An Efficient Dynamic and Decentralized Load Balancing Technique for Grid

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

by
Neeraj Kumar Rathore
(950903007)

under the guidance of

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

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

November, 2014

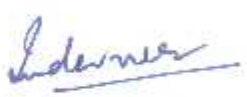
Certificate

I hereby certify that the work being presented in this thesis entitled, "**An Efficient Dynamic and Decentralized Load Balancing Technique for Grid**", 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 fully listed in the reference section.

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

(Neeraj Kumar Rathore)
Regn. No. - 950903007

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



(Dr. Inderveer Chana)
Associate Professor
Computer Science & Engineering Department
Thapar University
Patiala

Abstract

Grid computing has recently become one of the most important research topics in the field of computing. The Grid paradigm has gained popularity due to its capability to offer easier access to geographically distributed resources operating across multiple administrative domains. The grid environment is considered as a combination of dynamic, heterogeneous and shared resources in order to provide faster and reliable access to the Grid resources. For efficient resource management in Grid, the resource overloading must be prevented which can be obtained by proper Load Balancing and Job Migration mechanisms. In this scenario, dynamic and decentralized Load Balancing considers all the factors pertaining to the characteristics of the Grid computing environment. Dynamic load-balancing algorithms attempt to use the run-time state information to make more informative decisions in sharing the system load and in decentralization, algorithm is executed by all nodes in the system and the responsibility of Load Balancing is shared among all the nodes in the same pool. For this purpose, in this research work, an extensive survey of the existing Load Balancing and Job Migration techniques has been done. A detailed classification & gap analysis of the existing techniques is presented based on different parameters. A Job Migration and Job Migration approach has been proposed and designed to fulfill all the existing gaps.

The issue of Load Balancing in a Grid has been addressed while maintaining the resource utilization and response time for dynamic and decentralized Grid environment. Here, a hierarchical Load Balancing technique has been analyzed based on variable threshold value. The load is divided into different categories, like, lightly loaded, under-lightly loaded, overloaded, and normally loaded. A threshold value, which can be found out using load deviation, is responsible for transferring the task and flow of workload information. In order to improve response time and to increase throughput of the Grid, a random policy has been introduced to reduce the resource allocation capacity etc. Poission process has been used for random job arrival and then load calculation has been done for assigning job to the appropriate Processing Entity (PE) for balancing the load in the pool. After balancing the load, it comes into the normally loaded pool, then Job Migration process is executed.

In Job migration a process running on a Load Balancing resource is redeployed on another one in a way that the migration does not cause any change in the process execution. It means the process is temporarily suspended and later resumed on a new resource. For transferring the job from overloaded node to the underloaded node, the Job Migration

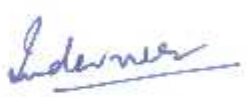
Certificate

I hereby certify that the work being presented in this thesis entitled, "**An Efficient Dynamic and Decentralized Load Balancing Technique for Grid**", 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 fully listed in the reference section.

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

(Neeraj Kumar Rathore)
Regn. No. - 950903007

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



(Dr. Inderveer Chana)
Associate Professor
Computer Science & Engineering Department
Thapar University
Patiala

system has also been proposed. Here, once the job known as Gridlet is submitted to a resource pool, job queue allocates the PE and starts the Job Migration. If any of the nodes, on which the job processes fail, Local Job Migration Scheduler suspends the job execution and tries to restart it on a new node in the same pool, else switches the job into other resource pool based on the availability of the under loaded node. Grid Information System periodically monitors the status of the job. If it detects that the job has been in the restart state for a long period, it tries to find a new resource pool to which the job can be migrated. Job migration can be achieved by following three procedures. Firstly, Job Migration with check-pointing, Checkpointing needs the small chunks of data to be migrated at the specific time requiring that data to be ready to get migrated. Application instance data on queues and failed events in the source node can be handled through checkpoining process. Next, Job Migration with scheduling, transfers the data on the basis of time and space. At last, Job Migration with replication requires migrating all the applications (such as code, database etc.)at the same time, from the source node to the destination node. So, for this, it needs all the applications on the source node to be ready to get migrated. For above techniques, Job Migration selection policy has been proposed for selecting one of the above techniques. Each of the above techniques has been executed according to its specific condition at run time.

To understand Job Migration with replication, a case study "Load Balancing and Job Migration in Social Grid Environment" has been considered further. In the case study, fault tolerance and Quality of Services (QoS) scheduling using Hash Table Functionality in Social Grid Computing has been proposed. Social Grid Computing is a computing model that includes devices to support user mobility. It connects with social networks to reflect real world user relationships, and therefore provides and shares Grid services directly among the members of a social network. Here, Hash Table Functionality (HTF) uses as the underlying Social Grid Computing (SGC) to logically manage the locations of the devices. Fault tolerance and QoS scheduling consist of four sub-scheduling algorithms: malicious-user filtering, Grid service delivery, QoS provisioning, and replication and load-balancing. Under the proposed scheduling, a device is used as a resource for providing Grid services, faults caused by user mobility or other reasons are tolerated and user requirements for QoS are considered. Simulation of scheduling is done, both with and without HTF. The experimantal results show that the proposed scheduling algorithm minimizes Grid service execution time, finish time and improves reliability reducing the Grid service error rate.

The performance of the proposed model, algorithms and techniques have been rigorously examined over the GridSim simulator using various parameters, such as response time, resource allocation efficiency, etc. Experimental results prove the superiority of the proposed techniques over the existing techniques.

Acknowledgements

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

It is with great pleasure that I wish to acknowledge several people that have helped me tremendously during the difficult, challenging, yet rewarding and exciting path towards a Ph.D. Without their help and support, none of this work could have been possible. First and foremost, I wish to express my sincere gratitude to my research guide, Dr. Inderveer Chana, Associate Professor for her guidance, encouragement, motivation, and continued support throughout my academic years at the Thapar University, Patiala, Punjab (India). She allowed me to pursue my research interests with sufficient freedom, while always being there to guide me. 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. Working with her has been one of the most rewarding experiences of my professional life.

I am also thankful to Dr. Deepak Garg, Head of Computer Science & Engineering Department, for his guidance through the early years of chaos and confusion. I would like to thank my Doctoral committee members Dr. Seema Bawa, Dr. V.P. Singh, and Dr. Rajesh Kumar for their constructive comments and regularly ensuring the progress of my research work. My deep regards to Prof. Prakash Gopalan, Director, Thapar University for all the facilities which have been immensely helpful for the completion of my work. I would like to sincerely thank Dr. Anil Kumar Verma, Associate Professor, Thapar University for helpful discussions and advices.

Many thanks go also to the faculties and staff of Jaypee University, Guna, M.P. (India), for my help. My fruitful discussions and interactions with them helped me grow professionally. I would also like to thank all the members of Centre of Excellence in Grid Computing at Thapar University, Patiala. I extend my thanks to Mr. Inderpreet Singh, Manager, Expicient Inc. and Dr. Ashutosh Singh, Assistant Professor, Jaypee University, Guna for his generous time and invaluable suggestions throughout the research work.

Last but not least, I am forever indebted to my wife, my parents, and the rest of my family. My wife, Smita Rathore, has been a great source of inspiration to me. None of this would have been possible without her love, support, and continuous encouragement. My deep regards to my father Mr. Mohan Lal Rathore and my mother Mrs. Santosh Rathore for their patience and love keeps me going. Without them this work would

never have come into existence. Their prayers have always accompanied me. They have provided me with lessons on honesty and ethics and their humbleness and patience have always amazed me. My sister's Jyoti & Priyanka and my brother-in-law Mr. Pramod Rathore have been the greatest source of motivation and inspiration during my Ph.D. I am grateful to all of them for their lifetime love, companionship, and support. This work is dedicated to my family. This thesis would have been impossible without the support of my family.

Neeraj Kumar Rathore

Contents

List of Figures

List of Tables

Abbreviations

LB	**L**oad **B**alancing
JM	**J**ob **M**igration
LGM	**L**ocal **G**rid **M**anager
JMR	**J**ob **M**igration with **R**eplication
JMC	**J**ob **M**igration with **C**heckpointing
JMS	**J**ob **M**igration with **S**cheduling
HTF	**H**ash **T**able **F**unction
SGC	**S**ocial **G**rid **C**omputing
PE	**P**rocessing **E**ntity
GB	**G**rid **B**roker
LJMS	**L**ocal **J**ob **M**igration **S**cheduler
LGM	**L**ocal **G**rid **M**anager
SM	**S**ite **M**anager
GIS	**G**rid **I**nformation **S**ystem
QoS	**Q**uality of **S**ervice
WLB	**W**ithout **L**oad **B**alancing
LBEGS	**L**oad **B**alancing in **E**nhanced **G**ridSim
PLBA	**P**roposed **L**oad **B**alancing **A**lgorithm
VO	**V**irtual **O**rganizations

I would like to dedicate this thesis to my Parents...

Chapter 1

Introduction

Distributed Computing Environment, like Grid, became popular due to growth of the Internet, and the emergence of low-cost solutions for end-user computing devices. Grid supports sharing and coordinated use of geographically distributed and multi-owner resources independently from their physical types and locations. Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities.

The outburst of the Internet, along with the increase in the computational power of the servers and emergence of high speed computer networks has given the power to Grid to share the computational resources. Grid is a vastly distributed enormous system that builds a global network of computational resources and may span many continents. The use of each small resource is flexible that combines computer resources and information systems for creating a ubiquitous source of computational power. Sharing resources between organizational and institutional boundaries needs an infrastructure to coordinate the resources across the boundaries, called the Virtual Organizations

This chapter provides a high level view of the thesis. It discusses the fundamental concept of the Grid technology, its evolution, architecture and key areas along with the major issues. It further provides the motivation to propose Load Balancing and Job Migration models & algorithms 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

"Grid is a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, Quality-of-Service requirement" -Rajkumar Buyya[1].

Grid Computing enables VO to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience, and an existing trust relationship.

FIGURE 1.1: Simplified Grid View

In other words:

(i) Grid is a service for sharing computer power and data storage capacity over the Internet and Intranet.

(ii) Grid is beyond simple communication between computers but it aims ultimately to turn the global network of computer into one vast computational resource.

(iii) Grid is used to coordinate resources which are not subjected to centralized control.

(iv) Grid uses standard, open, general-purpose protocols and interfaces.

(v) Grid delivers nontrivial Qualities of Service.

Grid Computing [2] resources have enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity

TABLE 1.1: Historical Background of the Grid [2]

S.No.	Technology	Year
1	Networked Operating Systems	1979-81
2	Distributed Operating Systems	1988-91
3	Heterogeneous Computing	1993-94
4	Parallel and Distributed Computing	1995-96
5	The Grid	1998

of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments as shown in Figure 1.1. Moreover, in recent times, rapid advances in networking, hardware and middleware technologies are facilitating the development and deployment of complex applications, such as large-scale distribution, collaborative scientific simulation, analysis of experiments in elementary particle physics, distributed mission training, etc. These predominantly collaborative applications are characterized by their very high demand for computing, storage and network bandwidth requirements, which can be fulfilled by Grid Computing. Recent researches on computing architectures have allowed the emergence of a new computing paradigm known as Grid Computing [3].

1.1.1 History of Grid

The theory behind "Grid Computing" is not to buy more resources but to borrow the power of the computational resources you need from where it's not being used. Grid has been around in one form or other throughout the history of computing as shown in Table 1.1. There is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented , it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rate. Due to the huge improvements of the underlying hardware (typically more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor.

1.1.2 Grid Architecture

The last decade has seen a considerable increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software.

The Architecture (presented in Figure 1.2) of a Global Grid has to be established on the following principles:

- Employ a common network infrastructure

- Transport any traffic type
- Integration of various transport media
- Adaptation to changes
- Provide Quality of Service

The architecture of the Grid is often described in terms of "layers" as shown in Figure 1.2, each providing a specific function. In general, the higher layer is user-centric, whereas the lower layers are more hardware-centric [4].

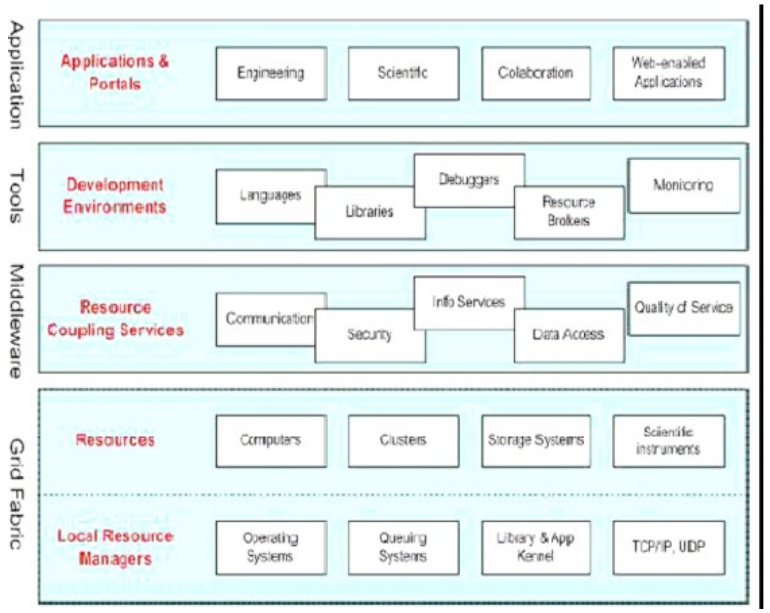


FIGURE 1.2: Grid Architecture [4]

(i) Network Layer: At the bottom is the Network Layer, which assures the connectivity for the resources in the Grid. On the top of it lies the Resource Layer, made up of the actual resources that are part of the Grid, such as computers, storage systems, electronic data catalogues, and even sensors, such as telescopes or other instruments, which can be connected directly to the network.

(ii) Middleware Layer: The Middleware Layer provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified Grid environment. The Middleware layer can be thought of as the intelligence that brings the various elements together.

(iii) Tool Layer: The Tool Layer is made up of tools that assure connectivity between Middleware and applications. It deals with several languages using a set of libraries to integrate the requests either from lower or higher layers.

(iv) Application Layer: It is the highest layer of the architecture that includes all different user applications. In this layer, Grid user will "see" and most of the time interacts through their browser. This layered structure can be defined in other ways. For example, the term fabric is often used for all the physical infrastructure of the Grid, including computers and the communication network. Within the Middleware layer, distinctions can be made between a layer of resource and connectivity protocols, and a higher layer of collective services. However, in all schemes, the Applications Layer remains the topmost layer.

1.1.3 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 [5]:

- Web Services: Grid Services, defined by OGSA, is an extension of web services. So, Grid service can leverage the available Web Service specifications. Web Service based on:

  (i) Extensible Markup Language (XML)is the basis of web services that is used to store and transport the data.

  (ii) Simple Object Access Protocol (SOAP) 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 Grid.

  (iii) Web Service Description Language (WSDL) [6] is an XML based language, used for describing the model of web services.

  (iv) Universal Description, Discovery, and Integration (UDDI) [7] is a key member of the web services stack and defines the ways in which web services are published and discovered across the network. It is approved by the organization for the Advancement of Structured Information Standard (OASIS) as a standard for service registries in the context of SOA.

- 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. OGSA [8] 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. OGSA provides services such as resource discovery, resource provisioning, resource management, data management, information services and security.

- Open Grid Services Infrastructure (OGSI): OGSI [9] 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 [10] is the de-facto standard for modeling and accessing resources using web services.
- Open Grid Service Architecture-Data Access and Integration (OGSA-DAI): OGSA-DAI [8] 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.

After discussing the Grid essentials such as History of Grid, Evolution, Type of Grid, VO, Grid Architecture, its characteristics and standards, the next section will focus on the key challenges of this area.

1.1.4 Concept of Virtual Organizations

The concept of a Virtual Organization (VO) is the key to Grid Computing. It is defined as a dynamic set of individuals and/or institutions defined around a set of resource-sharing rules and conditions. All VOs share some commonality among them, including common concerns and requirements, but may vary in size, scope, duration, sociology, and structure. The members of any VO negotiate on resource sharing based on the rules and conditions defined in order to share the resources from the thereby automatically constructed resource pool. Assigning users, resources, and organizations from different domains across multiple, worldwide geographic territories to a VO are one of the fundamental technical challenges in Grid Computing. This complexity includes the definitions of the resource discovery mechanism, resource sharing methods, rules and conditions by which following can be achieved, security federation and/or delegation, and access controls among the participants of the VO [11]. This challenge is both complex and complicated across several dimensions.

It undeniably creates the illusion of a simple large and powerful self-managing virtual computer, which gives the opportunity to use and reach heterogeneous systems sharing massive resources as shown in Figure 1.3.

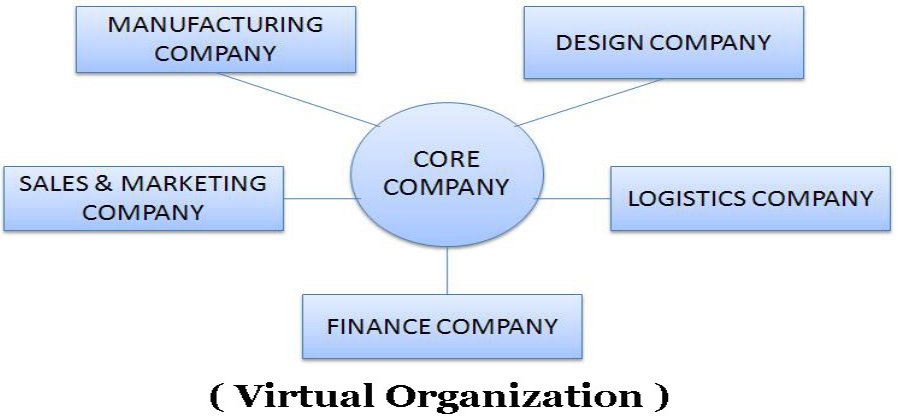


FIGURE 1.3: Grid Member submits a Task to the Grid via the Grid Interface

1.1.5 Grid Characteristics

In todays' complex world, computers have become extremely powerful and they become enough capable to run more complex problems, but still there are many areas left [12]. Grid Computing offers seamless access to distributed data and collaborative distributed environments, for running computationally intensive applications. Computing power available to any organization can thus be increased by using the idle periods of computing resources [13] [14]. Some of the characteristics of Grids are:

(i) Parallel CPU Capacity: The common attribute among such uses"medical fields,finnancial modeling, oil exploration, motion picture animation etc." are that the applications have been written to use algorithms that can be partitioned into independently running parts.

(ii) Collaboration of Virtual Resources: In the past, Grid Computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. The users of the grid can be organized dynamically into a number of virtual organizations each with different policy requirements. These VO can then share their resources collectively as a larger grid [13].

(iii) Access to Additional Resources: In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to

increase his total bandwidth to the internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the internet [14].

(iv) Reliability: Grid provides reliability in terms of failure at one location, while the other parts of the grid are not likely to be affected. Grid Management software can automatically or manually resubmit jobs most of the times to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the Grid. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering; thus offering much more reliability.

(v) Resource Balancing: For Grid application, the Grid can offer a resource balancing effect by scheduling Grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization.

(vi) Heterogeneity: A Grid hosts both software and hardware resources that can be extremely diverse ranging from data, files, software components or programs to sensors, scientific instruments, display devices, personal digital organizers, computers, super-computers and networks. A Grid involves a variety of resources that are heterogeneous in nature and might span several administrative domains across wide geographical distances. Resources are owned and managed by different, potentially mutually suspicious organizations and individuals that likely have different security policies and practices [13].

(vii) Scalability: It is a desirable property of a system, a network or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. A Grid might grow from few resources to the millions. This raises the problem of potential performance degradation as Grid‘s size increases. Consequently, applications that require large number of geographically located resources must be designed to be extremely latency tolerant.

(viii) Dynamicity or Adaptability: In a Grid, a resource failure is the rule, not the exception. As in a Grid there are numerous resources, the probability of some resource failing is naturally very high. The resource managers or applications must modify their behavior dynamically so as to extract the maximum performance from the available resources and services [14].

(ix) Resource Coordination: Resources in a Grid must be coordinated in order to provide aggregated computing capabilities.

(x) Reliable Access: A Grid must assure the delivery of services under established Quality of Service (QoS) requirements. The need for reliable service is elementary since administrators require assurances that they will receive expected, continuous and often high levels of performance.

1.1.6 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 different kinds of applications, ranging from High Performance Computing (HPC) to High Throughput Computing (HTC). 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. In other words, a Grid Application may consist of a number of jobs that together fulfill the whole task [1].

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), The Large Hadron Collider produces roughly 15 petabytes (15 million gigabytes) of data annually enough to parallelize 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. Similarly other applications like Enabling Grid for E-science (EGEE), SETI@home, 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 [15].

1.2 Grid Challenges

There are many challenges in Grid Environment, some of the key challenges are discussed below [16] [17] [18] [19] [20] [21] [22]:

(i) Adaptability: Adaptability refers to handle all the unexpected system activities [22].

It also provides a decision-making procedure to control these modifications, for example to increase the load distribution effort in the case of a wide load imbalance or to reduce it when the load of all nodes is so heavy or so light that no improvement can be achieved by such effort [20].

Adaptability can be classified into two categories:

In the *first* category the adaptability is included within the basic structure of the algorithm (infomation, transfer, and negotiation).

In the *second* approach, for making a scheduling algorithm adaptive to the system dynamic characteristics and workload conditions, the algorithm needs to dynamically assess the system environment and adjust the global scheduling strategy accordingly.

(ii) Scalability: As the number of nodes in the Grid grows and the range of workload fluctuations increases, scalability problems can arise. In order to cope with the communication and scheduling overhead resulting from the increased load distribution effort, a number of principles are to be observed during the design of these algorithms to make them more scalable:

- Symmetry: All nodes in the system should be allowed to play an equal role.
- Customer-server protocols: Each customer-server interaction should involve at most two nodes (one-to-one).
- Partiality:Every decision should be based on information from a bounded subset of the other nodes.
- Use randomness: The set of nodes with which a node interacts is chosen at random.

(iii) Resource Heterogeneity: Grid Computing solves massive computational problems by making use of unused resources of a largely heterogeneous collection of computers in a distributed infrastructure. A heterogeneous computational grid mainly has two categories of resources: First networks, which used to interconnect these computational resources. Second, computational resources may have different hardware and software [22].

(iv) Information Freshness: It is based on a hierarchical scheme that devises a set of nodes, each one hosting an Index Service. Each Index Service maintains a database on current resource availability for a group of grid nodes. Index Services can be also aggregated in higher-level Index Services, using a hierarchical approach typical of information repositories for widely distributed systems (such as the Internet domain name system). Even though it has been proved that hierarchical approaches are well suited to efficiently manage huge quantities of information in large distributed systems, the MDS approach suffers from a key problem: since resource information is advertised from grid nodes to index services, and then forwarded up in the hierarchy, inconsistences due to propagation latencies are possible, between the real information present in a grid node and the relevant copy stored in the repositories. As a result, matchmaking is performed using data that cannot be real. The communication overhead involved in capturing the load information of sites before making a dispatching decision can be a major issue negating the advantages of Job Migration.

