA) that addresses the need for standardization by defining a set of core capabilities and behaviors to address the key concerns in Grid systems [42]. OGSA provides services such as resource discovery, resource provisioning, resource management, data management, information services and security.

- Open Grid Services Infrastructure (OGSI): OGSI [43] defines mechanisms for creating, managing, and exchanging information among entities called Grid

services. OGSI provides the features that are needed for the implementation of the Grid services. It also provides the facility of Web Service Description Language (WSDL) that defines Grid services.

- Web Service Resource Framework (WSRF): WSRF is the de-facto standard for modeling and accessing resources using web services [44].

- Open Grid Service Architecture-Data Access and Integration (OGSA-DAI): OGSA-DAI project allows data resources to be federated and accessed via web services on the web or within Grids. With the help of these web services, data can be queried, updated, transformed and combined by different methods [45].

1.1.5 Grid Applications

A Grid application is a collection of work items to solve a certain problem or to achieve desired results using a Grid infrastructure. For example, Grids generally support many different kinds of applications, ranging from High Performance Computing (HPC) to High Throughput Computing (HTC) [46]. Grids evolved to comply with the computing needs of high performance scientific applications. As a result much of the driving force and assistance for Grids is provided by the academia. A Grid application can be the simulation of business scenarios, like stock market development, that requires a large amount of data as well as a high demand for computing resources in order to calculate and handle the large number of variables and their effects [47]. In other words, a Grid application may consist of a number of jobs that together fulfill the whole task.

Grid is utilized in a number of areas including:

- Molecular modelling for drug design

- Neuro science brain activity analysis

- Cellular microphysiology

- High Energy Physics and the Grid Network (HEP Grid)

- Access Grid

- Globus Applications

- The International Grid (iGrid) and many more

Other applications like LHC Computing Grid (LCG) [12], The Large Hadron Collider will produce roughly 15 petabytes (15 million gigabytes) of data annually enough to fill more than 1.7 million dual-layer DVDs a year. Thousands of scientists around the world at CERN, the European Organization for Nuclear Research want to access and analyze this data, so CERN organized LCG project by collaborating with institutions in 34 different countries to operate a distributed computing and data storage infrastructure [12]. Similarly other applications like Enabling Grid for E-science (EGEE) [17], SETI@home [48], etc require huge amount of power, usually over a short period of time. Grid provides the required computation power through the sharing of computational resources. Apart from these, Life sciences, Biology, Aerospace, Earth sciences and E-commerce are other grand challenge applications of Grid computing .

After discussing the Grid essentials such as Grid architecture, its characteristics and standards, the next section would focus on the key challenges of this area.

1.2 Grid Computing Key Issues

Grid computing also presents certain issues that needs to be addressed similar to the issues in other emerging technologies. Current obstacles related to the growth and adoption of Grid computing are:

- Data Management
- Resource Management and Scheduling
- Security

1.2.1 Data Management Issues

Data management is one of the key features of Grid computing, specifically for data Grid in which large amount of data are distributed over dynamic and remote sites, potentially all over the world [49]. Data submission, data location, data dissemination, data replication, data consistency control, multi-protocol reliable file transfer, autonomic data management, data security and privacy are the main issues of data management [50][51]. Data can be static or dynamic in nature. Static data cannot be modified and updated but can only be read and analyzed. On the flip side, dynamic data can be modified and updated. Data replication on distributed infrastructures is a static data problem while E-business type applications belong to dynamic data type category [39]. A data Grid must allow

dynamically scalable storage services to store data in such a way that data intensive applications can be handled efficiently and effectively.

1.2.2 Resource Management and Scheduling Issues

Resource management plays a vital role in Grid computing. Resource management issues often need to be addressed as the complexity of Grid increases. Grid resource management requires discovery, monitoring and scheduling of resources in an efficient manner and at an advance level. The resource management system also provides functionalities such as QoS that enables guaranteed provisioning of resource capacity [52]. An efficient functioning of a complicated and dynamic Grid environment requires resource manager to monitor and identify the idling resources and to schedule user's submitted jobs (or programs) accordingly [53]. In general, Grid scheduling is the process of mapping Grid jobs to resources over multiple administrative domains. Traditionally the scheduler is responsible for resource discovery, resource trading, resource selection and job assignment, but emerging deadline driven Grid applications require access to several resources and predictable QoS [54]. The resource allocation and job scheduling mechanism used at the global and local level play a crucial role for the performance and availability of Grid applications [6]. QoS is a constraint imposed on the scheduling process instead of the final objective function [26]. In Grid resource management system, QoS management aims to provide assurance for accessing resources, while maintaining the security level between domains. Scheduling should be done by considering the QoS parameters like cost of the resources at the time of job execution. QoS requirements must be fulfilled according to the user and provider's satisfaction. In such a complex, dynamic and distributed environment, resource provisioning and resource scheduling are the key issues which further need to be addressed, to improve the resource usage efficiency on the Grid [53].

1.2.3 Security Issues

Grid is increasingly being taken up and used by all sectors of business, industry, academia and the government as the middleware infrastructure of choice. So, Grid security is an essential aspect of its overall architecture [55]. Security within the Grid environment is driven by the need to support scalable, dynamic and distributed VOs.

Considering the Grid environment's diverse and geographically separated resources and wide variety of users, each with unique needs and goals for the Grid

system, the issue of managing the security of users and resources becomes an issue [56]. There are major security issues in Grid systems which can be categorized in the following disciplines: authentication, delegation, single sign-on, credential lifespan and renewal, confidentiality, message integrity, non repudiation, secure logging, privacy, trust, policy exchange, authorization, assurance and manageability [57]. Security is also of main concern at the time of job submission. OGSA Security Roadmap [58] itemizes the various security services: auditing, anonymity, credential processing services, credential conversion services, authorization, identity mapping services, etc. to provide the security in Grid computing. Trust being the fundamental and common security aspect at the time of resource sharing in Grid environments led to the development of several trust models.
This section discussed the major issues of Grid computing. The next section discusses the Resource Provisioning and Scheduling in the Grid resource management systems, which has been chosen as the area of research for this thesis.

1.3 Grid Resource Provisioning and Scheduling : The Research Motivation

A number of issues need to be dealt with in the area of Grid resource provisioning and scheduling. Resource provisioning allocates the resources dynamically and it also works at both the user's and provider's level. Resource provisioning and scheduling have 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. Due to this reason, many issues come to the surface, which have been summarized as follows:

- The resource owners and the users have different goals, objectives, strategies, and demand patterns. Compared to traditional distributed systems, the resource provisioning 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 [59].

- In Grid resource management system, QoS management aims to provide assurance for accessing resources, while maintaining the security level between domains. Scheduling can be done by considering the optimization of QoS

parameters like cost, time etc of the resources at the time of job execution.

- The complex, heterogeneous and dynamic nature of Grid systems presents new challenges in resource management such as the provision of QoS parameters like cost, time, reliability and security to resource consumers.
- In Grid resource scheduling, it's not an easy task to allocate resources. There is no facility of accessing the resources by calling an intermediate agent and the resources are available in two forms viz stable and un-stable. Moreover, the scheduling algorithms need to have less communication overhead and should meet user's QoS requirements in comparison to any other conventional algorithms available [26].
- Scheduling task on a set of heterogeneous and dynamically changing resources is itself a complex problem that requires sophisticated algorithms that take into account multiple-optimization criteria [5].
- In providing QoS, Grid resource management system must be able to handle resource provisioning and scheduling in an efficient manner. Investigation into related areas; such as real-time resource allocation strategies and capacity planning remains important research areas to support collaborative applications- such as those that require computational steering support [60].
- Due to the dynamic nature and an uncertainty of Grid computing system behavior, QoS depends crucially on the selection of appropriate subset of the available resources [4].
- A computational Grid must allow resource owners and resource consumers to make autonomous scheduling decisions and both parties must have sufficient incentives to stay and play in the Grid [61].
- One of the primary goals of the Grid is to provide non-trivial QoS which is very important for the stability distributed resources [60].

Due to all these factors, the resource provisioning and scheduling issues specific to the Grid environment have been the motivation behind this work. The next section discusses the organization of this thesis

1.4 Thesis Organization

After giving an Introduction to the thesis in chapter 1, the rest of the thesis is structured as follows:

Chapter 2 presents the literature survey on Grid resource management. Resource provisioning strategies and resource provisioning with and without QoS has been discussed. A detailed description of the existing Grid scheduling work has been presented. Existing scheduling techniques in different middleware have also been discussed. A taxonomy of Grid schedulers, types of Grid scheduling algorithms and heuristic approaches for Grid scheduling have been presented and a comparison of different heuristic techniques has also been discussed. The chapter finally concludes with the Problem Formulation. Chapter 2 is partially derived from:

- **Rajni**, Chana, I., "Resource Provisioning and Scheduling in Grids: Issues, Challenges and Future Directions", in IEEE International Conference on Computer and Communication Technology (ICCCT10), pp:306-310, MNNIT, Allahabad, September 17-19, 2010.
- **Aron R.**, Chana I., "Enhancement of Resource Provisioning in Grid Resource Management Systems", 11th annual Grace Hopper Celebration of Women in Computing, organized by Anita Borg Institute for Women and Technology and ACM, Oregon Convention Center, Portland, Oregon, November 9-12, 2011.

Chapter 3 describes the proposed Resource Provisioning and Scheduling Framework and the goals which it tends to achieve. The requirements analyzed on the basis of QoS parameter(s) have been illustrated in detail. Mode of Operation of QoS based Resource Provisioning and QoS parameters based Resource Provisioning Policies corresponding to resource provisioning and scheduling framework have been presented. Chapter 3 derives from:

- **Aron R.**, Chana I., "QoS based Resource Provisioning and Scheduling in Grids", Journal of Supercomputing, **Springer**, DOI 10.1007/s11227-013-0903-1, Impact Factor : 0.578, 2013.
- **Aron R.**, Chana I., "Formal QoS Policy based Grid Resource Provisioning Framework", Journal of Grid Computing, **Springer**, Impact Factor : 1.310, Vol 10, No 2, Pages: 249-264, June 2012.
- **Aron R.**, Chana I., "Cost based Resource Provisioning Policy for Grids", in International Conference on Parallel and Distributed Computing, Proceedings of the World Congress on Engineering , (Awarded Certificate of Merit) Vol II, WCE 2011, July 6 - 8, 2011, London, U.K.

- **Aron R.**, Chana I., "Resource Provisioning for Grids: A Policy Perspective", in International Conference on Contemporary Computing, LNCS, Springer, pp-546-547, JIIT University Noida, August 8-10, 2011.

Chapter 4 presents design of resource scheduling algorithm based on resource provisioning policies. This chapter provides the objective function formulation that takes care of the interests of both resource providers and resource consumers. A resource scheduling algorithm based on Bacterial Foraging Optimization (BFO) has been proposed in this chapter. Details about the proposed algorithm are discussed in this chapter. Chapter 4 derives from:

- **Aron R.**, Chana I., "QoS based Resource Provisioning and Scheduling in Grids", Journal of Supercomputing, **Springer**, DOI 10.1007/s11227-013-0903-1, 2013, Impact Factor : 0.578.

- **Rajni**, Chana I., "Bacterial Foraging based Hyper-heuristic for Resource Scheduling in Grid Computing", Future Generation of Computer Systems (FGCS), **Elsevier**, Vol 29, No 3, pp. 751-762, March 2013, Impact Factor: 1.978.

- **Rajni**, Chana I., "Resource Provisioning Policy based Scheduling for Grid Environment", 12th annual Grace Hopper Celebration of Women in Computing, organized by Anita Borg Institute for Women and Technology and ACM, Baltimore Convention Center, Baltimore, Maryland, October 3-6, 2012.

Chapter 5 illustrates the verification of the framework. Further, the framework has been validated and compared with the existing systems. Verification of the framework and QoS parameters(s) based resource provisioning policies have been done through Z formal specification language. Experimental results have been collected from the implementation of the policies using the GridSim toolkit. This chapter provides the design and implementation details of the proposed algorithm. The implementation of the proposed algorithm has been done in a Grid Environment using Ali's simulation model. Chapter 5 derives from:

- **Aron R.**, Chana I., "QoS based Resource Provisioning and Scheduling in Grids", Journal of Supercomputing, **Springer**, DOI 10.1007/s11227-013-0903-1, 2013, Impact Factor : 0.578,

- **Rajni**, Chana I., "Bacterial Foraging based Hyper-heuristic for Resource Scheduling in Grid Computing", Future Generation of Computer Systems (FGCS), **Elsevier**, Vol 29, No 3, pp. 751-762, March 2013, Impact Factor: 1.978.

- **Aron R.**, Chana I., "Formal QoS Policy based Grid Resource Provisioning Framework", Journal of Grid Computing, **Springer**, Vol 10, No 2, pp: 249-264, June 2012, Impact Factor: 1.310.

Chapter 6 finally concludes the thesis and discusses the future scope of the work.

1.5 Thesis Contribution

This Thesis contributes in the following ways:

- It critically analyzes the detailed literature of Grid resource provisioning and scheduling along with a detailed study of Grid schedulers and Grid middleware. It provides a qualitative comparison of the existing heuristic approaches with respect to their parameters like convergence, premature convergence, service, etc.

- QoS parameters have been identified to propose a resource provisioning and scheduling framework.

- To address the challenges of resource provisioning and scheduling in Grid resource management, a Resource Provisioning and Scheduling Framework has been proposed that offers resource provisioning policies which cater to provisioned resource allocation and resource scheduling.

- The proposed QoS parameter(s) based Resource Provisioning Policies have been designed using XML and the implementation of the policies has been shown in resource provisioning and scheduling framework.

- The usability of the proposed policies, their validation has been shown using Z formal specification language. Formal specification and verification of the framework helps in predicting possible errors before scheduling process itself and thus results in an efficient resource provisioning and scheduling of Grid resources.

- Grid resource scheduling problem has been formulated in the form of combinatorial optimization problem. The scheduling problem has been considered from both the user's and resource provider's point of view.

- Bacterial Foraging Optimization (BFO) based hyper-heuristic resource scheduling algorithm for scheduling of application with QoS constraints has been

proposed. Its main aim is to minimize the cost and execution time simultaneously by considering both the resource consumer's and resource provider's benefits. The algorithm has been evaluated by comparing it with other well-known scheduling algorithms.

- To demonstrate the validation of the proposed algorithm, it has been implemented in the GridSim toolkit using Ali's simulation model, the existing benchmark for heterogenous environments.

- The experimental results show that the proposed algorithm outperforms in comparison to the existing resource scheduling algorithms in all respects such as effect of resource heterogeneity, number of applications, number of resources, etc.

- Statistical analysis of simulation output has been performed in order to assess the accuracy of the estimated performance indices by using coefficient of variation. The coefficients of variation have been calculated for cost and makespan results achieved by the BFO based hyper-heuristic resource scheduling algorithm and existing scheduling algorithms.

Chapter 2

Literature Survey

Grid computing has emerged as an incredible technology to provide huge amount of power to large-scale scientific, innovative applications, and in some cases, high-performance orientation and e-commerce projects.

The deployment of Grid systems involves the efficient management of heterogeneous, geographically distributed and dynamically available resources. As Grid comprises of resources, Grid resource management is the fundamental task in Grid computing. The Grid resource management system is required to perform resource management decisions which include resource provisioning and scheduling, while maximizing the QoS metrics delivered to the clients when jobs have QoS constraints.

This chapter discusses the Grid resource management systems along with the description of Grid resource provisioning and scheduling requirements. It also discusses the key background information to facilitate a better understanding of resource provisioning and resource scheduling in Grid environment. A comparative study and analysis of the existing Grid schedulers and scheduling in Grid middleware has been done. Further, Grid scheduling heuristic approaches have been discussed and compared along with the study of QoS in Grids. The last section highlights the limitation of the existing approaches and finally lists down the objectives of the thesis.

2.1 Grid Resource Management Systems

Grid computing infrastructure promises to provide the facility of resource sharing in distributed systems in a flexible, secure and coordinated fashion [2] [62]. The main vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically as and when required, similar to a power Grid which supplies electricity to the end users [63]. Grid computing enables sharing of loosely coupled resources and services required by different applications on a large-scale. Resource is the basis of the Grid and managing the resources in Grids is a tedious task. Thus, resource management is the core of a Grid system as defined by Jyotishman Pathak et al. [64]. Resource management describes all aspects of the processes: locating a capability, arranging its use and utilizing its monitoring state. It also includes the efficient use of computing and storage resources. The basic function of a resource management system is to accept requests for resources and assign specific machine resources to a request from the overall pool of Grid resources for which the user has the access permission. Resource management includes various services such as launching a job on a particular resource, checking its status and retrieving results when the job is complete as descried by Li M. and Baker M in their book [55]. This mechanism can be further described as follows:

i. Resource Information Dissemination: It involves collection of information about all the resources required for the execution of application in the VO.

ii. Resource Discovery: Resource discovery is the process of matching a query for resources, described in terms of required characteristics, to a set of resources which meets the expressed requirements [59]. Authorization filtering and job requirement knowledge to meet the minimal job requirements' needs to be accomplished. Various approaches like agent based resource discovery, query based resource discovery, peer to peer approach, parameter based approach and QoS based approach can be used for resource discovery in Grid environment.

iii. Resource Provisioning: Resource provisioning allows users and providers to access the specified resources as per their availability in the virtual environment created in Grid infrastructure. Various strategies which can be used for resource provisioning in Grid environment includes static and dynamic resource provisioning, agreement based resource provisioning, model based resource provisioning and application based resource provisioning, etc.

iv. Resource Scheduling/Job execution: Resource scheduling allocates jobs on the provisioned resources and performs the task of job execution [65]. A variety of techniques and algorithms including static and dynamic scheduling, heuristic and approximate scheduling, cooperative and non-co-operative scheduling are utilized for job scheduling and execution. Other scheduling techniques like workflow scheduling, task level scheduling and adaptive scheduling are also explored for the Grid environments [26].

v. Resource Monitoring and Re-Scheduling: Resources must be monitored to track the status of allocated resources, available resources and required resources for application execution. In case a job fails to perform or a resource bottleneck occurs, then the resources are rescheduled. Static, dynamic and workflow monitoring are being done in the Grid environment.

With Grid becoming a viable high-performance alternative to the traditional supercomputing environment, various aspects of effective Grid resource utilization are gaining significance. With its multitude of heterogeneous resources, resource provisioning and proper scheduling in the Grid is required for improving the performance of the system [66]. As the Grid scales up, resource management complexity also increases and efficient resource management techniques are desired. Ramrez-Alcaraz J.M. et al. defined that resource management is one of the key components of Grid computing where QoS based resource provisioning and resource scheduling are challenging tasks [67]. To increase the efficiency of resource management systems, there is a need to concentrate on resource provisioning and scheduling. Resource provisioning and scheduling are the key components for Grid resource management besides numerous other components. In resource management systems, one of the key challenges is to design resource provisioning policies and resource provisioning based scheduling algorithms which would allow users to carry out their jobs by transparently accessing autonomous, distributed and heterogenous resources. Foster, I. et al [27][2] defined that resource provisioning allows the users and providers to access the specified resources according to the their availability of the resources in VOs. Grid scheduling makes scheduling decisions involving allocating jobs to resources over multiple administrative domains [65]. The main aim of the current resource provisioning and resource scheduling is to provide Grid's users and non- users with privileges of using resources in advance level while satisfying the QoS parameters. Deelman, E. suggests that resource scheduling can be done subsequently after resource provisioning as the user desires to discover the resources according to the requirements of the job.

Then resource mapping is done to that particular job. Resource provisioning is particularly useful for Grid scheduling because most often Grid scheduling is done on the best-effort basis in a Grid environment and due to this, overheads of Grid scheduling like inter-independent tasks can be very costly and time consuming [68]. For example:- If there are two jobs in a queue and the first job starts execution then the second job will not be released to the local resource manager until the first job finishes. But in case of resource provisioning, resources are provisioned and then the second job is released to the local resource manager immediately after the completion of the first job. Thus, queuing time delay will be less through resource provisioning.

The next section presents the requirements for the resource provisioning and scheduling in Grids.

2.1.1 Grid Resource Provisioning and Scheduling Requirements

There are various provisioning and scheduling requirements for Grids. Few of them have been identified by Dusseau A. C. A. in [69] as:
Efficiency: Resource provisioning provides the facility to minimize the Grid overheads. It requires efficient management and scheduling of the resources based on fair policies.
Efficient Resource Usage: It reduces the wastage of the resources. Jobs that are waiting for events (e.g., disk or user I/O, network latency, CPU usage, processor) should hand over the processor so that they do not waste resources.
Fair Allocation: The amount of resources allocated to each user should be independent of the number of jobs each user runs and provisioning of resources should also be fair.
Adaptability and scalability: A smart scheduler adapts as per the resources, i.e. even when the resources join or leave (dynamically), it manages the resources and jobs' execution process efficiently.

After discussing the Grid resource management systems, its importance and the requirements of resource provisioning and scheduling in the Grid, next section would focus on the resource provisioning and its existing practices in Grid computing.

2.2 Resource Provisioning

The resource availability in Grids is generally unpredictable due to the autonomous and shared nature of the Grid resources and stochastic nature of the workload resulting in a best effort Quality of Service [70]. Singh G. et al. defined that due to large-scale, shared and autonomous nature of the Grid resources, modeling the resources availability in discrete units that can be provisioned by the user is a more appropriate approach [71]. For resource provisioning, QoS parameters need to be defined and provided to the user. Guaranteeing QoS in a non- deterministically shared heterogeneous environment like Grid is a challenging task. Provisioning is slightly more complex than queuing in the way that it requires users to make more sophisticated resource allocation decisions [72]. As per Gideon Juve and Ewa Deelman, resource provisioning can be classified as follows:

Static provisioning: In static provisioning, application allocates the resources for job completion which are required for job processing. After the job completion, application releases the resources [72]. This method assumes that the number of resources required is known or can be predicted in advance.

Dynamic provisioning: In dynamic provisioning, system allocates the resources at run time. It allows the pool of available resources to grow and shrink according to the changing needs of the application [72]. Dynamic provisioning does not require advance knowledge of the resource but it requires the information about policies for acquiring and releasing the resources.
Resource provisioning can be implemented in the following ways:

Agreement based resource provisioning: The agreement-based management model allows the VO-level resource brokers to provision resources ahead of the execution of the applications by entering into agreements with the resource providers about the guaranteed availability of desired resources for a mutually agreed upon time-frame and cost [70].

Model based resource provisioning: Model-based approach to provisioning is multiple resource based that interacts in complex ways. In case of model-based resource provisioning, internal models capturing service workload and behavior can enable the utility to predict the effects of changes to the workload intensity or resource allotment [73].

Application based Resource Provisioning: Application based resource provisioning allows the user to control the scheduling and execution of the application on the provisioned resources [71]. This in turn enables the estimation of the application performance prior to its execution. It also provides the facility to explore

the space of resources to be provisioned and optimized for performance without worrying about external factors such as the workload of the resource or the policies of the resource provider to the user or workflow manager. Application based resource provisioning enables the adaptation of the application on the provisioned resources in order to achieve a certain level of performance. It also allows the execution of applications that require co-allocation of resources or have other constraints on the resource requirements.

AAA Policy based Provisioning: The main function of AAA policy based provisioning is to incorporate AAA (Authentication, Authorization and Accounting) policy into path computation resource allocation and signaling functions. It requires high-level association of policy with users as well as lower-level association of policy with actual network elements at a fidelity sufficient to implement meaningful policy based resource allocations [74].

Resource provisioning and scheduling are the core components of resource management in Grid computing. There has been little emphasis on resource provisioning for Grid resource management as described in the next section. The categorization of resource provisioning with QoS and without QoS has been discussed in the following section.

2.2.1 Resource Provisioning- without QoS

DRAGON [75] and GLARE [76] frameworks provide provisioning in heterogeneous Grid environments. As per DRAGON framework, network infrastructure is deployed, that allows dynamic provisioning of network resources in order to establish deterministic paths in direct response to end-user requests. It also allows advanced e-science applications to dynamically acquire dedicated and deterministic network resources to link computational clusters, storage arrays, visualization facilities, remote sensors and other instruments into globally distributed and application-specific topologies. GLARE provides distributed registries for activity types, activity deployments and services that perform registration, provisioning, monitoring and automatic deployment of new activities on different Grid computers in VO. Murphy et al. [77] have designed the Virtual Organization Clusters (VOC) architecture that provides a mechanism by which each VO may have its own dedicated clusters on Grid sites. It can be dynamically expanded, contracted, instantiated or terminated in response to the size and number of jobs submitted by the user. Dynamic provisioning has been done in VO.

Yu et al. introduced the Adaptive Online Job Provisioning (AOJP) that aims

to provide co-scheduling of computing and network resources in an optical Grid infrastructure. With the help of mixed integer liner programming, AOJP has solved the data aggregation problem which focuses on how to schedule network resources to fetch data from multiple sources to a single sink [78]. Byun et al. proposed an architecture to enable on demand resource provisioning and have developed a universal factory service that provides a dynamic Grid service deployment mechanism and a resource broker called door service for service providers [79].

Nau et al. have introduced an approach to allow heterogeneous applications to run together in a shared hosting platform, dynamically sharing the platform's resources [80]. Vazquez et al. addressed dynamic provisioning problem as interoperability, dynamic growth and enforcement of a budget by designing the architecture for an elastic Grid infrastructure [81]. Feasibility solution by harnessing resources of the Tera Grid, EGEE and Open Science Grid infrastructures through a single point of entry for the computational resources has been given for Grid infrastructures. AOJP and these solutions of dynamic provisioning do not support the QoS parameters for online job provisioning. QoS management in Grid computing has become an active area of research now-a-days. However, there are several applications that need to obtain the results for their tasks within time, hence they cannot wait for resources to become available [60]. Therefore, it is necessary to provide QoS at the time of resource provisioning.

2.2.2 Resource Provisioning- with QoS

OGSA based service ecosystem [82], a resilient infrastructure for services' provisioning on demand has been developed. OGSA based service ecosystem framework considers the status of Grid and customer requirements automatically to achieve the QoS parameters e.g. performance and reliability. This work provides services' provisioning using migration concept for commercial Grids and not for non-commercial Grids. Keller et al. presented strategies for Grid service provisioning to handle resource failures during Service Level Agreement (SLA) negotiations. Risk aware failure management has been considered in this work in the form of QoS but other important QoS parameters as time and cost have not been discussed [83].

Filali et al. [84][85] used the adaptive resource provisioning schema to achieve resource utilization in the best manner satisfying the required QoS by maximizing the network utilization, minimizing request blocking probability and maximizing the provider's revenues. Local optimization heuristic and global optimization

heuristic have been proposed by using the optimization techniques. An optimization model that enables the simultaneous allocation of interrelated resources as CPU and bandwidth for Grid applications has been proposed.

Kee et al. [86] proposed a high performance distributed computing resource management paradigm that defines a network of logical machines across time and space. A resource management system to provide resource provisioning, resource abstraction via resource programming over large-scale distributed resources has been used in this work. Virtualization for timely targeted applications in the Grid environment has been considered.

Foster et al. [87] described a General-purpose Architecture for Reservation and Allocation (GARA) that supports flow-specific QoS specification, immediate and advance reservation and online monitoring and control of both individual resources and heterogeneous resource ensembles.

QoS-GRAF framework [88] for QoS based Grid allocation with failure provisioning has been designed by Dasgupta et al. By considering SLA based service differentiation and failure provisioning, a linear relaxation based algorithm has been designed to improve the revenue.

Iosup et al. [89] have discussed about on demand resource provisioning for dynamic environment with worldwide sharing resources. A scheduling policy has been designed to solve the problem of resource shortage with heavy workload but security at the time of provisioning has not been considered for data sharing. Brocco et al. have proposed a flexible Grid service provisioning framework composed of two functional layers: a self-organized peer-to-peer overlay management layer, and a Grid services interface layer [90]. It has been designed to provide an adaptive, reliable and scalable communication platform for service provisioning in Grid network. Singh et al. [71][70] have presented a provisioning model for resource provisioning where Grid sites provide the information about resource availability in the form of time slots. The main aim of the model is to identify the subset aggregate resource availability so that cost and make span are minimized. A multi-objective genetic algorithm to find resource plan that corresponds to the Pareto optimal set has been used in this work. Raicu et al. [91] have proposed a dynamic resource provisioning architecture using the existing system *falkon*. Here allocation and de-allocation policies are presented and these policies are evaluated using metrics such as provisioning latency and accumulated CPU time.

The design of a gateway that provisions resources to dead-line applications relying on information given by current resource management services may be complex. It basically depends upon scheduling decisions that are far from optimal.

So, when providers and brokers use conflicting policies, the number of migrations can be high [54]. In the same vein, even owners of individual resources will exhibit different usage patterns and implement different policies for the time they make their machines available [92].

This section reported about the resource provisioning and existing work on resource provisioning with QoS and without QoS. The next section would present the Grid scheduling methods in existing literature.

2.3 Grid Scheduling

Grid scheduling is an essential part of Grid resource management. Khateeb et al. define Grid scheduling as the process of making scheduling decisions involving allocation of jobs to resources over multiple administrative domains [65]. The Grid Scheduling Architecture Research Group (GSA-RG) of the Open Grid Forum (OGF) provides different Grid scheduling use case scenarios and also describes common usage patterns in [93]. Grid scheduling involves three main stages: resource information gathering, resource selection & provisioning and scheduling.

- Resource information gathering : In this phase, Grid scheduler collects information about the status of the available resources. This step is very important due to the heterogenous and dynamic nature of the Grid resources. Grid Information Service (GIS) provides the information such as CPU capacity, memory size, software availabilities and network bandwidth to Grid schedulers.

- Resource Selection and Provisioning : The performance of a resource for different application species is also necessary for making a feasible schedule. Application profiling is used to extract properties of application and analogical benchmarking provides a measure of how well a resource can perform a given type of job. Another alternative is using prediction and historical information and user specification. After the selection of resources from a set of resource pool, provisioning is performed.

- Scheduling: In this phase, the mapping of the job to the ingredient resource is performed along with the job execution.

In this section, an overview of resource scheduling techniques used in existing literature has been discussed.

Resource scheduling is a major concern to achieve high performance on computational Grids. It is basically a large-scale optimization problem due to the heterogenous and dynamic nature of the Grid resources.

Several heuristic optimization methods such as Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) among others, have been proposed for resource scheduling in computational Grids. Abraham et al. used nature's heuristics namely GA, SA and TS for scheduling of jobs on computational Grids. They have shown that GA performs better than TS and SA for scheduling of the jobs to exact resources but hybrid heuristic algorithms perform better than GA approach as it minimizes the time required for scheduling the job[94]. Liu et al. [95] considered PSO to design a job scheduling algorithm in computational Grids. They have used the concept of a fuzzy matrix to design the scheduling algorithm. These scheduling methods try to minimize the execution time /makespan of the applications and as such are suitable for Grids. However, in Grids, there is another important parameter other than execution time, i.e., cost. Meta-heuristic approach has been used in all these methods but the hyper-heuristic approach would have performed better in comparison to the meta-heuristic.

Garg et al. proposed a model for meta-scheduling on utility Grids using linear programming and a genetic algorithm. This model minimizes the cost for scheduling of an independent task. They have considered multiple and concurrent users which are competing for the resources in a meta-scheduling environment so as to minimize their cost [96]. Garg et al. [97] proposed three novel heuristics for parallel applications on utility Grids. They evaluated the sensitivity of the proposed heuristics on the basis of changes in user's preference, application's execution time and resource's pricing.

Brun et al. [98] compared eleven static heuristics for mapping a class of independent tasks on heterogenous computing systems. They used a simulation model for comparing these static heuristics which returned the best optimum results. Xhafa and Abraham [10] surveyed the computational model and heuristic methods for Grid scheduling problems. They only surveyed the existing approaches. Gaoa et al. [99] developed two algorithms that used the predictive models to schedule jobs at both the system-level and application-level. In application-level scheduling, GA is used to minimize the average completion time of jobs through optimal job allocation on each node.

Golconda et al. did the comparison of five heuristics, namely QSMTS-IP, min-min, GA, least slack first and suffrage. The comparative study provides a

fair basis for the comparison of these heuristics [100]. Kim et al. [101] proposed a novel GA based approach that automatically decomposes data into communication and computation resources for scheduling of a divisible data intensive application. Konugurthi et al. [102] proposed a heuristic based GA for an optimal mapping of application/jobs to suitable resources.

Kondo et al. [103] designed three general techniques for resource selection: resource prioritization, resource exclusion and task duplication for scheduling of task parallel applications for rapid turnaround on enterprise desktop Grids. They have used techniques to instantiate several scheduling heuristics and investigated three approaches for task replication namely proactive, reactive and hybrid methods. The hybrid replication heuristic based on a probabilistic model of task completion did not perform any better than the best reactive heuristic. Jun et al. proposed a task scheduling heuristic algorithm based on linear programming for tree based Grid computing platforms. In this algorithm, they considered the computing power and bandwidth for each node in the model and assigned task for each node in an integrated manner. Then they used SimGrid to simulate and analyze the algorithm [104]. Chauhan et al. proposed QoS guided weighted mean time-min and QoS guided weighted mean time min-min max-min selective for QoS based Grid task scheduling. They have used the concept of historical information about the execution time of jobs to predict the performance of the algorithm [105].

Roy et al. [106] have discussed the resource brokering strategies within the multi-agent framework. These strategies help in finding out an optimal allocation of resources for executing multiple concurrent jobs in a Grid environment. Somasundaram et al. have proposed the CARE Resource Broker (CRB) [107] that addresses scheduling scenarios like unavailability of resources and the required execution environment for the resource providers. CRB is a Grid meta-scheduling framework that supports virtualization technology only.

Cameron et al.[108] have studied and analyzed the effects of various job scheduling and data replication strategies and compared them in a variety of Grid scenarios using several performance metrics. They used the Grid simulator OptorSim, and based their simulations on a world-wide Grid testbed for data intensive high energy physics experiments.

Saleh et al.[109] introduced adaptive Grid scheduling strategy that employs mobile agent technology in both monitoring and rescheduling processes. The main use of integrating mobile agent technology with computational Grids is to minimize the network traffic during the Grid resource discovery and monitoring processes. They only considered the scheduling and rescheduling of computational tasks ne-

glecting QoS parameters like cost and security. This approach does not take QoS requirements such as time into consideration.

Sanjay et al. have developed three strategies for deciding when and where to reschedule parallel applications that execute on multi-cluster Grids [110]. They only focused on the rescheduling instead of scheduling.

Desprez et al.[111] designed scheduling and replication algorithm that computes the mapping of data and computational requests at the same time. This approach used knowledge of the database usage scheme and of the target platform. Their main focus was to manage data and their replication. They mainly used the greedy algorithm to solve the mapping problem. The idea behind this algorithm is to try to map data that needs the maximum computational power to the server that has the maximum computation capacity. Pugliese et al. [112] analyzed the requirements of Grid resource management and provided a classification of schedulers. After that, they defined an extensible formal model for Grid scheduling activities. They designed only scheduling model instead of a resource scheduling algorithm to schedule jobs on the available resources. They did not consider the cost, makespan, security and reliability for independent job scheduling in the Grid environment.

Song et al. designed a new fuzzy-logic trust model for distributed trust aggregation through fuzzification and integration of security attributes [113]. The trust model aggregates many reputation attributes and measurable self-defense capabilities into scalar quantities, which can be easily applied to quantify the trust index of Grid resource sites. Their security-binding scheme scales well with increasing user jobs and Grid sites but they did not design scheduling algorithm for application schema. Additionally, fitness function of the genetic algorithm is dependent on the makespan of the solution while ignoring the other scheduling criteria like cost and security.

Cowling et al. stated that hyper-heuristic is a good approach to solve various problems such as personal scheduling, timetabling, nurse scheduling and resource scheduling. It is not problem specific like the meta-heuristic approach and it can be efficiently applied to an optimization problem [114]. Gonzalez et al. [115] used ad hoc (immediate and batch mode) scheduling methods to design a hyper-heuristic approach for scheduling of jobs on the Grid nodes according to the job and Grid characteristics. Bhanu et al. [116] designed a scheduling model for resource scheduling using heuristic methods. They have used Longest Job Faster Resource (LJFR) heuristic and Shortest Job Faster Resource (SJFR) heuristic method for resource scheduling.

The resource scheduling problem in Grid becomes more challenging as it is important to achieve not only the promising potentials of tremendous distributed resources, but also effective and efficient scheduling algorithms. Hence, a lot of algorithms have been developed for scheduling jobs in a computational Grid with an aim to minimize the job completion time [117].

Different Grid scheduling approaches have been investigated and applied to different Grid scenarios and requirements. Some of them have tried to schedule the jobs securely but there is not even a single approach that can be found in the literature that covers resources provisioning based scheduling. Kyriaki et al. have considered multi-criteria job scheduling using accelerated genetic algorithm [117]. They have considered security constraint for scheduling of jobs but resource provisioning based scheduling has not been considered. Kolodziej et al. [118][119] have presented an approach for independent task scheduling with security requirements in Grid computing environment. They have developed a scheduling model that enables the aggregation of task abortion and security requirements. They have used game-theoretic and a GA based approach for optimizing the makespan and flowtime. Menasce et al. stated that the main problem in Grid environment is the selection of services and service providers in an appropriate way so as to achieve global SLA with minimum cost [120].

The next section discusses scheduling in the existing Grid middleware.

2.3.1 Scheduling in Grid Middleware

Grid middleware is a layer between user's application and resources. The main aim of the Grid middleware is to find convenient places for applications to run, optimize the use of resources, organize the efficient access to data, deal with authentication to the different sites that are used, run the job & monitor progress, recover from problems and transfer the result back to the scientists. In this section, an overview of Grid middleware and their scheduling techniques have been discussed. The taxonomy followed is based on the scheduling models of gLite [121], UNICORE [122], Globus [123] and Legion [124]. Based on this taxonomy, an analysis of these middleware has also been done.

a) gLite middleware: gLite is the next generation middleware for Grid computing. It was born from the collaborative effort of more than 80 people in 12 different academic and industrial research centers as a part of the EGEE Project [121]. gLite provides a framework for building Grid applications

those tapping into the power of distributed computing and storage resources across the internet. gLite is a lightweight middleware for Grid computing and it provides scalability, performance, interoperability and modularity.

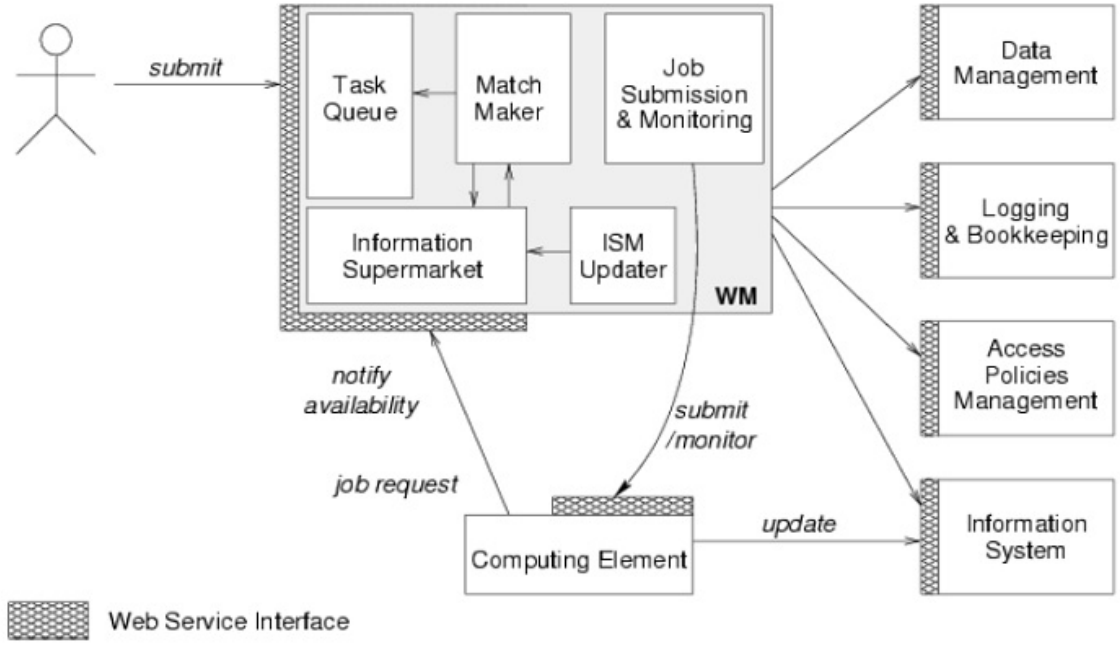

Figure 2.1: Job management services in gLite [125]

Scheduling in glite: gLite middleware achieves efficiently and reliably the scheduling of computational tasks on the available infrastructure. gLite consists of six principle components: Computing Element (CE), Workload Management (WM), Storage Element (SE), Catalog, Information and Monitoring and Security.

CE is a gateway to the local batch system & cluster. It provides the service that represents the computing resource that is responsible for the job management (submission, control, etc.). The CE may be used by a generic client, an end-user interacting directly with the computing element, or the workload manager, which submits a given job to an appropriate CE found by a matchmaking process [125].

Workload management acts as a resource broker in gLite middleware as shown in Figure 2.1. It allows user to submit the job and performs all tasks that are required to execute without exposing the user to the complexity of the Grid [126]. Workload management is primary job execution interface for users to find the best location for a job, considering job requirements and available resources (CPUs, files) and it gets resource information from

InfoSystem and File Catalogs. It is a set of middleware components that is responsible for the distribution and management of jobs across Grid resources [127]. In this, "matchmaking" the best available resource is assigned and "Logging & Bookeeping" keep track of job execution in term of events (Submitted, Running, Done,...).

Storage element is the gateway to a local storage (disk, tape) and a Gridftp server. Catalog remembers locations of files and it only deals with their locations.

In gLite, the task of information accumulation about the resource usage is done by users or group of users (VOs). Information on Grid services/resources needs sensors (resource metering, metering abstraction layer, usage records). Records are collected by the accounting system (Queries: Users, Groups, Resource). A system called Virtual Organization Membership Service (VOMS) is used to manage information about the roles and privileges of user with in a VO. This information is presented to service via an extension to the proxy [125].

b) UNICORE: In 1997, the development of the UNIform Interface to COmputer REsources (UNICORE) system was initiated to enable German super computer centers to provide their users with a seamless, secure and intuitive access to their heterogeneous computing resources [122][128].

Scheduling in UNICORE: The UNICORE architecture is based on three tier architecture: user tier, server tier and target tier. The user tier consists of the graphical user interface and offers the functions to prepare and control the UNICORE jobs and to set up and maintain the user's security environment [129]. A UNICORE (sub-) job is executed on behalf of a UNICORE user account. The interaction between a UNICORE client and its server is transaction based and usually asynchronous. So, the job is submitted to a UNICORE server and only the receipt of the job is acknowledged. The client does not wait for the completion of the job. Jobs may be submitted synchronously but the UNICORE server is free to convert this to asynchronous if the execution takes too long. The actual abstract job which is sent from the client to a Vsite is constructed in the client as an Abstract Job Object (AJO) [130]. The Java AJO class library is the basis for the modeling of UNICORE jobs and for the protocol between the UNICORE clients and the servers together with the abstract job specification generated from the user input [129][131]. The AJO is a key component in the architecture. Most

Grid scheduling actions within a UNICORE environment are currently carried out manually [132].

c) Globus: The Globus Alliance produced widely-used Grid middleware known as the Globus Toolkit. Globus is a software toolkit addressing key technical problems in the development of Grid enabled tools, services, and applications [123]. Globus is an open community project based on Apache Jakarta model. The Globus toolkit is a community-based, open-architecture, open-source set of services and software libraries that supports Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability.

Scheduling in Globus: Grid Resource Allocation Management (GRAM) is a software component of the Globus Toolkit that can locate, submit, monitor, and cancel jobs on Grid computing resources. It handles placement, provisioning and lifetime management of jobs. GRAM is a unifying remote interface for remote job submission and resource management. In globus, Grid job goals provide an environment for the job, stage files to/ from environment, causes, monitors the job execution, send job state change notifications and streams a job's stdout/err during execution. In GT4, remote job execution is carried out by the GRAM service, which is instrumental for providing a web service interface for submitting requests to execute jobs, defined in a job description language. It also monitors and controls the resulting job executions. GT4 includes two different GRAM services: the 'pre-WS GRAM', introduced in GT2, and the newer Web Services -based 'WS GRAM', introduced in GT4. When a job is submitted to the Globus, GRAM sends it to different distributed resources. When the user client sends the job request from the local user, the request is forwarded by this meta-scheduler to a Grid manager who in turn queries the Resource Manager (RM) for resources. RM stores information about local and remote resources. If available resources are found on a local site, job request is forwarded to the local scheduler. If the resources are not available locally, job request is sent to a remote site 's globus gate-keeper , which in turn forks the job to the job manager and is finally submitted to the remote scheduler for execution [133] [134]. The job details are specified through the Globus Resource Specification Language (RSL), which is the main part of the GRAM [134].

d) Legion: Legion is an object-based, meta-systems software project at the

University of Virginia [124]. It is a middleware system that combines very large number of independently administrated heterogeneous hosts, storage systems, databases legacy codes and user distributed objects distributed over wide area network into a single coherent computing platform [134]. The main goal of the legion middleware is to promote the principle design of the distributed systems software by providing standard object representation for file systems, data systems and processors, etc.

Scheduling in Legion: Scheduling in legion is not of a dictatorial nature;

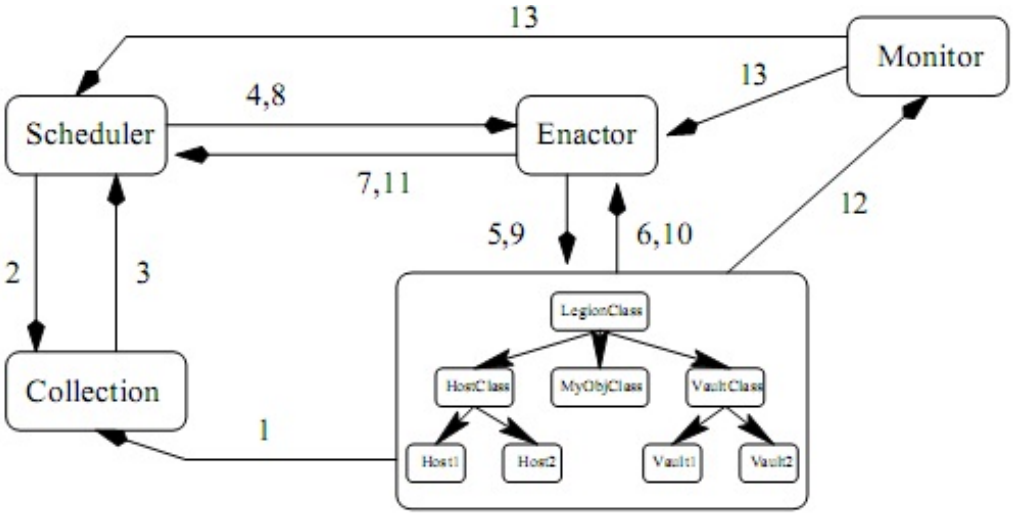

Figure 2.2: The use of Resource Management architecture in Legion [135]

request is made to the resource guardians who have the final authority [135]. Figure 2.2 shows that the user defined schedulers will directly interact with the infrastructure. Hosts and vault are the basic resources of the model. The collection is an information database and the enactor is the scheduler implementor of the resource management model. At first, the collection is populated with information describing the resources. Then the scheduler queries the collection and after getting the result of the query, computing of objects to resources is done. In case of application specific scheduler, there should be an appropriate knowledge of the application and then the mapping of application's classes is done. Scheduler passes the mapping to enactor, who invokes the methods based on hosts and vaults and gets the information of resource reservation from the mapping. After getting the information of resource reservation, enactor consults with the scheduler for confirmation, then the enactor instantiates objects through member function calls on the appropriate class objects. The class object gives the success or failure result which is informed to the scheduler via enactor. If during migration, there

is a need for the object to be migrated then this is notified to the scheduler and enactor and they reschedule the object.

e) Aneka: Aneka can be used as Grid middleware but recently aneka has also become market oriented Cloud development and management platform with rapid application development and workload distribution capabilities [136]. Aneka provides some advantages over Grid and cluster based workload distributions like provisioning based QoS/ SLA, support of multiple programming and application development, simultaneous support of multiple runtime environments. Aneka can be used in drug design, medical imaging, moduler & quantum mechanics and genomic research.
Scheduling in Aneka: In aneka, the scheduling services schedule work unit to execution nodes based on availability, capability and QoS requirements and the particular scheduling algorithms used. Pluggable scheduling algorithms, advance reservation and automatic/custom resubmission of failed task features are provided by Aneka scheduling services [137].

Table 2.1: Comparison of Grid Middleware

Middleware/ Features	**Advance reservation**	**QoS**	**Resource Provisioning**	**Scheduling Organization**	**Scheduling Policy**
gLite	Yes	Yes	Yes	Distributed	User Centric
Unicore	Yes	Yes	No	Distributed scheduler	User centric
Globus	Yes	Soft QoS	Yes	Hierarchical Scheduler	Ad-hoc extensible policy
Legion	Yes	Soft QoS	Yes	Hierarchical Scheduler	Ad-hoc extensible policy
Aneka	Yes	Yes	Yes	Decentralized & flexible	System centric

Table 2.1 describes the comparison between the above discussed middleware w.r.t to the features that they provide.

Next section discusses about a taxonomy of Grid schedulers.

2.3.2 A Taxonomy of Grid Schedulers

Grid scheduler works for the mapping of jobs to exact resources according to the requirements of the users. A Grid scheduler must make scheduling decisions in the best manner and submit the job to the selected resource. Grid scheduler is different from local scheduler [62]. Local scheduler only manages a single site or a cluster and usually owns the resources but this is not the case with the Grid scheduler. Grid scheduler is the key component of a computational Grid as it is responsible to optimize the use of Grid resources. A number of schedulers for Grid computing systems have been developed. In this section, following schedulers have been surveyed after considering their important features.

CONDOR-G: Condor [138] is a high throughput scheduler. It runs the job on single administrative domain but Globus runs the job on multiple administrative domains. By combining the strength of both, Condor- G is obtained. In Condor-G, Condor pool is divided into two parts: job management and condor software. Condor-G is the job management part of the Condor and Condor software does the resource management [139]. It uses the Globus toolkit to start a job on a remote machine instead of using condor-developed protocol. It uses globus protocols like Grid Resource Allocation & Management (GRAM), Global Access to Secondary Storage (GASS), Replica Location Service (RSL) and Grid Security Infrastructure (GSI) [140]. Condor-G also fulfills the requirements of the job' and tells the status of each job. Condor- G scheduler allows submitting the jobs into a queue, having log details of the life cycle of jobs and handling all input and output files. It provides a "window to the Grid" for users to both access resources and manage the jobs running on remote resources. The advantages of Condor- G is that it provides the facility of credential management, fault-tolerance and full -featured queuing services but the disadvantage is that it is platform dependent.

NIMROD-G: Nimrod-G is a resource broker and it performs resource management and scheduling of parameters sweep and task-farming applications on world-wide Grid resources [141] [142]. Nimrod-G uses globus and legion as Grid middleware for resource discovery/information and uses either network directory or object model based data organization. Load balancing facility is also provided by Nimrod-G by using rescheduling.

GridWay Scheduler: GridWay is a meta-scheduler (A meta-scheduler uses local schedulers of the particular systems. Thus, meta-schedulers coordinate local schedulers to compute an overall schedule [10]) which enables large-scale, reliable and efficient sharing of computing resources such as clusters, supercomputers and stand alone servers [143][144]. PBS, LSF, SUN GRID ENGINE, CONDOR are

supported by GridWay within a single organization or scattered across several administrative domains. As local resource management system, it uses globus WS, Pre WS, LCG, CREAM, ARC, etc. as a middleware for Grid applications [140]. First of all, client application agent uses client API to communicate with the request manager in order to submit the job. After this, client may also request the request manager to control the operation of job like start, resume and rescheduling of job. Then the dispatch manager submits pending jobs at each scheduling interval and reschedules the job. It also invokes the resource selector who gets information from monitoring and discovering services and short lists candidate hosts. The dispatch manager submits pending jobs by calling a submission manager, and also checks that wether the migration of rescheduled jobs is worthwhile or not. If this is the case, the dispatch manager triggers a migration event along with the new selected resource to the job submission manager, who manages the job migration [145]. The submission manager is responsible for the execution of the job during its life time, i.e. until it is done or stopped. It is initially invoked by the dispatch manager along with the first selected host, and is also responsible for performing job migration to a new resource [145]. The performance monitor periodically wakes up at each monitoring interval. It requests rescheduling actions to detect "better" resources when performance slowdown is detected [145][146].

SUN GRID ENGINE: Sun Grid Engine (SGE) is an open source and provides dynamic resource management services like load balancing, maximizes resource utilization, transparent job submission & machine selection, monitoring and accounting [22]. It acts as a resource broker for globus and takes scheduling decisions to select a remote site. Sun Grid engine broker provides services like guaranteeing required resources, full control over resource utilization, fair usage of resources that are otherwise shared and implementation of management policies. SGE/Broker submits and tracks jobs to remote systems using Globus services.

PBS: Portable Batch System (PBS) is a package which was designed and written by the Numerical Aerodynamic Simulation Complex, NASA, in 1994. PBS is a successor to Network Queuing System (NQS) and has addressed many of the deficiencies of NQS. It was designed to provide additional controls over the initiating, or scheduling, of execution of batch jobs [147] [148]. OpenPBS is the original version of the Portable Batch System (PBS) , a queuing system developed for NASA in the early 1990s [149]. OpenPBS operates on networked multi-platform UNIX environments. PBS Pro is used for job scheduling. In 1999, PBSpro and Globus were interoperable. Since 2000, it is a commercial product. It consists of a Server Daemon (1 server per cluster) and a Scheduler Daemon (1 scheduler for unique

cluster) - it is also possible to include an external scheduler. The Machine Oriented Mini-Server (MOM) Daemon runs on each execution node and is responsible for accounting and job processing. There are web-interfaces to CAE computations. It integrates Unix, Linux and there is a Grid front-end to Globus [150]. The future PBSpro offers fault tolerance and reliability, suspend/resume, checkpoint/restart, SMP cluster, floating licenses and supports system specific features (e.g. PSSP of IBM SP).

Gridbus Grid Service Broker: The Gridbus [151] project is an open source, multi-institutional project led by the GRIDS lab at the university of Melbourne. The Gridbus broker is designed for data Grid and computational Grid applications. The Gridbus broker follows a Service-Oriented Architecture (SOA) and is designed on object-oriented principles with a focus on the idea of promoting simplicity, modularity, reusability, extensibility and flexibility [152]. Gridbus emphasizes the end-to-end quality of services driven by computational economy at various levels - clusters, peer-to-peer (P2P) networks, and the Grid - for the management of distributed computational, data, and application services [134]. The Gridbus resource broker provides an abstraction to the complexity of Grids by ensuring transparent access to computational and data resources for executing a job on a Grid. It uses user requirements to create a set of jobs, discover resources, schedule, execute and monitor the jobs and retrieve their output once they are finished [153][134].

LSF: Load Sharing Facility (LSF) by Platform Computing is the most powerful workload manager for demanding distributed and mission-critical high performance computing environments [154]. It provides a complete set of workload management capabilities and is designed to work together to address high performance computing needs. Platform LSF includes a comprehensive set of intelligent, policy-driven scheduling features, enabling to fully utilize compute infrastructure resources. It is highly scalable and allows the user to schedule complex workloads[154]. With the best support in the HPC industry, Platform LSF provides the most complete HPC datacenter solution for workload management.

A comparison of Grid schedulers which classifies schedulers by characterizing different attributes is summarized in Table 2.2. The comparison focuses on their architecture, Grid type, language, platform, service and QoS parameters.

This section reported about the taxonomy of Grid schedulers and the next section would present the Grid scheduling algorithms.

Table 2.2: Comparative Analysis of Grid Schedulers

Scheduler/ Features	Architecture	Grid Type	Language	Platform	Services	QoS parameters
Condor-G	Independent Cluster	Computational Grid	Java	Linux, Solaris, Digital Unix, IRIX	Job management, resource selection, security, Fault Tolerance	Reliability
Nimrod-G	Resource Broker	Computational Grid, Service Grid	Parametric, declarative language	Platform Independence	Resource discovery,selection, scheduling, transparent execution of user jobs on remote resources	According to user requirements, Cost, Availability of resources, Hard and soft QoS
GridWay	Resource Broker	Partner, Supply Chain Grid, Enterprise Grid	C,java,perl, ruby, Python binding	Support existing remote platform	Advance scheduling, dynamic discovery, selection, opportunistic migration, performance slowdown detection, support for self-adaptive, fault detection, recovery	Flexiblity, Extensibility, Efficiency, Reliability
Sun Grid Engine	Independent Cluster	Data Grid, Computational Grid	DRMAA API,C, C++, Fortran, Netbeans	Multiple Cross Platforms	Distributed job scheduling,Resource balance,time management, authentication, authorization, job monitoring, Fault tolerance	Flexible administration, advance reservation, scalability
PBS	Independent Cluster	Computational Grid	Torque	Runs on most Unix-like systems: e.g. Linux, Irix, Unicos, HPUX, IA64, etc.	Preemptive job scheduling,Scheduler backfilling, Improved fault-tolerance, Desktop Cycle Harvesting	Advance reservation, flexibility, reliability, cost effective, scalability, serviceability
Gridbus Grid Service Broker	Resource Broker	Data Grid, Computational Grid	Java	Windows, Linux	Resource discovery, job scheduling, monitoring of jobs	Flexibility, Simplicity, HPC Solutions, Extensibility
LSF	Independent Cluster	Data Grid, Computational Grid, Enterprize Grid	C, Phython	Windows, Linux	Load sharing, Job scheduling	Flexibility, Visibility, Highly scalable, Advance Reservation

2.3.3 Grid Scheduling Algorithms

In literature, Grid scheduling algorithms have been discussed from different perspectives, such as static vs. dynamic policies, objective functions, application models, adaption QoS constraints, strategies dealing with dynamic behavior of resources and so on [26]. Further, scheduling can be done as static vs. dynamic, distributed vs. centralized and co-operative vs. non- co operative. Existing techniques and algorithms for scheduling of Grid resources are:

i. **Local versus Global**: The local scheduling discipline determines how the processes resident on a single CPU are allocated and executed where as a global scheduling policy uses information about the system to allocate processes to multiple processors so as to optimize a system-wide performance objective. Grid scheduling should be done as global scheduling [26].

ii. **Static versus Dynamic**: In Grid, both static and dynamic types of scheduling have been adopted. In static scheduling, information regarding all the resources in the Grid as well as all the tasks in an application are assumed to be available at the time of scheduling but in the case of dynamic scheduling, the basic idea is to perform task allocation on the fly as the application executes.

iii. **Centralized versus Decentralized**: In Grid scheduling, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In centralized scheduling, Grid schedular has more control on the resources and in this case efficient scheduler can be designed. Centralized Grid scheduling algorithm can be easily implemented but it suffers from the lack of scalability, fault tolerance, etc. Therefore, centralized scheduling is not useful for large-scale Grids. In decentralized scheduling, Grid schedulers have no centralized control over the resources. In this case, local schedulers play an important role in scheduling and also manage and monitor the status of the resources [10].

iv. **Co-operative versus Non-cooperative**: In co-operative scheduling, each Grid scheduler carries out its own scheduling tasks, but all schedulers are working toward a common system-wide goal. Scheduling is done through the co-operation of Grid users, rules and policies. In the non-cooperative case, each scheduler acts alone as an autonomous entity and arrives at decisions regarding their own optimum objects independent of the effects of the decision on the rest of the system [26].

v. **Approximation versus Heuristics**: The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. Heuristic algorithms are more adaptive to the Grid scenarios where both resources and applications are highly diverse and dynamic, so heuristics are considerably a de-facto approach for solving Grid scheduling problems [155].

The next section presents a detailed study of Grid scheduling heuristic approaches.

2.4 Grid Scheduling Heuristics

The mapping of jobs to appropriate resources for execution of application in heterogeneous environments like Grid computing is called an NP-complete problem [98]. NP-complete problems are often solved using heuristic methods. Heuristic approaches can be applied to Grid scheduling problems because Grid scheduling has various important issues that needs to be addressed such as heterogeneity of the resources, dynamic and autonomous nature of Grid resources and finally resource providers and resource consumers have different policies for the execution of their applications. Existing scheduling heuristics as given below are discussed and compared in [98].

2.4.1 Greedy Heuristic approaches

Greedy algorithms are intuitive heuristics in which greedy choices are made to achieve a certain goal [156]. Greedy heuristics are constructive heuristics since they construct feasible solutions for optimization problems from scratch by making the most favorable choice in each step of construction. By adding an element to the (partial) solution which promises to deliver the highest gain, the heuristic acts as a greedy constructor.

Opportunistic Load Balancing: Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the machine that is expected to be available next. This is regardless of the task's expected execution time on that machine [157]. The main aim of the OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity, but as OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans.

Minimum Execution Time: In contrast to OLB, Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task. This is regardless of that machine's availability [98]. The motivation behind MET is to give each task to its best machine. It does not consider the current load on a resource so will often cause load imbalance between the processors.

Minimum Completion Time: Minimum Completion Time (MCT) assigns each task, in arbitrary order, to the machine with the minimum expected completion time for that task [157]. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The intuition behind MCT is to combine the benefits of OLB and MET, while avoiding the circumstances in which OLB and MET perform poorly.

Min-min: The Min-min heuristic considers all the unmapped tasks with the known set of minimum completion times. Next, the task with the overall minimum completion time is selected and assigned to the corresponding machine (hence the name Min-min) [157] [158]. Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min-min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could.

Max-min: The Max-min heuristic is very similar to Min-min. The Max-min heuristic also begins with the set of all unmapped tasks with the known set of minimum completion times. Next, the task with the overall maximum completion time is selected and assigned to the corresponding machine (hence the name Max-min). Intuitively, Max-min attempts to minimize the penalties incurred from performing tasks with longer execution times [157] [158]. Assume, for example, that the metatask being mapped has many tasks with very short execution times and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a min-min mapping, where all the shorter tasks would execute first, and then the longer running task would execute while several machines sit idle. Thus, in cases similar to this example, the Max-min heuristic may give a mapping with a more balanced load across machines and a better makespan.

Duplex:The Duplex heuristic is literally a combination of the Min-min and Max-min heuristics. The Duplex heuristic performs both the Min-min and Max-

min heuristics and then uses the better solution. Duplex can be performed to exploit the conditions in which either Min-min or Max-min performs well, with negligible overhead [98].

2.4.2 Local Search based Heuristic approaches

Local search heuristic approaches is a family of methods that explore the solution space by starting at an initial solution, and constructs a path in solution space during the search process [10]. Local search heuristic approaches improve solutions through neighborhood search. The main objective of this local search based heuristic approach is to gain feasibility as soon as possible [159]. They have been applied successfully to many industrial problems and performance of local search based heuristic approaches depend on construction of neighborhood.