(v) Decision Making Structure: Decision-making Structure used to implement the different components. This structure can be centralized, hierarchical, decentralized, or a hybrid form. The centrally based algorithms such as Central suffer from the reliability problem due to a potential central point of failure and the potential bottleneck of the central node. To deal with the autonomous nature of the nodes in a distributed system, fully decentralized properties are needed. Most of the algorithms considered in this review fall under the decentralized category.

(vi) Load Redistribution: Different objectives can be pursued when performing a load redistribution within a distributed system.

- Load Sharing is used when the objective is to keep all the nodes busy.
- Load Balancing is used when the objective is not only to keep the nodes busy but also to attempt an equalization of the load over all the nodes.
- When the nodes are privately owned and their sharing is allowed only with the approval of the owner we use the term restricted sharing.

The computing power is sharable only during a specific period of time based on the discretion of the node owners. A finer objective is the type of load imbalance that the algorithm attempts to resolve. For a steady state imbalance the jobs are transferred between the nodes so that the arrival rates approach the mean arrival rate. Transient imbalance is resolved by assigning each new job to the node with the least number of jobs.

(vii) Resource Selection: Resource selection in grid environment is difficult due to the large no of resources, which are dynamically being joining grid and some are leaving the grid due to machine failure or connection problem. That's why the cost data staging is high in a grid environment. Data staging cost is most prominent obstacle in grid environment or a main problem in designing the efficient and most effective Load Balancing system in Grid Environment [22].

Another issue related to Load Balancing is that a computing job may not be arbitrarily divisible leading to certain constraints in dividing tasks. Each job consists of several tasks and each of those tasks can have different execution times. Also, the load on each resource as well as on the network can vary from time to time based on the workload brought about by the users. The resource capacity may be different from each other in architecture operation system, CPU speed, memory size and available disk space [20].

(viii) Dynamic Behavior: In Grid environment, there are more and more problems due to the heterogeneity of resources. Any time any resource may be available and can be unavailable due to machine failure or connection problems. This dynamic behavior always gives headache to the user of Grid environment.

(ix) Resource Non-Dedication: In this challenge, resource may join the many Grid systems simultaneously. A contention arises there when requests from many users come there. So due to resource non dedication resource usage contention is the major issue in Grid environment.

(x) Application Diversity: In Grid environment, there is different types of users having different type of applications and have different requirements. For example, some applications may have set of dependent jobs, other may have independent jobs, and on other side some application requires sequential execution. Keeping all aspects of jobs in mind, designing a general purpose Load Balancing system is extremely difficult [19].

After discussing the challenges in Grid Environment, the next section will focus on the research motivation in Load Balancing and Job Migration.

1.3 Grid Load Balancing and Job Migration : The Research Motivation

Load balancing is a key research area in Grid. The research topic of Grid Load Balancing is still developing. Many Load Balancing systems have been proposed and reported but these systems have several limitations. Hence, with the development and popularity of Grid computing, much effort is required to enable Grid computing to be real platform for delivering high performance services. Some of the challenges of Load Balancing algorithms and Job Migration are very closely tied to the applications that use migration. After extensive background analysis of Load Balancing following gaps have been identified, which have been summarized as follows:

- Most of the existing Load balancing algorithms work on decentralized approach which is better than centralized approach, but still faces problems like scalability, information gathering overhead etc and is prone to failures which need to be addressed [23] [24] [25].
- Dynamic Load balancing policy performs very competitively in heterogeneous Grids. This policy uses an effective mechanism for state information exchange, which significantly reduces the communication overhead but still it has issues like random behaviour, complexity etc. which need to be addressed [26].
- There is a need to achieve a greater overall improvement in system performance at a reasonable cost, e.g., reduce task response time while keeping acceptable delays, using fault tolerance under partial failure in the system [27].
- The complex, heterogeneous and dynamic nature of Grid systems presents new challenges in Load Balancing such as response time, resource allocation capacity etc [28].
- When an application is migrated through Load Balancing and executed in large scale distributed environment there may be bugs that manifest now that might not have become apparent in small scale configuration. Detecting and debugging errors in such large-scale distributed deployment is quite challenging. Further migration distance can also introduce errors [29].
- When a node is about to shutdown, the system can migrate the jobs which are running on it to another node, thereby enabling the jobs to go to completion, either on the destination node or on the source node by migrating it back [29].
- In Grid Job Migration, it's not an easy task to migrate task to overloaded to underloaded resources. There is no facility of accessing the resources by calling an

intermediate agent. Moreover, the Job Migration algorithms need to have better response time and hit ratio in comparison to any other conventional algorithms available [31].

- Migrating task on a set of heterogeneous and dynamically changing resources is itself a complex problem that requires sophisticated Job Migration techniques that take into account multiple-optimization criteria [19].
- It is expected that local jobs will be assigned higher priority and would be better served on the local resources but this needs to be incorporated in the scheduling technique during migration of resources [19].
- Due to the dynamic nature and an uncertainty of Grid computing system behavior, Load Balancing depends crucially on the selection of appropriate subset of the available resources [32].
- Job Migration can be used to migrate a distant process closer to the data "neighbour node" that it is processing, thereby ensuring it spends most of its time doing useful work [27].
- Existing Techniques lack in the efficiency of Job Migration for the load barrier policy and the minimal job turnaround time policy, requires worthy policy to support [30].
- Existing checkpointing lacks in providing check pointing for parallel-node jobs, and the realization of inter-cluster Grid migration [33].

Due to the above said limitations, this research work focuses on the design and development of dynamic and decentralized LB & JM technique in Grid. The next section will focus on thesis organization.

1.4 Thesis Contributions

This Thesis contributes in the following ways:

- A detailed analysis of the work done in the area of Grid Load Balancing and Job Migration has been done along with a detailed study of scheduling algorithms.
- Load Balancing threshold parameters have been used for three level Load Balancing model for equal workload distribution and fast response time.

- The Load Balancing problem has been formulated in the form of an optimization problem. A new dynamic, hierarchical Load Balancing algorithm based on variable threshold value has been proposed and implemented.
- The Job Migration model with decision based Job Migration policies has been designed.
- The Job Migration policies have been proposed for achieving fault tolerance also. JM can be done via check-pointing, scheduling and replication. For choosing one of the three techniques, JM policy has been proposed that takes decision at the time of migration of the load.
- A case study of JM with replication algorithm has been proposed and validated in the Social Grid Computing environment.
- The performance of the Load Balancing algorithms is evaluated using GridSim toolkit based on response time, resource allocation capacity etc.
- The simulation results clearly demonstrate that the proposed algorithm outperforms the existing Load Balancing algorithms in all respects such as effect of resource allocation capacity, response time, Grid service execution time, service reliability, error rate, makespan, complexity etc.

1.5 Thesis Organization

The work reported in the thesis is organized in six more chapters. The details are given below:

Chapter 2 presents a literature survey on Load Balancing and Job Migration techniques. A taxonomy of Grid Load Balancing components, policies, strategies, attributes, algorithm's types, classification and hierarchical approach for Load Balancing have been presented. A detailed description and comparison of the existing Grid Load Balancing algorithm in different environments has been included. The analysis of Job Migration techniques for different environments, Job Migration steps, policies and features has also been discussed. Various existing methods to tolerate faults and support QoS for users in many computing environments, system environment have been depicted. The chapter finally concludes with the Problem Formulation. Chapter 2 is partially derived from:

(i) N Rathore and I Chana, "Load Balancing and Job Migration Algorithm : A Survey of Recent Trends", Wireless Personal Communication, Springer Publication, ISSN print 0929-6212, IF - 0.979, DOI-10.1007/s11277-014-1975-9, July 2014.

(ii) Neeraj Kumar Rathore and Inderveer Chana, "Report on Hierarchal Load Balancing Technique in Grid Environment", International journal of scientific and innovative technology, i-manager's Journal on Information Technology, Vol. 2, No. 4, ISSN Print: 2277-5110, pp-21-35, Sep - Nov 2013.

(iii) N Rathore and I Chana, "A Cogitative Analysis of Load Balancing Technique with Job Migration in Grid Environment", World Congress on Information and Communication Technology (WICT), Mumbai, IEEE proceedings paper, ISBN - 978-1-4673-0127-5 pp- 77-82, December 2011

Chapter 3 presents design of hierarchal Load Balancing framework based on Variable threshold value. In order to overcome the scalability, response time and resource allocation efficiency, a dynamic threshold value with random policy has been used at each level in proposed technique to achieve the desired goal. Details about the proposed algorithm are discussed in this chapter. Chapter 3 derives from:

(i) N Rathore and I Chana, "Variable Threshold Based Hierarchical Load Balancing Technique in Grid", Engineering with computers, Springer publication, ISSN: 0177-0667 (print version), DOI-10.1007/s00366-014-0364-z ,IF- 1.088, June 2014.

(ii) N Rathore and I Chana, "Load Balancing and Job Migration Algorithm : A Survey of Recent Trends", Wireless Personal Communication, Springer Publication, ISSN print 0929-6212, IF - 0.979, DOI-10.1007/s11277-014-1975-9 ,July 2014.

(iii) N Rathore and I Chana, "A Sender Initiate Based Hierarchical Load Balancing Technique for Grid Using Variable Threshold Value", in International conference IEEE-ISPC, ISBN- 978-1-4673-6188-0, pp.1-6, 26-28 Sept. 2013.

Chapter 4 present design of Job Migration (JM) model based on JM decision policy. For Job Migration model, decision making JM policies have been proposed for achieving fault tolerant Job Migration. JM can be made via check-pointing, scheduling and replication of techniques. For choosing one of the three available techniques, JM policy takes decision at the time of migration of the load. The algorithms Chapter 4 derives from:

(i) N Rathore and I Chana, "A Sender Initiate Based Hierarchical Load Balancing Technique for Grid Using Variable Threshold Value", in International conference IEEE-ISPC, ISBN- 978-1-4673-6188-0, pp.1-6, 26-28 Sept. 2013.

(ii) Neeraj Kumar Rathore and Inderveer Chana, "An Efficient Hierarchical Load Balancing Technique for Grid" in 29th M.P. Young Scientist congress, Bhopal, M.P., pp. 55, Feb 28, 2014.

(iii) N Rathore and I Chana, "Job Migration policies for Load Balancing algorithms in Grid Computing", Journal of Grid Computing, SCI indexed, Print ISSN: 1570-7873, Impact Factor: 1.667, June 2014. (Communicated)

Chapter 5 explained the deployment of the model. The experimental results have been illustrated with the help of test cases. Further, the model has been validated by performing system model and also compared with the existing systems. Validation of the Load Balancing model has been done through variable threshold based dynamic Load Balancing algorithm with random policy for better response time, communication overhead and resource allocation efficiency, etc. wherein resources, machines and the Grid broker participates in the Load Balancing operations. The JM Model has been verified with JM policy to achieve fault tolerance for supporting LB. JM decision has been chosen at the time of migration according to scenario. This approach actually minimizes the searching time to find out the receiver machine in order to transfer the Gridlets. A rigorous examination of the proposed algorithm on the GridSim simulator has been carried out in order to judge the effectiveness of the algorithm. This chapter provides the design and implementation details of the proposed algorithms. Chapter 5 derives from:

(i) N Rathore and I Chana, "Variable Threshold Based Hierarchical Load Balancing Technique in Grid", Engineering with computers, Springer publication, ISSN: 0177-0667 (print version), DOI-10.1007/s00366-014-0364-z ,IF- 1.088, June 2014.

(ii) N Rathore and I Chana, "A Sender Initiate Based Hierarchical Load Balancing Technique for Grid Using Variable Threshold Value" in International conference IEEE-ISPC, ISBN- 978-1-4673-6188-0, pp.1-6, 26-28 Sept. 2013.

(iii) N Rathore and I Chana, "Job Migration policies for Load Balancing algorithms in Grid Computing", Journal of Grid Computing, SCI indexed, Print ISSN: 1570-7873, Impact Factor: 1.667, June 2014. (Communicated)

(iv) N Rathore and Inderveer Chana, "Performance of Load Balancing Algorithm in Complex Networks", Wireless Personal Communication, Springer Publication, SCI indexed, ISSN print 0929-6212,impact factor -0.979, July 2014 (Communicated)

Chapter 6 describes the case study based on Fault tolerance with replication algorithm has been proposed to support device users and share Grid services directly between users and provide fault tolerance based QoS to users in the SGC. To demonstrate the usability of the proposed scheduling algorithm, its implementation and validation has been performed using scheduling with and without HTF. The simulation results shows that by using all the four algorithms results have been improved. Therefore, the proposed

scheduling algorithms can be applied to improve execution time and reliability in a wide range of computing environments that provide shared services. In addition, to conduct a wider variety of experiments to study various QoS metrics and additional factors in the SGC and will apply this fault tolerance QoS scheduling algorithm to real world environments can also be done. Chapter 6 derives from:

(i) N Rathore and I Chana, "Load Balancing and Job Migration Algorithm : A Survey of Recent Trends", Wireless Personal Communication, Springer Publication, ISSN print 0929-6212, IF - 0.979, DOI-10.1007/s11277-014-1975-9 ,July 2014.

(ii) N Rathore and I Chana, "Job Migration with Fault Tolerance and QoS Scheduling using Hash Table Functionality in Social Grid Computing", Journal of Intelligent & Fuzzy Systems, IOS Press publication, ISSN print 1064-1246, ISSN online 1875-8967, DOI- 10.3233/IFS-141243 , IF- 0.936, June 2014.

(iii) N Rathore and I Chana, "Job Migration policies for Load Balancing algorithms in Grid Computing", Chiang Mai Journal of Sciences, SCI indexed, Print ISSN: 0125-2526, Impact Factor: 0.516, June 2014. (Communicated)

Chapter 7 finally concludes the thesis and discusses the future scope of the work.

Chapter 2

Literature Survey

The previous chapter provides a high level view of the thesis. It discusses the fundamental concept of the Grid technology, its evolution, architecture and key areas along with the major issues. 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.

In grid environment, tasks can be submitted at any host and the random arrival of tasks in such an environment can cause some hosts to be heavily loaded while others are idle or lightly loaded. Therefore, in such an environment, load imbalance can potentially be reduced by appropriate migration of the tasks from heavily loaded systems to idle or lightly loaded systems. Load Balancing with Job Migration are key Grid services. Load Balancing is effective for balancing the load of large scale heterogeneous Grid resources that are typically owned by different organizations. Job Migration is an important aspect of executing a parallel program in the Grid. It is a special event when a process running on a Load Balancing resource is redeployed on another one in a way that the migration does not cause any change in the process execution. The essential objective of Load Balancing primarily consists of optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the entire resources of a system.

This chapter discusses the classification and comparison of the different Load Balancing and Job Migration schemes. A Load Balancing taxonomy is presented to facilitate a better understanding of Load Balancing and Job Migration in Grid environment. Further, a survey of dynamic Load Balancing and Job Migration mechanisms has been presented. A detailed classification has also been included based on different parameters which depend on the analysis of the existing techniques . The last section highlights the adoption of Load Balancing and Job Migration techniques, limitation of the existing approaches and finally lists down the objectives as problem formulation of the thesis.

2.1 Load Balancing in Grid

Efficient Load Balancing across the grid can lead to improve overall system performance and a lower turnaround time for individual jobs. This is a very crucial concern in distributed environment like Grid, to assign fair jobs to resources. The main goal is to distribute the jobs among processors to maximize throughput, maintain stability, and resource utilization. This is achieved by proper Load Balancing techniques. The basic steps [17] for Load Balancing are shown in Figure 2.1.

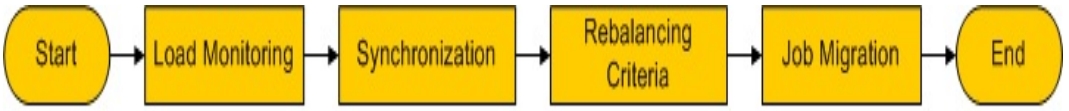


FIGURE 2.1: Load Balancing Steps

(i) Load Monitoring: Monitoring resource load and state.

(ii) Synchronization: Exchanging load and state information between resources.

(iii) Re-balancing Criteria: Calculating the new work distribution and making work moment decision.

(iv) Job Migration (Actual data movement): It needs, when a system decides to export a process

The problem of Load Balancing came into the limelight as soon as the concept of multi-processor as well as multi computation system architecture were documented. Today, the modern era of computing is mostly dominated by high speed processors with incredible processing power, advanced system architecture, complex storage hierarchy, lightning fast network services and powerful application supports, but the problem of Load Balancing has not been comprehensively solved yet.

Load Balancing algorithms vary in their complexity where complexity is measured by the amount of communication used to approximate the least loaded node. The most significant parameter of the system is the cost of transferring a job from one node to another [35]. Following terms are often used interchangeably but can also have distinct meaning, which is given as follows:

- Load: The ratio of number of processes in ready state to the performance, if the performances of servers are the same.
- Load Sharing: This is the coarsest form of load distribution. Load may only be placed on ideal resources and can be viewed as binary (either idle or busy).

- Load Balancing: Balance criteria of the workload present on every resource in the system and workload should be in equal degree.
- Load Leveling: load leveling introduces a third term to describe the middle ground between the two extremes of load sharing and Load Balancing. It seeks to avoid congestion on any resources by utilizing idle resources.

Next section discusses the taxonomy of Load Balancing in Grid Environment.

2.2 Load Balancing Taxonomy

Various Load Balancing algorithms have been proposed in the last twenty years or so. The taxonomy of the different existing LB schemes is shown in the literature in Figure 2.2 & the details are discussed in the subsequent sub-sections.

2.2.1 Scheduling for Balancing of Load

At the top level, the Load Balancing algorithm is divided into local and global scheduling.

(i) Local and Global Scheduling: Local balancing (LB) [36] is the assignment of processor time quantum to task as it is done by every traditional operating system.In a local Load Balancing, each resource polls other resources in its neighbourhood and uses this local information to decide on a load transfer. This local neighbourhood is usually denoted as the migration space. The primary objective is to minimize remote communication as well as effectively balance of the load on the resources. Global balancing, on the other hand, is the process of deciding the location to execute a task in a multicomputer environment. Global balancing can be implemented using a centralized approach, or it may be distributed among various processing elements.

In a global balancing scheme [36] [37], global information of all or part of the system is used to initiate the Load Balancing. This scheme requires a considerable amount of information to be exchanged in the system which may affect its scalability. Global Scheduling further classified into two major groups: Static Load Balancing and Dynamic Load Balancing.

(ii) Static Scheduling and Dynamic Scheduling: Dynamic Algorithms [38] balances the load at the time of the arrival of each job by a continuous assessment of the system load in real time environment of a fixed policy. LB could be done statically "At compile-time" and dynamically "At run-time".

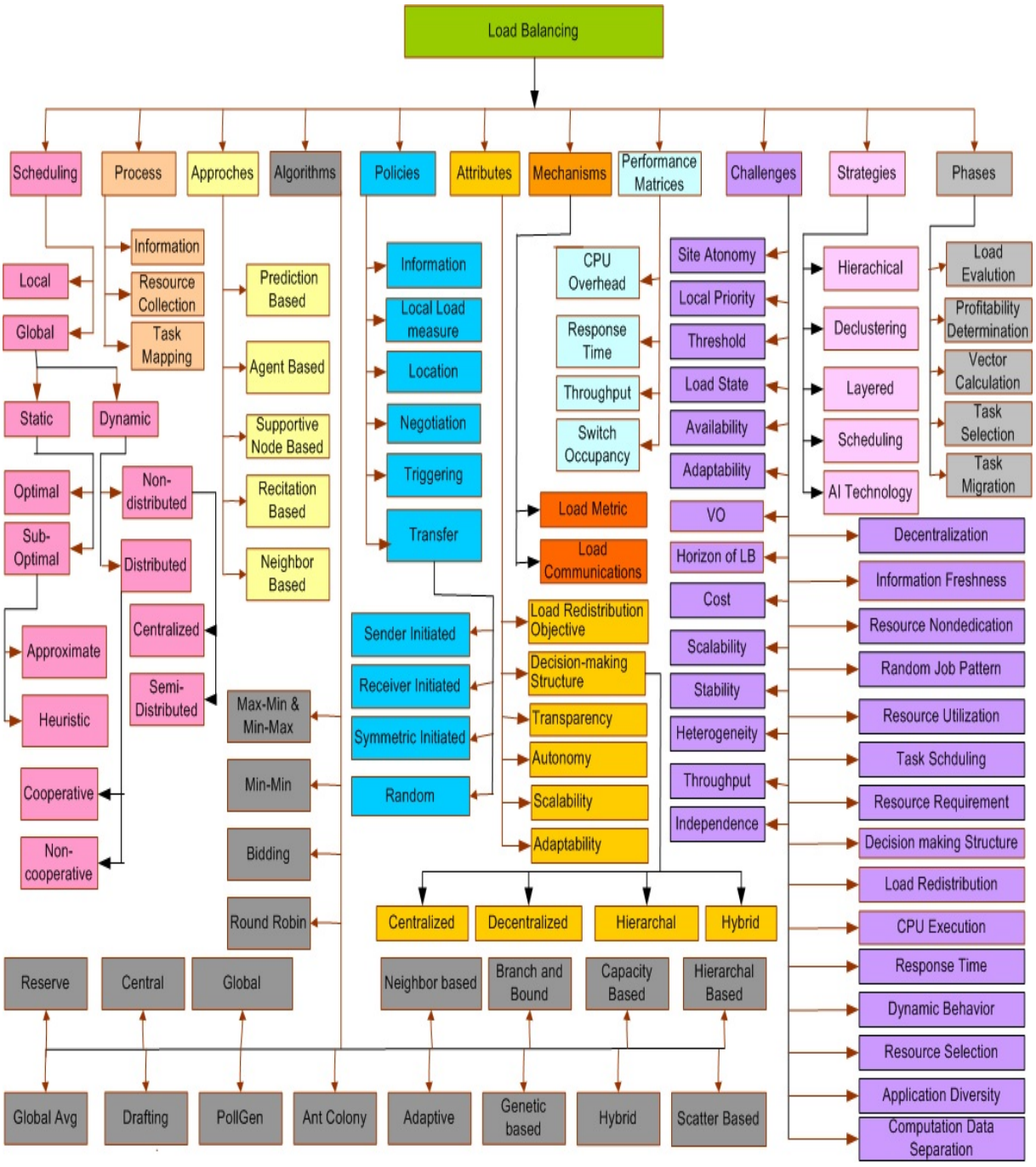


FIGURE 2.2: Load Balancing Taxonomy

(iii) Optimal and Suboptimal Scheduling: Static scheduling can be divided into optimal scheduling "Related information about the job and the resource is known by the job-scheduler and optimal allocation decisions can be calculated within a feasible period of time" and suboptimal scheduling "If these problems are not feasible computational and/or some related information is unknown, suboptimal allocation is adopted". Suboptimal static scheduling is divided into two parts: Approximate and Heuristic.

- In approximate scheduling, the algorithm will terminate when it finds a "good solution" instead of searching the entire solution space.
- Heuristic scheduling uses the most naturalistic assumptions about the applications and resources to make a "reasonable solution" that is not restricted by the assumptions, nor evaluated by an objective function.

(iv) Distributed and Non-Distributed /Centralized

Dynamic scheduling can be further bifurcated into distributed and non-distributed (centralized) scheduling.