Tabu Search: Tabu Search (TS) is a high level heuristic procedure for solving optimization problems and was proposed by Glover in 1986. It is a meta-heuristic that guides a local search procedure to explore the solution space beyond local optimality [160]. TS avoids entrapment in local minima and continues the search to give a near optimal final solution. It is very general and conceptually much simpler than other meta heuristic algorithms such as genetic algorithm, simulated annealing and ant colony optimization algorithms. TS is very easy to implement and does not require special memory space. TS takes short searching time to solve combinatorial optimization problems. It uses specific set of constraints, known as tabu conditions, in order to avoid blind search. TS has disadvantages like it often gets locked in looping from one local optimum to another and it has low global search capability.

Hill Climbing: Hill Climbing (HC) is a graph search algorithm where the current path is extended with a successor node which is closer to the solution than the end of the current path. It is logical and beneficial especially in situations where the search space is of simple nature with no more than a single maxima or minima [10]. HC is a local search heuristic technique and it is more simpler and straight forward in comparison to other heuristics. The main disadvantage in case of hill climbing is that, the solution is better than all of its neighbors, but it is not better than some other states far away. There is a flat area of the search space in which all the neighboring states have a same value. HC is a local method and it moves in many directions at a time.

Simulated Annealing: Simulated Annealing (SA) heuristic approach was proposed by kirkpatrick et al. in 1983. The simulated annealing process consists

of first melting the system being optimized at high effective temperature, then lowering the temperature by slow stages until the system freezes and no further changes occur [161]. It is an iterative technique that considers only one possible solution (mapping) for each meta task at a time [162]. This solution uses the same representation as the chromosome for the genetic algorithm. The initial implementation of SA was evaluated and then it was modified and refined to give a better final version in [98]. It uses a procedure that probabilistically allows poorer solutions to be accepted in an attempt to obtain a better search of the solution space.

SA has many advantages: (i) It is guaranteed to converge in asymptotic time. (ii) It can deal with arbitrary systems and cost functions. (iii) SA statically guaranties to find an optimal solution and it is relatively easy to code, even for complex problems. (iv) It is robust heuristic to implement and has an ability to provide reasonably good solutions for many combinatorial problems. The main disadvantage of SA is that there is a need for a great deal of computer time for many runs and carefully chosen turnable parameters. It has a difficulty in defining a good cooling schedule which is important both in single and multi objective optimization. In case of SA, if there is a repeated annealing with 1/logk then the scheduling is very slow, especially if the cost function is expensive to compute.

2.4.3 Population based Heuristic approaches

Population-based heuristic is a large family of methods which are highly efficient for solving combinatorial optimization problems. However, when the objective is to find feasible solutions of good quality in short execution times, as in the case of Grid scheduling, the inherent mechanisms of these methods can be exploited to increase the convergence of the method [10].

Genetic Algorithms: Genetic Algorithm (GA) was proposed by hollland et al [163]. GA is a famous stochastic optimization algorithm which uses biologically inspired techniques such as genetic inheritance, natural selection, mutation, and sexual reproduction (recombination, or crossover) [164]. It is a useful heuristic to find a near optimal solution in large search spaces [162]. In GA, a point in search space is represented by a set of parameters and these parameters are known as genes and a set of genes is known as string or a chromosome. A fitness function must be devised for each problem to be solved. Each chromosome is assigned a fitness value that indicates how closely it satisfies the desired objective. Given a particular chromosome, the fitness function returns a single numerical fitness or

figure of merit, which will determine the ability of the individual, which that chromosome represents [94]. A set of chromosomes is called population. Reproduction is another critical attribute of GAs where two individuals selected from the population are allowed to mate to produce offspring, which will comprise the next generation. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. Mutation provides a small amount of random search, and helps ensure that no point in the search space has a zero probability of being examined. If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum. The genetic algorithms have been found to be very powerful in finding out a global minima [165][98]. Genetic algorithms have been applied to many classification and performance tuning applications in the domain of knowledge discovery in databases [164] [166].

The important advantages of GA are: (i) Analytical knowledge is not required. (ii) It is easy to parallelize and no derivatives are required (iii) GA works on a wide range of problems and has better global capability. The disadvantages of genetic algorithm: (i) It requires much more evolution functions than linearized methods. (ii) There is no guaranty of convergence of local minima. (iii) It converges to local optima or arbitrary point rather than the global optima of the problem. (iv) GA has a slow convergence rate and premature convergence and (v) it can not use the feed back of a system.

Memetic Algorithm: Memetic Algorithm (MA) is an extension of genetic algorithm. Memetic algorithms are evolutionary algorithms that can be applied on a local search process to refine solutions for hard problems. It is the subject of intense scientific research and have been successfully applied to a multitude of real-world problems ranging from the construction of optimal university exam timetables, to the prediction of protein structures and the optimal design of spacecraft trajectories [167]. MA can handle complex objective functions. It combines the advantages of local search and genetic algorithm for optimization problems. MA can also be used for global search. It is based on a genetic algorithm and extended by a search technique to further improve individual fitness that may keep with population diversity and reduce the likelihood premature convergence. MA requires a considerable amount of time and memory needed for the improvement of their performances. MA can be used only in non-linear continuous multi-objective combinatorial optimization problems.

Ant Colony Optimization: Ant Colony Optimization (ACO) was proposed by

Marco Dorigo in 1992 [168]. The real power of ants resides in their colony brain. The self-organization of those individuals is very similar to the organization found in brain-like structures. Like neurons, ants use mainly chemical agents to communicate. One ant releases a molecule of pheromone that will influence the behavior of other ants [169]. Ant algorithms are non-deterministic and rely on heuristics to approximate to a sub-optimal solution in cases where the number of combinations is extremely huge and is impossible to calculate using a deterministic algorithm [170]. Ant algorithms are often compared with other evolutionary approaches such as Genetic Algorithms, Evolutionary Programming and Simulated Annealing.

Following advantages have been identified in [171]: (i) It is versatile and can be applied to similar versions of the same problem; for example, there is a straight forward extension from the Traveling Salesman Problem (TSP) to the Asymmetric Traveling Salesman Problem (ATSP). (ii) It is robust and can be applied with only minimal changes to other combinatorial optimization problems such as the Quadratic Assignment Problem (QAP) and the Job-shop Scheduling Problem (JSP). (iii) It can be used for static and dynamic combinatorial optimization problems. (iv) ACO convergence is guaranteed and it used for solving constrained discrete problems. (v) ACO has the powerful feedback capability which can increase the speed of evolution of algorithm to make algorithm convergence possible in the end. When the graph changes dynamically, the ant colony algorithm can run continuously and adapt to changes in real time. Some of the disadvantages are: (i) ACO's convergence rate is slow in comparison to other heuristics. (ii) ACO performs poorly for larger city in TSP. (iii) In ACO, there is no centralized control to guide and provide good solutions. (iv) ACO can be applied to only discrete problems and theoretical analysis in ACO is difficult.

Particle Swarm Optimization: Particle Swarm Optimization (PSO) is a method for performing numerical optimization without explicit knowledge of the gradient of the problem to be optimized. PSO is one of the latest evolutionary optimization techniques inspired by nature and was introduced in 1995 by Kennedy and Elberhart [172]. It simulates the process of a swarm of birds preying. It has a better ability of global searching and has been successfully applied to many areas. A flock or swarm of particles is randomly generated. Initially, each particle position represents a possible solution point in the problem space. The fitness value of each particle is evaluated by the objective function to be optimized. Each particle remembers the coordinates of the best solution (gbest) achieved so far. The coordinates of the current global best (pbest) are also stored. The key advantages of this approach are: (i) PSO is a robust stochastic optimization based

on the movement and intelligence of swarms. There is no selection and crossover parameter like genetic algorithm. (ii) PSO is easy to implement, few parameters to adjust, computationally efficient etc. (iii) PSO is efficient for global search algorithm. Some of the disadvantages are: (i) PSO has week local search. (ii) PSO has slow convergence rate in refined search strategy.

Bacterial Foraging Optimization: Bacterial Foraging Optimization (BFO) algorithm was proposed by Passino [173]. It is population based numerical optimization algorithm based on foraging behavior of Escherichia coli bacteria. In the foraging theory, the objective of the animal is to search and obtain nutrients in a fashion that energy intake per unit time (E/T) is maximized. Foraging is a process in which a group of bacteria moves in search of food in a region, they decide whether or not to enter into a possible food region and then search for a new food region so as to get high quality of nutrients. The bacterial foraging process consists of three main mechanisms: Chemotaxis, Swarming, Reproduction and Elimination-dispersal event. Chemotaxis is the process of simulating the movement of E.coli bacteria, which is carried in a flagella, through swimming and tumbling. The cell also repels a nearby cell in the sense that it consumes nearby nutrients and so it is not physically possible to have two cells at the same location. A bacterium in times of stress releases attractants to signal the bacteria to swarm together. After chemotaxis steps, a reproduction step is taken. Fitness value of bacteria is sorted in an ascending order.The least healthy bacteria eventually dies while each of the healthier bacteria (those yielding lower value of the objective function) asexually splits into two bacteria, which are then placed in the same location. This keeps the swarm size constant. Elimination event may occur due to sudden changes like a significant local rise of temperature or a part of them may move to other regions in the environment that will effect the behavior of bacteria heavily. The elimination and dispersal event destroys the performance of chemotaxis event but dispersal may place bacteria near good sources of food [174]. The main advantages of BFO are: (i) It is easy to implement, few parameters to adjust, computationally efficient, etc. (ii) The BFO algorithm is more adaptive and can be easily applied to real world optimization problems and is efficient for global search. (iii) It also avoids the chance of premature convergence. (iv) BFO is robust and flexible to implement. (v) Its performance is high with respect to speed of convergence, quality of solution and rate of success.

Table 2.3 describes the comparison between the above discussed Grid scheduling heuristic approaches w.r.t features. The comparison of the above described approaches depicts that BFO is better in comparison to other heuristic approaches

as BFO is the only technique which is able to attain optimal scheduling decision by satisfying QoS services and can thus be applied to a Grid environment.

Table 2.3: Comparison of Different Heuristic Approaches

Heuristics/ Features	Parameters	Convergence	Premature Convergence	Services	Local/Global Search	Optimization Problems
Tabu Search	Less parameters	Guaranteed convergence	Prevent premature convergence	conceptual, Simpler, easy to implement, no special memory requirement	Low global search	Combinatorial optimization
Hill climbing	Less functions	No guaranteed	Prevent premature convergence	Simpler and straight forward	Local search	Simple Optimization Problem
Simulated annealing	Less Functions	Converge In asymptotic time	Premature convergence	Easy to code, robust heuristic	Local search	Combinatorial optimization Problems
Genetic Algorithm	More functions	No guaranteed	premature Convergence	No need analytical knowledge, easy to run and implement	Global search capability	Multi objective optimization
Memetic Algorithm	More functions	Guaranteed convergence	Less chance of premature Convergence	Flexible	Global search	Complex objective functions, non-linear multi objective Combinatorial optimization Problems
Ant Colony Optimization	Less functions	Guaranteed convergence	avoid the premature Convergence	Versatile, robust	Global search	Static and dynamic Combinatorial optimization Problems
Particle Swarm Optimization	No function like genetic algorithm	Slow convergence rate	Less chance of premature Convergence	Robust	Global search	Stochastic optimization
Bacterial Foraging Optimization	No function like genetic algorithm	Better convergence rate	Avoid premature Convergence	Flexible & Robust	Global search	Real-world optimization

2.4.4 Meta-heuristics

Meta-heuristics support in decision-making with robust tools that provide high-quality solutions to important applications in business, engineering, economics and science in reasonable time horizons [175]. Some of the advantages of meta-heuristics are: (i) It is an iterative master process that guides and modifies the operations of subordinate heuristics in order to produce high quality solutions. (ii) Meta-heuristics are very flexible to solve real problems. (iii) They are often used for global optimizers. (iv) It often robust to problem size, problem instance and

random variables. Meta-heuristics have certain disadvantages which are as follows: (i) They perform well on a particular real-world problem but may not work on all problems. (ii) They may produce very poor solutions for other problems or even for other instances of the same problem. It requires extensive knowledge in both problem domain and appropriate heuristic techniques. (iv) They are quite expensive to implement. Meta-heuristics are not suitable in those situations when problems data and business requirements change frequently over time. (v) Optimality may not be guaranteed in meta-heuristics. (vi) There is a lack of theoretic basis and it requires multiple search parameters. (vii) Meta-heuristic algorithms like tabu search, ant colony etc have different searches but sometimes different searches may yield different solutions to the same problem.

2.4.5 Hybrid-heuristics

Hybrid strategies have been constructed to exploit the meta-heuristic techniques. To get a better result of the genetic algorithm, it has been hybridized with local search methods as tabu search, simulated annealing, etc. The advantage of parallel hybrids implemented on shared-memory parallel architectures is their simplicity [176]. The main advantage of hybrid-heuristic is that it has a better convergence. Hybrid-heuristics are more efficient in comparison to genetic algorithms. Some of the disadvantages are that hybrid-heuristics are not easy to implement and are time consuming.

2.4.6 Hyper-heuristics

The term Hyper-heuristics describe heuristics to choose heuristics in the context of combinatorial optimization. A hyper-heuristic can be seen as a high level methodology which when given a particular problem instance or class of instances and a number of low-level heuristics, automatically produces an adequate combination of the provided components to effectively solve the given problems [177]. This approach has several advantages: (i) Hyper-heuristics operate in a space of heuristics choosing and applying one low-level heuristic from a given set at each decision point. (ii) Hyper-heuristics do not require knowledge of each low-level heuristic, their working or the contents of the objective function of the problems. (iii) Hyper-heuristics are robustness and reapplicability heuristics. Some of the disadvantages of hyper-heuristics have been identified by Chakhlevitch K. et al. [178]. Some hyper-heuristic techniques make use of additional problem specific

knowledge. Such knowledge can be used to describe the current state of the problem in order to select a suitable low-level heuristic in hyper-heuristics employing learning classifier systems. In indirect GAs, a portion of problem-specific information is often injected into the chromosome. For many hyper-heuristics, a significant amount of parameter tuning is required in order to find good parameter settings for a given problem. A large number of problem instances may be required for training and testing of the method in order to accumulate enough knowledge to make the right choice of low-level heuristics. However, in case of any real-world problem, the data may not be easily available and randomly generated instances may not adequately represent the real distribution. Many hyper-heuristic methods are only tested on a relatively simple benchmark problems for which the best solutions (often optimal) as well as an effective low-level heuristics are known in advance. There is no evidence that such hyper-heuristics would be effective in more complex real-world situations.

After the analysis of heuristic approaches, it has been analyzed that local heuristics may not be sufficient to find an optimal solution to combinatorial optimization problems. So, meta-heuristic techniques are used to solve such type of problems efficiently but it also requires extensive knowledge in both problem domain and appropriate heuristic techniques. Moreover, meta-heuristics are quite expensive to implement. As compared to meta-heuristic, hyper-heuristic can be seen as a high-level methodology [177]. The main difference between meta-heuristic and hyper-heuristic is that the former operates directly on the problem search space to find an optimal solution where as the latter finds the heuristic/problem solver to solve the problems. After the analysis of the heuristic approaches, it has been analyzed that hyper-heuristic is better in comparison to other heuristic approaches like meta-heuristics and hybrid-heuristics. Hyper-heuristic approach is more pliable to Grid environment where both the resources and jobs are highly diverse and dynamic in nature.

This section discussed about the Grid scheduling heuristic approaches and the next section discusses about QoS in Grids.

2.5 QoS in Grids

QoS requirements can be of various kinds due to the heterogeneous nature of the Grid resources. In Grid environment, QoS issues not only include network QoS and device QoS issues, but also include resource QoS issues [179]. With the emergence of distributed multimedia applications however, QoS has become a major

issue in distributed systems research [180]. In distributed systems, QoS is applicable in the following areas described by Andrew Campbell et.al [180] (i) message passing services, which allows programmer to explicitly send a message between two or more processes in a distributed system (ii) remote invocation, which allows operations in a server process to be invoked by a client process and (iii) stream services, which are connections that support the transmission of continuous media flows [180]. QoS support is of paramount importance, given the inherent shared nature of Grid services and the limited capabilities of hardware and software resources available to satisfy client requests [181]. Sun H. X et al identified QoS policies based on resource sharing in grid computing in [181]. Along with the maturity of systems like Websphere, Gnutella, Skpe, Seti@home, Condor, PPlive and Globus, provision of QoS for these newly emerged computing platforms has also become one of the most important criteria [181]. In a shared network environment like Grid, there are several new issues related to QoS support that do not arise in a single computer system. He Sun. X et al has given resource availability, parallel processing and no-centralized control as the major issues. These issues of QoS make Grid computing extremely challenging. Providing non-trivial QoS is one of the primary goals of the Grid approach, as without such support, distributed resources becomes unusable [60]. In OGSA, the QoS characteristics of physical resource cannot represent that of logical resource. To convert the user's QoS request to particular QoS parameters that are in Grid becomes a problem which needs to be solved urgently [179].

2.5.1 QoS parameters

QoS can be categorized into various parameters. These parameters can be used for evaluating the Grid services. According to QoS parameters, user can easily evaluate the kind of services they are receiving.

QoS Parameter Categorization

QoS parameters categorization have been done by Kyriazis et al. in [182]. On logical basis, QoS parameters can be categorized as follows:

- User-defined Parameters: User defined parameters which relate to requirements / constraints that the user who initiates the workflow would like to pose, such as cost restrictions (e.g. maximum overall cost).

- Application Parameters: Application parameters which relate to the offered QoS from the application perspective. For example, the application configuration could play a significant role to the availability of the task to be executed.

- Resource Parameters: Resource parameters which relate to all types of resources, including computational, storage and network resources.

For Grids, QoS parameters are mainly related with resources like cost of the resources and resource utilization. Resources are of two types as defined by Xiaozhi Wang et.al in [179], soft resource and hard resources. Soft resources include data, information and software. Hard Resource includes devices like CPU, monitor, etc.

2.5.2 SLA in Grids

Managing QoS in distributed environment like Grid is a challenging task because SLA must be established and enforced both globally and locally at the components involved in the execution of tasks [6]. SLA is one of the most important elements of QoS in Grid computing. SLA defines a mutually agreed upon set of user's expectations and provider's obligations. SLA encodes QoS parameters such as resource availability, response time, advance reservation and completion deadlines [183]. The SLA is typically specified through a template containing both quantitative and qualitative information [184]. Due to the heterogeneous nature of the Grid environment, user must be given some form of commitment and assurances on the top of the allocated resources such as performance, security, availability, latency, throughput as well as in terms of responsibilities for dealing with invalid conditions fail over policies or backups and more. Those terms need to be agreed upon before use and manifested in the form of an SLA [185]. SLA allows clients to understand what to expect from the resource without knowledge of resource owner's policies [186]. SLA is a great tool to enforce different QoS policies in a Grid environment [181]. A detailed description of SLA components have been defined in [187]. Czajlowski K et al. defined three types of SLA's in [186]. Figure 2.3 describes the three types of SLAs as:

i. Task Service Level Agreements (TSLAs): In Task SLA, one negotiates for the performance of an activity or task.

ii. Resource Service Level Agreements (RSLAs): In Resource SLA, one negotiates for the right to consume a resource.

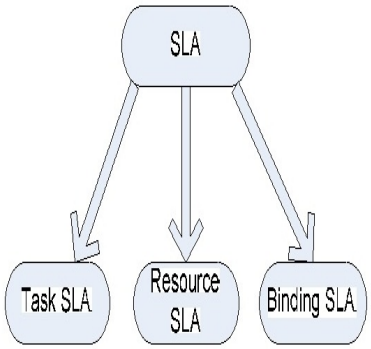

Figure 2.3: Types of SLA

iii. Binding Service Level Agreements (BSLAs): In Binding SLA, one negotiates for the application of a resource to a task.

2.6 Problem Formulation

Any distributed computing environment including Grid, needs to satisfy scalability, management and performance demands. Among what clearly distinguishes Grid from other distributed computing environments is its resource management requirements, which cannot be addressed by existing resource management mechanisms for distributed environments. It is also apparent from the literature survey that there are various mechanisms for Grid scheduling. The main features of the schedulers include advance reservation of the resources and job submission. In Grid environment, the resources are heterogeneous and can be geographically located all over the world. These resources i.e computing, storage, networking, scientific instruments can be accessed by using different policies. For the resource management technique to be efficient and in an understandable form, there is a need to associate resource provisioning with resource management i.e. there is a need to attach QoS parameters with resource provisioning and an efficient resource scheduling algorithm.

Based on the findings of the literature available and the trends in Grid computing, following problem has been formulated for this research work: Unless resource provisioning is considered a fundamental capability, predictable QoS cannot be delivered to Grid consumers. So it is required to design a resource provisioning policy based on QoS parameters for Grid environment. There is a need to design an efficient algorithm for scheduling the resources in a Grid environment, based on resource provisioning. This thesis attempts to propose and implement Resource

Provisioning and Scheduling Framework which provisions and schedules the resources in an effective way.

The main objectives of the proposed work are:

1. To analyze the Grid Resource Provisioning and Scheduling techniques.
2. To design QoS parameter(s) based Resource Provisioning Policy (RPP) for a Grid Environment.
3. To design an efficient resource scheduling algorithm based on above designed Resource Provisioning Policy.
4. To validate the proposed algorithm in a Grid Environment.

This chapter focused on exploring the existing resource provisioning and scheduling techniques in Grid environment. It analyzed existing resource provisioning strategies, scheduling in Grid middleware, Grid schedulers and heuristic approaches. The next chapter proposes a resource provisioning and scheduling framework to address the issues identified in problem formulation and to accomplish the objectives of this research work.

Chapter 3

Proposed Resource Provisioning and Scheduling Framework

The previous chapter discussed the research work accomplished in the area of Grid resource provisioning and resource scheduling. The study depicted that resource provisioning and scheduling challenges in Grid computing have not been addressed to a large extent. To provide a solution to resource provisioning and scheduling problems, a resource provisioning and scheduling framework has been proposed and designed in this chapter.

The proposed framework offers QoS parameter(s) based resource provisioning policies and a resource scheduling algorithm for Grid environment. A resource provisioning and scheduling framework provisions and schedules the resources along with achieving the practical constraints. QoS parameter(s) based resource provisioning policies have been designed and the policy rules have been specified in XML schema.

This chapter firstly discusses the requirements, architecture and components of the proposed framework. Then, it also discusses the objectives and commitments of QoS parameter(s) based resource provisioning policies. Finally, it illustrates the methods for expressing and exposing these policies in the Grid environment.

3.1 Goals of the Proposed Framework

The laid down research objectives have been accomplished through a proposed resource provisioning and scheduling framework where resource providers give the facility of resource provisioning to the user for optimum results, better services and avoid violations of the service level guarantees. Major Goals of the proposed framework are as follows:

- Providing QoS parameter based Resource provisioning policies and resource provisioning based scheduling.
- The implementation of this framework enables user to analyze customer requirements and define processes that contribute to the achievement of a product or service that is acceptable to the consumers.
- Framework assists organizations in enhancing customer satisfaction and contributes directly to the company's growth and institutional progress.

3.1.1 Framework Requirements

The detailed requirements for designing a system, which furnishes the solution for the desired problems are yet to be gathered. Unified Modelling Language (UML) [188] has been used to study, analyze and validate the functional requirements of the framework from different perspectives .
Requirement Analysis:

The key functions of the proposed resource provisioning and scheduling framework are:

- User Authentication
- SLA form for resource provisioning
- Searching Policy Repository
- Specification of the desired resources
- Execution of Resource Provisioning for Job Execution

These functional requirements of the Framework have been analyzed with the help of Use Case Diagrams and Sequence Diagrams as follows:

a. User Authentication- This use case shows the functionality of the system from the prospective of each user of the system. A use case is a coherent unit of functionality expressed as a transaction among actors and the system. In first use case, the successful registration of the user is demonstrated. User tries to login to access the resources and then he is asked to submit the basic information in registration page and password. Administrator confirms the registration of the user as shown in Figure 3.1.

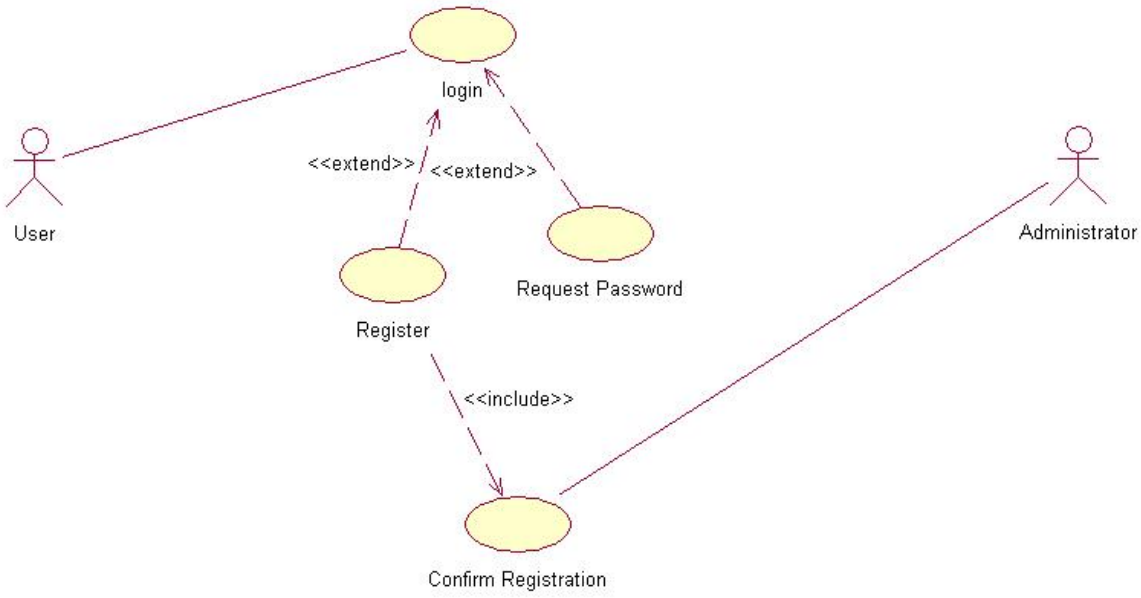

Figure 3.1: Use Case Diagram for User Authentication

After the authentication, user demands the resources for application's execution. Resource Information Center (RIC) will pass information about the availability of the resources to the Resource Provisioning Manager (RPM). RPM will provision resources to the user and then he will be able to access the resources as shown in Figure 3.2.

b. Sequence diagram of the successful execution of resource provisioning- In UML, sequence diagrams are useful design tools because they provide a dynamic view of the system behavior, which can be difficult to extract from static diagrams or specifications. In this case, the successful execution of resource provisioning has been shown using a sequence diagram in Figure 3.3. After login through the portal, user tries to access the resources. The user has to fill his requirements in the form of SLA. After going through

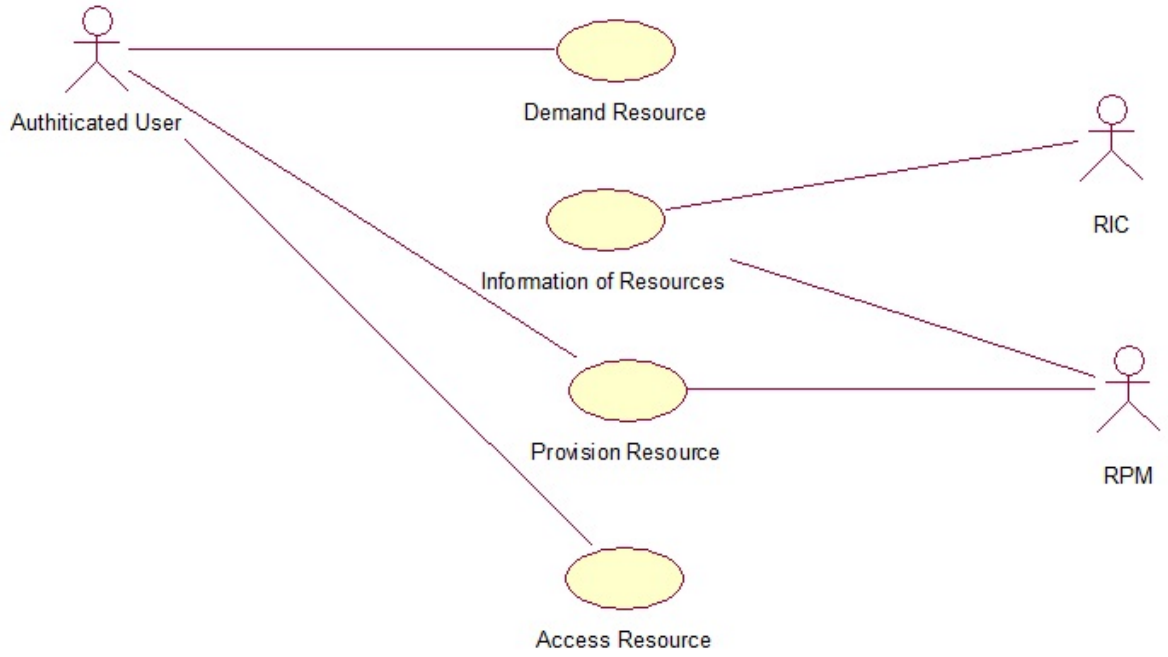

Figure 3.2: Use Case Diagram for Resource Provisioning

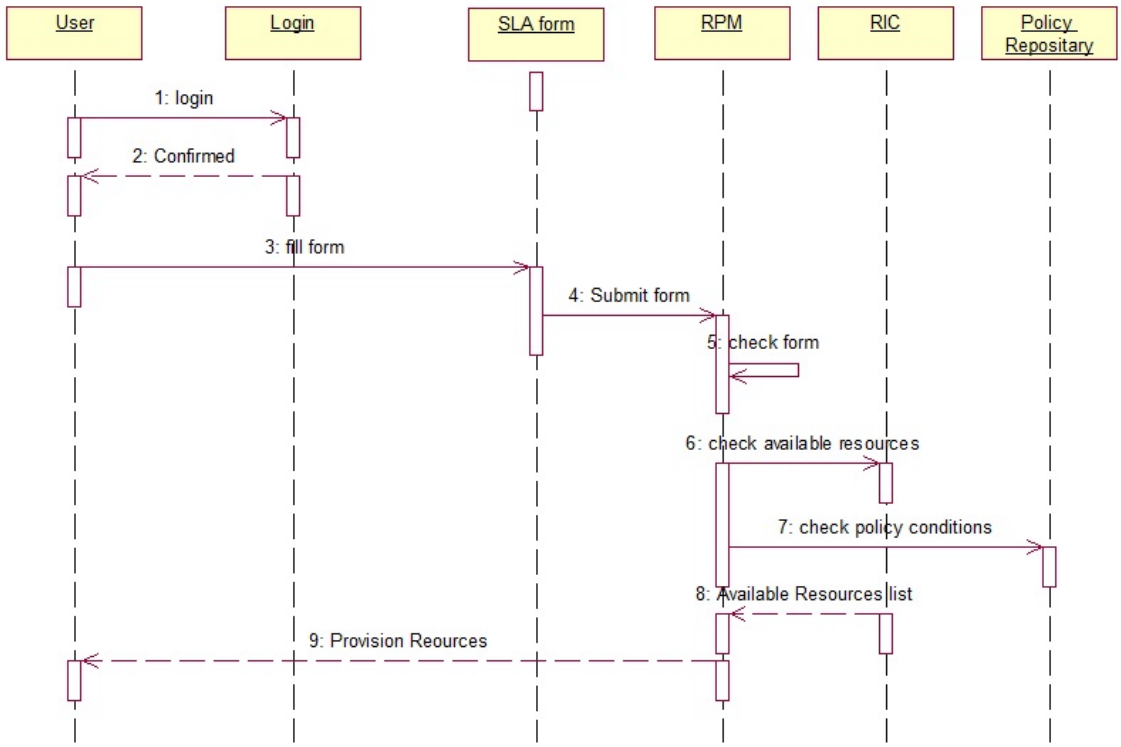

Figure 3.3: Sequence Diagram of Successful Execution of Resource Provisioning

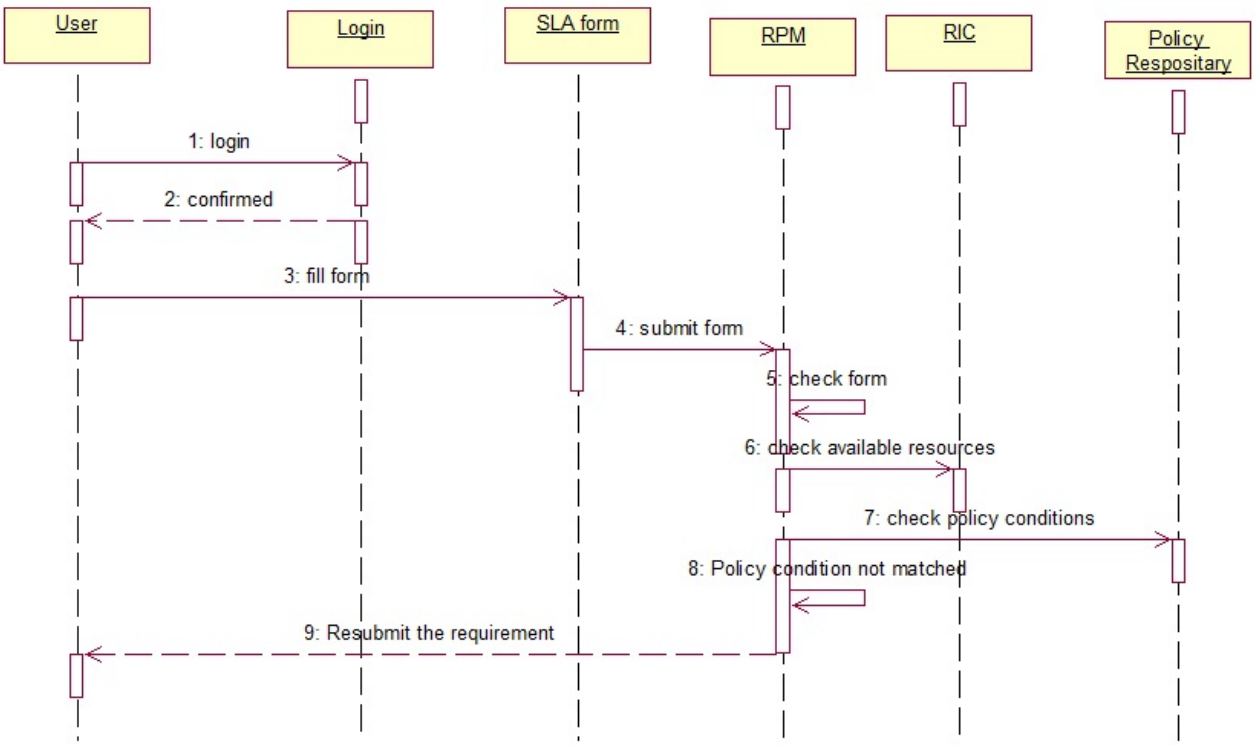

Figure 3.4: Sequence Diagram for resubmission of user's requirements

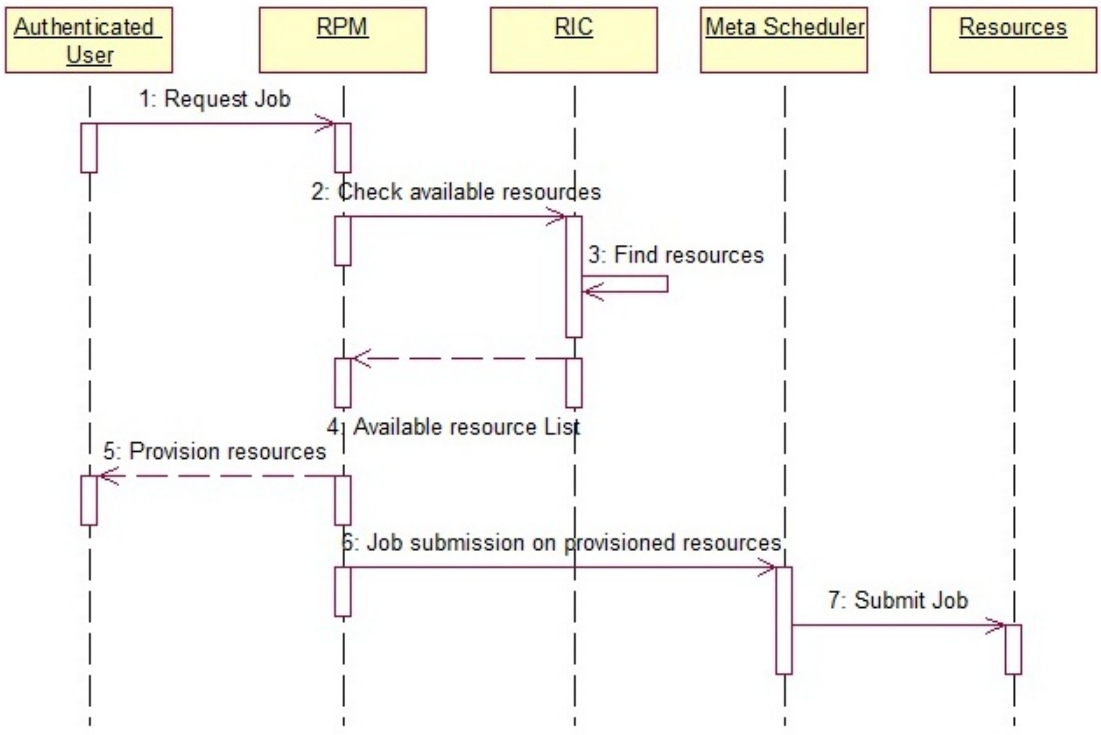

Figure 3.5: Sequence Diagram for job submission

the SLA form, RPM tries to take the information of available resources from the resource information center and simultaneously checks the policy conditions. If the policy conditions match then the RPM provisions the resources to users.

c. Sequence Diagram for resubmission of user's requirements - In this sequence diagram, the request of the resources for the execution of application through resource provisioning is shown. After login through the portal, user will try to access the resources. The user has to fill his requirements in the form of SLA. After going through the SLA form, resource provisioning manager tries to take the information of available resources from the resource information center and simultaneously will check the policy conditions. If the policy conditions do not match then the RPP will not provision the resources to users as shown in Figure 3.4.

d. Sequence Diagram for job submission - In this sequence diagram, job submission after resource provisioning is done. Authenticated user requests for job submission to resource provisioning manager. After resource provisioning, RPM submits the list of provisioned resources and applications to meta-scheduler. Then, meta-scheduler performs his task for optimal job's execution on the provisioned resources as shown in Figure 3.5.

3.1.2 QoS Requirements of the Framework

A Grid application is able to negotiate an SLA with the Resource Management System (RMS) on the basis of QoS [59]. The idea of QoS or SLA can be extended to placing bounds on the resource discovery or other Grid provided services such as data migration, scheduling, etc. So, to provide provisioning and scheduling, first of all, QoS parameters should be defined.

As per the literature survey, following QoS parameters have been identified for designing the resource provisioning and scheduling framework:-

a. Cost: Cost is identified as per unit of resources that are consumed by the users for execution of their applications.

b. Time: Time is calculated by subtracting the start time from the deadline time of user's application.

c. Security: Security is based on trust values of nodes and the trust of the node is identified on the basis of the past transactions/interactions and present

environmental characteristics.

d. Reliability: Reliability can be defined as the fault tolerance of the node. The fault tolerance of the node can be checked for data storage and other tasks. A performance criteria (like data transfer and computation capacity) is used to measure the reliability of any node.

The next section discusses the mode of operation of the proposed framework in detail.

3.2 The Proposed Framework: Mode of Operation

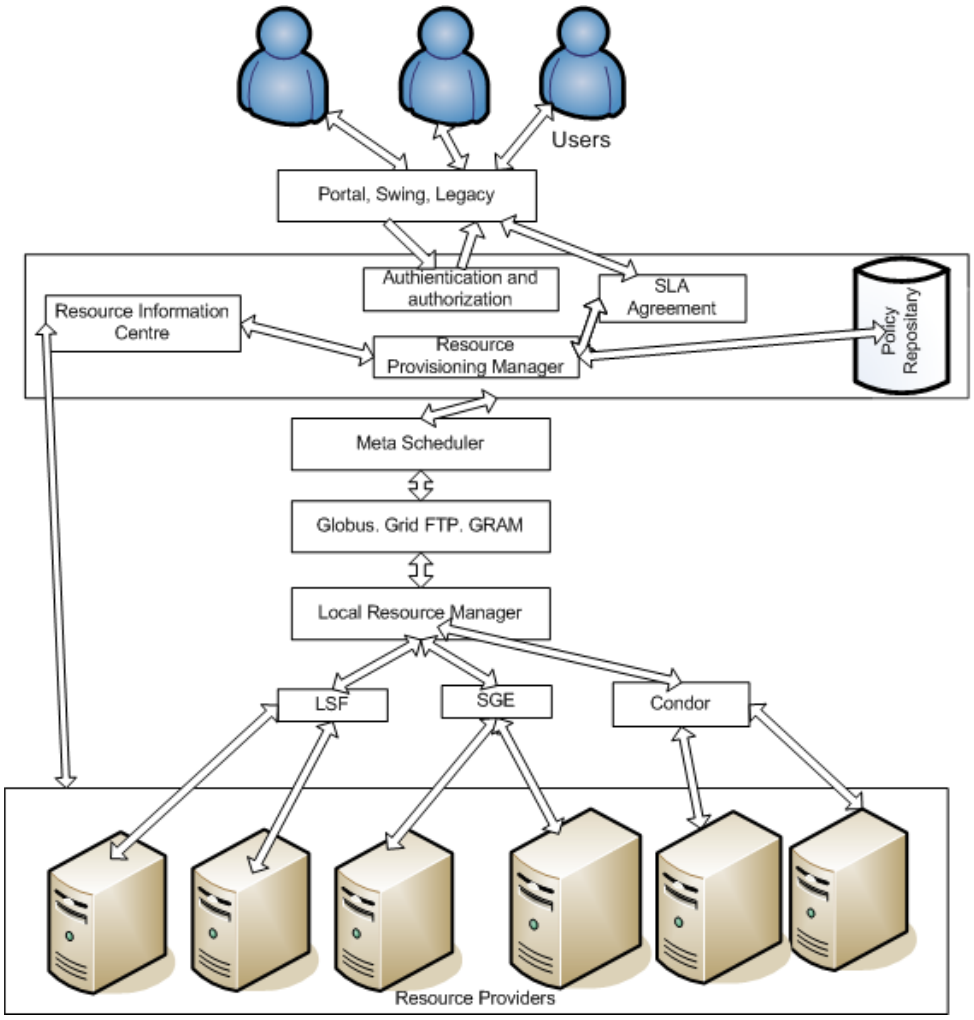

Figure 3.6: Resource Provisioning and Scheduling Framework

In the proposed framework (as shown in Figure 3.6), Resource Provisioning Manager (RPM) performs the tasks of managing the SLAs, maintaining information about the resources and dynamically updating the resources' status. In this process, RPM keeps a check over the provisioned resources and the proper execution of the jobs, information about SLAs and manages the resources' status and it also fills the policy repository. The XML formats of policies are stored in XML database that is depicted in the policy repository as shown in Figure 3.6. XML has been chosen as it is designed to be self-descriptive and it is a W3C recommendation [189]. The main aim of the resource provisioning and scheduling framework is to provision the resources to the Grid user with QoS and then schedule the application on the ingredient resources. The framework executes the requests in the following manner:

- In Resource Provisioning and Scheduling Framework, first of all user tries to access the resources through portal, swing or legacy. After that, the task of user's authentication and authorization is performed.

- After authentication, resource provisioning manager gives a SLA form to an authorized user and the user fills it to send the request for the availability of particular resources with proper specification for the execution of their application.

- RPM takes the information from the appropriate SLA. After studying the various parameters which the user has demanded, manager checks for the available resources. The selection of resources is made on the basis of QoS parameters defined in the SLA form.

- RPM then collects the information of available resources from the Resource Information Center (RIC). RIC contains all the information about the available resources.

- RPM provisions the requested resources to the user for the execution of application in the Grid environment only if the requested resources are available with the manager, according to the policy conditions stored in the policy repository.

- If the requested resources are not available according to the QoS based resource provisioning policies (discussed in the next section) to satisfy the QoS constraints then the resource provisioning manger asks to resubmit the QoS requirements again in the form of SLA.

- After provisioning of the resources, job is submitted through the Grid middleware.
- In the next step, the meta-scheduler via middleware communicates with the local resource manager for job submission. Local resource manager corresponds to different resource providers and the job is submitted to the provisioned resources.
- After getting the result, local resource manager gives it to the Meta scheduler through local scheduler. The result is then sent back to the RPM.
- Grid application gets the information and finally, the user collects the result.

Thus, this framework exhibits how QoS based resource provisioning can be done in the Grid environment. Resource provisioning policies have been identified as a part of this framework. QoS parameters based resource provisioning policies for the Grid computing environment are required to minimize the complexity of provisioning for job execution in Grid computing. With the use of a uniform policy, the number of migrations and complexity of policies can be less.

Next section discusses the detailed description of the QoS parameter based Resource Provisioning Policies.

3.3 QoS based Resource Provisioning Policies

Resource provisioning and scheduling framework provisions the resources on the basis of QoS based resource provisioning policies. The terminology associated with the policy is as follows:

- Resource Consumer (RC): Resource Consumer is an entity that demands resources for the execution of its applications.
- Resource Information Center (RIC): Resource Information Center collects all the information about the available resources.
- Resource Provisioning Manager (RPM): Resource Provisioning Manager is an entity that provides the resources to resource consumers as per the requirement of user's applications and keeps a track of provisioned resources.
- Resources: Resources can be clusters of computers, network latency, memory space, storage capacity, files and attached peripheral devices etc.

This policy standard is based on ISO:9000-2000, RFC 4745 [190].

3.3.1 Objectives and Commitments

The intent of QoS parameter(s) based Resource Provisioning Policy is to ensure that the policy will provision resources for the execution of job with QoS. It facilitates to:

i. Clearly understand the current and potential future requirements and expectations of the resource consumers.

ii. Deliver resources and services of the highest practicable quality, reliability and consistency that meet resource consumer's requirements.

iii. Establish and measure performance and customer satisfaction against appropriate objectives with security and reliability.

iv. Measure an appropriate level service performance and customer satisfaction by minimizing cost and time.

v. Enhance customer satisfaction by meeting all of its requirements.

Based on the four QoS parameters, four resource provisioning policies have been proposed and designed.

3.3.2 Cost based Resource Provisioning Policy (CRPP)

Under cost QoS parameter, a resource is used for the provisioning of job execution on consideration of cost of the resources. Cost is identified as per unit of resources that are consumed by the users for execution of their applications. It is an important aspect to be considered at the time of resource provisioning. After the minimization of cost of the resources, resources are provisioned. Thus, cost based QoS can be provided to the resource consumer. An XML schema of CRPP is as follows:

```
<?xmlversion="1.0"encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"attribute
Form Default="unqualified">
<xs:elementname="Cost">
<xs:annotation>
```

```
<xs:documentation>
Resource Provisioning Policy for Cost Operation
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:elementname="application execution service">
<xs:complexType>
<xs:attributename="resource"type="xs:string"
use="required"/>
<xs:attributename="type"type="xs:string"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:elementname="compute QoS parameter Cost"
minOccurs="0">
<xs:complexType>
<xs:attributename="capacity"type="xs:integer"
use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

3.3.3 Time based Resource Provisioning Policy (TRPP)

This resource provisioning policy has been made on the basis of time and takes static information as input. Time is calculated by the manager by subtracting the start time from the deadline time after the users have submitted their deadlines . The resource manager checks the resource providers' list to know about the resources that can satisfy the requirements of the users. This information is generally based on pre-execution analysis of the job or on historical data gathered after any previous execution. Suppose, if any node is available but no job is submitted to this node, then the performance of this node is predicted by its configuration like memory, processor's speed etc. If the configuration is high, then the response will be fast or vice versa. All the information is stored in RIC as shown in Figure 3.5. Then, the resource manager provisions the resources to users. The main aim of the TRPP is to minimize the time.

```
<?xmlversion="1.0"encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"attributeFormDefault="unqualified">
<xs:elementname="Time">
<xs:annotation>
<xs:documentation>
This is Time based Resource Provisioning Policy
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:elementname="application execution service">
<xs:complexType>
<xs:attributename="resource"type="xs:string"
use="required"/>
<xs:attributename="type"type="xs:string"
use="optional"/>
```

```
</xs:complexType>
</xs:element>
<xs:elementname="compute QoS parameter Time" minOccurs="0">
<xs:complexType>
<xs:attributename="startTime"type="xs:dateTime"
use="required"/>
<xs:attributename="endTime"type="xs:dateTime"
use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

3.3.4 Security based Resource Provisioning Policy (SRPP)

Under security QoS parameter, a resource is used for provisioning the job execution on consideration of security of the node. To enable more effective and security aware resource provisioning, it is desirable to know the Security Demand (SD) from Grid users at the time of job submission and the Trust Level (TL) assured by a resource provider at the Grid site. The first step is to issue a SD to all the available resource sites which is done by the user. The trust model requires assessing the resource site's trustworthiness, called the TL of a node. This information is taken from RIC unit. It is calculated to avoid run-time failure. Trust is the firm belief in the competence of an entity to behave as expected such that this firm belief is a dynamic value associated with the entity and it is also subject to the entity's behavior and applies only within a specific context at a given time [191]. When an application is scheduled to execute on a resource, the trustworthiness of node also reflects the reliability of the node's services. The TL quantifies how much a user can trust a site for successfully executing a given job. A job is expected to be carried out successfully when SD and TL satisfy a security-assurance condition ($SD \leq TL$) during the job mapping process [192].

The failure probability of a resource/machine as a function, which is dependent on the difference $SD_i - TL_k$ has been defined. The formula (4.1) presented below expresses the failure probability of a resource r_k with trust level value TL_k , to a job j_i with a specific SD_i value [193]. In the resource provisioning and scheduling framework, a job could be delayed or dropped, if the site TL is lower than the job SD. The SD is a real fraction in the range [0 , 1] with 0 representing the lowest and 1 the highest security requirement. The TL is in the same range with 0 for the most risky resource site and 1 for a risk-free or fully trusted site. The negative exponent indicates that the failure probability of a scheduling grows with the difference $SD_i - TL_k$. The failure probability of executing a job, with a job SD on a site with TL, is modeled by an exponential distributed failure function as follows:

$$P_f(j_i, r_k) = \begin{cases} 0, & \text{if} SD_i \leq TL_k \\ 1 - \mathrm{e}^{-\alpha(SD_i - TL_k)} & \text{if } SD_i > TL_k \end{cases} \tag{3.1}$$

All resources in Grid computing are shared and distributed, therefore trust relationship is very crucial for provisioning of resources for job execution. The provisioning of resources is based on the accuracy of the feedback provided about the resources. After the calculation of failure probability of executing a job, resources are provisioned.

```
<?xmlversion="1.0"encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"attributeFormDefault="unqualified">
<xs:elementname="Security">
<xs:annotation>
<xs:documentation>
This is Security based Resource Provisioning Policy
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:elementname="application execution service">
```

```
<xs:complexType>
<xs:attributename="resource"type="xs:string"
use="required"/>
<xs:attributename="type"type="xs:string"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:elementname="compute Trust of node" maxOccurs="0">
<xs:complexType>
<xs:attributename="value"type="xs:integer"
use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema >
```

3.3.5 Reliability based Resource Provisioning Policy (RRPP)

Reliability based RPP is based on reliability of the node. Reliability of the node has to be checked before provisioning of the resources. With the help of reliability parameter, the fault tolerance of the node can be checked for data storage and other tasks. A performance criteria (like data transfer and computation capacity) is used to measure the reliability of any node. Node can be ranked according to its ability and it can be measured in the form of computation power and data transfer capacity.

By analyzing the updated RM's reports, RIC generates the reliability probabilities $P_{r_k}, k = 1, ..., n$ for each resource i. The execution of a job on the resource i can then be aborted with the probability defined as follows:

$$P_{rb}(r_k) = (1 - P_{r_k}) \tag{3.2}$$

The reliability probability (also referred to as prediction of node failure) P_{r_k} was introduced by Rood and Lewis in [194] as the real fraction in the range [0,1], which utilizes historical data to forecast the availability of Grid nodes. The higher P_{r_k} value means the smaller probability of the failure of node execution due to some additional network problems like a disconnection of power and node's failure etc. After calculation of reliability of the node, resources are provisioned.

```
<?xmlversion="1.0"encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"attributeFormDefault="unqualified">
<xs:elementname="Reliability">
<xs:annotation>
<xs:documentation>
This is Reliability based Resource Provisioning Policy
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:elementname="application execution service">
<xs:complexType>
<xs:attributename="resource"type="xs:string"
use="required"/>
<xs:attributename="type"type="xs:string"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:elementname="Compute Reliability of resource or node ">
<xs:complexType>
<xs:attributename="startTime"type="xs:dateTime"
use="required"/>
```

```
<xs:attributename="endTime"type="xs:dateTime"
use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema >
```

3.4 Conclusion

This chapter discussed the proposed Resource Provisioning & Scheduling Framework and QoS parameter based Resource Provisioning Policies as an outcome of the research work. The detailed requirements of Resource Provisioning & Scheduling Framework have been analyzed. Further, the mode of operation of QoS based resource provisioning and scheduling in the Framework has been discussed. In the next chapter, resource scheduling algorithm which caters to the scheduling aspects of the framework has been proposed.

Chapter 4

Resource Scheduling Algorithm

The previous chapter discussed in detail the design of Resource Provisioning and Scheduling Framework and QoS parameter(s) based Resource Provisioning Policies.

In this chapter, a resource scheduling algorithm has been designed considering the QoS expectations of both, the resource providers and resource consumers. As hyper-heuristic is more pliable for Grid environment therefore a novel Bacterial Foraging Optimization (BFO) based hyper-heuristic resource scheduling algorithm has been proposed. This algorithm effectively schedules the jobs on available resources in a Grid environment and simultaneously minimizes the cost and the makespan. QoS parameter(s) based resource provisioning policies that have been discussed in the previous chapter are the basis of the proposed algorithm.

This chapter initially presents the Grid resource scheduling model. Then, the resource scheduling problem as a combinatorial optimization problem has been formulated that minimizes the cost and makespan. It then moves to the design of BFO based hyper-heuristic for resource scheduling algorithm. Finally, the last section provides the detailed description of the proposed algorithm.

4.1 Resource Scheduling Model

Grid scheduling is a two-step process. In step one, the required set of resources is identified as per the user's requests and in the second step, the jobs are mapped on to the actual set of resources thus further ensuring near optimal satisfaction of QoS parameters. For example, suppose that a person has to buy something from a shop. The shopkeeper would ask the buyer about his budget and then he will show the items accordingly. The buyer will now select the most appropriate items amongst all the items shown that would be an exact match for his cost and other specifications.

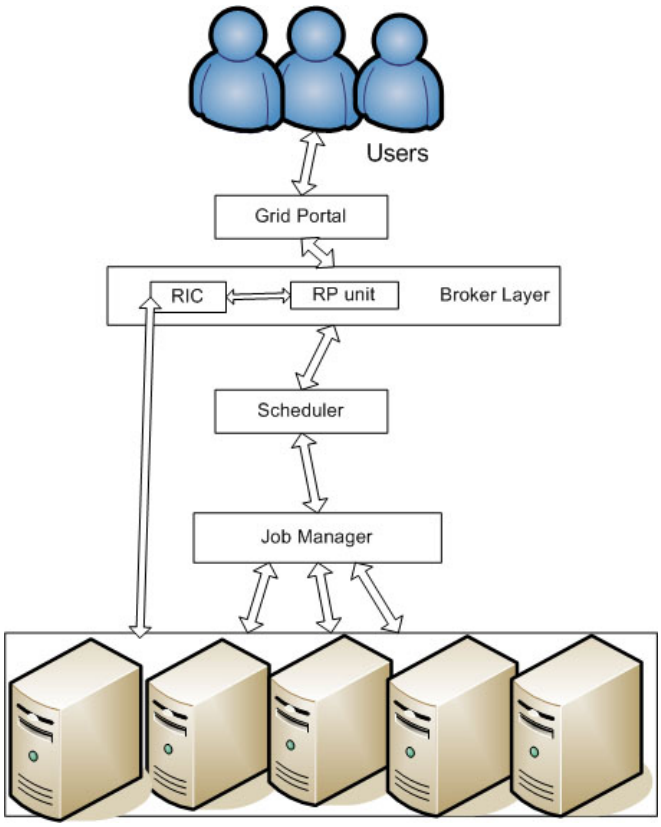

Figure 4.1: Resource Scheduling Model

Figure 4.1 shows the Grid resource scheduling model (elaborated from the proposed framework that has been discussed in the previous chapter).

- Each user tries to access the resources for application's execution through Grid portal. After this, the task of authorization and authentication is performed at the portal side.
- Resource Provisioning (RP) unit takes the information about the available resources from RIC. RIC collects all the information about the resource

provider's resources. Then, the RP unit performs preliminary provisioning for user's requests as shown in Figure 4.1. On the completion of resource provisioning, the task of mapping of jobs to exact resources is performed.

- Grid scheduler further takes all the information from RP unit who has a list of provisioned resources which are available in the Grid environment. According to the status of the resource, scheduler will consult with the Job Manger (JM) for job execution.

- Job Manager is a protocol engine for communicating with a range of different local resource schedulers using a standard message format [195]. Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs for high efficiency and performance. Then the mapping of jobs to resources is performed. Resources can be clusters of computers, memory space, storage capacity, files and attached peripheral devices etc.

This section described the resource scheduling model. The next section would discuss the problem formulation.

4.2 Resource Scheduling: Problem Formulation

To map the best resource to a corresponding job is a tedious task and the problem of finding the best resource - job pair according to user's application requirements is a combinatorial optimization problem. The main goal of the Grid scheduler is to schedule the resources effectively and efficiently. The resources and applications/jobs can leave and join the Grid dynamically. Grid resources are heterogenous and dynamic in nature which makes the scheduling problem an NP-complete problem [98]. In this model, independent jobs have been considered to handle the realistic scenarios as there are many scenarios in which the need of scheduling of independent jobs arises. Firstly, this problem is suitable to Grid systems because of the nature of Grid users, who submit jobs or monolithic applications in an independent manner to the system. Secondly, Grid systems are most useful for massive parallel processing, in which large amounts of data are processed independently [196].

In this model, scheduling problem has been considered from both the user's and resource provider's point of view. User wants to minimize the cost whereas

the resource provider wants to minimize the makespan. Makespan is used to indicate the general productivity of the Grid systems. Smaller values of makespan indicates that the scheduler is planning the jobs in an efficient manner. Cost is an another optimization criteria, which refers to the total cost of the job execution on a particle resource. In order to formalize the problem instance, the Ali et.al computational model has been used [197]. The problem has been mathematically formalized to get an optimal solution.

To consider this problem, a set of independent jobs $\{j_1, j_2, j_3, ...j_m\}$ to map on a set of heterogenous and dynamic resources $\{r_1, r_2, r_3.....r_n\}$ has been taken.

$R = \{r_k \mid 1 \leq k \leq n\}$ is the collection of resources and n is the total number of resources. $J = \{j_i \mid 1 \leq i \leq m\}$ is the collection of jobs and m is the total number of jobs.

The Expected Time To Compute (ETC) value of each application/ job on each resource is assumed to be given by the user-supplied information, experimental data, job profiling and analytical benchmarking. The performance estimation for resource services is achieved by using the existing performance estimation techniques such as: analytical modeling [198] and historical information[199] [200]. Under the ETC simulation model for problem formulation, the following constraints have been considered:

i. Each job to be scheduled for application's execution has a unique id.

ii. Jobs are independent.

iii. Arrival of jobs for execution of application is random and jobs are placed in a queue of unscheduled jobs.

iv. The computing capacity/speed of the resources is measured in Multiple Instruction Per Second (MIPS) as per Standard Performance Evaluation Corporation (SPEC) benchmark.

v. The processing requirement of job is measured in Million Instructions (MI).

vi. Execution time for every job on resource is obtained from the ETC matrix. No of jobs * no of resources gives the size of the matrix and its components are defined as $ETC(j_i, r_k)$. Rows of the ETC matrix demonstrate the estimated execution time for a job on each resource and the columns demonstrate the estimated execution time for a particular resource. $ETC(j_i, r_k)$ is the expected execution time of job j_i and the resource r_k.

A simple weighted sum function of makespan and cost has been used to deal with their simultaneous optimization.

4.2.1 Objective Function

In Grid scheduling, the main goal of the providers is to minimize the makespan where as the goal of the user is to minimize the cost for Grid application. Fitness value is thus calculated as:

$$FitnessFunction = \theta cost + \delta makespan \tag{4.1}$$

$$cost = min(c(r_k, j_i)) \quad for \quad 1 \leq k \leq n, 1 \leq i \leq m \tag{4.2}$$

$$makespan = min(F_{j_i}) \quad for \quad j_i \in J \tag{4.3}$$

where $0 \leq \theta < 1$ and $0 \leq \delta < 1$ are weights to prioritize components of the fitness function. The cost and makespan specified in Equations 4.2 and 4.3 have been used for the definition of an objective function for the independent parallel job scheduling as defined in Equation 4.1.

$$c(r_k, j_i) = c_e(r_k, j_i) \tag{4.4}$$

$$c_e(r_k, j_i) = \sum_{j_i \in J} completion(j_i, r_k) \Big/ completion_{m(j_i)} \times J \tag{4.5}$$

Where as :

$$completion_{m(j_i)} = max_{j_i \in J, r_k \in R} completion(j_i, r_k) \tag{4.6}$$

Cost c(r_k, j_i) is the cost of job j_i which executes on resource r_k. Equation 4.5 denotes $c_e(r_k, j_i)$ user's application execution cost. In Equation 4.6, the $completion_{m(j_i)}$ denotes the maximal completion time of the users job. Makespan is the finishing time F_j of the latest job and can be expressed as Expected Time to Compute (ETC) job j_i on resource r_k. The completion time of a machine must be defined before calculating the makespan. Avail $\_time_{r_k}$ is the time in which the machine can complete the execution of all the previous assigned jobs. Completion time $completion(j_i, r_k)$ indicates the time in which the machine/resource can complete the execution of all the previous assigned jobs in addition to the execution

time of job j_i on resource r_k, as defined below.

$$completion(j_i, r_k) = avail\_time_{r_k} \pm ETC(j_i, r_k) \tag{4.7}$$

The value of completion time has been used to compute the makespan. This mapping is done with an objective of minimizing the cost and makespan simultaneously.