- In Distributed Scheme [39], Dynamic Load Balancing algorithm is executed by all nodes in the system and the responsibility of Load Balancing is shared among them. It can take two forms: Cooperative "Multiple resources will work together to make dynamic and distributed job-scheduling decisions toward a common goal" and Non-cooperative scheduling "Each resource works alone making job-scheduling decisions".
- In Non-Distributed Scheme, the responsibility of LB is either taken single or shared by some nodes. In other words,all the decisions can be made at a centralized place at the user/resource management level. It can be categorized into centralized and semi distributed.
  - Semi-distributed, nodes of the distributed system are segmented into clusters. Load Balancing within each cluster is centralized; a central node is nominated to take charge of Load Balancing within this cluster. While in centralized, the central node is only responsible for Load Balancing of the whole distributed system [40].
  - Centralized node work as a load Balancing decision maker, which distributes jobs to different processing nodes. The centralized policies have the advantages of easy information collection about job arrivals and departures, and natural implementation that employs the server-client model of distributed processing and performance and reliability bottleneck due to the possible heavy load on the centralized job load-balancing decision maker are major drawbacks.

2.2.2 Load Balancing Process

In general, a Load Balancing process in the grid includes three stages:

(i) Information Collection: Information collection is the basis for providing current state information of the resources [41]. It should be performed during the whole course of the system running.

(ii) Resource Selection: Resource Selection is accomplished in two steps. First step, the initial filtering is done with the goal of identifying a list of authorized resources that are available to a given application. In the second step, those resources are aggregated into small collection such that each collection is expected to provide performance desired by the given application.

(iii) Tasks Mapping: The third phase involves mapping the given set of tasks onto a set of aggregated resources including both the computational resources and network resources. Various heuristics may be used to reach a near-optimal solution. The effort of mapping in conjunction of resource selection produces a set of candidate submission. Once the set of candidate submission is ready; the balancer starts to selects the best submission subject to the performance goal, based on the mechanisms provided by the performance model.

2.2.3 Load Balancing Approches

Till date, there have been a number of efforts at developing Load Balancing systems for grid environments. It is often difficult to make comparisons between distinct efforts because each Load Balancing approach is usually developed for a particular system environment or particularly greedy application with different assumptions and constraints. The comparison [42] of popular Load Balancing techniques like Agent Based, Prediction Based, Supportive Node Based, Recitation Based, Neighbour Based are shown in Table 2.1.

TABLE 2.1: Comparison of Load Balancing Techniques

LB Technique / Parameters	**Prediction Based Approach**	**Agent Based Approach**	**Supportive Node Approach**	**Recitation (Policy and Strategy) Based**	**Recent Neighbour Approach**
Environment	Heterogeneous (Centralized Load Manager)	Distributed heterogeneous P2P	Distributed Heterogeneous	Centralized and sender Initiate	Distributed Heterogeneous
Aim	Effective resources utilization and reduce average job response time	Self Adaptive, Avoiding large amount of data transfer and find under loaded nodes more quickly	Reduces communication delay using supporting nodes	Reduce response time	Reduce communication delay
Strength	Better resource utilization	Migrate load on the basis of state information exchange and location policy	Minimum traffic due to attached central node at each cluster	Based On Load Balancing Policy	Load index as a decision factor for scheduling tasks
Criteria	Selected task basis of I/O-intensive, CPU-intensive and memory intensive	Load estimates based on the job queue length and job arrival rates	Primary and centralized approach	Information, triggering and Selection Policies	Total Execution Time, Communication Delay, No. of Task and No. of cluster
Compose in Module/-Matrices	Predictor, Selector, Scheduler	Job Scheduler, Communication layer, Resource managers	Provide support while migrating load in individual node pool	Information, Triggering, Selection	Grid,Cluster, Leaf nodes
Behavior	Cluster Based Queuing model	Analyzed by varying the number of jobs in a nest	Performance criteria is load index and queue-length	Based on CPU consumption and Queue Length	Load Index based on CPU Utilization, Queue length and Communication delay
Gap	Need to evaluate in large scale cluster	Number of nodes increases in a nest will cause to increase the complexity of program	Complexity in implementation	Resources submit and withdrawn in any movement (Affect performance)	Performance degradation in nesting of cluster

2.2.4 Load Balancing Algorithms

On the basis of Load Balancing parameters, such as environment, load monitoring, synchronization etc. Different Load Balancing algorithms [17] [18] [36] [42] are compared in Table 2.2.

TABLE 2.2: Comparisons of Load Balancing Algorithms

Algorithms /Parameters	**Round Robin and Randomized Algorithms**	**Local Queue**	**Central Queue**	**Central Manager**	**Threshold**
Environment	Static and Decentralized	Dynamic and Decentralized	Dynamic and Centralized	Static and Centralized	Static and Decentralized
Load Monitoring	Round Robin fashion By Manager	By Local and global Load Manager	By Local and Central Load Manager	Through Central Manager Only	Through Distributed processes
Synchronization	Each processor maintain report of all resource	Randomly send Remote messages by load manager	Communication between Local and Central Load Manager	Inter process communication between Remote Processors	Each Process create private copy of system's load through remote messages
Rebalancing Criteria	Round Robin	Automatic Load Balancing	FIFO	Tree mechanism (parent children balancing)	Using Local and Global remote process
Job Migration	Randomly find underloaded node using round robin fashion	To find out Under loaded node in the ready Queue	To find out the Process in the process Request Queue	While Dynamic Activity created by different host in cluster	After checking the Load Limit of the process in the pool

2.2.4.1 Some more Generic Algorithm used for Load Balancing in Grid Architecture

(i) Min-Min Algorithm: Min-min is a simple algorithm which can be implemented easily. It provides a reduction in make span and maximum resource utilization. For all unmapped tasks, Min-min algorithm first calculates the Expected Time to Computed (ETC) in available machines. The task having the least ETC or Minimum Completion Time (MCT) is chosen first and assigned to the corresponding machine. The newly mapped task is removed from the set of unmapped tasks and the process repeats until all tasks are mapped.

In Min-Max approach [36], the shorter jobs get executed first while the longer jobs need to wait for long and hence many machines may stay idle. In Max-Min heuristic, this drawback has been addressed by making an effective Load Balancing. Here, the longest job gets executed first and hence other shorter job can also be executed parallel in other machines. So resource utilization is performed in an effective way in the Max- Min algorithm.

(ii) Bidding: Bidding [43] is a sender-initiated non-preemptive algorithm. The loaded node (based on threshold crossing) requests bids from neighbours or all nodes (i.e. Through broadcast). The bids (i.e. Current load) are sent by underloaded nodes. The node with the winning bid (i.e. Shortest load) is selected and will receive a transferred job. If no appropriate bids arrive within a time, then extend the request for further bids in the network or process locally.

(iii) Global (Centralized Control): In Global (centralized control), one host is designated as the load information centre, which assembles the load of all the hosts in a load vector and broadcasts the local vector to all the hosts every second. The placement policy is as follows: send new job to the host with the lowest load. If there is more than one host with the lowest value, select one arbitrarily [33].

(iv) Central (Centralized Control): The load information centre acts both as the load information centre and the central scheduler for all the processing elements. Such a distinguished agent requires less overheads making the algorithm more scalable [49].

(v) Reserve: It is a receiver-initiated algorithm based on job reservation. When a job arrives, it is sent to the node that made the most recent reservation. If the load falls below threshold, all reservations are cancelled.

(vi) Global Avg: Each node maintains a value for the network average load and strives to keep its own load within a pre-defined acceptable range. If the load is not within the acceptable range, then it attempts to find a transfer partner by broadcasting its conditions and waits for a reply within a reasonable time. If no complementary partner can be found, it updates the global average load by the amount and broadcasts the new average value of the other nodes otherwise it migrates an advantageous process with a complementary partner.

(vii) Drafting: Drafting is a receiver-initiated preemptive algorithm based on a drafting strategy. Each node maintains a load table of candidate processors from among its neighbours, but instead of using numerical values to describe the load of a node, three states are used: Low state when the node can accept remote processes, Normal state when no transfer in either direction is desirable and Heavy state when the node needs help from other nodes.

(viii) PollGen: PollGen is a preemptive algorithm with an adaptive feature. It is based on the threshold received-initiated algorithm.

(ix) Ant Colony: Ant Colony [44] [45] is based on meta-heuristic technique for grid Load Balancing. It has been used for improving the performance a new adaptive mechanism of intelligent ants. The ants being an intelligent agent can create a new one when they find themselves to be overloaded. Also, they can take decisions based upon the memory allocate to them and the decision making algorithms.

(x) Adaptive Technique: Adaptive technique [46] varies the number of processors during the run time in Grid environments. Adaptive technique balances loads upon the arrival of each job, but also balance loads whenever anomalies appear in the workload of the system or individual nodes. They exhibit more flexibility by adjusting their policy to match the dynamic system characteristics. The main disadvantage of this work is that the processors they do not consider the makespan, computational cost, overhead time.

(xi) Genetic Based Algorithm: The genetic based algorithm is a heterogeneous Load Balancing technique with speedy processors. They broadly classify the jobs into two categories and allocate the first set of jobs to speedy processors and the other set can be allocated to any Genetic resource [47].

(xii) Hybrid Method: The hybrid method [48] combines both the static and dynamic Load Balancing for addressing the problem of resource allocation. They use the metric of update interval for reducing the delay and deadlock. It has reduced the waiting time of the jobs and assigns priority.

(xiii) Load Balancing with Scatter Operations: Load Balancing with Scatter Operations approach describes about the method of balancing the load using scatter operations that run parallel codes on grids. It has been used for finding optimal distribution of data across processors by ordering them with the improvement of performance and cost.

(xiv) Hierarchical Load Balancing: The hierarchical Load Balancing method [49] comes up with the dynamic tree based model for managing the workload using the neighbouring property. It will decrease the amount of exchange messages in the grid environment and thereby lead to the decrease in communication overhead.

(xv) Capacity Based Technique: The Capacity based technique provides a two-level Load Balancing in a multicluster grid environment with each of the clusters located in different LANs. Minimization of the overall response time and maximization of system utilization and throughput has been achieved through the consideration of

the processing element's capacity. It achieves an appropriate load balance, when the workload increases and the system goes to the saturation state.

(xvi) Branch and Bound: Branch and Bound parallelization technique works with the construction of the search tree and exploring them through a depth first search that can be improved parallelization but has a scalability problem.

2.2.5 Load Balancing Policies

The LB policies' [17] issue are made to balance the load (which tasks should be executed remotely and where). The mechanism issue carries out the physical facilities to be used for remote task execution. Figure 2.2, illustrates a suitable decomposition with each leaf representing a distinct component of a load distribution scheme. The emphasis is about the components of the policy and the provision of information to the policy. The policies within the components are interrelated, fixing one would limit the options within the others. Nine main components can be identified below [24].

(i) Local Load Measure: Local Load Measure (load average) at each node is a mechanism that must be provided to give a good estimation of the current local load. There are two important aspects to be considered here: a load metric which has a correlation with the performance objective pursued, and the measuring mechanism that must give a quick and efficient evaluation of the local load state.

(ii) Information Policy: The information policy indicates the significance of various parameters (i.e. disk I/O, CPU queue, memory paging, CPU utilization, network, etc.) for the accuracy of evaluating the load status of a host. The information gathering process can be started periodically or independently of significant changes of system states. This component is responsible for the exchange and maintenance of the information about other individual or groups of nodes such as load level, nature of workload or the average load over the entire system. It is also responsible for the frequency of the state information update, the ways to exchange this information among the various nodes, the numbers of nodes involved in the exchange, and the amount of information made available to the decision makers. It must maintain consistent information about the global state at the distributed points of control

(iii) Transfer Policy: Transfer Policy [41] decides to transfer a process from the local workload and selects an overloaded node heuristically based on the local information. This component decides when it is beneficial to transfer a process from the local workload and selects which process to transfer/migrate. The overloaded

node chooses heuristically an advantageous process (for example; long lived process, availability of specific resources at specific nodes) to transfer, based on the local information, the remote information maintained locally, or acquired during the negotiation with other nodes. The transfer policy is responsible for requesting the transfer of work from other nodes when the local node is about to become idle. The transfer policy is the minimum component needed to implement a Load Balancing strategy

Sender/Receiver/Symmetrically/Random initiated [42] [50] balancing is the part of transfer policy. Transfer Load Balancing commonly used to start the Load Balancing activity, the time a new job arrives or is created on a node and the time a finished job departs from a node. This aspect mainly deals with allocation of nodes to jobs.

- Sender-Initiated, load distributing activity is initiated by an overloaded node (sender) trying to send a task to an under load node (receiver). It is convenient to remote invocation strategies.
- In Receiver-Initiated Algorithms, load distributing activity is initiated from an under load node (receiver), which tries to get a task from an overloaded node (sender).
- In Symmetrically Initiated Algorithms, combine the benefits of both senders and receivers to search for appropriate location.
- A Random Policy chooses the destination node randomly from all nodes in Grid system.

(iv) Location Policy: The duty of location policy is to find a suitable transfer partner in Grid environment.

(v) Negotiation Policy: In negotiation policy, Once a node has been decided that it is a suitable to transfer the load, it sets engaged in a pairing process. Once a node has decided that it is a suitable transfer client (overloaded i.e. that is wishing to get rid of some of its load) or it is a potential transfer server (lightly loaded or idle i.e. that is looking for work), it engages in a pairing process. This process consists of a search for a transfer partner, a node in a complementary state.

2.2.6 Load Balancing Attributes

The attributes of a LB Algorithm are described below [24] [41] [42] [51] [52].

(i) Load Redistribution: Different objectives can be tracked when performing a load redistribution within a Grid. The load sharing is used when the objective is to

keep all the nodes busy but also to attempt an equalization of the load over all the nodes. Restricted shares, when the nodes are privately owned and their sharing is allowed only with the approval of the owner.

(ii) Decision-making Structure: The LB algorithms can be distinguished based on the decision-making Structure used to implement the different components. This structure can be centralized, hierarchical, decentralized, or a hybrid form.

(iii) Transparency: Transparency, where the implementation of the Load Balancing scheme can be made transparent to users. This can be achieved by providing a system interface that automatically identifies the jobs eligible for transfer.

(iv) Autonomy:In Autonomy, a Load Balancing algorithm that has an autonomy attributes does not violate the control of the job allocation at individual nodes.

(v) Scalability: In Scalability, as the number of nodes in the distributed system raises and the range of workload fluctuations increases, scalability problems can arise.

(vi) Adaptability: In Adaptability, dynamic factors such as system load, network traffic, and the availability of computing nodes which characterize a distributed system, have a direct effect on the system performance. It is used to maintain the global scheduling scheme tuned to the variations in the environment, even when the system conditions change drastically.

2.2.7 Load Balancing Mechanisms

A Load Balancing mechanism is used to harness the computational power of the grid. Such mechanism attempts to balance the load with the results of maximum resource utilization and optimal performance. Two mechanisms commanly used are [17]:

(i) Load Metric Mechanism: It checks the type of information that makes a load index (queue CPU length, memory size) and the way such information is communicated to other loaders (broadcasting, focused addressing, polling).

(ii) Load Communication Mechanism: Load Communication Mechanism identifies the method by which information, such as the load on a resource, is communicated between the resource and the load distribution policy and mechanisms.

2.2.8 Load Balancing Performance Metrics

The performance impact of LB can be measured in many key areas like CPU Overhead, Response Time [53], Throughput, Switch Occupancy etc, as described in [42].

2.2.9 Load Balancing Strategies

Load Balancing strategies, try to distribute the workload uniformly across all computers in a grid. Load Balancing can be done without measuring the current load to avoid the overhead and temporary balancing. Many load-balancing strategies dynamically react to load imbalances by comparing a load metric to a threshold and transferring workloads to other computers if the threshold is exceeded. Other Load Balancing strategies use workload priorities and characteristics of the workload to do Load Balancing Collecting workload characteristics in advance can decrease the mean response time of batches of requests. It can be classified into many categories like Hierarchical, De-clustering, AI technologies, Layered Strategy, Scheduling strategy etc [18].

2.2.10 Load Balancing Phases

Dynamic LB can be divided into five phases i.e. Load Evaluation, Profitability Determination, Work Transfer Vector Calculation, Task Selection and Task Migration discussed in [17].

In a distributed computer system, some of jobs are distributed among the various nodes in the system. If the distribution of a job among the node can vary with time, then it calls this job a 'migratable' job. Redistribution of the user jobs and the migratable jobs among the nodes in the system is referred to as Job Migration. JM includes the migration of a task while it is still being executed.

2.3 Job Migration Taxonomy

Load Balancing is the main area of concern in Grid environments, whereas Job Migration is responsible to handle problems while balancing the load. Job Migration is an important aspect of executing a parallel program in the Grid. The basic steps of JM [50] are depicted in Figure 2.3

The JM mechanism can be useful in several scenarios as discussed below: [54]

- JM can solve the problem of Load Balancing. When a node in a pool becomes overloaded, the whole application can be migrated to the underloaded node.
- During JM, Grid Broker always works on priority basis. Migration can provide high throughput while Load Balancing, where unused/less used cycles of underloaded machines are collected.

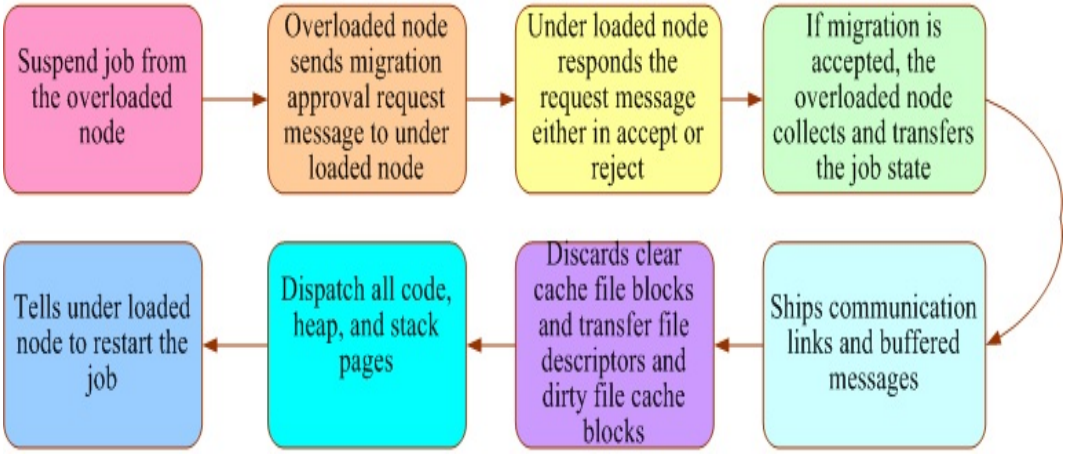


FIGURE 2.3: Job Migration Steps

- Migration can be used to provide fault-tolerance [55] capability using checkpointing [56], replication and scheduling.

Job Migration can be done if the processing performance of the underloaded node is higher than the overloaded node. A hierarchical taxonomy for JM in Grid environment has been presented in Figure 2.4.

2.3.1 Job Migration Algorithms

(i) Local Job Migration Algorithm: Each server has prior information about the amount of load to be transferred to its neighbouring servers.

(ii) Global Job Migration Algorithm: The Global Job Migration algorithm is an optimization algorithm that minimizes the amount of load for migration during the LB process [48].

(iii) Neighbour Job Migration Algorithm: The neighbour Job Migration algorithm is based on improving the fault tolerance with better resource utilization of Grid by using Load Balancing strategies along with the checkpointing to migrate a task from one node to another and resuming its execution from the point of failure [48] [42].

(iv) Heat Diffusion Migration Algorithm:In heat diffusion Migration algorithm, heat diffuses from a region of high temperature to low temperature over time. The diffusion follows the gradient of the temperature field and heat flows smoothly across regions of different temperature [57].

(v) Hybrid Job Migration Algorithm: The hybrid Job Migration algorithm works with limitations of local (less effective) and global (less efficient) Job Migration algorithms. It has reduced the average number of overloaded servers [55].

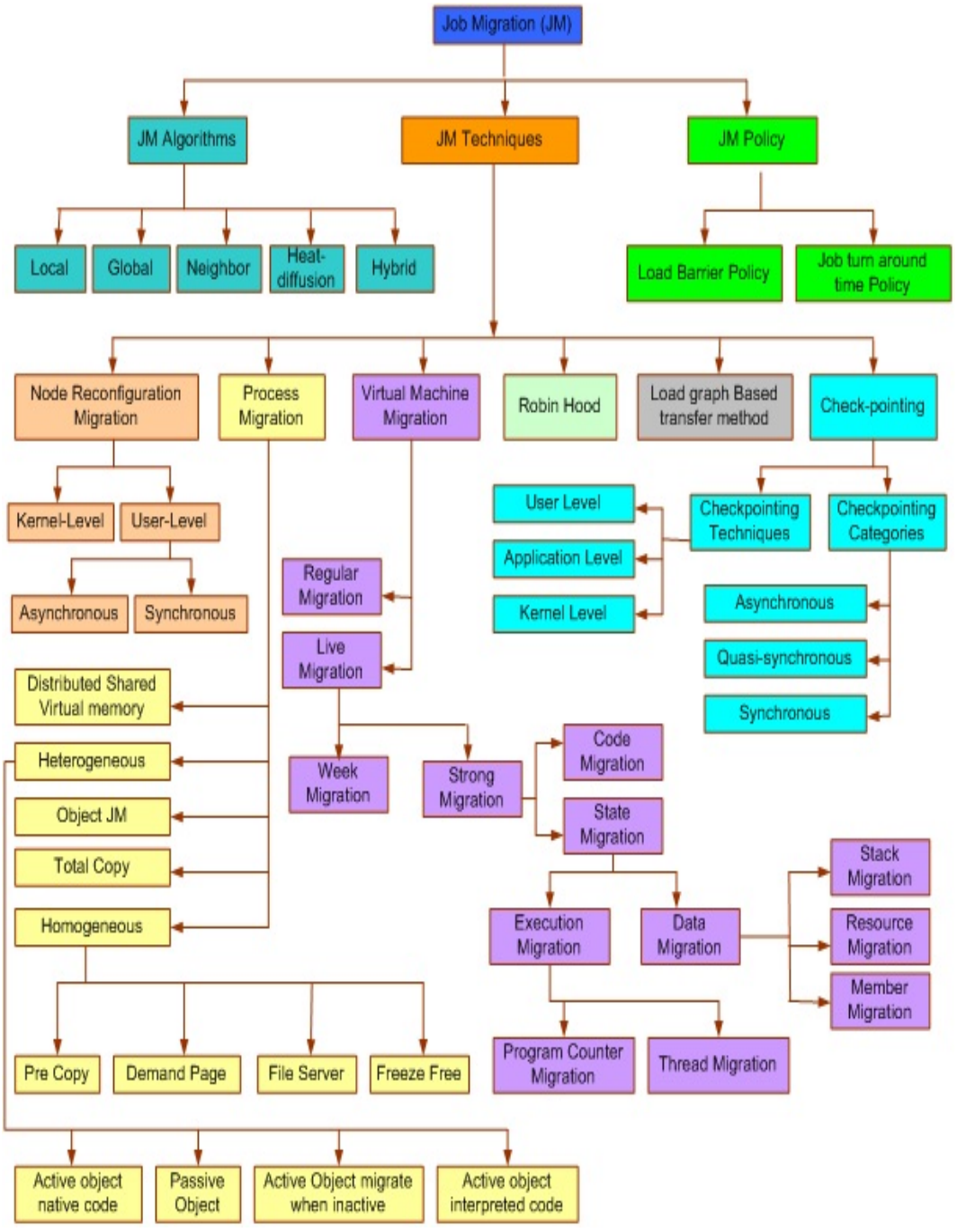


FIGURE 2.4: Job Migration Taxonomy

2.3.2 Job Migration Techniques

The Job Migration processes as studied here are homogeneous, heterogeneous, object based and distributed shared memory [18].

(I) Node Reconfiguration Migration [58]: This mechanism makes application workloads migrate from source node to a destination node and then allows a source node depart from the original computing environment. There are two mechanisms for this, node reconfiguration by user-level thread migration and node reconfiguration by kernel level thread migration.

  (a) Kernel Level Job Migration: Kernel level Job Migration techniques modify the operating system kernel to make JM easier and more efficient [59].

  (b) User-level Job Migration: User-level JM techniques support JM without changing the operating system kernel. Node reconfiguration by user level thread migration has been implemented in two methods:

    (i) In synchronous method, all nodes are paused during reconfiguration. Synchronous method may make performance down even though it is easier to design.

    (ii) In asynchronous method all nodes continue to work simultaneously with reconfiguration and better performance can be obtained as long as more attention is paid to correctly maintain the order of node reconfiguration messages.

(II) Process Migration:

  (a) Distributed Shared Virtual Memory (DSVM) transfers process data and state while migration.

  (b) Heterogeneous Process Migration is JM across machine architectures and operating systems. There are four basic types of heterogeneous migration i.e. Active object, native code,Passive object, Active object, migrate when inactive and Active object, interpreted code as shown in [60] .

  (c) Object Job Migration is possible on the persistent object basis. In these techniques, persistent objects move between machines, depending on usage or placement statistics.

  (d) The Total Copy Algorithm is popular in JM algorithm, in this algorithm all relevant job information including all virtual memory is transmitted to the new host during migration.

(e) Homogeneous Job Migration involves migrating jobs in a homogeneous environment where all systems have the same architecture and operating system, but not necessarily the same resources or capabilities. Homogeneous Job Migration Algorithms are five basic algorithms for homogeneous Job Migration.

(i) The Pre-Copy Algorithm optimizes the transfer of the virtual address space to the new node by performing it in parallel with the continued execution of the migrating jobs at the old node. The Pre-Copy algorithm ships all code, heap, and stack pages to the new node.

(ii) Demand Page Algorithm is similar to the Total Copy algorithm except that no virtual memory pages are transferred at migration time. When the job is restarted at the new node, it will page fault immediately. The job then requests the desired page from the old node. The new jobs will likely page fault three times very early in its execution; one page fault for the current code, stack, and heap pages respectively. Future page faults are satisfied by requesting the pages from the old node.

(iii) File Server Algorithm added a third machine to the Job Migration with old and new nodes. It can be done using the efficiency of demand paging without having a residual dependency on the old node.

(iv) Freeze Free Algorithm improves the latency time. Achieving this, speed up, eliminates the approval request message. Instead, the old node can assume the new node will accept the Job Migration and send the program counter and execution state of the new node. This saves waiting time, which is a costly round trip message.

(III) Virtual Machine (VM) Migration: In VM Migration [46] snapshots of machine are sent to another machine, that's why it is called the VM migration. There are two methods for VM migration. First one is live migration "Regular Migration the job is generally stopped and then migrated" and Live Migration "Running domain between the different host machines is migrated without stopping the job. In between, it stops the job and gathers all required data, and then resumes. But this happens only in same layer network and IP subnet".

The live migration minimizes the downtime, i.e., the time when no CPU cycle is devoted to any of the VM-resident applications, neither at the overloaded nor at the underloaded node. Live Migration can be bifurcated into strong or weak migration.

(a) In weak migration, both code and strong state transfer cannot be achieved together.

(b) In Strong migration [57] technique not only source code of the application migration, but also strong state migration, execution state and date information (such as the program counter and stack) should be migrated. Strong migration can be classified into different sub-types: code migration and state migration, that is described in [60].

(i) Code Migration means that all the source code could be migrated from the overloaded to the underloaded node.

(ii) State Migration contains the information of execution state and data that are being used. Therefore, state migration consists of two aspects: Execution Migration and Data Migration.

(IV) Robin Hood: The Robin Hood [35] technique is non- centralized Load Balancing mechanism, It multicasts channels to communicate, and synchronize the processors and proactive tools to migrate jobs between them. It is useful in the proactive library for the migration of jobs, a multicast channel for node coordination and provides the mobility and security in a uniform framework.

(V) Load Graph Based Transfer Method [35]: It is supported on a network graph where each node is represented with its load, whereas load can be the number of users, average queue length or the memory utilization. It uses an analytic model and single load determination policy throughout the system and the load is determined on the basis of memory utilization and average queue length. This algorithm is based on three- layered structure. Top layer is the Load Balancing layer which takes care of token generation, taking decision about task transfer; the middle one is called a monitoring layer and acts as an interface between the top and middle and monitors load changes and the third one is the communication layer which takes care of the actual task transfer.

(VI) Checkpointing: A checkpoint [23] is normal processing is interrupted specifically to preserve the status information, necessary to allow resumption of processing at a later time in the program. By periodically invoking the checkpointing process, one can save the status of a program at regular intervals. If there is a failure, one may restart the computation from the last checkpoint, thereby avoiding repeating the computation from the beginning. The process of resuming computation by rolling back to a saved state is called rollback recovery. There are three types of check pointing implementations: kernel-level, user-level and application-level. These implementations differ in levels of transparency, efficiency and mechanism used to initiate checkpoint and restart.

- In kernel level check pointing, the user does not have to change the application at all, so it is least efficient, because the system does not have the knowledge about the application.
- Creator achieves user level check pointing, and he puts or implements some set of procedures that handles check pointing and restarts. The creator knows all about the application, that's why this approach is more efficient.
- The developer itself achieves application-level check pointing. This approach is efficient, because the creator has detailed knowledge about the application.

Checkpoint [28] Algorithms can be classified into three broad categories: asynchronous, synchronous and Quasi-Synchronous or Communication Induced Checkpointing [23] [18].

- In asynchronous check pointing, each process take checkpoints independently. If all the processes take checkpoints at the same time instant, the set of checkpoints would be consistent.
- In the synchronous checkpointing process, synchronizes through message passing interface before taking checkpoints [61].
- Quasi-Synchronous or Communication Induced Checkpointing avoids the domino-effect without requiring all checkpoints to be coordinated. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forcing checkpoints are taken to guarantee the eventual progress of the recovery line and to minimize useless checkpoints.

2.3.3 Job Migration Policy

Two popular JM policies are the load barrier policy and the minimal job turnaround time policy [49] [62].

(i) Load Barrier Policy:The load barrier policy adopts the CPU load as the barrier of Job Migration. The mechanism starts to check the job queue for finding idle jobs [63]. If any idle job is found, the Job Migration mechanism would try to find a remote site and migrates this idle job to this remote site.

(ii) Minimal Job Turnaround Time Policy: The minimal job turnaround time policy extends the load barrier policy by taking into account the resource heterogeneity, network bandwidth, job transfer cost, CPU capability and job queue length. In this policy, the job response time and transmission cost is recorded.

2.3.4 Existing Job Migration Algorithms

Three Job Migration Algorithms MELISA, LBA and PIA address several issues that are imperative to Grid environments such as handling communication latency, resource heterogeneity and sharing [64].

(i) Modified ELISA Algorithm- In MLISA, the job will be transferred only if its expected finish time on destination node is less than expected finish time on source node. In ELISA, Load Balancing is carried out based on queue lengths. Whenever there is a difference in queue length, jobs will be migrated to the lightly loaded processor, ignoring the Job Migration cost. This cost becomes an important factor when the communication latency is very large such as for a Grid environment and/or the job size is large. For large-scale Grid systems (i.e. interGrid [65]), the Modified ELISA (MELISA) is shown to have better control in balancing the loads, in which we consider the Job Migration cost, resource heterogeneity and network heterogeneity when Load Balancing is considered [66]. In MELISA algorithm, each processor calculates its status parameters, which are the estimated arrival rate, service rate, and load at every status exchange period. This information gets exchanged to every buddy processor in the buddy set.

(ii) The Load Balancing on Arrival (LBA) Algorithm-In LBA, applicable to small-scale systems (that is, intraGrid), performs Load Balancing by estimating the expected finish time of a job on buddy processors on each job arrival.

(iii) Perfect Information Algorithm (PIA)-In PIA, each processor has up to-date information about all buddy processors requires plenty of message transmission [13].

In this section, comparative study of different Job Migration Algorithms has been done on the basis of parameters like arrival rate, service rate, load distribution, resource utilization etc.The comparative study depicts that no single algorithm is efficient enough to achieve all parameters as shown in Table 2.3.

TABLE 2.3: Comparisons of JM Algorithms Basis on Average Response Time [13]

Parameters	**MLISA**	**LBA**	**PIA**
ART execution of more jobs (N > 10)	High(N=128)	Moderate(N=26)	Low
ART comparison for random arrival and service rates	18% Better	12.8% Better	Low
Effect of the status exchange period on ART	Good	Average	Poor
Total execution time comparison for random arrival and service rates	[In between 1000-5000 jobs]	[In between 1000-5000 jobs]	[In between 1000-5000 jobs]
ART comparison for an uneven load distribution	High(approx 50k jobs)	Moderate	Low
Effect of job size on ART	High	Moderate	Low
Resource utilization comparison for an uneven load distribution	Wide	Average	Very little or Low
Arrival rate, service state and system state	Estimated Information	Estimated Information	Perfect Information
Load Unevenly Distributed	44.9% better then ELISA	25% better then ELISA	Average

After discussing the Job Migration algorithms, techniques and survey, the next part of the chapter cover presents an extensive and comparative survey of the existing Load Balancing and Job Migration techniques.

2.4 Job Migration and Load Balancing Survey

A survey of the existing Load Balancing and Job Migration technique used in grid computing since its inception until 2014 and it has been discussed in Table 2.4 and Table 2.5. Their research focus, contribution, compared model, strength, gap, future work and their suitability for usage in a dynamic grid environment has been analyzed. Above survey has been conducted on the basis of different parameters and is useful to find the appropriate technique in a different environment for optimization.

TABLE 2.4: Survey of Job Migration Techniques

Authors	**Techniques**	**Future Work**	**Existing Technique Suffers**	**Strength**
T. Altameem [2013] [67]	Fault Tolerance Techniques	Turnaround time, throughput, fail tendency and grid load	Checkpointing	Enhance grid throughput, utilization, response time and more economic profits. Replication and Check pointing are the major techniques used
Prasenjit Kumar Patra et al. [2013] [68]	Fault Tolerance Techniques	Not Given	Not Given	Comparisons of Fault tolerance technique
Shuangquan Yang [2012] [69]	Live migration scheduling	Not Given	Cost	Migrate jobs with minimum cost
Susanne Albers et al. [2012] [70]	On the Value of Job Migration in Online	Not Given	Limited number of job reassignment	Makespan Minimization
Zafer altug sayar et al. [2012] [71]	Strong Migration technique	Flexibility of the algorithm		Constraint Satisfaction Problem, LB, inter-messaging rates
Tariq Alwada'n et al. [2012] [72]	Heterogeneity, diversity of policies and attributes	Not Given	Limited CPU resources	User-submitted policy, data policy and multiple Virtual Organizations (VOs).
Said Fathy El-Zoghdy [2012] [73]	Load Balancing and job Migration algorithm	Not Given	Scheduling and Load Balancing	Introduce Random and uniform two-level Load Balancing policy and minimize the overall job mean response time and maximize the system utilization and throughput
Vasinee Siripoonya et al. [2011] [74]	Thread-Based Live Checkpointing (TLC)	Multiprocessor VMs Implementation, automatically switch TLC operation when high memory update locality is detected	Long delay and long disruption of services since they have to stop virtual machines to save state, which could be large	Save virtual machine state for recovery and reduced the check pointing overheads
Federico Calzolari et al. [2011] [75]	Job Migration Algorithm		Computing resource exploitation	Improve exploitation in terms of running jobs, and increase the computing efficiency, by reducing the free job slots

DaeWon Lee [2010] [76]	IP-Paging base Resource Management and Task Migration in Mobile Grid Environments	Challenges-Disconnected operation, job scheduling, device heterogeneity, and security	Mobility management, disconnected operation, device heterogeneity, service discovery, and resource sharing	User-defined checkpoint technique decreases total execution time and total job completion time.
Shakti Mishra et al. [2010] [77]	Priority and criticality based Optimized Scheduling Algorithm for Migration	Not Given	Availability, Reliability and Performance	Increased Throughput, Reduced latency time along with no starvation policy for any process.
David Cuesta et al. [2010] [78]	Adaptive Task Migration Policies	Not Given	Not Given	Low and balanced temperatures and performance overhead
John Mehnert-Spahn et al. [2010] [79]	Checkpointing and Migration of Communication Channels in Grid	Implementation optimizations to improve scalability and channel flushing	Single node checkpoints	It offers transparent migration and supports recreation of sockets, shared by multiple threads of one or more processes
Remi Busseuil et al. [2010] [80]	Close-communication and forwarding-communication migration	Strategies of a dynamic selection of the best suited migration protocol	Adaptability	Scalability, global performance.
A. Satheesh et al. [2008] [81]	CPU-based load sharing policies	Use of distributed operating system	Homogeneous networks and preemptive migrations	Usage of global memory in addition to CPU load sharing, reduce the number of page faults caused by unbalanced memory allocations for jobs
Zhirou Zhang et al. [2007] [82]	Path Optimization Algorithm	Not Given	Optimization in migration	Bi-pheromone Ant Colony Optimization algorithm âĂŞ network Load, service node‘s ability and load are considered
Ruchir Shah et al. [2007] [64]	MELISA Job Migration algorithms	Extended to add Fault tolerance technique	For large-scale systems	Work on job size, data transfer rate, status exchange period, and migration limit
Vipin Chaudhary et al.[2006] [83]	Mig-thread migrating technique	Not Given	Client/server and distributed computing systems	Clean off the obstacles for grid computing

Abdullah Azfar et al. [2005] [84]	Ubiquitous migration technique	Ubiquitous Computing	Individual computing resources	Checkpointing mechanism utilizes and centralize different Interrelated computing resources.
HwaMin Lee et al. [[2005]] [85]	A resource management and fault tolerance services	Work on the Globus toolkit for measuring efficiency	Fault tolerance service	Resource manager for optimal resource selection
Sathish S. Vadhiyar et al. [2003] [86]	Oriented Migration Framework	Rescheduling threshold dynamically	Simple Policies	Improve the response times for individual applications
Sriram Satish Tadepalli [2003] [87]	Fault tolerance	Priority based scheduling algorithms	Checkpointing	Identify the role of local resource managers in the fault detection and migration of Grid applications
Po-Cheng Chen et al.[2003] [88]	Virtual Machine Migration vs. Thread Migration for Grid Systems	Grid resource utilization	Non-dedication and dynamic behaviour	The performance difference between these two evaluated mechanisms
T. N. Ellahi et al. [2002] [89]	DGET Middleware	Migration at any arbitrary code location	Making the migration more fine-grained.	Supported multi-threaded migration and asynchronous migration. Supported transparent strong migration in grid environments. Least possible space and time overhead.
P. Emerald Chung et al. [1997] [90]	User-level process migration environment	Includes checkpointing script applications, transparently migrating inter-process communications for message-passing applications, and integrating the migration techniques	Checkpoint-restart techniques	To utilize the CPU power and balance load on all machines in an environment; To provide fault tolerance by migrating a process from a failed machine to another machine.

TABLE 2.5: Survey of Load Balancing Techniques

Authors	**Technique/ Approaches**	**Compared Model**	**Gap/ Future work**	**Research focus/Contribution/Features**
Azin & Simone et al. [2011] [91]	State broadcast and two random Approaches	Ant colony Approach	Scalability, resource failure and robustness	Using an ant colony optimization approach. It invokes a response to submit a job to the Grid, and surf the network to find the best resource to deliver the job. Using Particle swarm optimization algorithm, Each node plays a role as a particle and moves toward other particles by sharing its workload between them and optimizing load locally by sending or receiving jobs from its neighbors.
Qingqi et al. [2011] [92]	Agent based distributed optimized scheduling algorithm with partial information	Heuristic Algorithms and static random agent distribution algorithm	Migration cost, Security and agent scheduling for fault tolerance	Integrates the distributed mode, approximate optimization and agent set scheduling approach to achieve short execution time in large-scale agent scheduling. It describes the multi-level framework, agent distribution, communication architecture, event-driven conservative time synchronization mechanism to avoid potential failure problem caused by the centralized mode as well as improve execution efficiency through parallel computing.
Sotirious et al.[2011] [93]	Cluster-based web systems&RR, Least Connections	Least Loaded algorithm.	Incorporation of prediction metrics	Provide features like scalability, adaptive fully utilization, clustering prediction, bandwidth and web farm utilization estimation and sharing mechanisms among classes of HTTP traffic is introduced. Its uses for Non-content aware Load Balancing algorithm.
Hemant et al. [2011] [94]	Dynamic distributed Scheduling	Not Given	E-Learning & data mining applications	Usage strategy like service based approach to data access, integration & resource registration and discovery applicable in large distributed computing environment. This strategy improves the overall performance of the existing Distributed computing environment (DCEs).
PENG et al.[2011] [49]	Proximity-aware LB approach	Not given	Not given	Load Balancing is performed independently and ensures fair load distribution with reduced overhead. It provides rapid convergence on load balance and reduces the load transfer cost in the heterogeneous and dynamic network state.
Robson et al.[2011] [95]	Local and cluster monitoring mechanisms	Not given	Better detection and reactivity	Monitor resources, load reallocation, and load migration. Minimizes balancing overhead while load redistribution. Balancing involves normalizing load distribution of shared resources and maximizing overall resource utilization. Migration technique performs reliable and low latency load transfers.
Deepti et al.[2011] [96]	Cluster & non cluster based LB Algorithm	GA, Fuzzy, DDLB, Adaptive	Centralized Main server controller	Its ability to trace dead machines, framework works well with heterogeneous web servers where the incoming load is high and the response is given within few seconds without any bottleneck.

Hung-Chang et al.[2011] [97]	Hierarchical based LB algorithm	Centralized directory approach	Movement cost of virtual servers and protocol message overhead	Structured P2P networks either explicitly construct auxiliary networks to manipulate global information or implicitly demand the P2P substrates organized in a hierarchical fashion. Its uses for Load Balance with imperfect information in Structured Peer-to-Peer systems.
T.R.V. Anandharajan et al.[2011] [98]	Dynamic distributed Scheduling	Not Given	Boundary value approach	Combines the inherent efficiency of the centralized approach, energy efficient and the fault-tolerant nature of the distributed environment. Used for job scheduling and job selection algorithm.
Jasma et al. [2010] [99]	Decentralized Recent Neighbour (RN) LBA	Desirability aware Load Balancing	Nesting of clusters & network bandwidth	RN performs intra-cluster and intercluster (grid) Load Balancing. It is feasible and improves the system performance considerably, provides shorter response time, enhances resource utilization and availability of tasks. This technique uses sender initiates strategy in the algorithm.
Yuan Rao et al.[2010] [100]	Agent based	Not Given	Computational and storage complexities	Achieves better Load Balancing, decrease packet loss ratio efficiency, guarantee better throughput and end-to-end delay bound in case of high traffic load. It has lower on-board computation, storage, signalling requirements than other on-board routing schemes.
Safa Khalouli et al.[2010] [101]	NP Hard Problem	Heuristics on Dispatching Rules	Multi-criterion Scheduling	Earliness tardiness penalties are addressed to complete each job close to due date using just in time approach.
Leandro dos et al. [2010] [44]	Ant colony	A Greedy algorithm & Ant Colony Optimization	Different chaotic systems	Synchronization of identical discrete chaotic systems and optimization of proportion integral derivative.
M.H. Afshar et al. [2010] [102]	To effectively find the optimal solution & found insensitive	Asynchronous parallel implementation	Includes many constraints like size, velocity	Nodal elevations of the network are considered as decision variables Incremental solution building capacity, is used for Ant Colony optimization.
Al-Dahoud et al. [2010] [103]	Ant colony based	Work-stealing for LBA in distributed systems	Not Given	Raising the rate of information exchange all over the nodes in the system. Improved efficiency in terms of the number of busy nodes and the elapsed time. It uses for Ant Colony Optimization (ACO).

Liang Bai et al. [2010] [104]	Multiple ant colony optimization approach	FCFS and ACS approach	Ant Colony	To minimize the execution time of tasks as well as workload across all nodes and achieve task scheduling with Load Balancing.
Husna Jamal et al. [2010] [105]	Agent Based	Not Given	Not Given	This facilitates in scheduling jobs to available resources efficiently, which will be enable jobs to be processed in minimum time and also balance all the resource in the grid system.
Antony Lidya Therasa.S et al.[2010] [106]	Dynamic Adaptation of Checkpoints based	Not Given	Not Given	Achieves fault tolerance by dynamically adapting the checkpoint frequency, which reduces checkpoint overhead and increases the throughput in case of resource failure. Fault Index Based Rescheduling (FIBR) algorithm effectively reschedules the job from failed resource to any other available resource with fewer fault index value.
Biagio Cosenza et al. [2010] [107]	Agent Based	Not Given	Extend the technique to multidimensional space	It achieves a high scalability and low communication overhead. Calculations for the balancing take place on every worker at each computational step and influences the successive step. Each communication is local (i.e., between neighbor workers), due to the local worker interaction, is dominated by the speedup achieved better Load Balancing.
Magdy Saeb et al.[2010] [108]	Cluster Based	Client-Server paradigm	Prototype for large scale	Study the impact of the cluster size, the agent lifetime on the algorithm and the variance of the workload over the cluster. It introduces a packet format for the information agent, the location agent, and the routing agent that provides better average response time and requires relatively short time to implement.
Ghada F et al.[2010] [109]	Direct Acyclic Graph (DAG) based	Not Given	Not Given	Application independent, dynamic algorithm for scheduling tasks and Load Balancing in message passing systems.Its uses to reduce response time of the newly arriving jobs while maintaining the time constrains of the existing DAG.
P. K. Suri et al.[2010] [110]	Intra and inter cluster (grid) LB	Not Given	Partition of task into subtasks	Efficient utilization of resources and enhancing the performance of computational grids. It improves the system performance, reduce the execution time and cost, enhances the resource utilization and balance load in an effective manner.
Weiwei lin et al.[2010] [111]	Centralized & distributed hierarchical method	Master-Slave Model in parallel Network	Work in simulated environment only	To schedule a large number of parallel tasks and it can make full use of the Internet resources to solve the high-performance computing problems.