To provide the security and reliability as QoS parameters, makespan and cost have been calculated using the below mentioned equations. The objective function in this case is same as in Equation 4.1. but it includes the values of the $P_f(j_i, r_k)$ and $P_{rb}(r_k)$ probabilities. Makespan has been updated by using Equation 4.9.

$$\widetilde{ETC}(j_i, r_k) = (1 + P_f(j_i, r_k) + P_{rb}(r_k))ETC(j_i, r_k) \tag{4.8}$$

$$completion(j_i, r_k) = avail\_time_{r_k} \pm \widetilde{ETC}(j_i, r_k) \tag{4.9}$$

Cost $c(r_k, j_i)$ is the cost of job j_i which executes on resource r_k in addition to security-assurance cost that is defined in Equation 4.11.

$$cost = Min(c(r_k, j_i)) for 1 \leq k \leq n, 1 \leq i \leq m \tag{4.10}$$

$$c(r_k, j_i) = c_e(r_k, j_i) + c_s(r_k, j_i) \tag{4.11}$$

Equation 4.5 denotes $c_e(r_k, j_i)$ user's application execution cost and Equation 4.12 indicates $c_s(r_k, j_i)$ the cost of security-assured mapping of the user applications.

$$c_s(r_k, j_i) = \sum_{j_i \in J} (P_f(j_i, r_k) \times ETC(j_i, r_k)) \Big/ ETC_{m(j_i)} \times J \tag{4.12}$$

where as :

$$ETC_{m(j_i)} = max_{j_i \in J, r_k \in R} ETC(j_i, r_k) \tag{4.13}$$

In Equation 4.13, $ETC_{m(j_i)}$ is the (expected) maximal computation time of the tasks of the user.

The next section describes the hyper-heuristic based scheduling algorithm.

4.3 Hyper-heuristic based Scheduling Algorithm

Meta-heuristics are heuristics which control the search in a space of solutions performed by a single low-level heuristic. Optimality is not guaranteed in meta-heuristics. It requires multiple search parameters as there is a lack of theoretical basis. Meta-heuristic algorithms like tabu search, ant colony, etc., have different search methods but sometimes different searches may yield different solutions to the same problem. Thus, meta-heuristic approaches that perform well on a particular real-world problem may not work on all the problems. They may produce very poor solutions for other problems or even for other instances of the same problem. Thus, an extensive knowledge in both problem domain and appropriate heuristic techniques is required. A simplistic way of solving problem using meta-heuristic has been shown below:

```
if(problem type(p)==p1) then
apply(heuristic1, P);
elseif(problem(p)==p2) then
apply(heuristic2,P);
elseif(problem(p)==p3) then
apply(heuristic3,P);
elseif.....................
```

One logical extreme of such an approach would be an algorithm containing an infinite switch statements enumerating all finite problems and then applying the best known heuristic for each problem, which is not a good approach. A better approach is to find the best heuristic under the problem condition and then apply different heuristics to different parts or phases of the solution process. A hyper-heuristic operates at a higher-level of abstraction. It selects a low-level heuristic that should be applied at any given time, depending upon the characteristics of the region of solution space currently under exploration [177]. The main aim of the hyper-heuristic is not to compete with the state of the art problem specific approaches, but to provide a general framework which is able to deliver solutions of good quality for a wide range of optimization problems [178]. Figure 4.2 shows the basic hyper-heuristic framework.

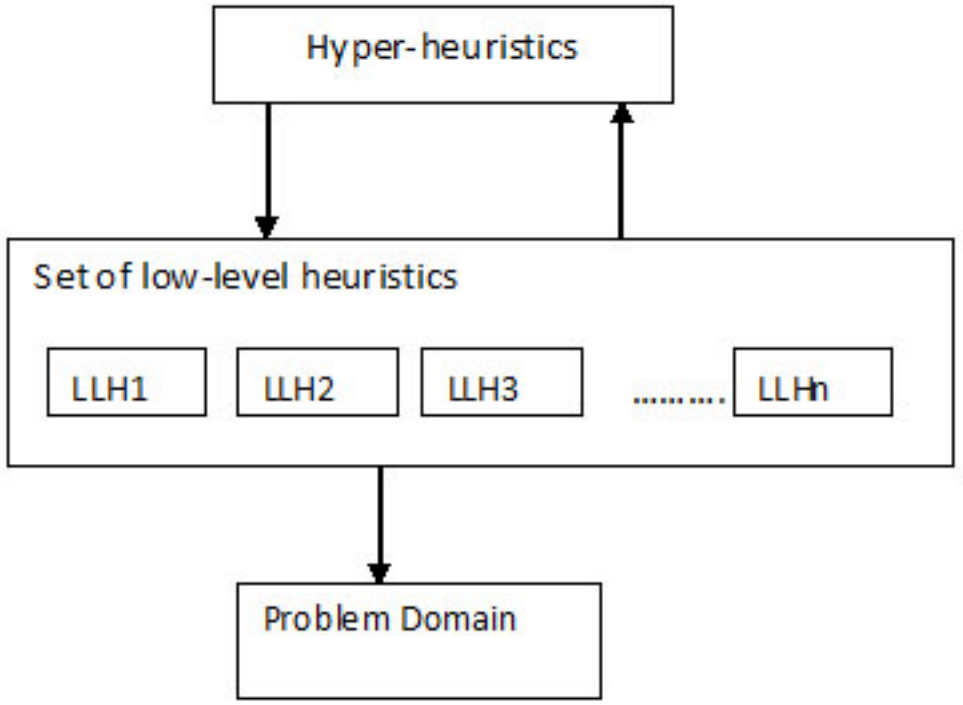

Figure 4.2: Hyper-heuristic Framework[201]

4.3.1 Bacterial Foraging Optimization

Bacterial Foraging Optimization (BFO) algorithm was proposed by Passino et al. [173][202]. It is population based numerical optimization algorithm based on foraging behavior of Escherichia coli bacteria. E.coli bacteria has a control system, which directs its behavior in food foraging. In the foraging theory, the objective of the animal is to search and obtain nutrients so that energy intake per unit time (E/T) is maximized. Foraging is a process in which a group of bacteria move in search of food in a region. They decide whether or not to enter into a possible food region and then search for a new food region so as to get high quality of nutrients. The bacterial foraging process consists of four main mechanisms: Chemotaxis, Swarming, Reproduction and Elimination-dispersal event.

Chemotaxis: It is the process of simulating the movement of E.coli bacteria, which is carried in a flagella, through swimming and tumbling. Two different ways of movement of E.coli bacteria are : Swim in the same direction and tumble in a random direction.

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j) \tag{4.14}$$

Where $\theta^i(j,k,l)$ stands for the current position of the i^{th} individual; j,k and l indicate the numbers of chemotaxis, reproduction and elimination-dispersal events respectively. $C(i)$ is the size of the step taken in a random direction and $\phi(j)$ is the direction angle of the j^{th} step.

If the cost at $\theta^i(j+1;\ k;\ l)$ is better than the cost at $\theta^i(j;\ k;\ l)$ then the bacterium takes another step of size $C(i)$ in that direction otherwise it is allowed to tumble in a random direction $\phi(j)$. This process is repeated until the number of steps taken is greater than the number of iterations in chemotaxis loop, N_c.

Swarming: The cell also repels a nearby cell in the sense that it consumes nearby nutrients and so it is not physically possible to have two cells at the same location. A bacterium in times of stress releases attractants to signal the bacteria to swarm together. Each bacterium also releases repellants to signal the others to be at a minimum distance from it. Thus all of them will have a cell to cell attraction via attractant and cell to cell repulsion via repellant. The cell to cell signaling in E.coli swarm may be represented by the following function:

$$\begin{aligned} J_{cc}(\theta, p(j,k,l)) &= \textstyle\sum_{i=1}^{S} J_{cc}(\theta, \theta^i(j,k,l) \\ &= \textstyle\sum_{i=1}^{S}[-d_{attractant}\, exp(-w_{attractant} \sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] \\ &+ \textstyle\sum_{i=1}^{S}[h_{repeelant}\, exp(-w_{repellant} \sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] \end{aligned} \tag{4.15}$$

Here $J_{cc}(\theta, p(j,k,l))$ represents objective function value to be added to the actual objective function, S is the total number of bacteria, p is the number of parameters to be optimized which are present in each bacterium. It denotes the combined cell-to-cell attraction and repelling effects, where $\theta = [\theta_1, \theta_2, ...\theta p]^T$ is a point on the optimization domain and θ_m^i is the m^{th} component of the i^{th} bacterium position θ^i. $d_{attractant}$ is the depth of the attractant released by the cell (a quantification of how much attractant is released) and $w_{attractant}$ is a measure of the width of the attractant signal. $h_{repellant} = d_{attractant}$ is the height of the repellant effect (magnitude of its effect) and $w_{repellant}$ is a measure of the width of the repellant.

Reproduction: After N_c chemotaxis steps, a reproduction step is taken. Fitness value of the bacteria is sorted in an ascending order.The least healthy bacteria eventually dies while each of the healthier bacteria (those yielding lower value of the objective function) asexually splits into two bacteria, which are then placed in the same location. This keeps the swarm size constant.

Elimination and Dispersal: Elimination event may occur due to some sudden changes like a significant local rise of temperature or a part of them may move to

some other regions in the environment that will effect the overall behavior of the bacteria heavily. The elimination and dispersal event destroys the performance of chemotaxis event but dispersal may place bacteria near good sources of food [174].

4.3.2 Scheduling Algorithm

In this section, the pseudo code of bacterial foraging based hyper-heuristic for resource scheduling in the Grid environment has been discussed. Each bacterium in genome is a partial solution and is represented as a heuristic (eg: select, move, swap, drop) or a sequence of heuristics. A low-level heuristic is operated upon by the hyper-heuristic. Low-level heuristics can be simple or complex and can be implemented as follows :

i. First of all, select the job to be scheduled. The heuristic selects a job from the list of unscheduled jobs and schedule it to the best available resource that is filtered from the resource provisioning list.

ii. Move job j_i from its current resource to some other resource.

iii. Randomly select a job and swap it with some other job.

iv. Finally, remove a randomly selected job from the job pool already scheduled. This is the only heuristic which will move the search into an infeasible region because any job may be unscheduled. But, the search will surely move back into its feasible region by un-scheduling the job that has other valid resources so that it can move into the next iteration.

BFO can be used as a top-level heuristic to manage the overall performance and quality of all the mechanisms. Objective of the BFO is to find the best low-level heuristic that generates the best solution for the resource scheduling problem. The low-level heuristics are then applied in a specific order to find an optimal solution of the problem instance. The selection process of a low-level heuristic in hyper-heuristic stops after a pre-defined number of iterations. The number of iterations have been fixed so as to keep the computation time low. Region of the BFO search space is the possible combination of low-level heuristics. A swarm of bacteria has been endowed with pre-constructed solutions and let them collectively learn the heuristic space, so as to guide their selection of an appropriate low-level heuristic to improve the solution. The pseudo code of bacterial foraging based hyper-heuristic is given in Algorithm 1. The detailed description of the proposed algorithm is as follows:-

Algorithm 1: Bacterial foraging based hyper-heuristic

Data: Number of jobs and number of available resources.
Result: Mapping of the each job to the resources.
begin
initialize resourcelist[Number of Resources]
initialize joblist[Number of Jobs]
Input r = number of iterations
Input n= number of heuristics
Initialize a random feasible solution
S= The number of bacteria in the population
P = Dimensions of the search space
$C(i)$ = The size of the step taken in the random direction specified by the tumble
N_c = Chemotaxis steps
N_s = Swimming length
N_{re} = Reproduction steps
N_{ed} = Elimination and dispersal events
P_{ed} = Elimination and dispersal probability
joblist= get job to schedule(), resourcelist= get available resources()
Elimination and dispersal loop $l = l + 1$
Reproduction loop $k = k + 1$
Chemotaxis loop $j = j + 1$
For $i = 1, 2, 3, ..., S$ take a chemotaxis step for bacterium i as follows.
choose the heuristic
Compute $Fitness(i, j, k, l)$
$Fitness_{last} = Fitness(i, j, k, l)$
Tumble: Generate a random vector $\Delta(i) \epsilon R^n$ with each element $\Delta_m(i)$, $m = 1, 2, 3....p$; a random number on[-1,1].
Move: $\theta(i, j + 1, k, l) = \theta(i, j, k, l) + c(i) * \Delta(n, i)$
$ComputeFitness(i, j + 1, k, l)$
Swim, let $m = 0$
while *($m < N_s$)* **do**
  let $m = m + 1$
  if *($Fitness(i, j + 1, k, l) < Fitness_{last}$)* **then**
    $Fitness_{last} = Fitness(i, j + 1, k, l)$
    update
    $\theta(i, j + 1, k, l) = \theta(i, j + 1, k, l) + c(i) * \Delta(n, i)$
    $Fitness(i, j + 1, k, l) = \theta(i, j + 1, k, l))$
  else
    let $m = N_s$, This is the end of while statement.
go to next bacterium $(i + 1)$ if $i \neq S$ (i.e., go to step Compute $Fitness(i, j, k, l)$) to process next bacterium
if *($j < N_c$)* **then**
  go to step chemotaxis loop
for *($i = 1 to S$)* **do**
  $Fitness^i_{health} = \sum_{j=1}^{N_c} Fitness(i, j, k, l)$

The S_r bacteria with the highest $Fitness_{health}$ values die and the other S_r bacteria with the best values split (and the copies that are made are placed at the same location as their parent) ;
if $(k < N_{re})$ **then**
  next k ;
Elimination-dispersal: For $i = 1, 2, ...S$ with probability P_{ed}, eliminate and disperse each bacterium ;
if $(l < N_{ed})$ **then**
  next l ;
Apply the heuristic ;
while *there are unscheduled jobs in the queue* **do**
  for *every resource is in resource list* **do**
    get the next job from queue ;
    schedule the job on the resource on the basis of fitness;
Repeat each and every step till all the jobs are allocated;

- A resource list is obtained from the resource provisioning unit after provisioning of user's requests. Once the resource list has been obtained, a job list and a random feasible solution are initialized.

- The task to choose the best heuristic from low-level heuristics is started. There are a number of bacterium, each of which represents a hyper-heuristic agent supplied with an initial solution in the solution space and an access to the evaluation function. The bacterium constructs a sequence of heuristics through chemotaxis steps.

- It will then select a low-level heuristic at each decision point and compute fitness function $Fitness(i, j, k, l)$.

- $Fitness(i, j + 1, k, l)$ will be computed and then the process of swim will be started till the bacteria have not climbed too long. If at $\theta^i(j + 1;\ k;\ l)$ the cost ($Fitness(i, j + 1, k, l)$ is better than at $\theta^i(j, k, l)$, $Fitness_{last}$ then the bacterium takes another step of size $C(i)$ in that direction. This swim is continued as long as it continues to reduce the cost, but only up to a maximum number of steps, N_s.

- If $j < N_c$, go to chemotaxis loop. In this case, continue chemotaxis, since the life of bacteria is not over.

- For the given k, l and for each $i = 1, 2, 3, ..., S$, let $Fitness^i_{health} = \sum_{j=1}^{N_c} Fitness(i, j, k, l)$ be the health of the bacterium i (a measure of how many

nutrients it got over its lifetime and how successful it was at avoiding noxious substances). Sort bacteria in order of ascending cost $Fitness_{health}$ (higher cost means lower health). (After each bacterium has visited a certain number of heuristics, the fitness value of each bacterium is sorted in an ascending order to encode a good solution).

- During the reproduction step, a bacterium may choose to reject a new solution if it discovers that it is poorer than the current solution. If $k < N_{re}$ then go to reproduction loop: $k = k + 1$.

- For $i = 1, 2, ...S$ with probability P_{ed}, eliminate and disperse each bacterium which results in keeping the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse one more bacterium to a random location on the optimization domain. If $l < N_{ed}$, then go to the elimination and dispersal loop; otherwise end.

- After selection of a low-level heuristic, it is then applied to the problem. Resource scheduling is performed till there are no unscheduled jobs in the queue.

4.3.3 Algorithm Complexity

For each generation, a given population of bacteria representing a sequence of heuristics is evaluated for fitness. Bacterial Foraging Optimization based Hyper-heuristic (BFOHH) performance is usually measured by the number of fitness function evaluations done during the course of a run. The main operations performed during BFOHH are as follows:

- For fixed population size which is the usual case in BFOHH implementations, the number of fitness function evaluations is given by the product of population size and the number of generations.

- To allocate any resource to an application, a number of iterations is to be done over each user application/job and resource i,e O(mn).

- All the operations are done for each application,i.e., n times. Thus, complexity of the bacterial foraging based hyper-heuristic resource scheduling algorithm is given by $O(mn^2 \times generations \times populationSize)$.

4.4 Conclusion

This chapter presented the Grid resource scheduling model and then formulated the objective function based on QoS parameters. This objective function tries to take care of interests of both providers and users by minimizing the makespan and cost simultaneously. Further, a BFO based hyper-heuristic resource scheduling algorithm has been proposed for the mapping of applications on the ingredient resources. The next chapter discusses the verification and validation of QoS parameter(s) based resource provisioning policies and resource scheduling algorithm.

Chapter 5

Verification and Validation of the Proposed Solution

The previous chapter discussed in detail the problem formulation of resource scheduling and the design of the proposed Bacterial Foraging based Hyper-heuristic resource scheduling algorithm.

This chapter focuses on the verification and validation of the proposed framework. Formal verification of QoS based resource provisioning policies has been done through Z formal specification language. The proposed resource scheduling algorithm has been implemented in GridSim toolkit. Statistical analysis of simulation output has been performed in order to assess the accuracy of the estimated performance indices by using coefficient of variation. The framework has also been validated from various other perspectives similar to the existing frameworks like Dragon, GARA, Intra Grid etc.

At the outset, this chapter outlines the details of verification and the implementation of the proposed work through Z formal specification language. Then the experimental results obtained through GridSim have been discussed from various perspectives such as effect of resource heterogeneity, number of applications, number of resources, etc. Finally, the framework has been validated.

5.1 Verification of Resource Provisioning and Scheduling Framework

Formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy those needs and those who test the results [203]. Resource provisioning and scheduling framework and its policy (stated in chapter 3)can be verified using a formal language like Z specification. In Z [203][204], schemas are used to describe both static and dynamic aspects of a system. Z decomposes specifications into manageably sized modules, called schemas: that are divided into 3 parts: (i) a state, (ii) a collection of state variables and their values, (iii) operations that can change its state.

Resource provisioning policies deal with the resources and the consumers. In cost, time, security and reliability based resource provisioning policy, the resources are provisioned to those consumers whose requirements such as *cost*, *deadline*, *security* and *reliability* are fulfilled by the resource provisioning manager. The set of all resources and consumers are introduced as *basic types* of the specification:

$$[RESOURCE, CONSUMER].$$

The first aspect of the system to describe is its *state space*:

ResourceProvisioningPolicy

$$list : \mathbb{P}\, RESOURCE$$
$$provision : RESOURCE \nrightarrow CONSUMER$$

$$list = rajniprovision$$

In this schema, list and provision are variables. A relationship between the values of the variables has also been established.

- *list* is the set of resources recorded for allocation;
- *provision* is a function which when applied to certain resources, gives the user associated with them who fulfills all the conditions of the policy.

For example:

$$list = \{ \text{Resource1}, \text{Resource2}, \text{Resource3} \}$$
$$provision = \{ \text{Resource1} \mapsto \text{consumer 2}, \text{Resource2} \mapsto \text{consumer 3}, \text{Resource3} \mapsto \text{consumer1} \}.$$

In the schema Add Resource for Provisioning, the resources which are not in the list, are added in Grid environment, for provisioning to users' applications. Resource manager selects the resource providers who have not submitted the resources to the Grid environment earlier.

AddResourceforProvisioning

$\Delta ResourceProvisioningPolicy$
$consume? : CONSUMER$
$resource? : RESOURCE$

$resource? \notin list$

$provision' = provision \cup \{resource? \mapsto consumer?\}$

$list' = list \cup \{resource?\}.$

In Find Resource for Provisioning schema, resource manager finds the resources for provisioning the execution of users' applications.

FindResourceforProvisioning

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$consumer! : CONSUMER$

$resource? \in list$

$consumer! = provision(resource?)$

$list' = list$

$provision' = provision$

In this section, test cases for resource provisioning have been considered which are as follows:
Case 1: Initial state of the resource provisioning policy

InitResourceProvisioningPolicy

ResourceProvisioningPolicy

$list =$

This schema describes resource provisioning policy in initial state when the set *list* is empty: in consequence, the function *provision* is empty too.

Case 2: Resource provisioning operation is successful

$REPORT ::= ok \mid already\_in\ list \mid not\_in\ list.$

Success

$result! : REPORT$

$result! = ok$

This case shows a correct implementation of QoS based resource provisioning policy. It faithfully records resources, provisions them and gives the list of resource consumers associated with them which fulfill conditions of CRPP, TRPP, SRPP and RRPP policies. Following cases result in errors:

- Resource provider tries to add a resource already in the list of the Grid environment.

- Resource manager tries to find the resource for provisioning which is not in the list.

Case 2.1: Add same resource for provisioning in the list

AlreadyList

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$result! : REPORT$

$resource? \in list$
$result! = already\_list$

In this case, when the resource providers try to add the resources which are already in the list of resources, the schema shows this result.

Case 2.2: Find that resource which is not in the list

NotinList

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$result! : REPORT$

$resource? \notin list$
$result! = not\_list$

Now, the case in which the resource manager tries to find that resource which is not in the list of resource allocation is shown. Detail of resources has been shown in Table 5.1. Array of resources and consumers are:

$$resources : \quad ARRAY[1\,..\,]RESOURCE;$$
$$consumers : \quad ARRAY[1\,..\,]CONSUMER;$$

These 'infinite' arrays are taken for the sake of simplicity of Grid environment. In a real Grid environment, the schema is used to calculate and specify a limit on the number of entries of resources.

$\mathbb{N}_1$ is strictly positive integer to *Resource* or *Consumer*:

$$resources : \mathbb{N}_1 \rightarrow RESOURCE$$
$$consumer : \mathbb{N}_1 \rightarrow CONSUMER.$$

The element $resource[i]$ of the array is simply the value $resources(i)$ of the function, and the assignment $resources[i] := v$ is exactly described by the specification

$$resources' = resources \oplus \{i \mapsto v\}.$$

The right-hand side of this equation is a function which takes the same value as *resources* everywhere except at the argument i, where it takes the value v.

*ResourceProvisioningPolicy*1

$$resource : \mathbb{N}_1 \rightarrow RESOURCE$$
$$consumer : \mathbb{N}_1 \rightarrow CONSUMER$$
$$hwm : \mathbb{N}$$

$$\forall i, j : 1 \,..\, hwm \bullet$$
$$\quad i \neq j \Rightarrow resources(i) \neq resources(j)$$

The state space of the program of resource provisioning as a schema is described. There is another variable *hwm* (for 'high water mark') which shows the number of arrays in use. The idea of this representation is that each resource is linked with the consumers in the corresponding element of the array *consumers*. This can be documented with a schema *Abs* that defines the *abstraction relation* between the abstract state space *ResourceProvisioningPolicy* and the concrete state space *ResourceProvisioningPolicy*1:

Abs

$$ResourceProvisioningPolicy$$
$$ResourceProvisioningPolicy1$$

$$list = \{\, i : 1 \,..\, hwm \bullet resources(i) \,\}$$

$$\forall i : 1 \,..\, hwm \bullet$$
$$\quad provision(resources(i)) = consumers(i)$$

Here, Add Resource for Provisioning1 as concrete state is defined.

*AddResourceForProvisioning*1

$\Delta ResourceProvisioningPolicy1$
$resource? : RESOURCE$
$consumer? : CONSUMER$

$\forall i : 1 \,..\, hwm \bullet resources? \neq resources(i)$

$hwm' = hwm + 1$
$resources' = resources \oplus \{hwm' \mapsto resource?\}$
$consumers' = consumers \oplus \{hwm' \mapsto consumer?\}$

This schema describes an operation which has the same inputs and outputs as *AddResourceforprovisioning*, but operates on the concrete instead of the abstract state. It is a correct implementation of *AddResourcefor provisioning*, because of the fact:

- Whenever *AddResourceforprovisioning* is legal in some abstract state, the implementation *AddResourcefor provisioning*1 is legal in any corresponding concrete state.

Precondition: Add Resource Provisioning in List
The operation *AddResourceforProvisioning* is legal if its precondition $resource? \notin list$ is exactly satisfied. If this is so, the predicate

$$list = \{ i : 1 \,..\, hwm \bullet resources(i) \}$$

from *Abs* tells us that *resource*? is not one of the element $resources(i)$:

$$\forall i : 1 \,..\, hwm \bullet resource? \neq resources(i).$$

This is the pre-condition of *AddResourceforProvisioning*1.

The second operation, *FindResourceforProvisioning*, is implemented by the following operation, again described in terms of the concrete state:

*FindResourceforProvisioning*1

Ξ*ResourceforProvisioning*1

resource? : *RESOURCE*

consumer! : *CONSUMER*

$\exists\, i : 1 \,..\, hwm \bullet$

$\quad resource? = resources(i) \land consumer! = consumers(i)$

The predicate indicates that there is an index i at which the *resources* array contains the input *resource*?, and the output *consumer*! is the corresponding element of the array *consumers*.

Precondition: Find Resource for Provisioning

For this schema to be possible, *resource*? must in fact appear somewhere in the array *resources*: this is the pre-condition of the operation. The pre-conditions of the abstract and concrete operations are in fact the same: i.e the input *resource*? is known. The output is correct because for some i, $resource? = resources(i)$ and $consumer! = consumers(i)$, so

$$
\begin{aligned}
& consumer! = consumers(i) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{[spec. of } \mathit{FindResourceforProvisioning}1\text{]} \\
& \quad = provision(resources(i)) \qquad\qquad\qquad\qquad\qquad \text{[from } \mathit{Abs}\text{]} \\
& \quad = provision(resource?). \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{[spec. of } \mathit{FindResourceforProvisioning}1\text{]}
\end{aligned}
$$

The existential quantifier $\exists$ i: in the description of *FindResourceforProvisioning*1 leads to a loop in the program code, searching for a suitable value of i:

$$
\begin{aligned}
& \mathit{FindResourceforProvisioning}(resource : RESOURCE; \\
& consumer : CONSUMER); \\
& \quad i : INTEGER; \\
& \quad i := 1; \\
& \quad resources[i] \neq resourcei := i + 1; \\
& \quad consumer := consumers[i] \\
& ;
\end{aligned}
$$

The idea of this representation is that the resource providers give the facility of resource provisioning to the user for optimum results, better services and avoid the violations of the service level guarantees. The implementation of this policy enables the users to analyze customer requirements and define processes that contribute to the achievement of a product or service that is acceptable to their resource consumers.

5.2 Simulation Model: GridSim Toolkit

Due to the inherent heterogeneity of Grid computing environment, it is difficult to evaluate the performance of the proposed schema in a controlled manner. The simulation model enables us to perform repeatable experiments and the cost incurred by performing experiments on real infrastructure would be prohibitively expensive. In addition, Grid testbed is limited to a few resources and domains. Thus for experimental results, GridSim toolkit [205] has been used. GridSim toolkit provides facilities for modeling and simulation of resources and network connectivity with different capabilities, configurations and domain. It also supports primitives for application composition, information services for resource discovery and interfaces for assigning application tasks to resources and managing their execution. The following are the reasons for the GridSim toolkit to be used for evaluation [205].

- It allows modeling of heterogeneous resources.
- Resources capability can be defined in the form of Millions instructions Per Second (MIPS) as per Standard Performance Evaluation Corporation (SPEC) benchmark.
- There is no limit on the number of application jobs that can be simulated.
- Multiple user entities can submit tasks for execution simultaneously.
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.
- It supports simulation of both static and dynamic schedulers.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.

For experimental results, heterogeneous resources have been considered. In general, each resource may contain a different number of machines, and each

machine may have one or more than one processing element with different MIPS. In results, it has been assumed that each application/task which is submitted to the Grid may require varying processing time and input size and such type of tasks are defined in the form of Gridlets. A Gridlet is a package that contains all information related to the job and its execution management details such as job length, I/O operations, the size of input/output files, etc. The processing requirement of Gridlets is measured in Multiple Instructions (MI).

5.2.1 Performance Metrics

In this section, a performance evaluation criteria to evaluate the performance of a resource provisioning and scheduling framework has been defined. Two matrices, namely submission burst and cost for evaluating the performance of QoS parameter(s) based resource provisioning policy have been selected. The former indicates the total time obtained since the start of Gridlet's waiting in the queue till the resources are provisioned where as the latter indicates, the cost per unit resources that are consumed by the users for execution of their applications. The submission burst and cost are measured in seconds and Grid dollars (G$) (in-built units) respectively.

5.2.2 Experimental Results

To validate the QoS based resource provisioning approach, 250 jobs and 75 resources have been considered. An average of thirty to fifty runs have been considered in order to guarantee statistical correctness. In addition, a comparison of submission burst with QoS enabled resource provisioning vs QoS disabled resource provisioning has been presented. In order to evaluate the performance of the proposed approach, the effects of different values for the parameters of QoS based and non-QoS based resource provisioning have been investigated. In all the experiments, a comparison of QoS based resource provisioning with non-QoS based resource provisioning has been done.

Test Case 1: In the first test case, the submission burst time and cost of Grid applications have been evaluated in two scenarios as (i) the same number of applications/jobs are sent and (ii) different number of applications are sent. The pricing of resources may or may not be related to CPU speed. Thus, minimization of both submission burst time and cost of an application may conflict with each other depending on the price of the resources. Figure 5.1, 5.2 show the submission

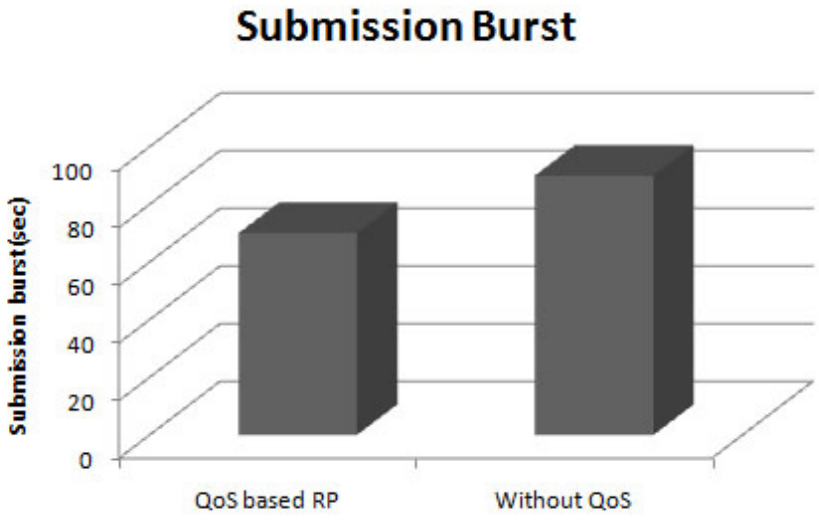

Figure 5.1: Comparison of the submission burst of QoS based resource provisioning QoS vs without QoS

burst and cost of QoS based resource provisioning vs non-QoS based resource provisioning respectively. The results show that in case of non-QoS based resource provisioning, if the same number of applications/jobs are sent to the Grid, submission burst and cost increases while in the other case, they decreases. This is expected as the QoS based resource provisioning approach is keeping track of the state of all resources at each point of the time which enables it to take an optimal decision than non-QoS based resource provisioning approach.

Experiment to determine the effect of increasing the number of applications on the cost and submission burst has also been performed. From the experimental results shown in Figures 5.3, 5.4, it can be concluded that the time taken to submit an application reduces by using QoS based resource provisioning approach. Figure 5.4 shows that cost per application increases as the number of submitted applications increases. Non-QoS based resource provisioning resulted in a schedule which is expensive in comparison to QoS based resource provisioning approach as the number of applications increases. The reason is that non-QoS based resource provisioning approach does not consider the effect of other applications in the meta scheduler at the time of job submission but in QoS based resource provisioning approach, RPM considers the effect of applications in meta scheduler before execution of application according to both user and resource providers's perspectives. In resource provisioning and scheduling framework, RPM considers the cost, time and other parameters of other applications before job submission

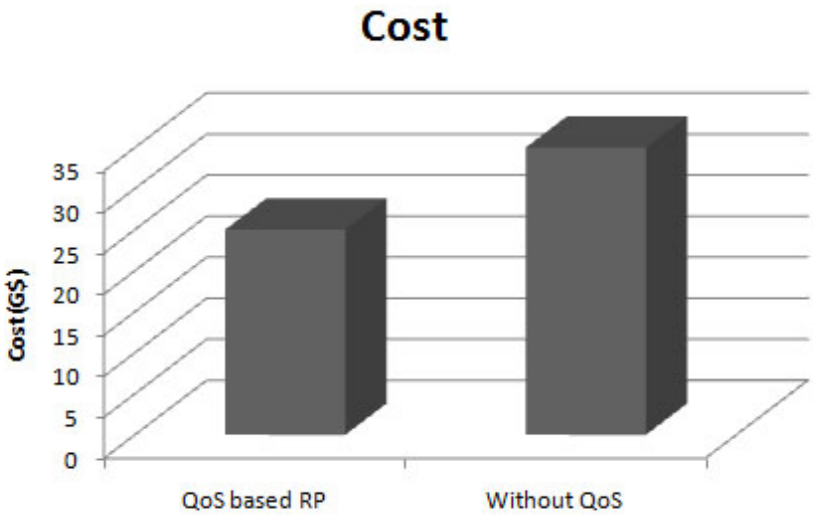

Figure 5.2: Comparison of the cost of QoS based resource provisioning vs without QoS

to meta-scheduler.

Test Case 2: The second test case evaluates the submission burst time and cost for job submission based on QoS based provisioned approach and best-effort approach. Best-effort approach does not support QoS and same priorities for all users and also there is no assurance about resource provisioning. Figures 5.5, 5.6 show the submission burst and cost of the application's execution using the best-effort and QoS based provision approach at different resource utilization levels. At 50% resource utilization level, the best effort submission burst id is 10 to 15% higher than QoS based provision approach as shown in Fig 5.5. This time variation with resource utilization is quite significant. The best-effort submission burst time sharply increases as the resource utilization reaches at 80% but the provisioned approach based submission burst time is less. The cost of the best-effort and provisioned approach differs less significantly in comparison to submission time at the resource utilization levels. This is because when the number of applications is large, they are scheduled on left out resources which may not be very cost effective. The cost of resources may or may not vary with resource's configuration.

From the above results, it has been observed that application's execution using the proposed QoS based resource provisioning approach provides the following advantages: The provisioned based submission burst time is 60% lower than the best effort approach. The time variation in execution of applications is about 5-10 %, compared to non QoS based resource provisioning of 50- 60 % using the same

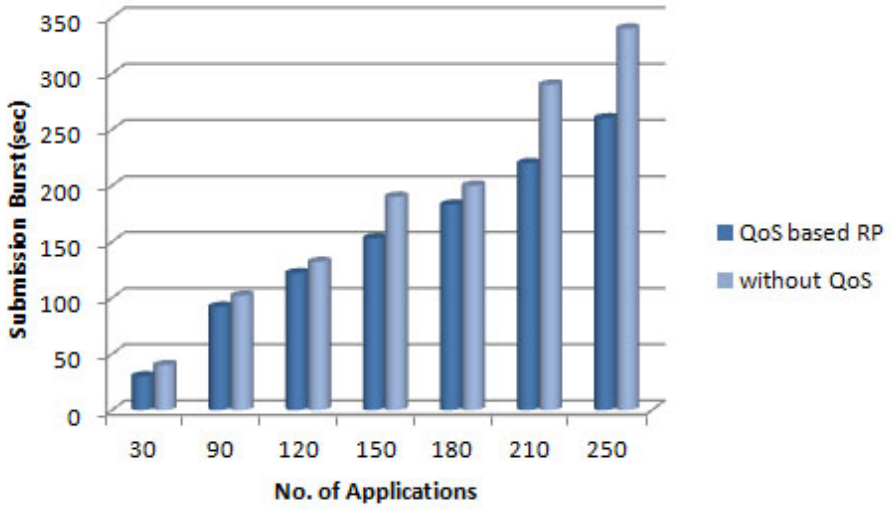

Figure 5.3: Effect of change in number of application submitted on submission burst

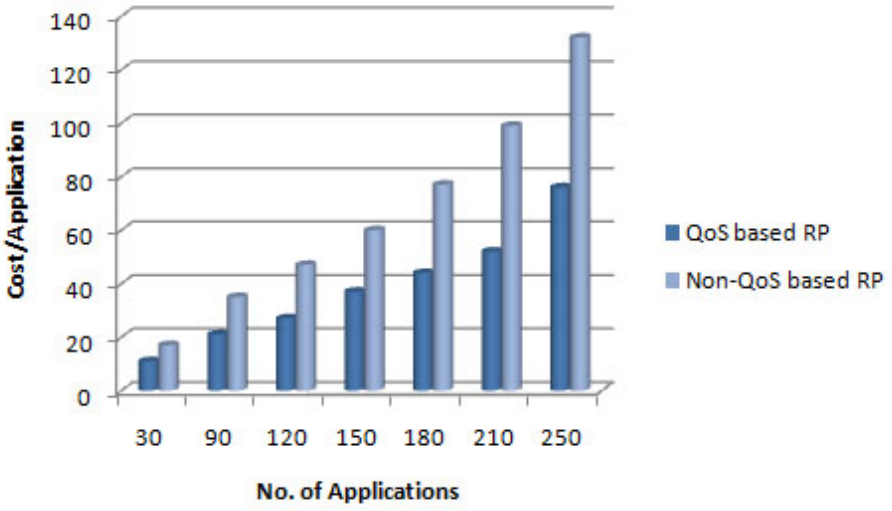

Figure 5.4: Effect of change in number of application submitted on cost

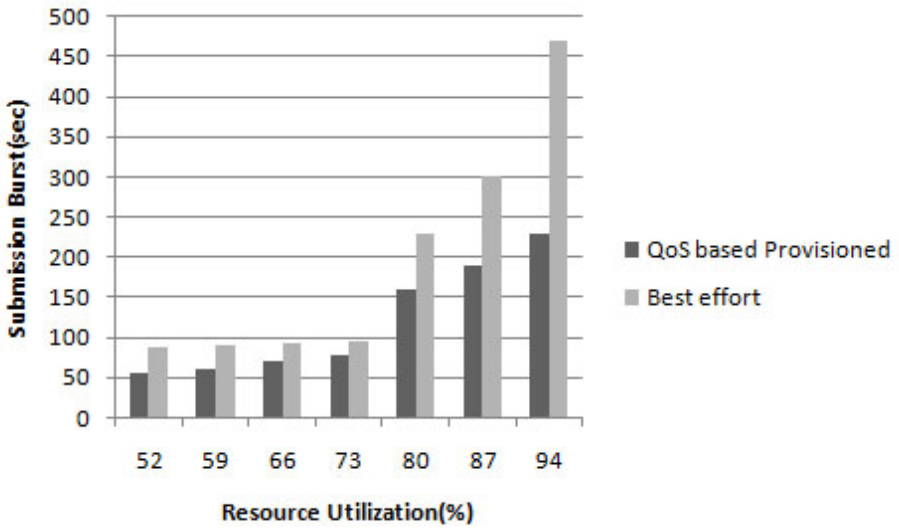

Figure 5.5: Submission burst of best effort and QoS based provisioned approach with resource utilization

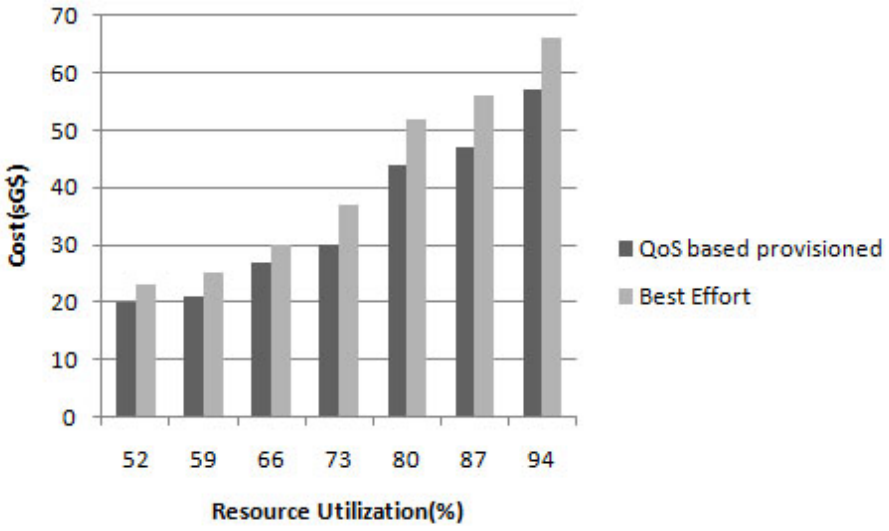

Figure 5.6: Cost of best effort and QoS based provisioned approach with resource utilization

set of applications. This time variation is quite significant. It also maintains the resource utilization and cost for user's application execution.

5.3 Experimental Scenario

The evaluation of the strategies has been performed using GridSim. Table 5.1 shows the characteristics of resources and Gridlets, which have been used for all the experiments. Table 5.2 below provides the proposed scheduling algorithm parameters names and their values taken in the experiments. User applications have been modeled as independent parallel applications which are computation intensive. Thus the data dependency among the tasks in the parallel applications is negligible. Each task is parallel and is hence considered to be independent of any other subtask. For evaluation, a suitable workload from real machine traces have been derived. These traces have been obtained from Grid workload archive website [1]. 5000 user applications are generated according to the Lublin workload model [206]. The model specifies the arrival time, number of CPUs required, and execution time μ of each application. This model is derived from existing workload traces for rigid jobs and incorporates correlations between job runtimes, job sizes, and daytime cycles in job-interarrival times. Using this generated workload, an ETC matrix has been generated which is computed as the ratio of workload and computing capacity of machine vectors. No of jobs * no of resources gives the size of the matrix and its components are defined as $ETC(j_i, r_k)$. Rows of the ETC matrix demonstrate the estimated execution time for a job on each resource and the columns demonstrate the estimated execution time for a particular resource. $ETC(j_i, r_k)$ is the expected execution time of job j_i and the resource r_k. Each job can execute on each resource, and the estimated execution times of each job on each resource is known. ETC matrices are classified into consistent and inconsistent matrices. Consistent matrix means that whenever a resource r_k executes the job j_i faster in comparison to r_l then the resource r_k executes all the jobs faster than r_l. Inconsistent matrix means that r_q may be faster in job execution than r_s for some cases and slower for others [197]. Resource heterogeneity represents the variation that is possible among the execution times for a given job across all the resources. The variation of the application's execution time on different resources can be high or low. A high variation in execution time of the same application is generated using the gamma distribution method. In the gamma distribution method, a mean

[1] More information about the real trace used can be obtained from the Grid Workload Archive at http://gwa.ewi.tudelft.nl/pmwiki/

Table 5.1: Scheduling parameters and their values

Parameter	Value
Number of Resource	150-250
Number of Gridlets	5000
Length of Job	1000 - 6000
Bandwidth	3000 or 7000 B/S
Number of machine per resource	1
Number of PEs per machine	1-5
PE ratings	10-60 MIPS
Cost per job	3 G$-5G$
File size	100 + (10-30%) MB
Job Output size	250 + (10-40%)MB

task execution time and Coefficient of Variation (CV) are used to generate ETC matrices [197]. The mean task execution time of an application is set to μ and a CV value of 0.9 is used. Similarly, the low variation in the execution time is generated using uniform distribution with mean value of μ and a CV value of 0.3. The prices of resources are generated using weibull distribution with parameters α=0.3 and β=0.7.

Table 5.2: BFO parameters and its values

Parameter	Value
Population size	100
Chemotaxis steps	20
Swim length	4
Reproduction steps	4
Ci (step size)	0.05
Elimination and dispersal with probability	.25

The main flow of scheduling in GridSim toolkit can be briefly described as follows. When a scheduling event is started, an instance of the scheduling problem is created by the simulator which is based on the current jobs and available resource pools as shown in the Figure 5.7. The instance contains: (a) workload (b) computing capacity of machines; (c) prior load of machines and (d) the ETC

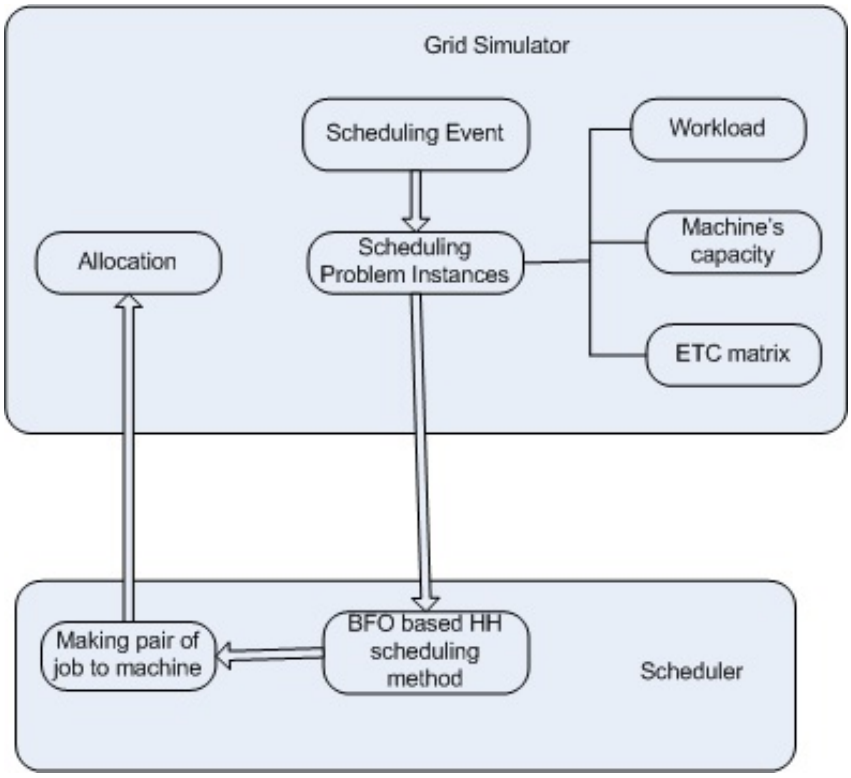

Figure 5.7: Flowchart of scheduling in Simulator

matrix. The defined instance is then passed on to the scheduler which computes the planning of jobs to resources.

5.3.1 Performance Evaluation Criteria

In this section, the performance evaluation criteria has been defined to evaluate the performance of bacterial foraging based hyper-heuristic for resource scheduling. Two matrices, namely makespan and cost have been selected and calculated using the equations 4.2 and 4.3 (described in Chapter 3) for evaluating the performance. The former indicates the total execution time where as the latter indicates the cost per unit resources that are consumed by the users for the execution of their applications. The makespan and cost are measured in seconds and Grid dollars (G$) respectively.

5.4 Analysis of Results

To validate the proposed algorithm, 5000 jobs/applications and 150 - 250 resources have been considered. An average of fifty runs has been used in order to guarantee

statistical correctness. The simulation results have been presented using Ali et al. [197] simulation model with the GridSim discrete event simulation so as to test the performance of hyper-heuristic based algorithm. In addition, a comparison of makespan and cost of the proposed algorithm with existing heuristic algorithms such as GA, SA, GA-TS has been presented. In order to evaluate the performance of the proposed approach, the effects of different number of applications have been investigated . In all the experiments, a comparison for consistent and inconsistent matrix has been done.

5.4.1 Test case 1: Performance for the low Heterogeneous case

In this case, makespan and cost of the Grid applications for low resource/machine heterogeneity with inconsistent and consistent matrices have been evaluated. The pricing of resources may or may not be related to CPU speed. Thus, minimization of both makespan and cost of an application may conflict with each other depending on the price of the resources. The most important characteristic applicable to real-world scenarios is that how each algorithm responds to different heterogeneity of jobs and resources. A comparison of both cost and makespan for low resource/machine heterogeneity has been shown for consistent and inconsistent matrices.

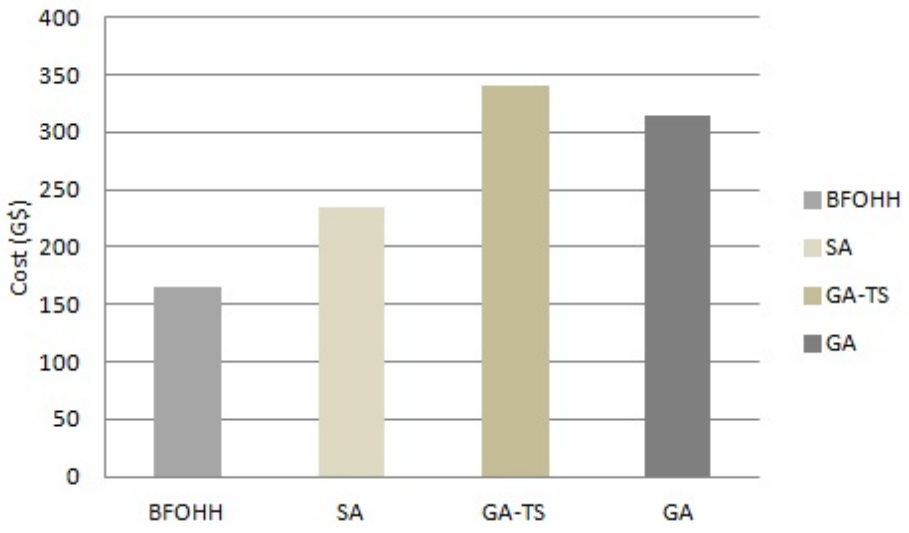

Figure 5.8: Cost comparison result for inconsistent and low machine heterogeneity

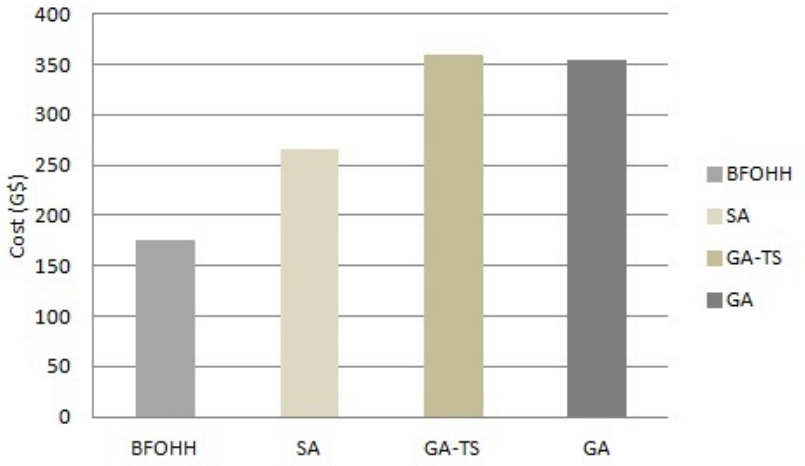

Figure 5.9: Cost comparison result for consistent and low machine heterogeneity

The low resource heterogeneity is simulated by resources having the number of PEs between 1 and 5. Figures 5.8 and 5.9 show the effect on cost by the four heuristics in case of low resource heterogeneity with inconsistent and consistent matrices. Figure 5.8 shows that the lowest cost was achieved in case of Bacterial Foraging based Hyper-heuristic (BFOHH) resource scheduling algorithm where as GA-TS resulted in the highest cost. A similar trend can be seen in case of low resource heterogeneity with consistent matrix. When having low resource heterogeneity, BFOHH outperforms all the other approaches for both consistent and inconsistent matrices. The other three heuristics have higher cost in comparison to BFOHH for application execution.

Figures 5.10 and 5.11 show the makespan comparison for inconsistent and consistent matrices with low machine heterogeneity. It can be seen from the figures that the makespan is lowest in case of BFOHH for both the consistent and inconsistent matrices. This is because of the low variation in execution time across various resources as the resource list that is obtained from the resource provisioning unit is already filtered. This also demonstrates the effectiveness of the BFOHH in managing the time requirement of the user. The results show that in case of GA, SA and GA-TS algorithms, if the same number of applications/jobs are sent to the Grid, makespan and cost increases whereas in the case of bacterial foraging based hyper-heuristic resource scheduling algorithm both makespan and cost decreases.

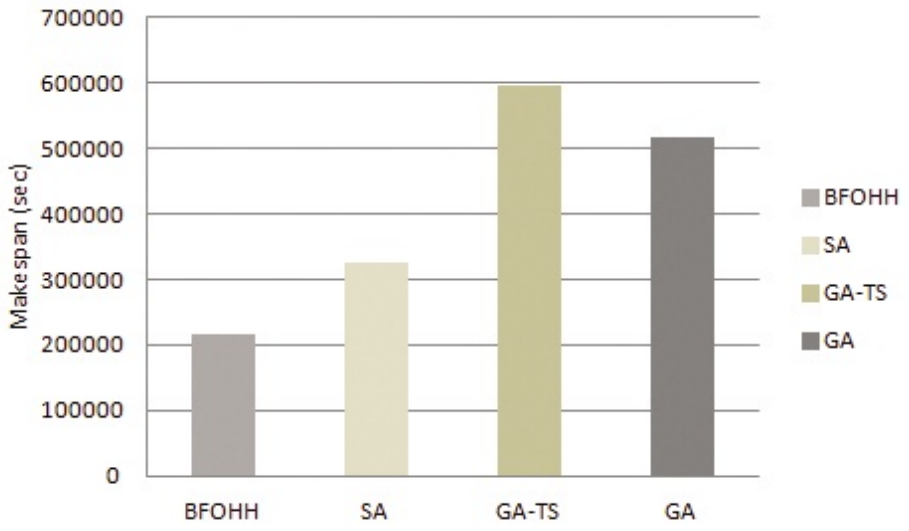

Figure 5.10: Makespan comparison result for inconsistent and low machine heterogeneity

5.4.2 Test case 2: Performance for the high heterogeneous case

In this case, makespan and cost of the Grid applications for high resource/machine heterogeneity case with inconsistent and consistent matrices have been evaluated. In this case, high resource heterogeneity is simulated by each resource having a random number of PEs between 7 and 30.

Figures 5.12 and 5.13 show the makespan comparison for inconsistent and consistent cases with high machine heterogeneity. As can be seen from Figures 5.12 and 5.13, the performance of BFO based hyper heuristic is far better than that of existing scheduling algorithms i.e GA, SA and GA-TS for the case of inconsistent and high machine heterogeneity. Figures 5.14 and 5.15 show the cost comparison for inconsistent and consistent ETC matrix with high machine heterogeneity. The cost reduction in GA was the least in comparison to other algorithms. It can be concluded from the figures that when the same number of applications are sent, the cost and makespan are the least for the proposed algorithm. This is because more applications are scheduled on cheaper and efficient resources, due to the ability of BFOHH to find optimal resources. By analyzing the results in the Figures 5.8 - 5.15, it can be concluded that BFO based hyper-heuristic outperforms all the other approaches in the cases with both low or high machine heterogeneity.

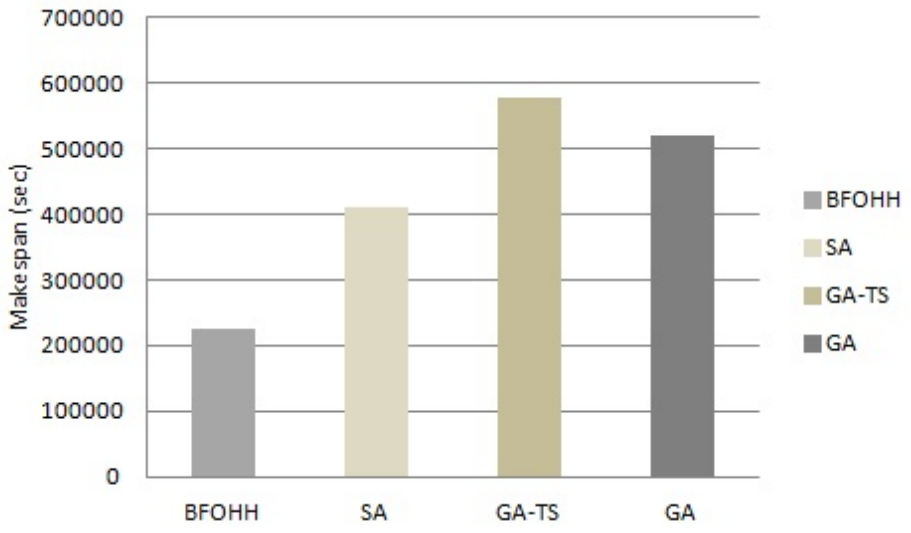

Figure 5.11: Makespan comparison result for inconsistent and low machine heterogeneity

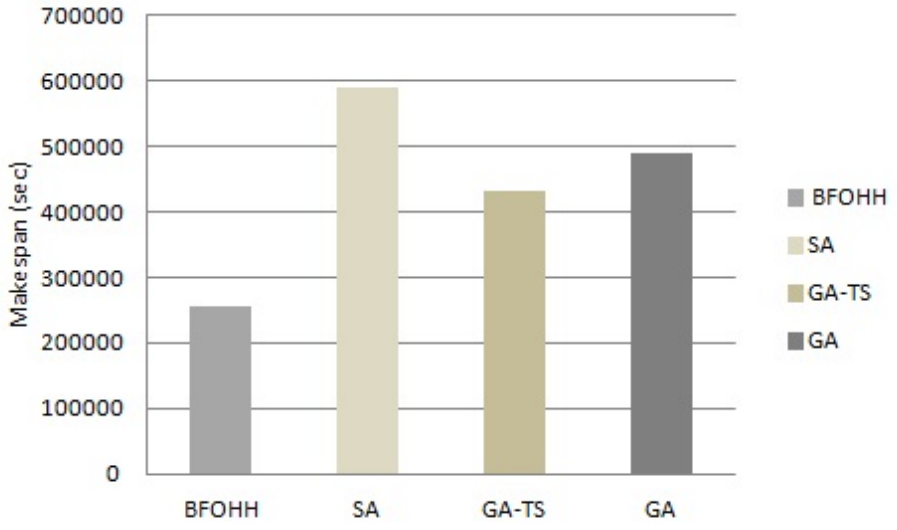

Figure 5.12: Makespan comparison result for inconsistent and high machine heterogeneity

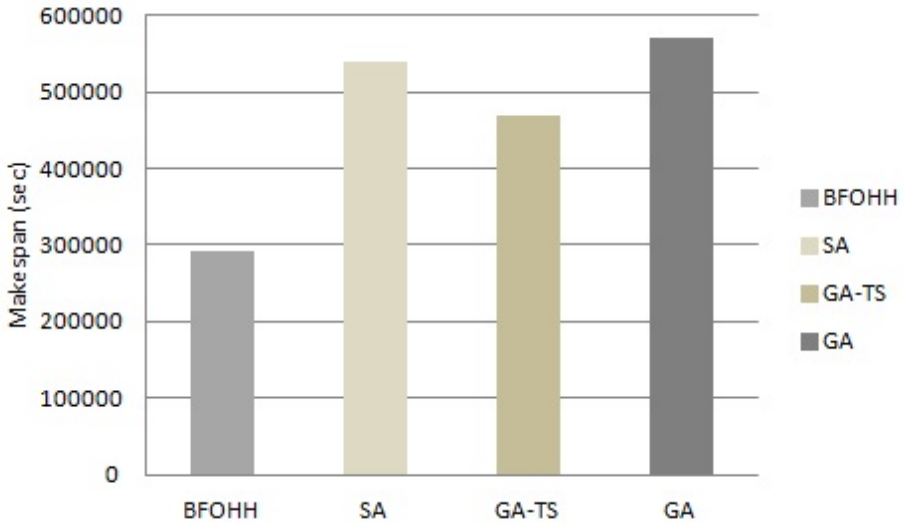

Figure 5.13: Makespan comparison result for consistent and high machine heterogeneity

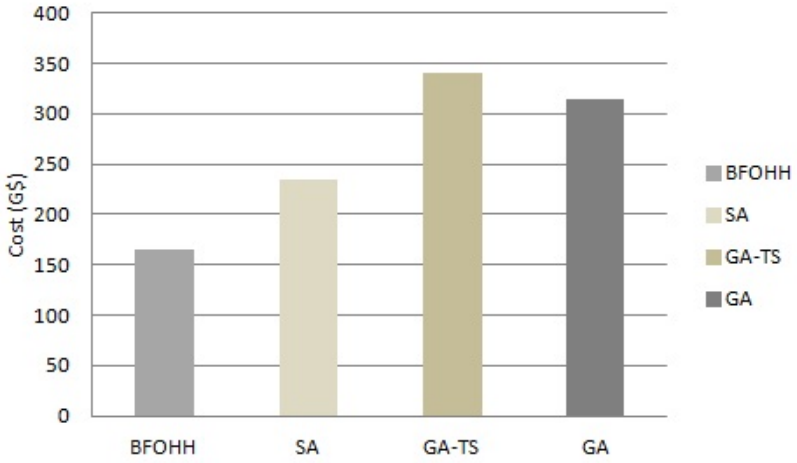

Figure 5.14: Cost comparison result for inconsistent and high machine heterogeneity

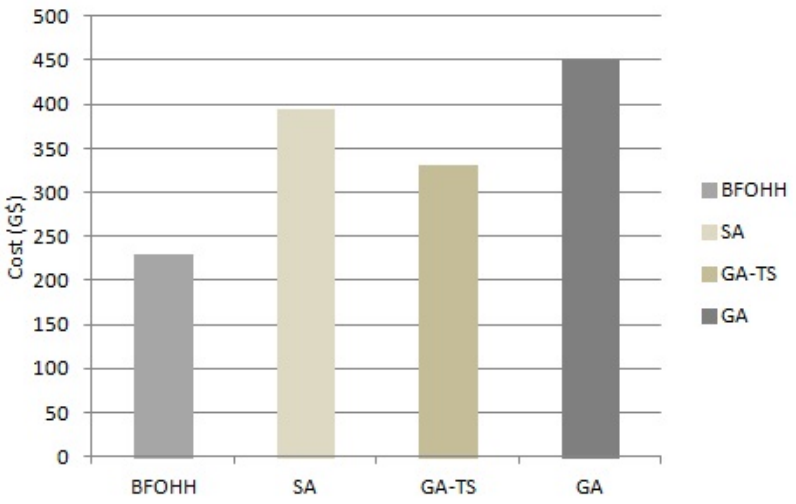

Figure 5.15: Cost comparison result for consistent and high machine heterogeneity

5.4.3 Test case 3: Effect of the number of resources and its capacity

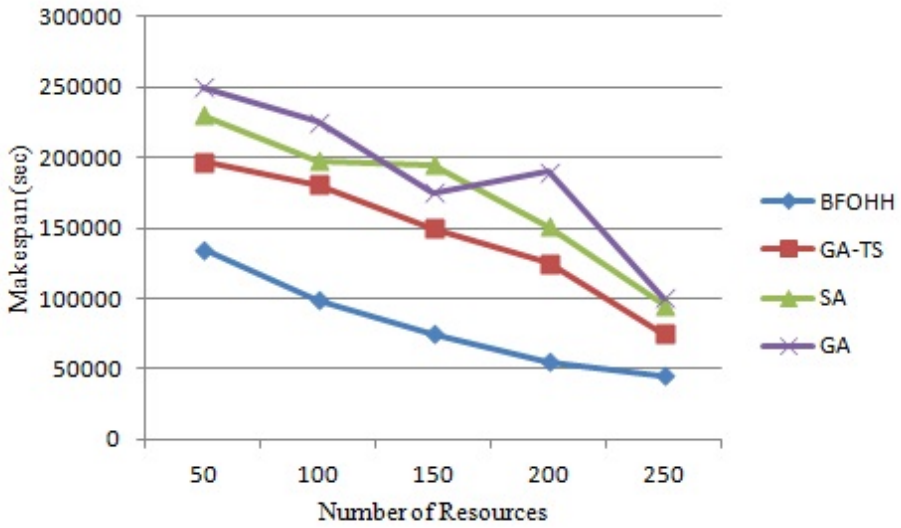

Figure 5.16: Effect of the number of resource on the makespan

Figure 5.16 shows the effect of increasing the number of resources, while keeping the number of jobs being sent to the Grid constant. In this experiment, 1000 jobs were sent to the Grid with varying number of resources. The results depict that by increasing the number of resources, the execution time increases thus decreasing the performance of the Grid. BFOHH and GA-TS performs better