Somayeh Abdi et al.[2010] [112]	Job Scheduling policy and Data Replication Mechanism	Not Given	High latencies of Wide Area Networks and Internet	To achieve good network bandwidth utilization, get better performances and reduce data access time. Uses for Hierarchical Replication Strategy (HRS).
Malarvizhi Nandagopal et al.[2010] [113]	Parallel and Distributed Computing	Not Given	No precedence constraint among different tasks of a job	It addresses the problem of Load Balancing using Min-Load and Min-Cost policies while scheduling jobs to the resources in the multi-cluster environment. Heuristically estimate the completion time of executing jobs in remote clusters, improves system performance in terms of mean response time and average slowdown. Fault tolerant measures increase the reliability of the algorithm. Uses for Hierarchical Status Information Exchange Scheduling.
Malarvizhi Nandagopal et al.[2010] [114]	Sender Initiated Decentralized Dynamic Load Balancing	Non Migration (NM) algorithm	Real world grid computing environment	Highly decentralized, distributed and scalable algorithm for scheduling jobs and balance the load across the resources in heterogeneous computational grid environment is to minimize response time and reduce the communication overhead.
Gengbin Zheng et al.[2010] [115]	Periodic and hierarchical LB Approach	Centralized schemes	Extend hierarchical LBA	It overcomes the scalability challenges and considerably reduces the memory requirements, the running time and provide interconnect topology aware strategies that map the communication graph on the processor topology to minimize network contention.
Bin Wang et al.[2010] [116]	P2P System (Distributed Hash Table)	Not Given	Not Given	Even node distribution is achieved in key space according to top-down ID allocation. In heterogeneous system, the algorithm performs well in Load Balancing for applications with even distribution of keys and queries.
Robson E. De Grande et al.[2010] [117]	High Level Architecture (HLA) based	Not Given	Improved technique for detecting communicative federates	Using local and cluster monitoring mechanisms identify imbalances, improves the use of shared resources, increases distributed simulation performance by minimizing communication latencies and partitioning the load evenly. The hierarchical architecture minimizes the overhead produced by the balancing system and overcome the heterogeneity issues.
Xiao Qin et al.[2010] [118]	Based on parallel applications in clusters	Existing memory-aware and I/O-aware schemes	Develop a prototype of the COM-aware load balancer	It is capable of improving the performance of communication-intensive applications by increasing the effective utilization of networks in cluster environments. Dynamic communication-aware Load Balancing scheme (COM-aware) for non dedicated clusters where the resources of the cluster are shared among multiple parallel applications being executed concurrently.

Shoukat Ali et al.[2010] [119]	Genetic Based	Across the Machine load Balance	Not Given	It is allocating Robust Resource and enhances the tolerance of power grid against fluctuations in the load.
Chao-Chin Wu et al.[2010] [120]	Heterogeneity of fault-tolerance mechanisms	Min-Min and suffering algorithm	Not Given	Support four kinds fault-tolerance mechanisms, including the job retry, the Job Migration without checkpointing, the Job Migration with checkpointing, and the job replication mechanisms.
P. V. Prasad et al.[2010] [121]	Network reconfiguration based distribution system	Not Given	Not Given	Ability to solve the radial distribution system reconfiguration problem for Load Balancing. It eliminates infeasible solutions, optimal configuration and improves the voltage profile of the network.
Wenmin et al.[2010] [122]	Load-balancing dynamic scheduling algorithm	Bottleneck Dynamics (BD) algorithm	Not Given	Priority rule regarding makespan and due date, priority rule regarding processing time and machine unused time segment, and the rule that all machines will be added unused time segment if there is no matching one.
Jie Chang et al.[2010] [123]	Polling scheduling dynamic Load Balancing algorithm	Conventional static policy framework	Not Given	Avoids some shortages of the polling scheduling algorithm. The value of the threshold is based on the principle of prediction and feedback which significantly enhancing the working efficiency.
Youngjoon et al.[2010] [124]	Guided self scheduling for data centric Load Balancing	Locality-aware work-stealing scheduler	Not Given	To achieve high performance on amorphous data parallel Programs that address additional factors beyond runtime overhead and load balance.
Jaehwan Lee et al.[2010] [125]	Based on PUSH and PULL JM mechanisms (Decentralized)	Centralized scheduler	Accommodate asymmetric multiprocessors with General Purpose Graphics Processing Units	Local scheduling uses a variant of backfilling to leverage residual resources in a single machine. Internode scheduling extends the local scheduling across machines by migrating jobs when that will allow a job to run immediately. Queue balancing proactively balances job waiting queues across machines to avoid highly skewed queues, to overlap Job Migration with other system activities and Improves total system throughput. The algorithms also avoid starvation of large jobs through the backfilling counter mechanism.
Shakti Mishra et al.[2010] [126]	Prioritizing processes based	Not Given	Latency time for Process Migration Server	Incorporating an optimized Load Balancing strategy in a trusted cluster.

Mohsen Moradi et al.[2010] [127]	Time Optimizing Probabilistic LB Approach	Not Given	Not Given	Chooses the resources that have better past and least completion time. It is establishing Load Balancing with reducing the response time and task failure percentage.
Dongliang Zhang et al.[2009] [128]	Based on BST (Adaptive balancing)	Artificial and a real distributed	Limitation in adaptability on large networks	This method can adjust the sub-domains with heavy loads and decompose their loads faster with a lower communication overhead. Loads are balanced from a global view, and sub-domains are stretched and compressed in the group.
Yong Hee Kim et al. [2009] [129]	Multi-Agent System (MAS) based	Not Given	Overhead of load data collection	This scheme significantly reduces the response time by migrating some agents to balance the loads of the system. Architecture effectively supports the registration, modification, and deletion that provides easy retrieval and provisioning of registered services for the users like web service.
Chang Hui et al.[2009] [130]	Deferred shading parallel rendering applications	Coarse or CPU bounded algorithms	Not Given	It determines the performance and scalability of sort-first rendering clusters. Which can distribute rendering load more uniformly among the rendering node and improve the system render performance prominently.
Azzedine Boukerche et al.[2009] [131]	DLB for HLA-Based	Not Given	Communication overhead	It minimizes the communication, can lead to performance improvement.
Abhinav Bhatele et al.[2009] [132]	Dynamic Topology Aware LBA	Other topology aware LBA	Hierarchical & Heuristic-based LBA	Load Balancing in parallel MD programs is crucial for good performance on large parallel machines and used to improve scalability and performance.
Xiao Qin et al.[2009] [133]	DLB for I/O-Intensive Applications on clusters based	Existing non-I/O-aware load-balancing including CPU and Memory aware schemes	Not Given	Effective I/O-aware load-balancing schemes for two types of clusters: (1) homogeneous clusters where nodes are identical and (2) heterogeneous clusters, which are comprised of a variety of nodes with different performance Characteristics.
P Visalakshi et al.[2009] [134]	Hybrid Particle Swarm Optimization based	Particle Swarm Optimization, GA	Non-preemptive task scheduling	This algorithm solving the Task Assignment Problem (TAP) np-hard problem.

Yajun Li et al.[2009] [135]	Averages and instantaneous measures based hybrid LB	Static or Dynamic Algorithms	Not Given	FCFS with GA algorithms perform Load Balancing of sequential tasks with achieving minimum execution time, reduce the system response time, maximum node utilization and a well-balanced load across all the nodes involved in grid computing environments.
Thein et al.[2008] [136]	Decentralized dynamic Load Balancing	Not Given	Not Given	Providing faster communication time and minimize execution time. It is improving the performance of a cluster computer system with excellent scalability & reliability.
Minglong et al.[2008] [137]	Fault Tolerance based scheduling	REM	Loss of data	Provide Load Balancing & fault tolerance hierarchically. Introduce the state balancing policy in large scale distributed Multimedia service.
Zhengping Qian et al.[2008] [138]	Distributed multiprocessor cluster based	Static job scheduling algorithms	Cost of process migration	Exibits Deadlock Detection & resolving algorithm and based on dynamic recorded events of each task at runtime.
K. Saruladha et al.[2007] [139]	Agent Based	Dynamically varying number of nodes in a nest	Scalability	Improves the response time and overall execution time of user submitted jobs. It explores and find the under load nodes more quickly.
Ana et al.[2000] [140]	Nearest neighbor approach	Sender Initiate diffusion	No estimating cost of data	Provides even load distribution considering loads indivisible units. Facilitating load movements b/w non-directed connected processors, and algorithm globally converges. Uses for Diffusion algorithm Searching unbalanced domains.
Chingjung et al.[1999] [141]	Mapping/load-balancing based cluster approach	Comparisons between all three methods	Not Given	Use three trees-based parallel load-balancing methods, maximum cost spanning tree (MCSTLB), BTLB and Condensed Binary Tree (CBTLB), balance the computational load of each processor of solution-adaptive finite element application programs with speedup performance.
Mohammed et al.[1997] [142]	Compile-time and run-time modelling	Receiver-initiate DLB	Not Given	Automatically transform an annotated sequential program to a parallel program in a dynamic environment with less complexity.

The survey depicted that Load Balancing and Job Migration challenges in Grid computing have not been addressed to a large extent and their are many issues that need to be addressed. After discussing the survey of Load Balancing and Job Migration, the next section will focus on adoption of Load Balancing and Job Migration over the years in graphical view.

2.5 Adoption Graph of Load Balancing and Job Migration Techniques

On the basis of the survey, a meta analysis of temporal trends in publication of Load Balancing and Job Migration techniques in Grid have been described in Figure 2.5 and Figure 2.6. Below comparison easily shows the usage probability of techniques according to the year wise fashion.

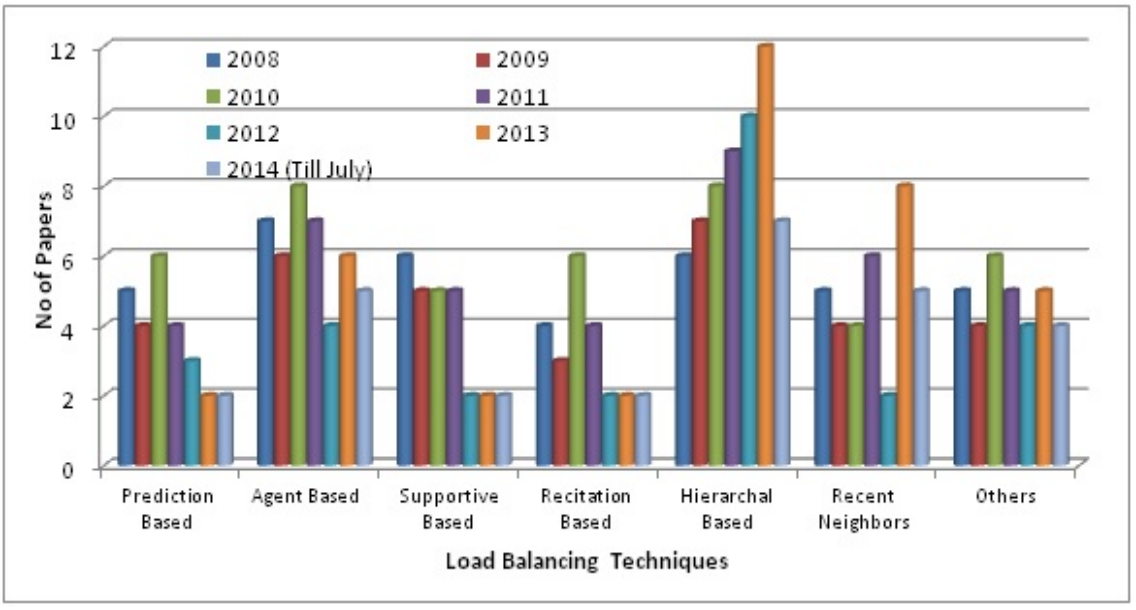


FIGURE 2.5: Adoption of Load Balancing Techniques

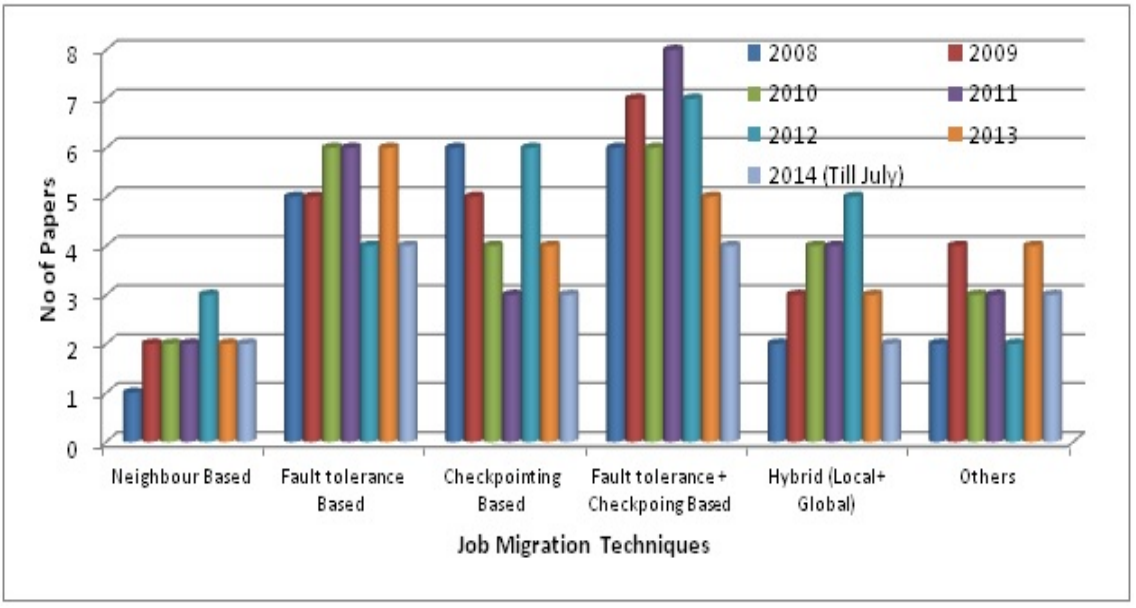


FIGURE 2.6: Adoption of Job Migration Techniques

2.6 Problem Formulation

Grid Computing encompasses various features which make Load Balancing in Grid challenging to accomplish. These challenges further pose significant obstacles for designing an efficient and effective Load Balancing technique for Grid environments. Many Load Balancing systems have been proposed and reported, as discussed in literature review but these systems have several limitations like heterogeneity, adaptability etc. and issues like applicability problem in dynamic and decentralized environment, lack of scalability, optimization of performance and cost etc. yet need to be addressed for optimal utilization of Grids. Further the Job Migration technique used in Load Balancing should be capable to decide that it should make changes of load distribution during execution of job or not. Most of the existing algorithms address Load Balancing technique by using round robin, FIFO etc. After an exhaustive literature review, it is apparent that there is a need of innovative Load Balancing technique to enable grid computing to be real platform delivering high performance services. If Load Balancing algorithm is complemented with efficient Job Migration technique then a Load Balancing algorithm has flexibility to migrate the jobs to nodes pragmatically. Existing dynamic decentralized Load Balancing systems in computational grids tend to improve the overall utilization and performance by developing effective load-balancing strategies but lack in scalability, information gathering overhead etc. which needs to be addressed. Further the Job Migration techniques in these systems are not efficient in cost/time optimization, fault tolerance and bandwidth for transfer of data into large scale locations which needs to be effectively addressed. Therefore this work focuses on the design & development of an efficient dynamic decentralized algorithm catering to challenges of Load Balancing along with an efficient Job Migration technique proposed in Grid environment.

The main objectives of the proposed work are:

i. To analyze the existing Load Balancing and Job Migration techniques in grid environments.

ii. To propose and design a dynamic, decentralized Load Balancing algorithm for grids.

iii. To design and develop an efficient Job Migration technique for the above designed Load Balancing algorithm.

iv. To test the proposed Load Balancing algorithm and Job Migration technique in a grid Environment.

2.7 Conclusion

This chapter presents a detailed survey of Load Balancing and Job Migration techniques based on various parameters: compared model, strength, gap, future work, techniques, research focus. The different techniques that are available in the literature have been appropriately classified under different categories. As, Job Migration also plays an important role in achieving effective fault tolerance, sufficient light has been thrown on those techniques. The algorithm, description, merits and demerits of each technique have been analyzed and presented. Next chapter presents the proposed design of Load Balancing algorithm.

Chapter 3

Proposed Load Balancing Technique

The previous chapter discusses the classification and comparison of the different Load Balancing and Job Migration schemes. It also includes taxonomy to facilitate a better understanding of Load Balancing & Job Migration in Grid environment.A Survey of Dynamic Load Balancing and Job Migration mechanisms have been presented and compared along with the existing techniques.

In the existing work, a dynamic, decentralized Load Balancing technique is proposed that considers all the factors pertaining to the characteristics of the Grid computing environment. A dynamic threshold value is used at each level and the value of the threshold is dynamically changing according to the Grid size in the network. A well-designed information exchange scheduling scheme is adopted in the proposed technique to enhance the efficiency of the Load Balancing model. A comparative analysis between the existing technique and proposed exhibits why the proposed technique is better than other existing algorithms. The proposed technique is a new version of the LB mechanism based on random policy. The design of the system environment, LB process for random job arrival through Poisson process has been developed and the flowchart of proposed Hierarchical Load Balancing technique along with fitness function has been presented in this chapter.

This chapter firstly analyses the proposed and existing approaches based on various parameters like: categorizes, techniques, comparisons, scheduling, policy, response time etc. In the next section the system model and the LB process have been designed for the proposed Load Balancing algorithms. Then, the problem description, requirements,job arrival process of Local Grid Manager, flowchart, fitness functions have been presented for the proposed model. Finally, Load Balancing algorithms have been proposed for the model in the Grid environment.

3.1 Preliminaries

Qureshi and Rehman [143] had documented Grid architecture for Load Balancing, as illustrated in Figure 3.1. They have used, Poisson process for random job arrival with a random computation length. Considering that the jobs are sequenced, mutually independent with the arrival rate and can be executed on any site. Furthermore, the site should meet the jobs demand for the computing resource and the amount of data transmitted. Each processor can only execute one job at a time and execution of a job cannot be interrupted or moved to another process during current execution. This model has been divided into three levels: Level-0 Broker, Level-1 Resource, and Level-2 Machine Level.

When a new job known as Gridlet arrives on a machine, it may go to underlightly-loaded, lightly-loaded, overloaded and normal loaded resources by load calculation being computed at each node. In order to compute the mean job response time analysis one Grid Broker (GB) section as a simplified Grid model has been considered. Grid Broker is the top manager of a Grid environment. It is liable for sustaining the overall Grid activities of scheduling and rescheduling. It acquires the information of the work load from Grid resources and sends the tasks to resources for optimization of load. Resource that comes next to Grid Broker in the hierarchy, are connected through internet. The resource manager is responsible to maintain the scheduling and Load Balancing of its machines and it also sends an event to the Grid broker during overload. The machine is a Processing Entity (PE) manager, responsible for task scheduling and Load Balancing of its PE's connected with various resources via LAN. PE manager also sends an event to resource during overload. PE's next to machines are mainly responsible for calculating workload and threshold values for Load Balancing, Job Migration and passes the load information upwards to machines via buses. Gridlet is considered as a load and assigned to any of the PE's according to their capability (i.e. Computational speed, queue length, etc.).

3.2 System Model and Problem Description

The work, proposed here, is an enhancement of the work [143] discussed above.

It suggests the necessity of quantification of the load in order to achieve Load Balancing in computational grid. Quantification of the load is done and the objective function is derived based on the load distribution of the computational nodes. Response time and resource allocation have been recorded as a fair contribution of this research. Furthermore, it extends the existing technique into two cases as discussed in Table 3.1.