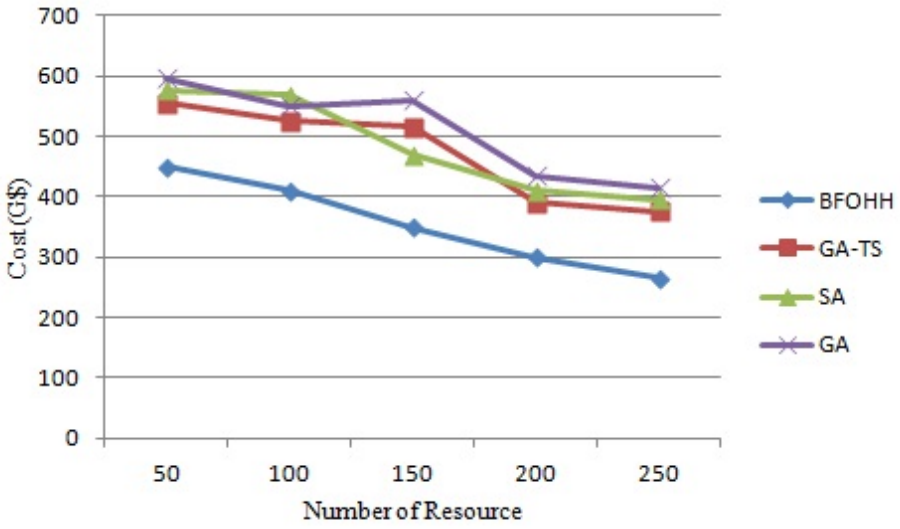

Figure 5.17: Effect of the number of resource on the cost

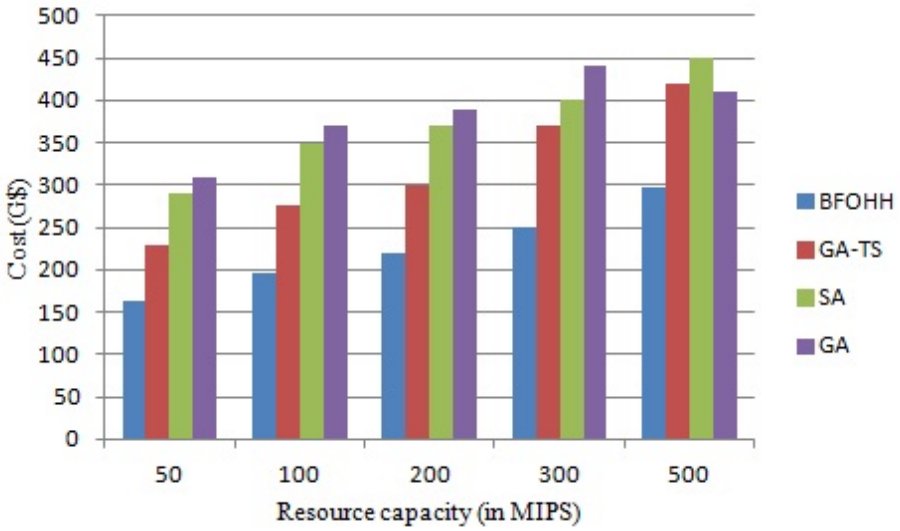

Figure 5.18: Effect of the resource capacity on the cost

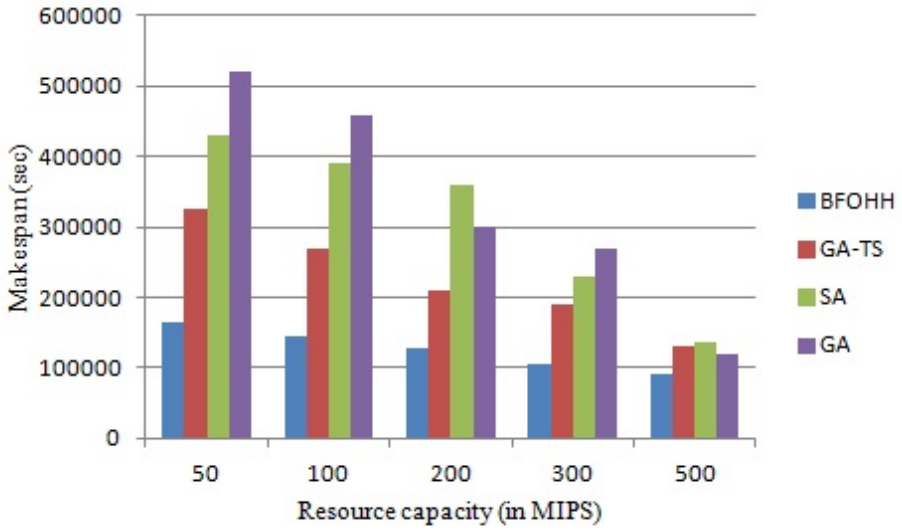

Figure 5.19: Effect of the resource capacity on the makespan

when the number of resources is less in comparison to the number of jobs. As shown in Figure 5.16, the makespan time decreases for all the algorithms in the same proportion on an increase in the number of resources. This observation indicates that our algorithm gives an equally good performance in comparison to that given by other algorithms.

The cost of application execution using BFOHH is much less in comparison to the execution cost using existing scheduling algorithms. As the cost variations within the Grid resources are not significant (i.e. 4G$ with 0.5G$) so the cost benefits of only 5%-7% were noticed. However, more benefits can be anticipated if the variations are higher. Figures 5.18 and 5.19 show cost and makespan of the user application by varying the capacity of the resources respectively. In this case, the cost of the resource in terms of its speed has been considered. An increase in the capacity of the resource will also increase its cost but with a reduction in the execution time. Figure 5.19 depicts that an increase in the resource's capacity will decrease the makespan thus increasing the overall performance of our algorithm in comparison to GA, SA and GA-TS. Thus BFOHH outperforms all the existing scheduling algorithms when the application execution cost is high, which is an important case for a large-scale Grid computing environment.

5.4.4 Test case 4: Effect of the number of jobs

Experiments have also been performed to determine the effect of an increase in the number of applications on cost and makespan. A hundred-node simulated

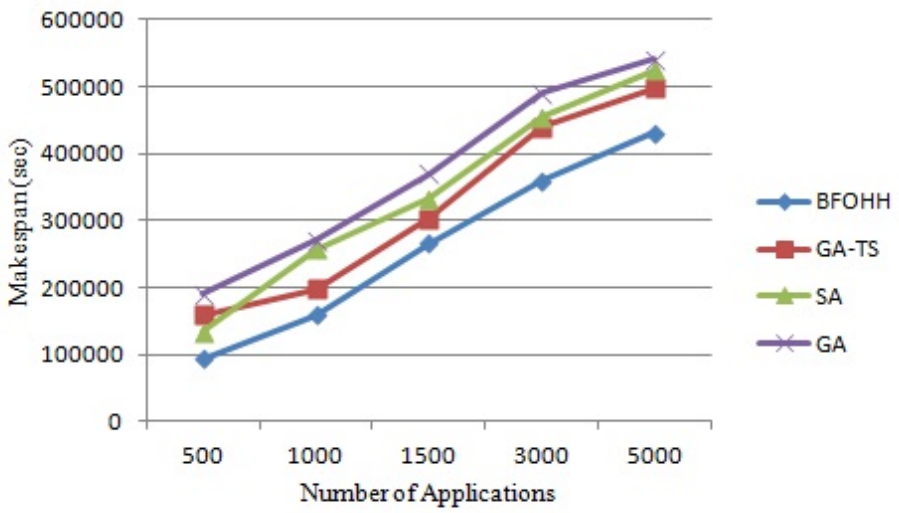

Figure 5.20: Effect of the number of applications on the makespan

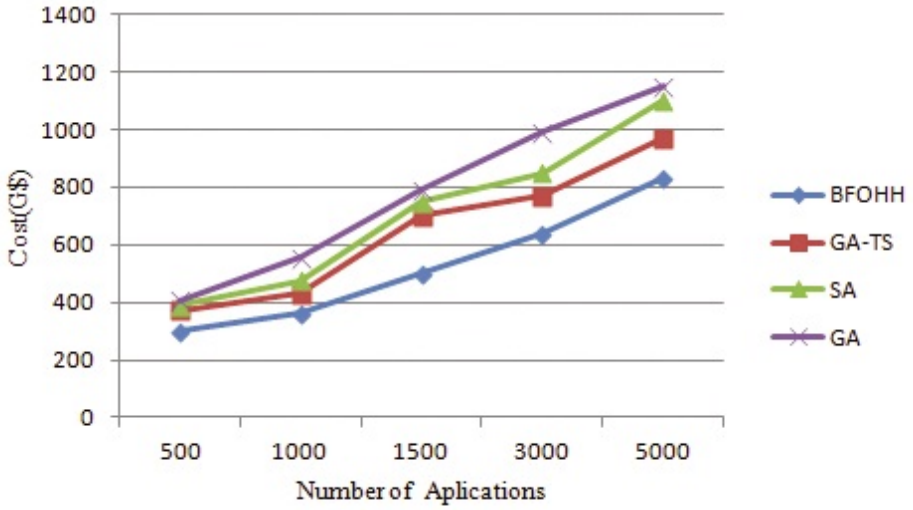

Figure 5.21: Effect of the number of applications on the cost

Grid with two thousand jobs being sent to the Grid has been considered. From the experimental results shown in Figure 5.20, it can be concluded that the time taken to execute an application reduces by using BFO based hyper-heuristic algorithm. Figure 5.21 shows that cost per application increases as the number of submitted applications increases. The existing algorithm based application's execution resulted in a schedule which is expensive in comparison to BFO based hyper-heuristic algorithm. From all the experimental results, it was observed that application's execution using BFO based hyper-heuristic algorithm provides the following advantages: The makespan is much lower in comparison to the GA, SA and GA-TS. The time variation in applications execution is about 5-11 %, which is much less in comparison to the 50-70 % variation in the existing algorithms using the same set of applications. The overall cost for user's application execution is less.

5.4.5 Statistical Analysis of Results

In this section, statistical method, namely the Coefficient of Variation (CV), has been employed to analyze the statistical significance of the results. The coefficient of variation is defined as the statistical measure of the dispersion of data around the average value. For comparison between datasets with different units or widely different means, coefficient of variation is used instead of the standard deviation. It expresses the variation of the data as a percentage of its mean value, and is calculated as follows:

$$CV = (standard\ deviation/mean) * 100 \tag{5.1}$$

The CV statistic is very useful in the analysis of the data series. It can also provide a general analysis of performance of the method used for generating the data. In Figures 5.22 and 5.23, the CV of the makepsan and cost of application's execution of each scheduling algorithm has been thoroughly examined.

It can be observed that in all instances, the values of CVs are in the range 0% – 2% which confirms the stability of the proposed algorithm. As the coefficient of variation is small, it means that BFO based hyper-heuristic resource scheduling algorithm is more effective in scheduling of independent parallel jobs in the cases where the number of application changes. As the number of applications increases, the CV value decreases. It shows that our proposed algorithm outperforms other existing algorithms for large number of applications. As, the system

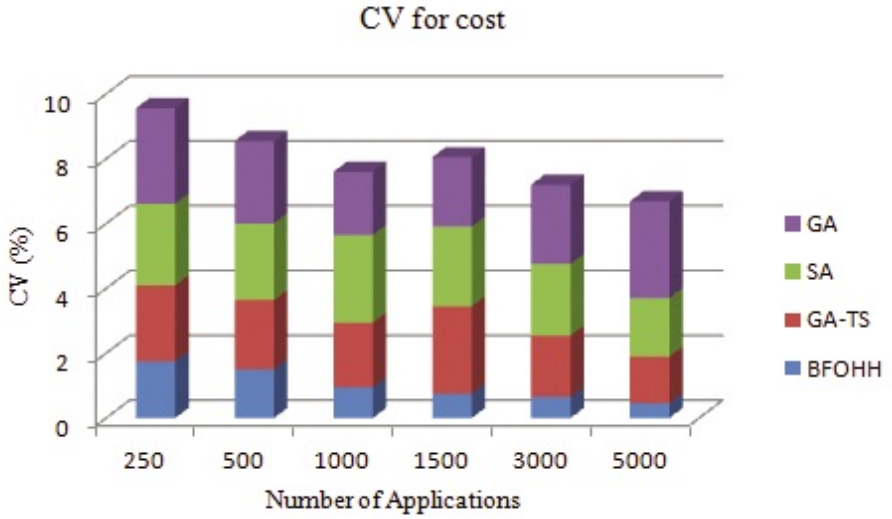

Figure 5.22: Coefficient of variation for the cost with each algorithm

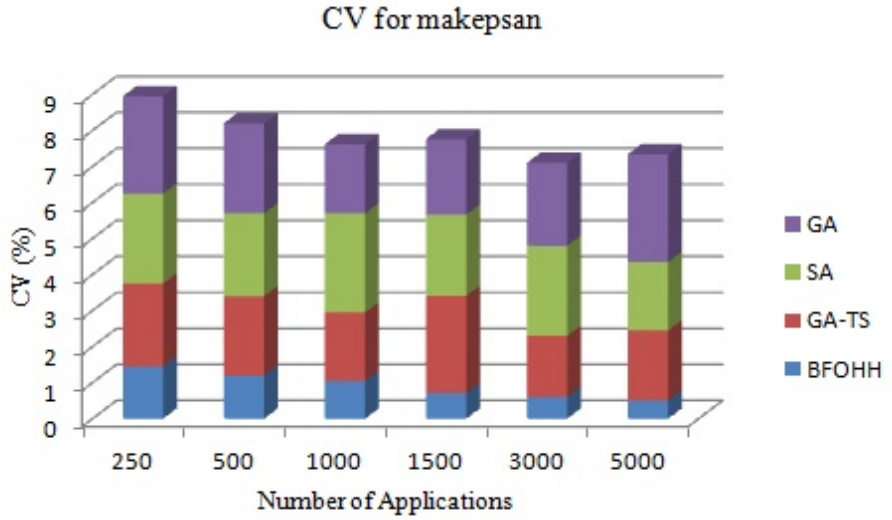

Figure 5.23: Coefficient of variation for the makespan with each algorithm

with small coefficient of variation value is more balanced, so the proposed algorithm achieved the best results in the Grid with regard to both makespan and cost as QoS parameters.

5.5 Framework Validation

In the proposed framework, resource scheduling has been done on the basis of the QoS parameter(s) based resource provisioning policy which was not considered traditionally. Figures (in experimental results section) show that QoS based provisioned approach generates better results and resource provisioning based scheduling algorithm is able to schedule the job on the ingredient resources more efficiently. Resource provisioning and scheduling framework has been validated against the features of the existing resource provisioning and scheduling frameworks and its result has been depicted in Table 5.3.

Table 5.3: Comparison of Resource Provisioning and Scheduling Framework with existing frameworks

Framework /Features	**RPSF**	**Dragon**	**Glare**	**GARA**	**QoS-GRAF**	**RAA/Falkon**	**Intra Grid**	**Ali's Framework**
Usage	Resource provisioning and Scheduling Framework provides the facility of provisioning of the resources.	It discussed about the Nw architecture which can provide dynamically provisioning.	It provides the facility of service provisioning.	It supports flow-specific QoS specification, immediate and advance reservation and online monitoring and control of both individual resources and heterogeneous resource ensembles.	It optimized business matrices based on SLA driven pricing policies.	It is the combination of DRP & falkon.	It enable the deployment of applications across multiple Grids.	It worked before resource provisioning to discover resources.
Resource Provisioning based Scheduling	Yes	NO	No	No	No	No	No	No
Policies	CRPP, SRPP, TRPP, RRPP	No Policies	No Policies	No Policies	No Policies	Allocation & Deallocation policies	Conservative Backfilling & Multiple site partition policies	Reservation Policy
QoS Parameters	Submission burst, Cost, Time, Security and Reliability	No parameters	Security, Reliability	Reservation	Revenue	Time	Response Time	Advance Reservation
Validation	Formal Methods, Z formal specification Language	No	No	No	No formal method verification	No	No	No

5.6 Conclusion

In this chapter verification details, experimental setup and testing results of the proposed work have been demonstrated. Formal specifications of resource provisioning policies have ensured that the customer gets resources as per the desire and the scheduling process is more clear. The experimental results have been illustrated for resource provisioning and resource scheduling. The performance of the BFO based hyper-heuristic algorithm has been compared with well-known scheduling algorithms such as GA, SA and GA-TS. The performance of the proposed algorithm with variation in both the number of jobs and the number of resources have been analyzed, which are expected to vary in the real Grid environment. The algorithm's performance has been evaluated with respect to both makespan and cost. Makespan allows the evaluation of the algorithm which results in better scheduling in the sense of the duration of job execution, while the cost allows the comparison for resource selection. The proposed algorithm helps to achieve high performance and simultaneously it also helps to satisfy the user's requirements. The testing results demonstrate that the proposed solutions are working and can be effectively used to address resource provisioning and scheduling challenges to establish an efficient Grid.

The next chapter concludes the thesis and also discusses the future directions.

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 issues related to it have been identified and addressed. This thesis focuses upon various issues including resource provisioning and resource scheduling in Grid environments. A resource provisioning and scheduling framework and QoS parameters based resource provisioning policies have been proposed, which addresse these issues and a Resource scheduling algorithm has also been designed and implemented. Further, the framework has been designed, developed and validated in this thesis.

This chapter concludes the thesis by highlighting the main contributions of this research work. This chapter commences with explaining outcome of each chapter and then discusses contributions of the Resource Provisioning and Resource Scheduling for Grids. Later, it focusses upon the future scope of the work.

6.1 Conclusions

The thesis, "QoS based Resource Provisioning and Scheduling in Grids" addresses resource provisioning and resource scheduling challenges in Grid computing.

Literature Review as presented in Chapter 2 highlights the resource provisioning and scheduling in Grid computing infrastructure. After an exhaustive review, it becomes apparent that there are no existing Grid resource provisioning and scheduling frameworks for provisioning of the resources using a standard policy. Also, an important resource management challenge i.e., Resource provisioning based scheduling has not been addressed in a Grid environment using all the necessary QoS parameters. Based on these observations, a resource provisioning and scheduling framework, QoS based resource provisioning policies have been proposed and designed in chapter 3. Resource provisioning and scheduling framework with requirements and mode of operation of the QoS based resource provisioning has been discussed thoroughly. This framework offers resource provisioning policies that caters to provisioned resource allocation and resource scheduling. The policies rules have been specified in XML schema.

The challenge was to schedule various applications in a coordinated manner by satisfying both the users and providers by minimizing the cost and time and maximizing the security and reliability of resources for all Grid users. This scheduling problem, was found to be NP-complete due to its combinatorial nature. The problem becomes more challenging when users may prefer cheaper services with a relaxed QoS, if it is sufficient to meet their requirements. To address these issues of scheduling in a Grid, BFO based hyper-heuristic resource scheduling algorithm has been proposed in Chapter 4. A combinatorial optimization model to produce near-optimal schedules for the independent parallel scheduling problem has been described in this chapter. A novel bacterial foraging based hyper-heuristic resource scheduling algorithm has been designed to effectively schedule the jobs on available resources in a Grid environment. The proposed algorithm is used in the scheduling of independent parallel jobs in the Grid environment so as to simultaneously minimize the cost and the makespan. Optimization technique for optimizing the cost and makespan for resource scheduling simultaneously through a fitness function has been discussed in detail in this chapter.

Chapter 5 gives the verification details of Grid Resource Provisioning and Scheduling framework and QoS parameter based Resource Provisioning Policies. The framework has also been validated by comparing it with existing frameworks. The experimental results have been shown and the solution has been presented

as a novel approach to address Resource Provisioning and Scheduling challenges. The proposed algorithm was evaluated by an extensive simulation study and it gave the lowest makespan and cost in almost all scenarios considered. Statistical analysis of the results has also been done in this chapter. The contributions of this thesis are as follows:

- A detailed literature review of the work done in the area of Grid Resource Management and addresses challenges such as Resource Provisioning and Scheduling.
- Resource provisioning and scheduling framework provides the facility of both the resource provisioning and resource scheduling to execute the user's application optimally.
- Efficient resource utilization can be done by the proposed solution of resource provisioning.
- The scheduling algorithm proposed in this thesis have taken both the user's and resource provider's constraints such as cost, makespan.
- The Thesis presents the development and implementation of the proposed Grid Resource Provisioning and Scheduling framework and QoS parameter(s) based Resource Provisioning Policies and Resource Scheduling Algorithm.
- It also demonstrates the applicability of the proposed framework, policies and algorithm in a Grid environment.

6.2 Future Directions

In this thesis, the problem of resource provisioning and scheduling with QoS constraints in Grid computing was addressed. It also introduces resource provisioning and scheduling framework, QoS parameter(s) based resource provisioning policies and resource provisioning policy based scheduling algorithm to build scalable, robust and an efficient Grid resource management system. It demonstrated the benefits of the proposed framework, policies and algorithm in terms of user's QoS satisfaction and provider's profit function. However, the work can be further enhanced and extended in future. Some of the future directions are:

- The framework can be further enhanced by introducing dynamic load balancing techniques at the time of scheduling.
- The existing solutions have been designed and tested only for Resource provisioning and scheduling challenges. In future, this can be extended for addressing the other resource management challenges such as Re-scheduling and resource monitoring, etc.
- Scheduling based on energy efficiency is an important aspect for the development of Grid computing and environmental sustainability.
- It will be interesting to enhance the proposed algorithm to schedule jobs with different application models such as workflows and bag-of-tasks.
- In addition, the proposed framework and resource scheduling algorithm in this thesis can also be enhanced by considering more QoS needs such as memory and network bandwidth.
- Today, there are a lot of real-time applications which require huge amount of data processing and data storage requirements, etc. So, this research work can also be focused on developing the programming and scheduling frameworks for data intensive applications from commerce, science and medicine that require large processing power as well as data analysis / processing on large distributed databases.
- Cloud computing has emerged as a computing paradigm for sharing computation of storage resources. It also faces QoS based resource provisioning and scheduling challenges that need to be addressed. The proposed solutions can be extended to be used in Cloud computing so as to enable usage and adoption of the same.

References

[1] U. Schwiegelshohn, R. M. Badia, M. Bubak, M. Danelutto, S. Dustdar, F. Gagliardi, A. Geiger, L. Hluchy, D. Kranzlmller, E. Laure, T. Priol, A. Reinefeld, M. Resch, A. Reuter, O. Rienhoff, T. Rter, P. Sloot, D. Talia, K. Ullmann, and R. Yahyapour, "Perspectives on Grid Computing," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1104–1115, 2010.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[3] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid An Open Grid Services Architecture for Distributed Systems Integration," *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[4] Y. Han and C. H. Youn, "A New Grid Resource Management Mechanism with Resource Aware Policy Administrator for SLA- Constrained Aapplications," *Future Generation Computer Systems*, vol. 25, pp. 768–778, April 2009.

[5] T. Stevens, M. D. Leennheer, C. Develder, B. Dhoedt, K. Christodulopoulus, P. Kokkinos, and E. Varvarigos, "Multi-Cost Job Routing and Scheduling in Grid networks," *Future Generation Computer Systems*, vol. 25, no. 8, pp. 912–925, 2009.

[6] S. Kounev, R. Nou, and J. Torres, "Autonomic QoS aware Resource Management in Grid computing using Online Performance Models," in *Proceedings of the 2nd International conference on Performance evaluation methodologies and tools*, 2007.

[7] T. Tachibana, K. Kgiso, and K. Sugimoto, "Dynamic Management of Computing and Network Resources with PID Control in Optical Grid Networks,"

in *Proceedings of the 2008 IEEE International Conference on Communications, ICC'08,*, pp. 396–400, 2008.

[8] E. Varvarigo, V. Sourlas, and K. Christodoulopoulos, "Routing and Scheduling Connections in Networks that Support Advance Reservations," *Computer Networks*, vol. 52, pp. 2988–3006, 2008.

[9] M. Baker, R. Buyya, and D. Laforenza, "The Grid : International Efforts in Global Computing," in *Proceedings of the 2000 International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet (SSGRR 2000)*, (l'Aquila, Rome, Ital), July 2000.

[10] F. Xhafa and A. Abraham, "Computational Models and Heuristics methods for Grid Scheduling Problems," *Future Generation Computer Systems*, vol. 26, pp. 608–621, 2010.

[11] I. Foster, "What is the Grid? A Three Point CheckList, booktitle = GRID Today, year = 2002, address = Online at http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf,,"

[12] "Large Hadron Collider Computing Grid," September 2009. http://lcg.web.cern.ch/LCG/.

[13] GridPhyN. http://www.usatlas.bnl.gov/computing/grid/griphyn/.

[14] C. Catlett, "The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility," in *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, (Berlin, Germany), 2002.

[15] C. Catlett, P. Beckman, D. Skow, and I. Foster, "Creating and Operating National-scale Cyberinfrastructure Services," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 2, pp. 2–10, 2006.

[16] "Particle Physics Data Grid." http://www.http://ppdg.net/.

[17] "Enabling Grids for E-sciencE. (EGEE) project," September 2009. http://www.eu-egee.org/.

[18] T. Hey and A. E. Trefethen, "The UK E-science Core Programme and the Grid," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1017–1031, 2002.

[19] "German D-Grid." http://www.d-grid.de/.

[20] "BiG Grid- the Dutch e-science Grid." http://www.biggrid.nl/.

[21] R. Bolze, F. Cappello, E. Caron, M. D. e, F. Desprez, E. Jeannot, Y. J. . egou, S. L. eri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E. Talbi, and T. I. ea, "Grid5000: A Large scale and Highly Reconfigurable Experimental Grid Testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[22] P. T. Bul$\tilde{h}$oes, C. Byun, R. Castrapel, and O. Hassaine, "Sun N1 Grid Engine 6 Features and Capabilities," 2004.

[23] "IBMGrid." http://www-935.ibm.com/industries/energy/.

[24] OracleGrid. http://docs.oracle.com/cd/E1188201/install.112/e24321/oraclerestart.htm.

[25] "HPGrid." http://www.thehpgrid.com/.

[26] F. Dong and S. G. AKI, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems," technical report 504, School of Computing, Queens University, Kingston, Ontario, Canada, 2006.

[27] I. Foster and C. Kesselman, *The Grid: Blueprint for a Future Computing Infrastructure*. USA: Morgan Kaufmann, 2nd ed., 2003.

[28] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, I. F. F. W"urthwein, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The Open Science Grid," *Journal of Physics: Conference Series*, vol. 78, 2007.

[29] J. O'Callaghan, "A National Grid infrastructure for Australian researchers," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 1, 2006.

[30] H. Park, P. Lee, J. R. Lee, S. Kim, J. Kwak, K. W. Cho, S. B. Lim, and J. Lee, "Construction and Uutilization of the Cyberinfrastructure in Korea," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 1, 2006.

[31] K. Miura, "Overview of Japanese Science Grid Project NAREGI," *Progress in Informatics*, vol. 3, pp. 67–75, 2006.

[32] N. M. Ram and S. Ramakrishnan, "GARUDA: India's National Grid Computing Initiative," *Cyberinfrastructure Technology Watch Quarterly*, vol. 2, no. 1, 2006.

[33] N. Andrade, L. Costa, G. Germoglio, and W. Cirne, "Peer-to-Peer Grid Computing with the OurGrid Community," in *Proceeding of the 23rd Brazilian Symposium on Computer Networks, IV Special Tools Session*, (Brazilian Computer Society), 2005.

[34] N. Jacq, J. Salzemann, F. Jacq, Y. Legré, E. Medernach, J. Montagnat, A. Maa, M. Reichstadt, H. Schwichtenberg, M. Sridhar, V. Kasam, M. Zimmermann, M. Hofmann, and V. Breton, "Grid-enabled Virtual Screening against Malaria," *Journal of Grid Computing*, vol. 6, no. 1, pp. 29–43, 2008.

[35] AuverGrid. http://www.auverGrid.fr.

[36] DAS, 2006. The Distributed ASCI Supercomputer 2 (DAS-2).

[37] M. D. D. Assuncao, *Provisioning Techniques and Policies for Resource Sharing between Grids*. PhD thesis, University of Melbourne, MARCH 2009.

[38] M. L. Bote-lorenzo, Y. A. Dimitriadis, and E. Gmez-snchez, "Grid Characteristics and Uses: A Grid Definition," in *Proceedings of the 2003 International Conferecen on Across Grids, LNCS*, pp. 291–298, 2003.

[39] F. Magoules, J. Pan, K. A. Tan, and A. Kumar, *Introduction to Grid Computing*. CRC Press, Taylor & Francis Group, 2009.

[40] E. Christensen, F. Curbera, G. Meredith, and S. Weerarawana, "Web Service Description Language." W3Cnote15, 2001.

[41] "Universal Description Discovery and Integration." http://www.uddi.org.

[42] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich, "GFD-I.030 Open Grid Services Architecture," (Available online at:http://forge.gridforum.org/projects/ogsa-wg), 2003.

[43] K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, and S. Tuecke, "GWD-R draft-GGF-OGSI-Grid Service-33," (http://www.ggf.org/ogsi-wg), june 2003.

[44] T. Banks, "OASIS: Web Services Resource Framework," tech. rep., 2007.

[45] "OGSA: What is OGSA-DAI?," 2007.

[46] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of the 2008 IEEE International workshop on Grid Computing Environments*, pp. 1–10, 2008.

[47] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Y. Girard, R. Strachowski, and S. S. Yu, *Enabling Applications for Grid Computing with Globus*. An IBM Red books, June 2003.

[48] Seti@home. http://setiathome.ssl.berkeley.edu/.

[49] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing," *ACM Computing Surveys*, vol. 38, pp. 1–53, March 2006.

[50] E. Pacitti and P. Valduriez, "Grid Data Management: Open Problems and News Issues," *Journal of Grid Computing, Springer*, vol. 5, no. 3, pp. 273–281, 2007.

[51] A. Hameurlain, F. Morvan, and M. E. Samad, "Large Scale Data Management in Grid Systems: A Survey, Invited Paper," in *Proceedings of the 3rd IEEE International Conference onInformation and Communication Technologies: From Theory to Applications, ICTTA 2008.*, pp. 1–6, 7-11 April 2008.

[52] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Plateforms: Vision, Hype and Reality for Delivering Computing as 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[53] M. Kiran, A. H. A. Hashim, L. M. Kuan, and Y. Y. Jiun, "Execution Time Prediction of Imperative Paradigm Tasks for Grid Scheduling Optimization," *International Journal of Computer Science and Network Security*, vol. 9, no. 2, pp. 155–163, 2009.

[54] M. D. D. Assuncao and R. Buyya, "Performance Analysis of Allocation Policies for InterGrid Resource Provisioning," *Information and Sotware Technology*, vol. 51, pp. 42–55, 2009.

[55] M. Li and M. Baker, *The Grid: Core Technologies*. John Wiley & Sons ltd, 1 ed., 2005.

[56] E. Cody, R. Sharman, R. H. Rao, and S. Upadhyaya, "Security in Grid Computing: A Review and Synthesis," *Journal of Decision Support Systems*, vol. 44, pp. 749–764, 2008.

[57] P. J. Broadfoot and A. P. Martin, "A Critical Survey of Grid Security Requirements and Technologies," Tech. Rep. PRG-RR-03-15, Programming research group, Oxford university computing laboratory, 2003.

[58] F. Siebenlistand, V. Welch, S. Tuecke, I. foster, N. Nagaratnam, P. Janson, J. Dayka, and A. Nadalin, "OGSA Security Roadmap," 2002. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.4881&rep=rep1&type=pdf.

[59] I. Chana, *A Framework for Resource Management in Grid Environment*. PhD thesis, Thapar University, 2009.

[60] R. J. Al-Ali, K. Amin, G. Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. Zaluzec, "Analysis and Provision of QoS for Distributed Grid Applications," *Journal of Grid Computing*, vol. 2, no. 2, pp. 163–182, 2004.

[61] D. Nurmi, R.Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L.Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud Computing System," in *Proceeding of the 9th IEEE/ACM International Symposium on Cluster Computing and Grid, CCGRID'09*, 2009.

[62] F. Magoules, T. M. H. Nguyen, and L. Yu, *Grid Resource Management: Toward Virtual and Services Compliant Grid Computing*, ch. Scheduling Grid Services, pp. 161–193. (Chapman & Hall/CRC Numerical Analysis and Scientific Computing Series), 2008.

[63] S. Ludwig and A. Moallem, "Swarm Intelligence Approaches for Grid Load Balancing," *Journal of Grid Computing*, vol. 9, no. 3, pp. 279–301, 2011.

[64] J. Pathak, J. Treadwell, R. Kumar, P. Vitale, and F. Fraticelli, "A Framework for Dynamic Resource Management on the Grid," Tech. Rep. HPL-2005-153, HP Laboratories Palo Alto., 2005.

[65] A. A. Khateeb, R. Abdullah, and N. A. Rashid, "Job Type Approach for Deciding Job Scheduling in Grid Computing Systems," *Journal of Computer Science*, vol. 5, no. 10, pp. 745–750, 2009.

[66] R. Shah, B. Veeravalli, and M. Misra, "On the Design of Adaptive and Decentralized Load Balancing Algorithms with Load Estimation for Computational Grid Environments," *IEEE Transactions on Parallel an ddistributed Systems*, vol. 18, no. 12, pp. 1675–1686, 2007.

[67] J. M. Ramrez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J. Gonzlez-Garca, and A. Hirales-Carbaja, "Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids," *Journal of Grid Computing*, vol. 9, no. 1, p. 95116, 2011.

[68] E. Deelman, "Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments," *International Journal High Performance Computing Applications*, vol. 24, pp. 286–298, 2010.

[69] A. C. A. Dusseau, *Implicit Co-Scheduling: Coordinated Scheduling with Implicit Information in Distributed Systems.* PhD thesis, University of California, Berkeley, 1998.

[70] G. Singh, C. Kesselman, and E. Deelman, "A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids," (Monterey, California, USA), pp. 117–126, Proceedings of the 16th international symposium on High performance distributed computing, June 25-29 2007.

[71] G. Singh, C. Kesselman, and E. Deelman, "Application-Level Resource Provisioning on the Grid," in *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, 2006.

[72] G. Juve and E. Deelman, "Resource Provisioning Options for Large-scale Scientific Workflows," in *Proceeding of the IEEE Fourth International Conference on eScience*, pp. 608–613, December 2008.

[73] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat, "Model-based Resource Provisioning in a Web Service Utility," in *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems(USITS)*, March 2003.

[74] X. Yang, T. Lehman, C. Tracy, J. Sobieski, S. Gong, P. Torab, and B. Jabbari, "Policy-Based Resource Management and Service Provisioning in GMPLS Networks," in *Proceedings of the 25th IEEE International Conference on Computer Communications.*, pp. 1–12, 2006.

[75] T. Lehman, J. Sobieski, and B. Jabbari, "DRAGON: A Technique for Service Provisioning in Heterogeneous Grid Networks," *IEEE Communications Magazine*, vol. 44, pp. 84–90, March 2006.

[76] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer, "GLARE: A Grid Activity Registration, Deployment and Provisioning Framework.," in *Proceedings of the 2005 ACM/IEEE International Conference on Supercomputing*, November 12 - 18, 2005.

[77] M. A. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic Provisioning of Virtual Organization Clusters," in *Proceedings of the 9th IEEE International Symposium on Cluster Computing and the GridCCGrid09*, (Shanghai, China), pp. 364–371, 2009.

[78] X. Yu and C. Qiao, "Online Job Provisioning for Large Scale Science Experiments over an Optical Grid Infrastructure," in *Proceedings of the 2009 Internationa workshop on INFOCOM*, pp. 1–6, 19-25, April 2009.

[79] E. K. Byun, J. W. Jang, W. Jung, and J. Kim, "A Dynamic Grid Services Deployment Mechanism for On-Demand Resource Provisioning," in *Proceedings of teh 2005 IEEE International Symposium on Cluster Computing and the Grid*, pp. 863–870, 2005.

[80] R. Nou, F. Juliá, J. Guitart, and J. Torres, "Dynamic Resource Provisioning for Self-adaptive Heterogenous Workload in SMP Hosting Platforms," in *Proceedings of the 2007 International Conference on E-Business*, (Barcelona,Spain), July 2007.

[81] C. Vazquez, E. Huedo, R. S. Montero, and I. Llorente, "Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers," in *Proceedings of 2009 International Conference of Grid and Pervasive Computing*, pp. 113–119, 2009.

[82] Y. Li, F. Rao, Y. Chen, D. Liu, and T. Li, "Services Ecosystem : Towards a Resilient Infrastructure for On Demand Services Provisioning in Grid," in *Proceedings of 2004 International Conferenece on Web Services*, 2004.

[83] A. Keller, K. Voss, D. Battre, M. Hovestadt, and O. Kao, "Quality Assurance of Grid Service Provisioning by Risk Aware Managing of Resource Failures," in *Proceedings of the 3rd International Conference Risks and Security of Internet and Systems: CRiSIS2008*, pp. 149–157, 2008.

[84] A. Filali, A. Hafid, and M. Gendreau, "Adaptive Resources Provisioning for Grid Applications and Services," in *Proceedings of the IEEE International Conference on Communications, ICC'08*, (China), pp. 186 – 191, 2008.

[85] A. Filali, A. Hafid, and M. Gendreau, "Bandwidth and Computing Resources Provisioning for Grid Applications and Services," in *Proceedings of the IEEE Internatioanl Conference on Communications, ICC'09*, pp. 1–9, 2009.

[86] Y. Kee and C. Kesselman, "Grid Resource Abstraction, Virtualization, and Provisioning for Time-targeted Applications," in *Proceedings of the 2008 ACM/IEEE International Symposiumon Cluster Computing and the Grid(CCGRID 08)*, 2008.

[87] I. Foster, M. Fidler, A. Royd, V. Sander, and L. Winkler, "End-to-End Quality of Service for High-End Applications," *Elsevier Computer Communications Journal*, vol. 27, no. 14, pp. 1375–1388, 2004.

[88] G. Dasgupta, K. Dasgupta, A. Purohit, and B. Viswanathan, "QoS-GRAF: A Framework for QoS based Grid Resource Allocation with Failure Provisioning," in *Proceedings of the 14th IEEE International Workshopon QoS (IWQOS06)*, (NewHeaven, CT, USA), pp. 281–283, June19- 21 2006.

[89] A. Iosup, P. Garbacki, and D. H. Epema, "Provisioning and Scheduling Resources for World-Wide Data-Sharing Services," in *Proceedings of the 2nd IEEE International Conference e-science and Grid computing*, 2006.

[90] A. Brocco and B. Hirsbrunner, "Service Provisioning Framework for a Self-Organized Grid," in *Proceedings of the 18th International Conference Computer Communications and Networks (ICCCN 2009)*, pp. 1–6, 2009. ISSN: 1095-2055, Print ISBN: 978-1-4244-4581-3, Digital Object Identifier: 10.1109/ICCCN.

[91] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Dynamic Resource Provisioning in Grid Environments," in *Proceedings of the 2007 International Conference on Tera Grid*, june 2007.

[92] B. Rood and M. Lewis, "Grid Resource Availability Prediction-Based Scheduling and Task Replication," *Journal of Grid Computing*, vol. 7, no. 4, pp. 479–500, 2009.

[93] R. Yahyapour and P. Wieder, "Grid Scheduling Use Cases," in *Grid Scheduling Architecture Research Group GSA-RG*, no. GFD-I.064, Open Grid Forum OGF, March 26, 2006.

[94] A. Abraham, R. Buyya, and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids," in *Proceedings of the 8th IEEE Conference on Advanced Computing and Communications*, 2000.

[95] H. Liu, A. Abraham, and A. E. Hassanien, "Scheduling Jobs on Computational Grids using a Fuzzy Particle Swarm Optimization Algorithm," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1336–1343, 2010.

[96] S. Garg, P. Konugurthi, and R. Buyya, "A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids," in *Proceedings of the 16th International Conference on Advanced Computing and Communication (ADCOM 2008)*, (Chennai, India), IEEE Press, New York, USA, 2008.

[97] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and Cost Trade-off Management for Scheduling Parallel Applications on Utility Grids," *Future Generation Computer Systems*, pp. 1344–1355.

[98] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[99] Y. Gaoa, H. Rongb, and J. Z. Huangc, "Adaptive Grid Job Scheduling with Genetic Algorithms," *Future Generation Computer Systems*, vol. 21, no. 1, 2005.

[100] K. Golconda and F. Ozguner, "A Comparison of Static QoS-based Scheduling Heuristics for a Meta-Task with Multiple QoS Dimensions in Heterogeneous Computing," in *Proceedings of the 18th International Symposium on Parallel and Distributed Processing*, 2004.

[101] S. Kim and J. B. Weissman, "A Genetic Algorithm based Approach for Scheduling Decomposable Data Grid Applications," in *International Conference on Parallel Processing*, pp. 406 – 413, 2004.

[102] P. K. Konugurthi, K. Ramakrishnan, and R. Buyya, "A Heuristic Genetic Algorithm based Scheduler for Clearing House Grid Broker," Tech. Rep. Technical Report, GRIDS-TR-2007-22, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2007.

[103] D. Kondo, A. A. Chien, and H. Casanova, "Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids," *Journal of Grid Computing*, vol. 5, pp. 379–405, 2007.

[104] L. Jun, L. Chunlin, and L. Qingqing, "A Research about Independent Tasks Scheduling on Tree-Based Grid Computing Platforms," in *Proceedings of the 2nd International Workshop on Intelligent Systems and Applications*, (Institute of Computer Science, Wuhan University of Technology Wuhan, CHINA), pp. 1–4, 2010.

[105] S. S. Chauhan and R. C. Joshi, "QoS Guided Heuristic Algorithms for Grid Task Scheduling," *International Journal of Modeling and Optimization*, vol. 2, no. 3, pp. 356–359, 2012.

[106] S. Roy and N. Mukherjee, "Efficient Resource Management for Running Multiple Concurrent Jobs in a Computational Grid Environment," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1070–1082, 2011.

[107] T. S. Somasundaram, B. R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajendar, R. Rajiv, G. Kannan, G. R. Britto, E. Mahendran, and B. Madusudhanan, "CARE Resource Broker: A Framework for Scheduling and Supporting Virtual Resource Management," *Future Generation Computer Systems*, vol. 26, pp. 337–347, 2010.

[108] D. G. Cameron, A. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger, and F. Zini, "Analysis of Scheduling and Replica Optimisation Strategies for Data Grids Using OptorSim," *Journal of Grid Computing*, vol. 2, pp. 57–69, 2004.

[109] A. I. Saleh, A. M. Sarhan, and A. M. Hamed, "A New Grid Scheduler with Failure Recovery and Rescheduling Mechanisms: Discussion and Analysis," *Journal of Grid Computing*, vol. 10, no. 2, pp. 211–235, 2012.

[110] H. A. Sanjay and S. S. Vadhiyar, "Strategies for Rescheduling Tightly-Coupled Parallel Applications in Multi-Cluster Grids," *Journal of Grid Computing*, vol. 9, no. 3, pp. 379–403, 2011.

[111] F. Desprez and A. Vernois, "Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid," *Journal of Grid Computing*, vol. 4, no. 1, pp. 19–31, 2006.

[112] A. Pugliese, D. Talia, and R. Yahyapour, "Modeling and Supporting Grid Scheduling," *Journal of Grid Computing*, vol. 6, no. 2, pp. 195–213, 2008.

[113] S. Song, K. Hwang, and Y. K. Kwok, "Trusted Grid Computing with Security Binding and Trust Integration," *Journal of Grid Computing*, vol. 3, pp. 53–57, 2005.

[114] P. Cowling, G. Kendall, and L. Han, "An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem," in *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pp. 1185–1190, 2002.

[115] J. A. Gonzalez, M. Serna, and F. Xhafa, "A Hyper-heuristic for Schedulingin Independent Jobs in Computational Grids," in *Proceedings of the 2007 International conference on software and data technologies, ICSOFT*, 2007.

[116] S. M. S. Bhanu and N. P. Gopalan, "A Hyper-heuristic Approach for Efcient Resource Scheduling in Grid," *International Journal of Computers, Communications / Control*, vol. 3, no. 3, pp. 249–258, 2008.

[117] K. Z. Gkoutioudi and H. D. Karatza, "Multi-Criteria Job Scheduling in Grid Using an Accelerated Genetic Algorithm," *Journal of Grid Computing*, vol. 10, pp. 311–323, 2012. DOI 10.1007/s10723-012-9210-y.

[118] J. Kolodziej and F. Xhafa, "Integration of Task Abortion and Security Requirements in GA-based Meta-Heuristics for Independent Batch Grid Scheduling," *Computers and Mathematics with Applications, Elsevier*, vol. 63, pp. 350–364, 2012. DOI: 10.1016/j.camwa.2011.07.038.

[119] J. Kolodziej and F. Xhafa, "Meeting Security and User Behaviour Requirements in Grid Scheduling," *Simulation Modelling Practice and Theory, International Journal of the Federation of European Simulation Societies, Elsevier*, vol. 19, pp. 213–226, 2011. doi:10.1016/j.simpat.2010.06.007.

[120] D. Menasce and E. Casalicchio, "QoS in Grid Computing," *IEEE Internet Computing Journal*, vol. 8, july 2004.

[121] gLite: Lightweight Middleware for Grid Computing. http://glite.cern.ch/.

[122] J. Almond and D. Snelling, "UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce," *Future Generation Computer Systems*, vol. 15, no. 5-6, p. 539548, 1999.

[123] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[124] Legion. http://legion.virginia.edu/.

[125] "The gLite Middleware Architecture and Components." Ariel Garcia Forschungszentrum Karlsruhe, 2005.

[126] S. Burke, S. Campana, A. D. Peris, F. Donno, P. M. Lorenzo, R. Santinelli, and A. Sciab‘a, *gLite 3 User Guide Manual Series*, version 1.1 ed., 17 jan 2007.

[127] E2GRIS1. www.eu-eela.eu o Itacuru (Brazil).

[128] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M.Romberg, B. Schuller, and P. Wieder, *UNICORE- From Project Results to Production Grids*. Corr, 2005.

[129] M. Romberg, *UNICORE: Beyond Web-based Job-Submission*. Research Center Julich Central Institute for Applied Mathematics, d-52425 julich, germany ed., 2002.

[130] F. J. Gmb, R. U. Stuttgart, D. W. Offenbach, and P. Gmb, "UNICORE Plus Final Report-Uniform Interface to Computing Resources," tech. rep., Joint Project Report for the BMBF Project UNICORE Plus Grant Number: 01 IR 001 A-D, January 2000 to December 2002.

[131] M. Romberg, *The UNICORE Grid Infrastructure*. Research Center Julich Central Institute for Applied Mathematics, 2002.

[132] A. Streit, O. Waldrich, P. Wieder, and W. Ziegler, "On Scheduling in UNICORE- Extending the Web Services Agreement based Resource Management Framework," in *In Parallel Computing: Current & Future Issues of High-End Computing* (F. P. O. P. P. T. G. Joubert, W. Nagel and E. Zapata, eds.), pp. 57–64, John von Neumann Institute for Computing (NIC), 2006.

[133] R. Wanker, "Grid Computing with Globus: An Overview and Research Challenges," *International Journal of Computer Science and Applications*, vol. 5, no. 3, pp. 56–69, 2008.

[134] P. Asadzadeh, R. Buyya, C. L. Kei, D. Nayer, and S. Venugopal, "Global Grids and Softwae toolkits: A Study of Four Grid Middleware Technologies," in *Proceeding of the High Performance Computing: Paradigm and Infrastructure* (L. Yang and M. Guo(edS), eds.), (New Jersey, USA), Wiley Press, June 2005.

[135] J. S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "Resource Management in Legion," *Future Generation Computer Systems - Special issue on metacomputing archive*, vol. 15, no. 5-6, pp. 583 – 594, 1999.

[136] Aneka. http://www.manjrasoft.com/products.html.

[137] D. Karunamoorthy, "Gridbus Workflow Management System and Aneka Enterprice Middleware, A Project on the Integration of Two Technologies," tech. rep., The University of Melbourne, June 2009.

[138] M. J. Litzkow, M. Livn, and M. W. Mutka, "Condor A Hunter of Idle Workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, (San Jose, USA.), p. 104111, 1988.

[139] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC 2001), IEEE Computer Society*, (San Francisco, USA.), pp. 55–63, 2001.

[140] Globus. http://www.globus.org/service/.

[141] M. Baker, R. Buyya, and D. Laforenza, *Grids and Grid Technologies for wide-area Distributed Computing*. John Wiley & Sons, Ltd, 2002.

[142] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," in *Proceeding of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, (Beijing, China), pp. 283–289, 2000.

[143] Gridway. http://www.gridway.org/doku.php.

[144] E. Huedo, R. S. Montero, and I. M. Llorente, "A Fframework for Adaptive Execution in Grids," *Software Practice and Experience*, vol. 34, no. 7, pp. 631–651, 2004.

[145] R. Yahyapour and P. Wieder, "Grid Scheduling Use Cases," Global Grid forum, Grid Scheduling Architecture Research Group (GSA-RG), GWD-I.64, March 2006.

[146] E. Huedo, R. S. Montero, and I. M. Llorente, "An Experimental Framework for Executing Applications in Dynamic Grid Environments," ICASE Report 2002-43, NASA/CR-2002-211960, 2002.

[147] R. Henderson, "Job Scheduling Under the Portable Batch System," in *Proceedings of the 1995 Workshop on Job Scheduling Strategies for Parallel Processing*, (Santa Barbara, CA, USA), 1995.

[148] H. A. James, *Scheduling in Metacomputing Ssystems*. PhD thesis, University of Adelaide, July 1999.

[149] "OpenPBS: The Portable Batch System Software." http://www.openpbs.org/scheduler.html. Accessed March 3, 2010.

[150] PBSPro. http://www.hoise.com/primeur/03/articles/monthly/UH-PR-01-03-9.html.

[151] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling E-Science Applications on Global Data Grids," *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 18, no. 6, pp. 685–699, 2006.

[152] *The Gridbus Grid Service Broker and Scheduler*. User Guide, v.3.0 ed.

[153] Gridbus. http://www.gridbus.org/broker/3.0.

[154] LSF. http://www.platform.com/workload-management/high-performance-computing/lp.

[155] K. Vivekanandan and D. Ramyachitra, "A Study on Scheduling in Grid Environment," *International Journal on Computer Science and Engineering*, vol. 3, pp. 940–950, Feb 2011.

[156] P. Merz and B. Freisleben, "Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming," *Journal of Heuristics*, vol. 8, pp. 197–213, 2002.

[157] R. Armstrong, D. Hensgen, and T. Kidd, "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions," in *Proceedings of the 7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pp. 79–87, 1998.

[158] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling Resources in Multi-user, Heterogeneous, Computing Environments with SmartNet," in *Proceedings of the 7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pp. 184–199, 1998.

[159] P. Cappanera and M. Trubian, "A Local Search Based Hheuristic for the Demand-Constrained Multidimensional Knapsack Problem," *INFORMS Journal On Computing*, vol. 17, pp. 82–98, 2005.

[160] F. Glover and M. Laguna, *Tabu Search*, vol. 1. Springer, 1998.

[161] S. Kirkpatrick, C. D. Gelatt, and P. M. Vechhi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, May 1983. DOI: 10.1126/science.220.4598.671.

[162] M. D. Theys, T. D. Braun, H. J. Siegal, A. A. Maciejewski, and Y. K. Kwok, *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach*, ch. 6, pp. 135–178. New York, USA: John Wiley and Sons, 2006.

[163] J. H. Holland, *Adaptation in Natural and Artificial Systems.* AnnArbor: The University of Michigan Press, 1975.

[164] W. H. Hsu, "Genetic Algorithms," Tech. Rep. 66506-2302, Department of Computing and Information Sciences, Kansas State University, 234 Nichols Hall, Manhattan, KS, USA.

[165] K. R. Kumar and D. I. G. Amalarethinam, "Applying Non-Traditional Optimization Techniques to Task Scheduling in Grid Computing An Overview," *International Journal of Research and Reviews in Computer Science(IJRRCS)*, vol. 1, December 2010.

[166] M. V. Judy and B. Ramadoss, "An Enhanced Solution to the Protein Folding Problem Using a Hybrid Genetic Algorithm with G-Bit Improvement

Strategy," *International Journal of Modeling and Optimization*, vol. 2, no. 3, pp. 356–359, 2012.

[167] W. E. Hart, N. Krasnogor, and J. E. Smith, *Recent Advancement in Memtic Algorithm*, vol. 166. New York: Springer, Heidelberg, 2004.

[168] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed Optimization by Ant Colonies," in *Proceedings of the 1991 European Conference on Artificial Life*, (Springer US), pp. 134–142, 1991.

[169] P. E. Merloti, "Optimization Algorithms Inspired by Biological Ants and Swarm Behavior," Artificial Intelligence Technical Report CS550, San Diego State University, San Diego, 2004.

[170] M. Dorigo and L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem," *BioSystems*, vol. 43, pp. 73–81, 1997.

[171] M. Dorigo and A. Colorni, "The Ant System: Optimization by A Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 1–113, 1996.

[172] J. Kennedy and R.Eberhart, "Particle Swarm Optimization," in *Proceedings of the Fourth IEEE International Conference on Neural Networks*, p. 19421948., 1995.

[173] K. M. Passino, "Biomimicry of Bacterial Foraging for Distributed Optimization and Control," *IEEE Control Systems Magazine*, pp. 52–67, 2002.

[174] S. Dasgupta, S. Das, A. Abraham, and A. Biswas, "Adaptive Computational Chemotaxis in Bacterial Forgaing Optimization: An Analysis," *IEEE Transactions on Evolutionary Computing*, vol. 13, pp. 919– 941, 2009.

[175] S. Vo, "Meta-heuristics:the state of the art," in *Local Search for Planning and Scheduling, A.Nareyek(Ed.):, LNAI 2148*, (Berlin Heidelberg), pp. 1–23, Springer-Verlag, 2001.

[176] E. G. TALBI, "A Taxonomy of Hybrid Heuristics," *Journal of Heuristics*, vol. 8, pp. 541–564, 2002. Kluwer Academic Publishers.

[177] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A Survey of the State of the Art," Tech. Rep. Technical Report, University of Nottingham, 2009.

[178] K. Chakhlevitch and P. Cowling, "Hyperheurictics: Recent Developments," *Adaptive and Multilevel Metaheuristics, Studiesin Computational Intelligence*, pp. 3–29.

[179] X. Wang and J. Luo, "Architecture of Grid Resource Allocation Management Based on QoS," in *International conference Grid and Cooperative Computing, LNCS*, (Germany, Springer-Verlag), pp. 81–88, 2004. ISBN:0302-9743.

[180] A. Campbell, C. Aurrecoechea, and L. Hauw, "A Review of QoS Architectures," in *Proceedings of the 4th IFIP International Workshop on Quality of Service*, (Paris, France), March 1996.

[181] X. H. Sun and M. Wu, "Quality of Service of Grid Computing: Resource Sharing," in *Proceedings of the 6th International Conference on Grid and Cooperative Computing*, pp. 395–402, 2007.

[182] D. Kyriazis, K. Dolkas, A. Menychtas, and T. Varvarigou, "A New Workflow Mapping Mechanism for Grids," in *Proceeding of the 1st International ConferenceMobile and Wireless Communication Summit*, (Myconos, Greece), 4-8 june 2006.

[183] M. A. S. Netto, K. Bubendorfer, and R. Buyya, "SLA-based Advance Reservations with Flexible and Adaptive Time QoS Parameters," in *Proceedings of the 5th International Conference On Service-Oriented Computing*, (Vienna, Austria), Springer-Verlag, Berlin, Germany, September 2007.

[184] D. Colling, T. Ferrari, Y. Hassoun, C. Huang, C. Kotsokalis, A. S. McGough, E. Ronchieri, Y.Patel, and P. Tsanakas, "On Quality of Service Support for Grid Computing," in *Proceedings of the 2nd International Workshop on Distributed Cooperative Laboratories*, (Italy), 2006.

[185] M. Karsten, N. Berir, L. Wolf, and R. Steinmetz, "A Policy-Based Service Specification for Resource Reservation in Advance," in *Proceedings of the 1999 International Conference on Computer Communications*, 1999.

[186] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiation of Service Level Agreements and Coordinated Resource Management in Distributed Systems," in *Proceeding of the 2002 International Conference Job Scheduling Strategies for Parallel Processing*, April 30 2002.

[187] A. Sahai, S. Graupner, V. Machiraju, and A. V. Moorsel, "Specifying and Monitoring Guarantees in Commercial Grids through SLA," in *Proceeding of the 3rd IEEE/ACM International Conference on Cluster Computing and Grid (CCGrid)*, 2003.

[188] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, New York,, 2004.

[189] "Extended Markup Language." http://www.w3.org/standards/xml/core.

[190] H. Schulzrinne, H. Tschofenig, J. Morris, J. Cuellar, J. Polk, and J. Rosenberg, "Common Policy: A Document Format for Expressing Privacy Preferences," Tech. Rep. RFC 4745, February 2007.

[191] W. W. K. Lai, K. Ng, and M. R. Lyu, "Integrating Trust in Grid Computing Systems," in *Proceedings of the 2004 Grid and Cooperative Computing*, pp. 887–890, 2004.

[192] S. Song, K. Hwang, and Y. K. Kwok, "Risk-Resilient Heuristics and Genetic Algorithms for Security-Assured Grid Scheduling," *IEEE Transanctions on Computers*, vol. 55, p. 703719, 2006.

[193] K. Z. Gkoutioudi and H. D. Karatza, "Multi-Criteria Job Scheduling in Grid Using an Accelerated Genetic Algorithm," *Journal of Grid Computing*.

[194] B. Rood and M. J. Lewis, "Resource Availability Prediction for Improved Grid Scheduling," in *Proceeding of the 4th IEEE International of Conference on eScience*, eScience2008, p. 711718, 2008.

[195] I. Foster, "Website of the Gt 5.2.0 Gram5 Key Concepts, http://www.globus.org/toolkit/docs/5.2/5.2.0/gram5/key/," 2009.

[196] J. Kolodziej and F. Xhafa, "Enhancing the Genetic-based Scheduling in Computational Grids by a Structured Hierarchical Population," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1035–1046, 2011.

[197] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems," *Tamkang Journal of Science and Engineering*, vol. 3, no. 3, pp. 195–207, 2000.

[198] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox, "Pace- A Toolset for the Performance Prediction of Parallel and Distributed Systems," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 228–251, 2000.

[199] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times using Historical Information," in *Proceedings of the 1998 International Workshop on Job Scheduling Strategies for ParallelProcessing (IPPS/SPDP'98)*, (FL,USA), 1998.

[200] S. Hotovy, "Workload Evolution on the Cornell Theory Center IBM SP2," in *Proceedings of the 1996 International Workshop on Job Scheduling Strategies for Parallel Processing, IPPS96*, pp. 27–40, 1996.

[201] P. Cowling, G. Kendall, and E. Soubeiga, "A Hyper-heuristic Approach to Scheduling a Sales Summit," in *Proceedings of the 3rd International Conference on the Practice And Theory of Automated Timetabling, Springer,LNCS*, pp. 176–190, 2001.

[202] K. Y. Liuand and M. Passino, "Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles and Emergent Behaviors," *Journal of Optimization Theory Application*, vol. 115, no. 3, pp. 603–628, 2002.

[203] J. M. Spivey, *The Z Notation: A Reference Manual*. University of Oxford, Programming Research Group, second ed., 1998.

[204] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 7 ed., 2009.

[205] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1175–1220, 2002.

[206] U. Lublin and D. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *Journal of Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1105–1122, 2003.

List of Publications

International Journals (indexed by SCI)

1. Aron R., Chana I., "QoS based Resource Provisioning and Scheduling in Grids", Journal of Supercomputing, **Springer**, DOI 10.1007/s11227-013-0903-1, 2013, Impact Factor:0.578.

2. Rajni, Chana I., "Bacterial Foraging based Hyper-heuristic for Resource Scheduling in Grid Computing", Future Generation Computer Systems (**FGCS**), **Elsevier**, Vol 29, No 3, pp. 751-762, March 2013, doi:10.1016/j.future.2012.09.005, Impact Factor: 1.978.

3. Aron R., Chana I., "Formal QoS Policy based Grid Resource Provisioning Framework", Journal of Grid Computing, **Springer**, Vol 10, No 2, pp. 249-264, June 2012, doi: 10.1007/s10723-012-9202-y, Impact Factor: 1.310.

International Conferences

1. Rajni, Chana I., "Enhancement of Resource Provisioning in Grid Resource Management Systems", 11th annual Grace Hopper Celebration of Women in Computing, organized by Anita Borg Institute for Women and Technology and the ACM, Oregon Convention Center, Portland, Oregon, November 9-12, 2011. Online Available at: http://gracehopper.org/2011/assets/GHC2011ProgramProceedingsFinal.pdf

2. Aron R., Chana I., "Resource Provisioning for Grids: A Policy Perspective", in International Conference on Contemporary Computing, LNCS, Springer, pp-546-547, JIIT University Noida, August 8-10, 2011.

3. Aron R., Chana I., "Cost based Resource Provisioning Policy for Grids", in International Conference on Parallel and Distributed Computing, Proceedings of the World Congress on Engineering 2011 Vol II, WCE 2011, London, U.K, July 6 - 8, 2011 (**Awarded Certificate of Merit**)

4. Rajni, Chana I., "Resource Provisioning and Scheduling in Grids: Issues, Challenges and Future Directions", in IEEE International Conference on Computer and Communication Technology (ICCCT,10), pp:306-310, MNNIT, Allahabad, September 17-19, 2010.

5. Rajni, Chana I., "Resource Provisioning Policy based Scheduling for Grid Environment", 12th annual Grace Hopper Celebration of Women in Computing, organized by Anita Borg Institute for Women and Technology and

the ACM, Baltimore Convention Center, Baltimore, Maryland, October 3-6, 2012.

Paper Communicated

1. Rajni, Chana I., "A Hyper-heuristic approach for Resource Provisioning based Scheduling in Grid Environment", Concurrency and Computation: Practice and Experience, John Wiley & Sons, Ltd (Communicated).

Scholarship Awarded

1. GOOGLE COMMUNITY SCHOLARSHIP 2011 Awarded Google Community Scholarship for presenting Research paper entitled Enhancement of Resource Provisioning in Grid Resource Management Systems " and attending ANITA BORG Women in Technology, GHC-2011 International Conference which includes travel, registration and accommodation.

2. Microsoft sponsored "MSR India 2012 Summer School on Distributed Algorithms, Systems and Programming", IISC Bangalore, May 28-June 8, 2012