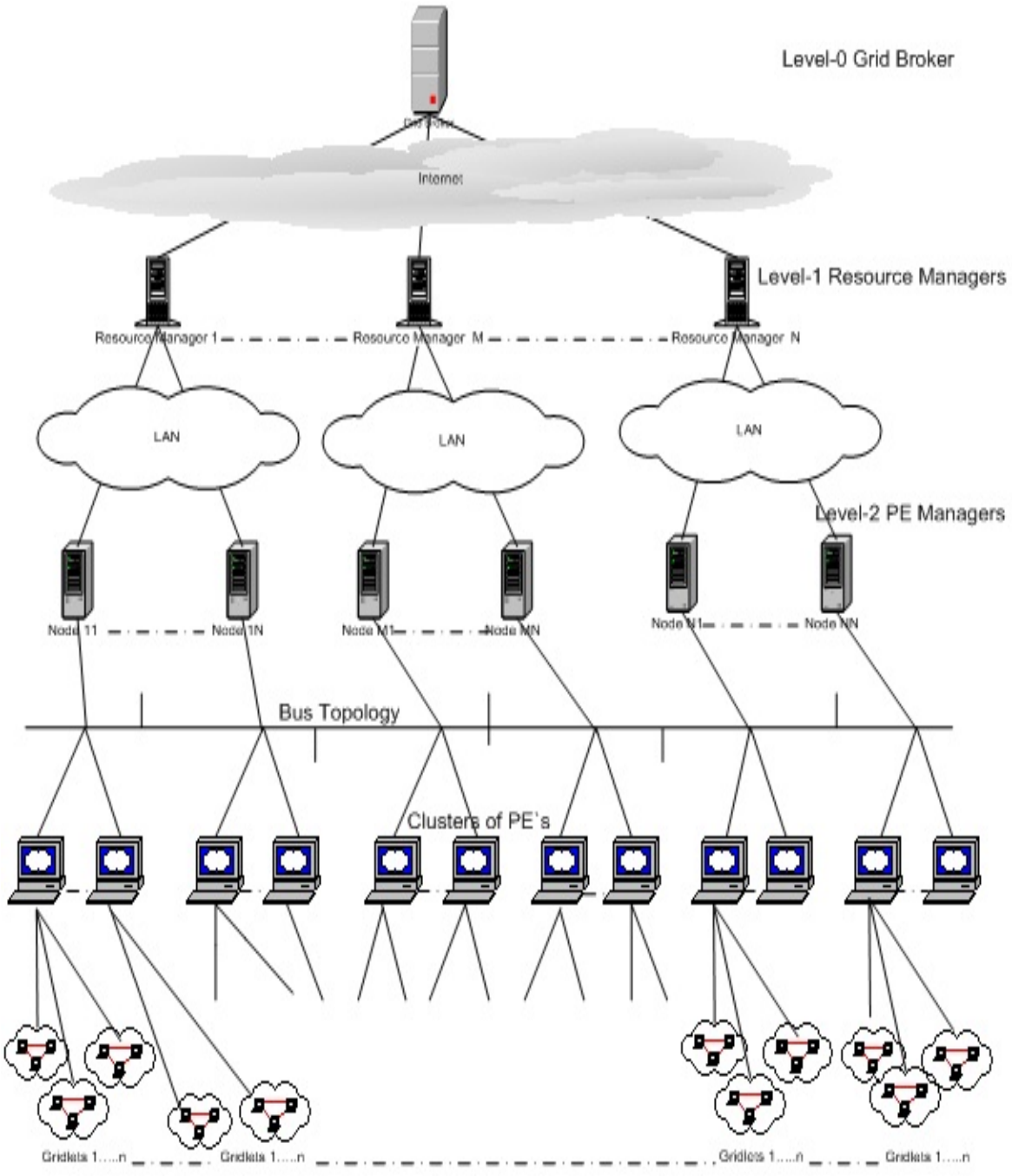


FIGURE 3.1: Hierarchical Load Balancing Model

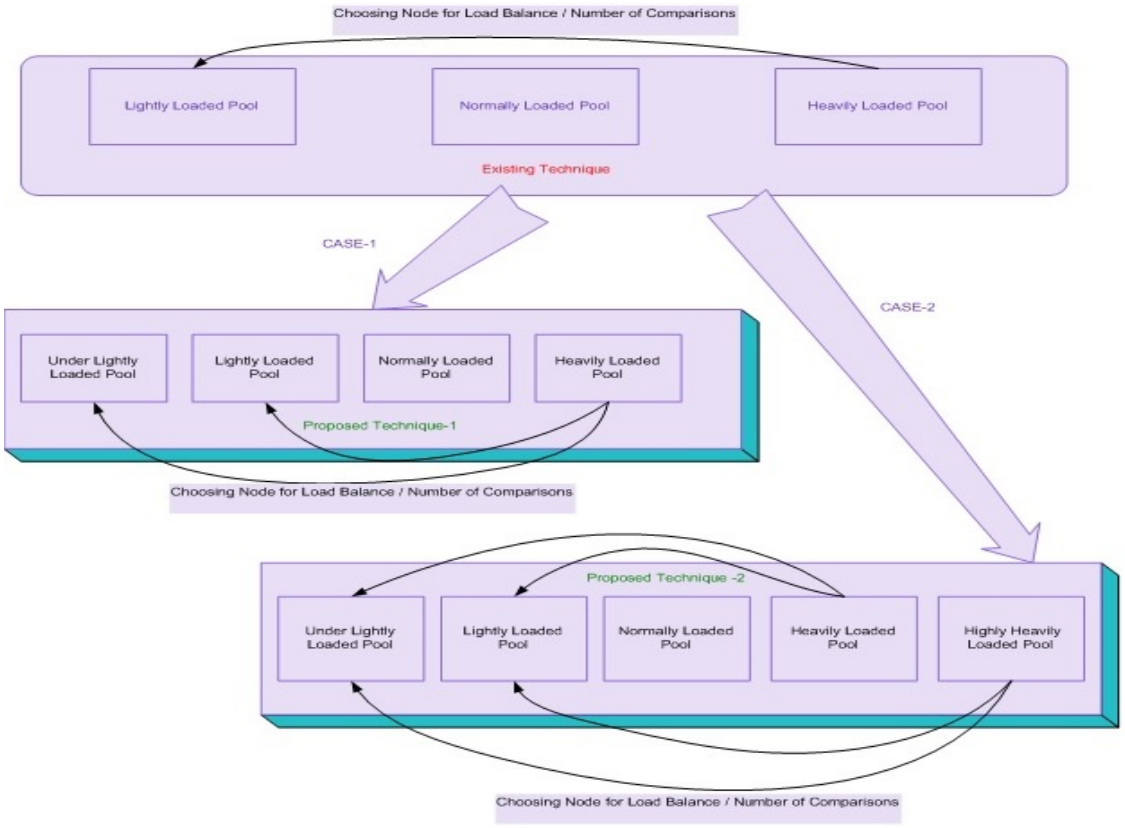


FIGURE 3.2: Basic Comparison between Three Techniques

TABLE 3.1: Comparison between Three Techniques

Techniques	**Existing Technique**	**Proposed Technique (Case-1)**	**Proposed Technique (Case-2)**
Categories	Divided into three pools like: lightly, normally, heavily	Divided into four pools like: under lightly, lightly, normally, heavily	Divided Into Five pools Like: under lightly, lightly, normally, heavily, Highly heavily
Techniques	Each heavily loaded node has to check all the nodes in the Lightly loaded pool	According to load percentage overloaded node is checked into one pool either lightly or under lightly loaded pool on the basis of sorting	Checking has been done in all four pools
Comparisons	1	2	4 (Exponential increase)
Scheduling	No	Yes	Yes
Sorting	No	Yes Ascending -Under lightly and Lightly loaded pool Descending -Heavily Loaded pool	Yes
Policy	Not Applicable	Random: means process can be executed vice-versa also. (Lightly loaded node can also search heavily loaded node)	Any
Response Time	Low	High (Know about the status of each pool & sorted all nodes in order)	Medium
Throughput	Low	High (High Availability)	Medium

Resource Allocation Capacity	Low	High (Due to idea of load persentage)	Medium
Cost	Low	Medium	High
Time	High (Each node has explored in the pool)	Low (Due to Sorting Technique)	Medium (More comparisons)

Two cases for the proposed technique have been considered. In first case, lightly loaded node is divided into lightly and under lightly loaded categories in the context of variable threshold and standard deviation. Thus, the nodes are divided into four pools. This technique minimizes the searching time to find out the receiver machine in order to transfer the Gridlets. A well-designed information exchange scheduling scheme is adopted in the proposed technique to enhance the efficiency of the Load Balancing model.

In the second case, load is divided lightly loaded into lightly and under lightly loaded categories and heavily loaded node into heavily and highly heavily loaded categories in the context of variable threshold and standard deviation. Thus, here the nodes are divided into five pools. This technique minimizes the searching time (as compared to the existing technique) but increases the ambiguity in the form of comparison that can increase exponentially.

To find an appropriate node for migration is complex task that can enhance the cost and storage capacity as compared to the existing techniques. Therefore, first case is better than existing and second case in all perspectives. Thus, the first case has been implemented in this thesis work. A comparative analysis depicts the difference between all the techniques and exhibits why the proposed algorithm is better than any other existing algorithms presented in Figure 3.2.

The proposed technique is thus the enhanced version of the Load Balancing mechanism using random policy. The proposed random LB policy chooses the target endpoint randomly from the specified list. The proposed Load Balancing technique has not only increased resource allocation efficiency of Grid resources, but also cuts down the response time of the entire Grid. This algorithm is rigorously examined on the GridSim simulator to judge the effectiveness of the algorithm, especially on Grid platform. The detailed steps and architecture of the proposed technique are explained in the subsequent sections.

3.2.1 Load Balancing Process

Grid mapping has been shown as sequence diagram in Figure 3.3. The interaction of each site by computing nodes in a network is given through the following steps:

(i) In the network, each site may contain multiple machines and each machine may have a single or multiple PEs. The Grid scheduler is a software component that runs on each site. It manages the system related information (i.e. CPU utilization, remaining CPU capability, remaining memory, computing nodes, sites etc.)in order to join the Grid. It provides resources, receives jobs from Grid clients and assigns them to the processors in the Grid system.

(ii) It provides information back to the jobs and discovers the candidate set of the nearest site based on the job requirements, resource characteristics. It creates a list of these nearest nodes & sends this list to the Load Balancing decision maker.

(iii) The decision maker decides on the basis of the minimum Grid site, whether the job can be executed at the local site or remote site and transfers the job accordingly.

(iv) The job dispatcher ships the jobs, checks the site availability through fault detector and monitors the state of sites.

(v) If site failures or system degradation occurs, the faults are detected or a failure message is sent to the fault manager. After the fault manager receives a failure message, it reschedules the job using a primary backup approach.

In this system, sites play the role of assisting the execution of jobs. When any site is in idle state and can provide its resource, a join message and related hardware information will be transmitted to the Grid scheduler of the nearest neighbour site. When it can no longer provide its resource, it will transmit an exit message to the Grid scheduler of the nearest neighbour site. The Grid scheduler uses a derived threshold to effectively select suitable sites. This threshold value is based on the load and demands for resource needed to execute the job and will evaluate the sites in the system. Thus providing a dynamic Load Balancing method over static Load Balancing. In static Load Balancing state, when a job has to be executed, it will be assigned to an appropriate site. The minimum requirement of job's needs for sites is considered as the threshold for the table of effective sites. Sites passing this threshold are called effective sites. As a grid is composed of sites with different performances, the execution time of a job on each site is different.

In dynamic Load Balancing state, the system will adjust dynamically until it reaches a balance. In a grid environment, the effectiveness of sites varies with time. Thus, the

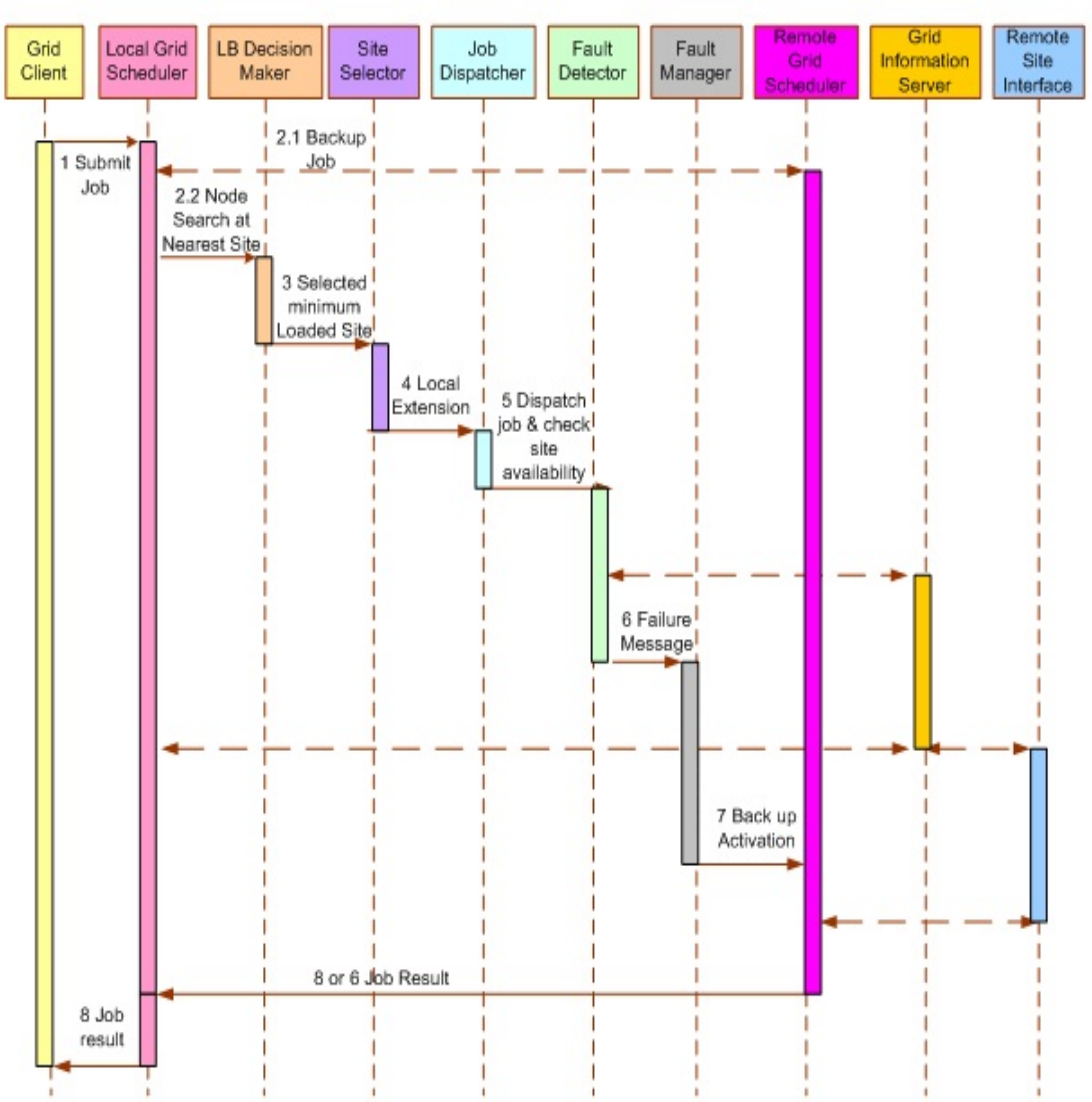


FIGURE 3.3: Sequence Diagram of Components and their interaction

assignment of jobs must be adjusted dynamically in accordance with the variation of the site status. The flowchart of the proposed Load Balancing technique has been shown in Figure3.4.

3.2.2 Job Arrival Process at Local Grid Manager

To clearly understand the model analytically, a Local grid Manager (LGM) section has been considered as a simplified grid model, in which the external tasks arrive sequentially at the LGM according to a time-invariant Poisson process [55], with inter-arrival times

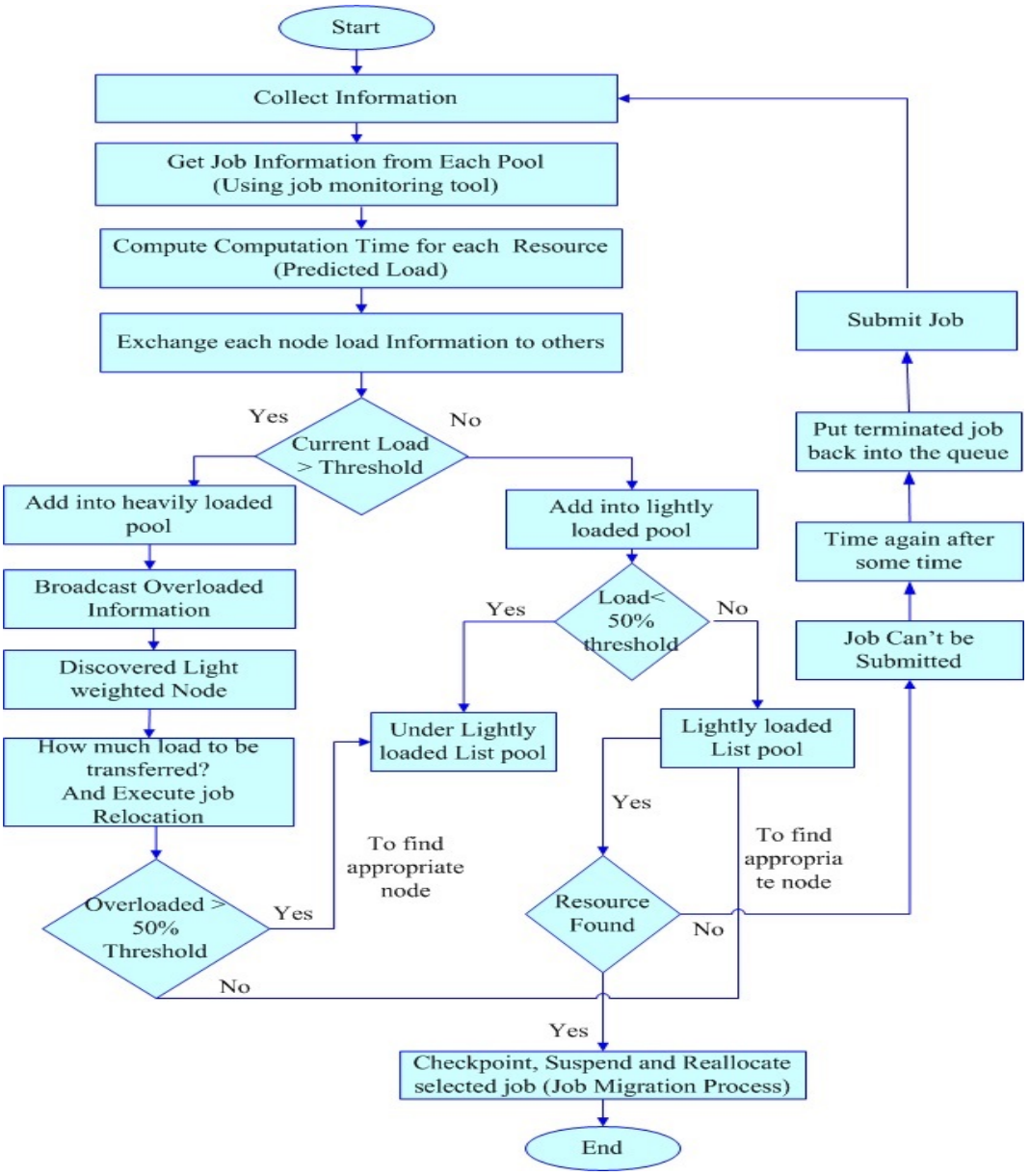


FIGURE 3.4: Flowchart of Proposed Hierarchical Load Balancing Technique

which are independently, identically, and exponentially distributed. Considering the following system parameters:

- λ is external (remote) task arrival rate from grid clients and other LGMs to a LGM out of which, the rate λ_a is accepted for processing by the LGM, and the rate λ_r is rejected up arrival and forwarded upon arrival through the communication means to another LGM.
  Hence, $\lambda = \lambda_a + \lambda_r$

- ${\lambda_i}^a$ is the external task flow rate from the LGM to the i^{th} SM which is managed by that LGM. ${\lambda_i}^L$ and is the local task arrival rate to the i^{th} SM.

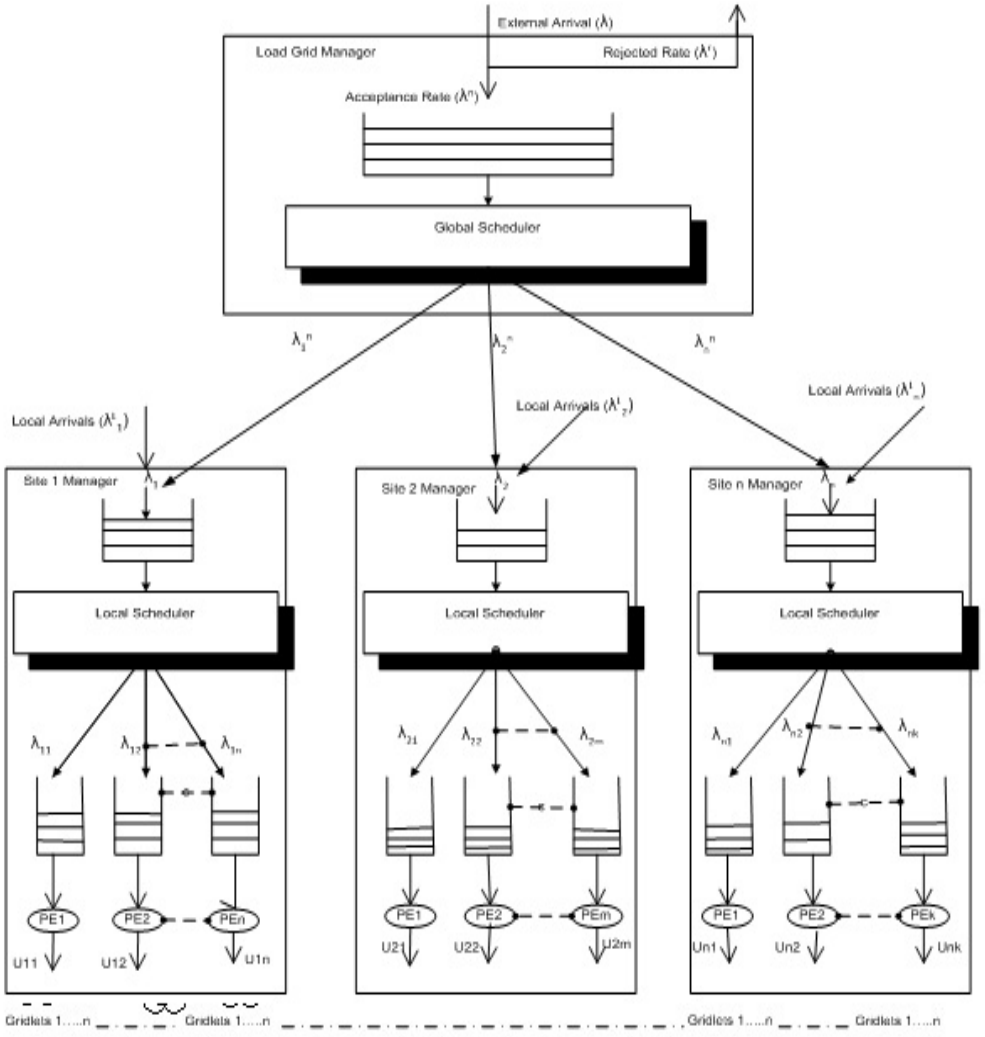


FIGURE 3.5: Queuing Model of Grid Manager

- λ^L is the total local tasks arrival rate to the Site Manager (SMs) managed by a LGM, hence $\lambda^L = \sum_{i=1}^{m} \lambda_i^L$ and i=1,2,3...m is the number of sites managed by that LGM.
- $\lambda_i$is the total arrival rate (external and local) to the i^{th} SM that is $\lambda_i = \lambda_i^a + \lambda_i^L$
- λ_{ij} is the job flow rate from the i^{th} SM to the j^{th} PE managed by that SM.
- μ is the LGM service rate.
- μ_i is service rate of the i^{th} SM.
- μ_{ij} is the service rate of the j^{th} PE which is managed by i^{th} SM. Hence, the LGM traffic intensity is given by:

$$\rho = \frac{\lambda + \lambda^L}{\mu} = \frac{\lambda + \sum_{i=1}^{m} \lambda_i^L}{\sum_{i=1}^{m} \mu_i} = \frac{\lambda + \sum_{i=1}^{m} \lambda_i^L}{\sum_{i=1}^{m} \sum_{j=1}^{n} \mu_{ij}} \tag{3.1}$$

Where, i=1...m is the SMs number, and j=1...n is the number of PEs in every SM. For the LGM to be stable ρ must be less than 1. The i^{th} SM traffic intensity is given by:

$\rho = \frac{\lambda_i}{\mu_i} = \frac{\lambda_i^a + \lambda_i^L}{\sum_{j=1}^{n} \mu_{ij}}$

Where, j=1...n is the number of PEs at the i^{th} SM. For the SM to be stable ρ_i must be less than 1. The traffic intensity of the j^{th} PE which is managed by i^{th} SM $\rho_{ij} = \frac{\lambda_{ij}}{\mu_{ij}}$, similarly ρ_{ij} must be less than one for that PE to be stable. During the analysis process, the following facts are used:

- Merging of k Poisson streams with mean rate λ_i results in a Poisson Stream with mean rate λ given $\lambda = \sum_{i=1}^{k} \lambda_i$
- If a Poisson stream is split into k substreams such that the probability of a task going to the i^{th} substream is p_i, each substream is Poisson with a mean rate of $p_i\lambda$.

Since the communication links between the LGM and its SMs are heterogeneous i.e., every communication link has an individual bandwidth and latency, the G (λ_a) is used as a general function for the overall mean expected communication (including queuing) time of forwarding external accepted task flow rate λ_a from the LGM to its SMs. Refer to the length of time between the instant when a task arrives at the LGM and the instant when it leaves, after all processing and communication, if any, are over as task response time. Hence, the overall mean task response time T can be computed as follows:

$$\left.\begin{array}{ll} T = G(\lambda_a) + T_S & for\ external\ tasks \\ T = T_S & for\ local\ tasks \end{array}\right\} \tag{3.2}$$

Where, T_S is the overall mean task response time all over the sites managed by the LGM. As mentioned earlier, the system is studied in the steady state that is the traffic intensity is less than one i.e., $\rho < 1$. To compute, the SM is modelled as a variant M/M/n queue where the service rates of any two PE are not identical which is similar to the case of heterogeneous multiprocessor system. Figure 3.5 shows a LGM Queuing Model, the queuing structure of a site is a part of this figure.

Define the state of the SM by (k_1, k_2, k_n), where k_i is the number of tasks at the i^{th} PE queue, i =1,2,...n. Reorder the PE's in a list such that the occupation ratio $\frac{k_1}{\mu_{i1}} \frac{k_2}{\mu_{i2}}, ., \frac{k_n}{\mu_{in}}$. The LS at the SM services the arriving tasks (local or remote) based on their arrival time (FCFS). It distributes tasks on the fastest PE (i.e., the PE has the lowest occupation ratio). The traffic intensity of ith SM is given by, $\rho = \frac{\lambda_i}{\mu_i} = \frac{\lambda_i^a + \lambda_i^L}{\sum_{j=1}^{n} \mu_{ij}}$, n is the number of PEs in that site.

The mean number of tasks $E[N]^i_{SM}$at the i_{th} site is then given by:
Here,

$$\left.\begin{aligned} X &= \frac{(1+2\rho_i)\Pi_{j=1}^{n}\mu_{ij}}{\lambda_i \sum_{j=1}^{n}(\lambda_i+\mu_{ij})} + \frac{1}{1-\rho_i} \\ E[N]^i_{SM} &= \frac{1}{X(1-\rho_i)^2} \end{aligned}\right\} \tag{3.3}$$

To compute the expected mean task response time:
Let $E[T]^i_{SM}$ denotes the mean time spent by a task at the i_{th} SM to the arrival rate λ_i and $E[N]^i_{SM}$ denotes the mean number of tasks at that site. Hence, $E[T]^i_{SM}$ can be computed as follows:

$E[T]^i_{SM} = \frac{1}{\lambda}(E[N]^i_{SM})$
So, the overall mean task response time T_S all over the sites at a LGM is then given by:

$$T_S = \frac{1}{m}\sum_{i=1}^{m} E[T]^i_{SM} \tag{3.4}$$

Where, m is the number of sites managed by that LGM. By substituting about the value of T_s in Equation 3.4, the overall mean task (external and local) response time can be computed.

Next section presents the internal calculation to determine the load on the node and cluster, after job arrival in the hierarchical Load Balancing model.

3.2.3 The Fitness Function

The fitness function used to calculate load at PE, machine & broker level are described in this section.

3.2.3.1 Machine level Calculation

When the scheduler receives a job submitted by a user, it will transfer a request to the information service in order to obtain the necessary information such as the idle PE percentage of each resource, average load of each cluster and average load of the system. Then, the scheduler chooses a cluster which has the fastest Average Computing Power (ACP).
Firstly, the Average Computing Power of the cluster $ACP(PE_i)$ is defined as:

$$ACP(PE_i) = \sum_{k=1}^{n} PE\_Speed_k * \frac{1-PE_k}{n} \tag{3.5}$$

TABLE 3.2: Stored Information on Grid Server

S.No.	Notations	Description
1	PE	Processing Element (Single CPU); Processing capability of each PE is same for a single machine
2	G	Number of Gridlets (Applications/Task) on any PE
3	P	Number of PEs on any Machine
4	PELoad	Load on any PE
5	PEsLoad	Total load of available PE in any Machine
6	OPEList	List of PE with overloaded Gridlets
7	OP	Size of OPEList (O: Overloaded and P: PE)
8	LPEList	List of PE with Lightly loaded Gridlets
9	LP	Size of LPEList (L: Lightly Loaded and P: PE)
10	ULPEList	List of PE with under lightly loaded Gridlets
11	UP	Size of ULPEList (U: Underlightly loaded and P: PE)
12	NPEList	List of PE with normal task
13	NP	Size of NPEList (N: Normal-loaded and P: PE)
14	$(PELoad)_D$	Load of Processing element for lightly loaded PE list
15	$(TaskLoad)_L$	Load of Gridlets
16	$(PELoad)_E$	Load of Processing element for Under lightly loaded PE list
17	Rating	Actual capacity of PE i.e. CPU speed in terms of MIPS rating
18	$(Curr\_Load_{PE})$	Current load of PE
19	$(TaskLoad)_L$	Load of Gridlets
20	File_size	Is a workload in bytes
21	Gridlets	An arrival task as a load
22	ALC_i	Average load on each cluster (collection of PE's)
23	ALM_i	Average machine load
24	σ	Standard deviation

Where, PE_Speed_k = Processing speed of processing entity k in cluster i (MIPS)
PE_k = Current CPU utilization of the PEs in cluster i, expressed as a percentage
PE= Processing Entity (Single CPU- Processing capability of each PE is same for a single machine) and
n = number of processing entity in cluster i.

After that machine scheduler selects the PE which has the fastest ACP. The average load of the cluster is defined by the average load of each processing entity in cluster i.
The current load of PE $Curr\_Load_{PE}$ is defined as:

$$Curr\_Load_{PE} = \sum_{1}^{g} Filesize \tag{3.6}$$

Where, g = Number of Gridlets (Application/Task) on any PE.

The average Load (Avg_load) of each PE is defined as:

$$Avg\_Load = \sum_{1}^{p} \frac{Curr_Load_p}{P} \tag{3.7}$$

Where, p=number of PE in cluster (Total load of available PE in any Machine)

To measure the load of each PE $PELoad_{k,i}$, as

$$PELoad_{k,i} = \frac{Curr\_Load_{PE} - Avg\_load}{Rating} \tag{3.8}$$

Where, $PELoad_{k,i}$ = load of each processing entity (PE)k in the cluster i
Rating=Actual capacity of PE i.e. CPU speed in terms of MIPS rating.

Calculate average load of each cluster i ALC_i is defined as:

$$ALC_i = \frac{1}{p}\sum_{k=1}^{p} PELoad_{k,i} \tag{3.9}$$

Where, $PELoad_{k,i}$ =load of all PEs in cluster i
p=number of PE in cluster (Total load of available PE in any Machine).

The average load of the machine system ALM_i is defined as:

$$ALM_i = \frac{1}{m}\sum_{i=1}^{m} ALC_i \tag{3.10}$$

Where, m=number of cluster in the machine system
We set the average load of each cluster, ALC_i to be less than the balance threshold of the machine system. Hence, this threshold is called as balance threshold.

The threshold value of the machine level denoted as ∂ and defined as:

$$\partial = ALM_i + \sigma \tag{3.11}$$

Where, σ =Standard Deviation of the load of the system and defined as:

$$\sigma = \frac{1}{N}\sum_{1}^{N}(x_i - x)^2, \qquad foralli \tag{3.12}$$

Where, x = average loads of the machine system which equals ALM
x_i is the average load of each cluster i which equals ALC and
n is the number of clusters in the machine system

3.2.3.2 Resource level Calculation

Firstly, the Average Computing Power of the machine is defined as:

$$ACM_i = \sum_{k=1}^{n} \frac{Mac_Speed_k * (1 - Mac_k)}{n} \tag{3.13}$$

Where, mac_Speed_k = Speed of machine k in resource i (MIPS)
mac_k =current CPU utilization of the k machines in resource i (in percentage)
n = number of machine in resource i

The average load of the resource system ALR_i is defined as:

$$ALR_i = \frac{1}{r} \sum_{k=1}^{r} ALM_i \tag{3.14}$$

Where, r is the number of machines in the resource system
The average load of each ALM_i, is set as less than the balance threshold of the resource system. Hence, a threshold is set called balance threshold.

The threshold value of the resource level denoted as Îů defined as:

$$\eta = ALR_i + \sigma_1 \tag{3.15}$$

Where, σ_1 is the Standard Deviation of the load of the system and defined as below:

$$\sigma_1 = \frac{1}{N} = (x_i - x)^2, \qquad foralli \tag{3.16}$$

Where, x=average loads of the resource system which equals ALR
x_i = average loads of each machine i which equals ALM
N = number of machines in the resource system.

3.2.3.3 Broker Level Calculation

Firstly, the Average Computing Power of the resource is defined as:

$$ACR_i = \sum_{k=1}^{n} \frac{Rec\_Speed_k * (1 - Rec_k)}{n} \tag{3.17}$$

Where, Rec_Speed_k = Speed of resource k in the broker (MIPS)
Rec_k = current CPU utilization of resource i (in percentage)
n = number of resources in broker

The average load of the Broker system ALB_i is defined as:

$$ALB_i = \frac{1}{b}\sum_{k=1}^{b} ALR_i \tag{3.18}$$

Where, b = number of resources in the broker system the average load of each resource ALR_i, is set as less than the balance threshold of the system.
Hence, a threshold is set called balance threshold.

The threshold value of the machine level denoted as ρ defined as:

$$\rho = ALB_i + \sigma_2 \tag{3.19}$$

Where, σ_2 = Standard Deviation of the load of the system and defined as:

$$\sigma_2 = \frac{1}{N}(x_i - x)^2, \qquad foralli \tag{3.20}$$

Where, x = average load of the broker system which equals ALB
x_i = average load of each resource i which equals ALR
N = number of resources in the broker system.

The complexity of the hierarchical network is depending on max number of nodes at any level multiply with the level of the tree. In our network, generally number of PE's is always more than number of machine and resources under same broker. So, Total complexity of the network is O(logp), where p = no of PE's.

After discussing the Load Balancing Model with supporting fitness function, the next section introduce Load Balancing Algorithm in Grid environment.

3.3 Proposed Load Balancing Algorithms

This section deals with a LB algorithm that can be run into three levels like resource, machine and PE's of LB model as shown in Figure 3.1. The proposed algorithm has been executed individually, according to the threshold value at each level and only PE level algorithms at level-2 are considered. Initially, the load has been calculated according to Gridlet (task) arrival on each node and it is connected to the corresponding PE's. The entire tasks are Compute Intensive that maximize the system resources for processing large computations for simulation. Furthermore, the Load Balancing operation is performed according to the availability of lightly loaded or under lightly loaded node list

and at the same time it passes information to the associated machines. To perform Load Balancing, a random strategy is used for selecting tasks randomly either from overloaded or under loaded node in the pool. It can reduce the time to manage the load and improve execution time as well. A random strategy is implemented at the machine level algorithm. At last, the Job Migration operation has been performed till the overload queue is not empty and same procedure has been followed at machine level and resource level. PE level load threshold, machine level load threshold and the resource level load threshold are denoted by ∂, η, and ρ have been taken from [113] [143] respectively in Algorithms 1-4. These values has been found according to the task arrival.

TABLE 3.3: Load Parameters

S.No.	Threshold Load Parameters	Level
1	$\partial = 0.6$	PE level Threshold
2	$\eta = 0.75$	Machine level Threshold
3	$\rho = 0.8$	Resource level Threshold

Description: This algorithm is executed on each node $(0 \leq i \leq n)$
Input: List of available Gridlets activity ready for execution
Output: Queued into PE list for the job execution

for *(Check all machines to the corresponding resources)* **do**
  Wait for activity (i.e. Load change in any machine);
  if *(Activity occurs)* **then**
    Call Algorithm-2 to find the PEs category (i.e. OP, UP, LP, NP);
  else
    go to step 2;
  end
end

Algorithm 1: Load Balancing Activity at PE Level

Description: This algorithm is executed for each PE till $PE \neq 0$
Input: File Size (as a workload on PEs), Rating
Output: Calculate load of each PEs

Initialized PELoad and check all Gridlets
if *(PELoad= 0)* **then**
  Go to step 9;
else
  Check Load for all PE's using corresponding Gridlets ;
  Calculate current load of $PE = \sum_1^g FileSize_j$;
  Aggregate PELoad is $PE = \frac{((Curr\_Load_{PE} - Avg\_load)}{Rating)}$;
  Return PELoad ;
end

Algorithm 2: Load Calculation Algorithm at PE Level

Description: To divide the Gridlets loads into four PE list executed on all the machine
Input: Threshold Value, PELoad, Algorithm 2
Output: Distributed job according to load in all PE lst

```
Create OPEList, ULPEList, LPEList and NPEList ;
Calculate ALM_i and σ ;
for (All PEs associated with Machine which are processed/ empty) do
    CALL PELoad_Calc_Algo ( ) //Calculate PELoad (work load of PEs);
    if (ALC_i == ∂) then
        Add this PE to NPEList;
    else
        if (ALC_i > ∂) then
            Add this PE to OPEList ;
        else
            if (ALC_i < 0.5 * ∂ ) then
                Add this PE to ULPEList;
            else
                Add this PE to LPEList;
            end
        end
    end
    Sort OPEList in descending and ULPEList and LPEList in ascending order of
    loads;
    CALL Algorithm-4 ;
end
```

Algorithm 3: Load Balancing Algorithm at Machine Level

Description: Load calculation for the Job Migration process
Input: PE load, Task load
Output: Send load information to the resources on upper level for Job Migration

for *(Check PE's from the overloaded PE List (Descending Order) from selected PE)* **do**
  if *(Check for all Gridlets processed or empty)* **then**
    Go to step 18;
  else
    Check for all PE's in the lightly loaded PE list (One by one ascending order) ;
    if *(Calculate and check* $(PELoad_D + TaskLoad_L < 0.5 * \partial)$*)* **then**
      Shift Gridlet Load of Overload to lightly loaded PE List ;
    else
      Check for all PE's in the under lightly loaded PElist (One by one in ascending order) ;
      if *(Calculate and check* $(PELoad_E + TaskLoad_L < 0.5 * \partial)$ *)* **then**
        Shift Gridlet Load of Overload PE List to under lightly loaded PE List ;
      else
        Gridlet Load will execute in its originator ;
      end
    end
    Gridlet Load will execute in its originator;
  end
  Call Algorithm 3 ;
  if $(ALC_i \geq \eta)$ **then**
    Send information to resource level for further calculation ;
  end
end

Algorithm 4: Load calculation for Job Migration Algorithm at Machine Level

The mentioned algorithms propose an efficient Load Balancing algorithm using random policy with variable threshold value. Here, machine level algorithms have been discussed. The same algorithm procedure has been followed at resource and PE level.

3.4 Conclusion

In this chapter, Hierarchical Load Balancing technique has been proposed and compared with existing techniques. LB process for job arrival has been designed and validate with the fitness functions at all three levels. Finally Load Balancing algorithms have been proposed in the Grid environment. The next chapter discusses the Job Migration techniques in the Grid environment.

Chapter 4

Proposed Job Migration Techniques

The previous chapter discussed the detailed analysis of the proposed Load Balancing algorithm on the basis of variable threshold values defined on system model. The problem description requirements, analytical view of Local Grid Manager, Load Balancing process and fitness functions have also been described for the model in the Grid environment.

Job Migration is a solution for the load dis-balancing problem as it can handle traffic balancing whenever imbalance occurs in the grid environment. For this purpose, Job Migration model has been designed in this chapter. The proposed Job Migration model offers decision making policy based on the scheduling, checkpointing and replication for Grid environment. The Job Migration problem has been formulated as an optimization problem for checkpointing, replication and resource scheduling algorithm that minimizes the average finish time and makespan. A moving scheduling framework is considered and an algorithm is formulated to achieve practical constraints. Finally, Job Migration policies have been designed and the policy rules have been specified in XML schema.

This chapter focus on all three categories of Job Migration model. Further, Job Migration algorithms have been proposed for all three categories along with mathematical formulation and execution for states of model which is presented in next section. Job Migration algorithm effectively schedules the jobs on available resources in a Grid environment and simultaneously minimizes the response time and the resource allocation capacity. Finally, QoS parameter(s) based Job Migration policies have been discussed at the last in this chapter on the basis of proposed Job Migration Algorithms.

4.1 Proposed Job Migration Model

The Figure 4.1 shows the overall architecture of the proposed Job Migration system. Grid Broker (GB) finds the appropriate resource pool. Once the Gridlet is submitted to a resource pool, job queue allocates the Processing Elements (PE) and starts the job. If any of the nodes on which the job processes are running fails, Local Job Migration Scheduler (LJMS) suspends the job execution and tries to restart it on a new node in the same pool, else switch into another the resource pool based on the availability of underloaded node. GIS periodically monitors the status of the job. If it detects that the job has been in the restart state for a long period, it tries to find a new resource pool to which the job can be migrated.

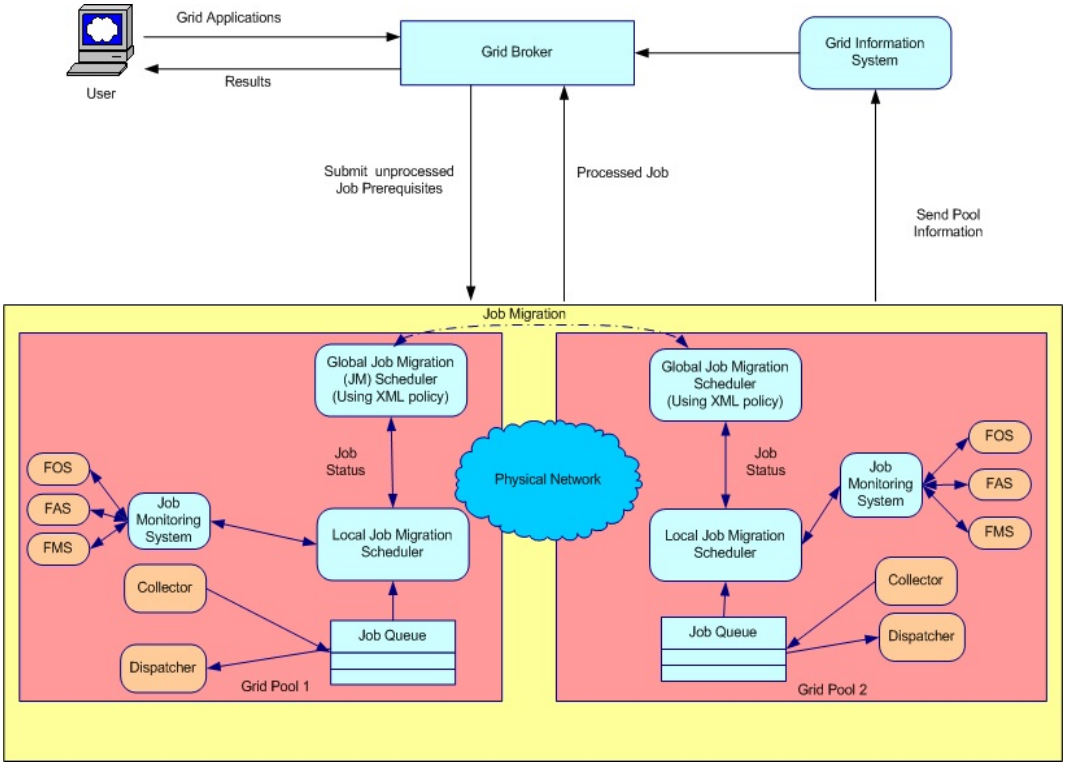


FIGURE 4.1: Job Migration Model

Components of the proposed Job Migration model are:

(i) Job Queue- Job Queue is a local queuing system of the pool for the management of Gridlets. Gridlets submitted to job queue must be specified depending on their type. Job queue has two main components: dispatcher and collector. The dispatcher performs the dispatching of Gridlets to another node and the collector is responsible of confining Gridlets from another node.

(ii) Local Job Migration Scheduler (LJMS)- LJMS is liable for scheduling of submitted Gridlets. Many processes have been executed in this component. Different states through which the job goes in the LJMS are explained below:

- Waiting- The job is initially put into a waiting queue.
- Running- Once the scheduler finds unloaded PEs that can satisfy the requirement of overloaded PEs, it assigns these Gridlets to the PEs. The PE starts the execution by sending a message to Job Monitoring System (JMS) on these pools.
- Failed- Due to hardware or software failure occurrence.
- Restart- After failure job goes to a restart state. It remains in this state until the restart scheduler assigns a PE from the restart pool. After restart, the job then comes into the running state.
- Finish- If it can successfully complete its execution

(iii) Global Job Migration Scheduler (GJMS) - It maintains the load information along with registration information of each LJMS in the grid as discussed in Section 4.2 .

(iv) Grid Information System (GIS) - The GIS consists of the monitoring system which resides on each pool. GIS directly connects the Grid Broker to share the pool and job status. It receives messages from the GJMS system in the form of Job Status and Grid Pool Status.

- Job Status: if any Gridlets executing on its PE exit, JMS sends message to GIS. The message specifies if the Gridlet departed gracefully or due to some fault/failure.
- Grid Pool Status: The GIS system periodically sends a heartbeat message to the GJMS. Through this message (i.e. Contains details like the number of Gridlets executing, free memory, and the average load of the PE) the GJMS account the health of the PE‘s.

After receiving status of PEs, If Gridlets have executed gracefully then appropriate log files are updated which are stored by GIS otherwise (in case of failure) Gridlets are terminated and the reason is written to log files. JMS also uses the log files that have been stored in GIS for monitoring purpose. It periodically checks to see if messages have been received from all the PEs. If messages are not received from a PE for a long period, then the JMS confirms the PE that are down. It makes the PE as offline and initiates the restart process for the Gridlets running on the PE.

(v) Fault Tolerance : A Gridlet can fail if any of the PEs fail. The rest of the Gridlets keeps waiting in the job queue for the execution messages from this process and the application hangs. The JMS handles two modes of failure. The following paragraph describe these failures and how the JMS detects them:

- Hardware Failures: If any Gridlet running in PE goes down, then the PE is killed. The PE cannot communicate with its peers if any hardware failure (e.g., network card failure) occurs on the PE. In these cases the JMS also cannot send its heartbeat message to the LJMS. The JMS periodically checks if a heartbeat is received from every PE in the pool. If the heartbeats from a PE are continuously missing, then the JMS concludes that a hardware failure occurred on that PE.
- Software Failure: A Gridlet process can fail due to a program error. Whenever a Gridlet exits the JMS receives a signal. The JMS can analyse this signal and know the reason for the Gridlet exit. Hence the JMS can detect if the Gridlet has exited due to an error in the application code or finished its execution gracefully.

(vi) Grid Broker (GB): It can fully control over the whole process within it, and directly operates the GJMS of all the Grid pools.

(vii) Job Monitoring System (JMS): JMS determines the load on PE's and provides this information to the Grid broker through GIS. It performs three important functions:

- Fault Observation System (FOS): The JMS receives the job startup request from the LJMS. It detects the failure and passes the information to the GIS through FOS.
- Fault Alert System (FAS): FAS counts the event and type of PE failure by analyzing the information about the state of a PE and transfers the information about the failure to FMS.
- Fault Management System (FMS): If FMS receives the information about failure, it tries to resolve failures. PE failures can be either hardware or software failure. FMS guarantees that the Gridlet submitted are completely executed using available PEs. It is also responsible for the check-pointing.

4.1.1 Execution States of Model

A FOS is associated to each of the PE managers in the LJMS and a FAS and FMS to Grid Broker. To understand the work of FOS, FAS and FMS, if any fault occurs while executing the Gridlets, three cases have been proposed for specified conditions:

(i) If failed node is not a Local grid Manager (LGM) of the pool then:

- The FOS detects the failure node and the FMS associated with the nodes report to GIS to update their status.
- FAS informs the FMS and it decides whether or not to start a local Job Migration operation from faulty node to under load node. If it decides to start a Job Migration operation, then it's load balances its workload among the nodes.
- If FMS fails to migrate jobs of the faulty node among other underloaded node, then jobs are automatically migrated to lightly under load node according to selection criteria that have been chosen for calculating the load. The chosen lightly under-loaded node are those which need minimal communication cost for transferring tasks from faulty node.

(ii) If failed node is a LGM of the pool then:

- The FOS detects the faulty node in the pool, FMS reports to the GIS and updates its information.
- The lightly under-loaded node of the pool becomes the new LGM in the pool and GIS update their log file and pass the information to GB.
- FAS detects the failure and reports to the FMS. FMS decides whether to start or not a Gridlets assignment operation of the new PEs in the pool by consulting the new LGM.

(iii) The failed node is the Global Grid Manager (GGM) of the pool then:

- The FOS detects the failure. FAS reports to GIS and updates its log file.
- The lightly under-loaded LGM of the pool becomes the new GGM in the pool. Other LGM on the grid pool update their log files and GIS passes the information to GB.
- Lightly under-loaded LGM of the grid becomes new GGM and request all LGM for their current task assignment information.

4.2 Global Job Migration Scheduler

Global Job Migration Scheduler plays an important role in achieving Load Balancing. In this work JM can be done on the basis of check-pointing, scheduling and replication. For choosing one of the three techniques, JM selection policy (discussed in Section 4.2.1)

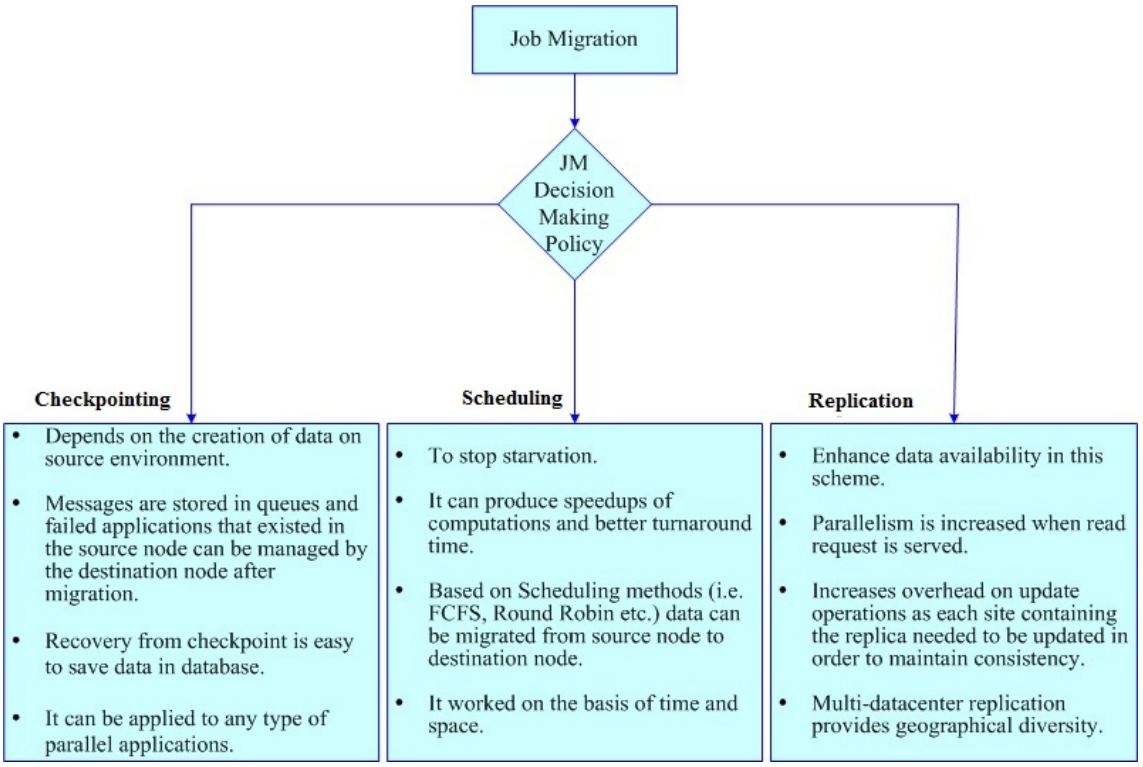


FIGURE 4.2: Job Migration Categorization

takes decision at the time of migration of the load by analysin the requirement for each job as shown in Figure 4.2.

JM decision based policy has been designed to select one of them at run time.

(i) Replication requires migrating all the applications (such as code, database etc.)at same instance, from the source node to the destination node. So, for this, it needs all the applications on the source node to be ready for migration. Critical tasks may choose this type of policy.

(ii) Checkpointing needs the small chunks of data to be migrated at the specific time requiring that data to be ready to get migrated. Process and human tasks can start in the source node and compete in the destination node. Application instance data on queues and failed events in the source node can be handled through checkpoining process. A rollback plan handles a possible migration failure.Information system or data oriented tasks may choose.

(iii) To transfer the data on the basis of time and space, scheduling can be used. Scheduling requires a minimum response time and throughput, when, the databases are being upgraded and migrated prior to start migrating the data. For this, scheduler temporarily removes processes from the source node "swapping out" and places them on the destination node "swapping in".

4.2.1 Job Migration Selection Policy

The Job Migration selection policy has been proposed for making decisions dynamically. The proposed policy has been done designed in XML for the three of the above mentioned approaches.

$\langle ?xmlversion = "1.0" encoding = "UTF-8"?\rangle$
$\langle JMpolicy\rangle$
$\langle select = "technique"\rangle$
$\langle technique1 = Checkpointing\rangle$
$\langle create\rangle Checkpoint\langle /create\rangle$
$\langle redo\rangle data\langle /redo\rangle$
$\langle undo\rangle data\langle /undo\rangle$
$\langle /technique1\rangle$
$\langle technique2 = Scheduling\rangle$
$\langle find\rangle node\langle /find\rangle$
$\langle map\rangle data\langle /map\rangle$
$\langle execute\rangle job\langle /execute\rangle$
$\langle /technique2\rangle$
$\langle technique3 = Replication\rangle$
$\langle find\rangle destinationnode\langle /find\rangle$
$\langle migrate\rangle data\langle /migrate\rangle$
$\langle /technique3\rangle$
$\langle /select\rangle$
$\langle /JMpolicy\rangle$

$\langle ?xmlversion = "1.0" encoding = "UTF-8"?\rangle$
$\langle xs : schemaxmlns : xs = "http://www.w3.org/2001/XMLSchema" elementFormDefault =$
$"qualified" attributeformDefault = "unqualified"\rangle$
$\langle decision\rangle$
$\langle choiceid = IDmaxOccurs = nonNegativeInteger|unboundedminOccurs = nonNegativeIntegerany$
$(annotation?, (Replication|checkpointing|scheduling)$
$\langle decision\rangle$
$\langle xs : elementname = "replication"\rangle$
$\langle xs : annotation\rangle$
$\langle xs : documentation\rangle$
$JobMigrationwithReplicationOperation$
$\langle /xs : documentation\rangle$
$\langle /xs : annotation\rangle$

$\langle xs : complexType\rangle$
$\langle xs : sequence\rangle$
$\langle xs : elementname = "Jobexecutionservice"\rangle$
$\langle xs : complexType\rangle$
$\langle xs : attributename = "Servers" type = "xs : string" use = "required"/\rangle$
$\langle xs : attributename = "Storage" type = "xs : string" use = "required"/\rangle$
$\langle /xs : complexType\rangle$
$\langle /xs : element\rangle$
$\langle xs : elementname = "ComputeQoSParameterforReplication" minOccurs = "0"\rangle$
$\langle xs : complexType\rangle$
$\langle xs : attributename = "Cost" type = "xs : integer" use = "required"/\rangle$
$\langle xs : attributename = "Frequency" type = "xs : integer" use = "required"/\rangle$
$\langle xs : attributename = "Reliability" type = "xs : integer" use = "required"/\rangle$
$\langle /xs : complexType\rangle$
$\langle /xs : element\rangle$
$\langle /xs : sequence\rangle$
$\langle /xs : complexType\rangle$
$\langle xs : elementname = "checkpointing"\rangle$
$\langle xs : annotation\rangle$
$\langle xs : documentation\rangle$
$JobMigrationwithCheckpointingOperation$
$\langle /xs : documentation\rangle$
$\langle /xs : annotation\rangle$
$\langle xs : complexType\rangle$
$\langle xs : sequence\rangle$
$\langle xs : elementname = "Jobexecutionservice"\rangle$
$\langle xs : complexType\rangle$
$\langle xs : attributename = "ProcessingElement" type = "xs : string" use = "required"/\rangle$
$\langle xs : attributename = "CommunicationLink" type = "xs : string" use = "required"/\rangle$
$\langle /xs : complexType\rangle$
$\langle /xs : element\rangle$
$\langle xs : elementname = "ComputeQoSParameterforCheckpointing" minOccurs = "0"\rangle$
$\langle xs : complexType\rangle$
$\langle xs : attributename = "undo" type = "xs : integer" use = "required"/\rangle$
$\langle xs : attributename = "redo" type = "xs : integer" use = "required"/\rangle$
$\langle /xs : complexType\rangle$
$\langle /xs : element\rangle$
$\langle /xs : sequence\rangle$
$\langle /xs : complexType\rangle$

$\langle xs : elementname = "scheduling" \rangle$
$\langle xs : annotation \rangle$
$\langle xs : documentation \rangle$
$JobMigrationwithSchedulingOperation$
$\langle /xs : documentation \rangle$
$\langle /xs : annotation \rangle$
$\langle xs : complexType \rangle$
$\langle xs : sequence \rangle$
$\langle xs : elementname = "Jobexecutionservice" \rangle$
$\langle xs : complexType \rangle$
$\langle xs : attributename = "TimeBasedScheduling" type = "xs : string" use = "required" / \rangle$
$\langle xs : attributename = "SpaceBasedScheduling" type = "xs : string" use = "required" / \rangle$
$\langle /xs : complexType \rangle$
$\langle /xs : element \rangle$
$\langle xs : elementname = "ComputeQoSParameterforScheduling" minOccurs = "0" \rangle$
$\langle xs : complexType \rangle$
$\langle xs : attributename = "ResponseTime" type = "xs : integer" use = "required" / \rangle$
$\langle xs : attributename = "ResourceCapacity" type = "xs : integer" use = "required" / \rangle$
$\langle /xs : complexType \rangle$
$\langle /xs : element \rangle$
$\langle /xs : sequence \rangle$
$\langle /xs : complexType \rangle$
$\langle /xs : element \rangle$
$\langle /xs : schema \rangle$

After discussing about the selection policy for the JM technique in XML, the next subsection describes each JM techniques in detail.

4.2.2 Job Migration with Checkpointing Algorithms

Checkpointing is the technique of saving computation state in stable storage, to be reconstructed later. At checkpoint time, the checkpointing subsystem will first freeze the application, so that its state is stable and persistent. It supports Time Based and Record Based checkpoint algorithms [144]. In time-based checkpointing, algorithm commits global transactions at a specified time interval. In record-based checkpointing, algorithm commits global transactions at a specified number of iterations of batch step. The basic steps of checkpointing are given below:

(i) PE chooses job to take decision for start checkpoining.

(ii) Receiving checkpoint request at the receiver PE.

(iii) After receiving an acknowledgment message from the sender, Algorithm is executed on the receiver PE.

(iv) If any, fault has been occurred then recovery process has been started.

(v) It start from the sender PE, where algorithm is executed and faulty PE at the source end has been resorted.

(vi) Send recovery message to receiver PE.

(vii) Resume the receiver message again.

Algorithms 5-10 shows the checkpointing and recovery scheme.

Description: PE chooses job to take decision for start checkpoint (CP)
if *(No provisional checkpoint)* **then**
    Take a new provisional CP with new checkpoint_id ;
    Set originator_flag=True and originator_id =X ;
    Send a CP request to the next PEs with originator_id ;
end

Algorithm 5: Beginning of Check pointing from starting PE_X

```
Description: PE chooses job to take decision for start checkpoint (CP)
if (Provisional checkpoint exists) then
    if Y is the immediate ancestor of the receiving originator_id then
        Remove the existing stable checkpoint ;
        Create the presented provisional checkpoint stable ;
        Set acknowledgement originator_id = Y ;
        Generate and send acknowledgement message to the next PE with
        acknowledgement originator_id ;
    else
        if (Y > receiver originator_id) then
            Promote the checkpoint request to the next PE;
        else
            if (originator_flag X = False) then
                Advance the checkpoint request to the next PE;
            else
                Reject the checkpoint request;
            end
        end
    end
else
    if (Y is the immediate ancestor of the receive originator_id) then
        Remove old stable checkpoint ;
        Take a new stable checkpoint with new checkpoint_id ;
        Set acknowledgement originator_id = Y ;
        Generate and send acknowledgement message to the next PE with
        acknowledgement originator_id ;
    else
        Take a new provisional checkpoint with new checkpoint_id forward the
        checkpoint request to the next PE ;
    end
end
```

Algorithm 6: Receiving CP request at the receiver PE_Y

Description: After receiving an acknowledgement message from sender the algorithm executed on receiving PE_Y

if *(Y is not the immediate predecessor of the originator of this acknowledgement message)* **then**
  if *(Provisional Checkpoint Exists)* **then**
    Remove the stable checkpoint make the presenting provisional checkpoint stable ;
  else
    Forward acknowledgement message to the next PE ;
  end
else
  if *(provisional checkpoint exists)* **then**
    Remove the stable checkpoint make the presenting provisional checkpoint stable ;
  else
    Reject acknowledgement message;
  end
end

Algorithm 7: Acknowledgement Message Receives PE_Y

Description: After receiving an acknowledgement message from sender the algorithm executed on receiving PE_Y

Set recovery originator_id ;
Send recovery message to the next process with latest checkpoint_id and recovery originator id ;

Algorithm 8: Originator Recovery (Algorithm is executed by PE_x when the faulty PE_x is restored)

Description: Receiver Recovery Message at PE_Y

if *(PE_Y and checkpoint_ id of the newest checkpoint match with the originator_ id and checkpoint_ id of the recovery message)* **then**
  if *newest checkpoint is provisional* **then**
    Remove the stable checkpoint ;
    Create the provisional checkpoint stable ;
  else
    Produce resume message ;
    Set resume originator id=Y ;
    Send resume message to the next PE with resume originator_ id ;
  end
else
end
if *(Checkpoint_ id of the latest checkpoint does not match with checkpoint_ id of the recovery message but $Y \neq$ originator_ id)* **then**
  if *(Provisional checkpoint exists)* **then**
    Remove the provisional checkpoint ;
    Rollback to its previous stable checkpoint ;
    Forward recovery message to the next process ;
  else
    Remove recovery message ;
    Set PE_y as a new recovery originator ;
    if *(checkpoint_ id of the latest checkpoint match with checkpoint_ id of the recovery message but $Y \neq$ originator_ id)* **then**
      Promote recovery message to the next PE ;
    end
  end
end

Algorithm 9: Receiver Recovery Message at PE_Y

Description: (Resume Receiver Message)

if *(PE_Y is not the immediate predecessor of the originator of this resume message)* **then**
  if *Provisional checkpoint exists* **then**
    Remove the presenting stable checkpoint ;
    Create the presented provisional checkpoint stable ;
  end
  Advance resume message to the next PE ;
else
  if *(Provisional checkpoint exists)* **then**
    Remove the presented stable checkpoint ;
    Create the presenting provisional checkpoint stable ;
  end
  Terminate resume message ;
end

Algorithm 10: Resume receiver message

Proposed algorithms have $O(n^2)$ complexity as compare to existing checkpointing techniques (S-K and P-K) presented in [144] [145]. It needs only one temporal and permanent chekpoint for each process at run time, and one temporal checkpoint at the time when rollback recovery has been done.

4.2.3 Job Migration with Scheduling Algorithms

Grid scheduler [148] schedules the task and finds the appropriate resource for each task. Scheduling policies for Grid systems can be classified into space sharing and time sharing policies. In time sharing scheme, processors are shared over time by executing different applications on the same processors during different time intervals, which is commonly known as time slice or quantum. In contrast, in the space sharing approach, processors are partitioned into disjoint sets and each application executes in isolation on one of these sets.

The users submit the tasks to the grid scheduler and the grid scheduler assigns the submitted tasks to the available resources based on their requirements. Each PE is different in many aspects (i.e. the number of processing elements, processing speed, internal scheduling policy, load factor, etc.). Similarly, each job is different from others in terms of execution time, deadline and time zone etc. The terms and abbreviations used in the algorithm are given in Table 4.1

TABLE 4.1: Abbrevations for Scheduling Algorithm

S.No.	Abbreviation	Description
1	$CL(J_x)$	Client Limitation of Job in x sec
2	$PT(J_x, PE_y)$	Projection Time to compute p*q size matrix Where p=No. of jobs and q=Processing Elements (PE)
3	$ST(J_x, PE_y)$	Starting Time of (PE_y)
4	$FT(J_x, PE_y)$	Finish Time of each job
5	$AFT(J_x, PE_y)$	Aggregate Finish Time
6	$CT(J_x, PE_y)$	Communication Time of each job
7	$IF(J_x, PE_y)$	Time taken by (J_x) for transfer of input files to the (PE_y)
8	$OF(J_x, PE_y)$	Time taken by (J_x) for transfer of output files to the (PE_y)
9	$CR(PE_y)$	Crash Rate of (PE_y)
10	$Failed_{Gridlets}$	Number of job crashed to be executed previously in (PE_y)
11	J_{sub}	Number of jobs submitted to be executed previously in (PE_y)
12	$HR(J_x, PE_y)$	Health Rate of job (J_x)
13	CHF	Crash rate Health Function
14	H_{var}	Variation in health rate
15	J_{succ}	Number of jobs successfully completed by a (PE_y)
16	$Timespan_x$	Total time span of job in the queue
17	$Processingpower_y$	Processing power of PE in MIPS
18	CR_{min}	Minimum of crash rates of all the (PE_y)
19	$Var(J_x)$	Minimum variation of (J_x) in $(PE_y) \in$ q
20	$Task\_Load_L$	Task Load

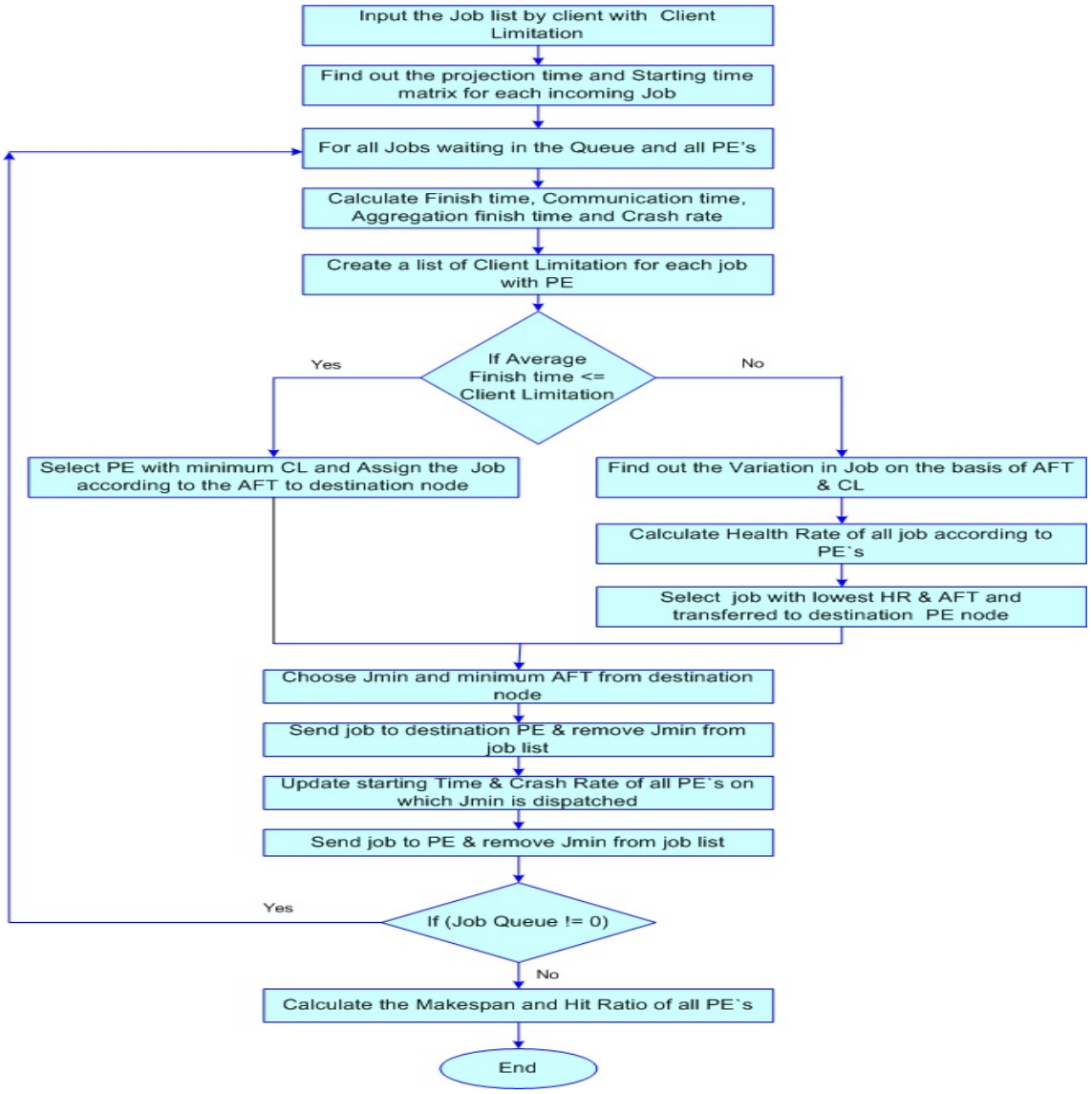


FIGURE 4.3: Job Migration Flowchart

For an assumption, each machine executes one job at a time and they have prior knowledge about the size of the jobs, number of PEs, and PT (Projection Time). PT matrix is constructed using the AT (Approximate time) for job x on Processing Element PE_y. Here, in PT matrix, rows contain the information about estimated projection times for a given job on each machine and columns consist of the estimated projection times of a given machine for each job. Thus, for random job J_x and random PE_y , $PT(J_x, PE_y)$ is the approximate time of J_x on PE_y.

The procedure of Job Migration scheduling and proposed Job Migration algorithm have been shown in Figure 4.3 and Algorithm 11 respectively.

Description: Job Migration with Scheduling Algorithm.
Input: the Job list by the client with its limitations $CL(J_x)$;

Create projection time $PT(J_xPE_y)$ a matrix of size p*q ;
Create starting time $ST(J_xPE_y)$ a matrix of size 1*q ;
for *(All Job Jx in the queue and for each PE_y where y∈q)* **do**
  Create Finish time $FT(J_xPE_y)$ matrix of size p*q;
  Communication time $CT(J_xPE_y)$ matrix of size p*q ;
  Aggregate finish time $AFT(J_x, PE_y)$ matrix of size p*q ;
  Calculate crash rate $CR(PE_y) = \frac{Failed_{Gridlets}}{Job_{submit}}$,for all PEs ∈ q ;
  Create a list of client limitation $CL((J_x)$ for each job task Jx with PE which has $AFT(J_x, PE_y) \leq CL((J_x)$;
  if *(Entries in $CL((J_x))$)* **then**
    Select the PE with minimum $CR(PE_y)$;
    Set Job J_x, PE_y, and $AFT(J_x, PE_y)$ to Destination node $(J_x, PE_y, AFT(J_x, PE_y)$;
  else
    Construct $Var(J_x) = AFT(J_x, PE_y) - CL(J_x)$ and $HR(J_x, PE_y) = CHF + H_{Var}$;
    Set job J_x, PE_y with lowest HR and $AFT(J_x, PE_y)$ to Destination node $(J_x, PE_y, AFT(J_x, PE_y)$;
  end
  Choose Job J_{min} with minimum $AFT(J_x, PE_y)$ from Destination node $(J_x, PE_y, AFT(J_x, PE_y)$;
  Send out Job J_{min} to PE_y and remove Jmin from the job list ;
  Update $ST(PE_y)$ where y is the PE to which the job J_{min} is dispatched ;
  Update $CR(PE_y))$, if PE_y fails, where y is the PE to which the task $J_{min}$is dispatched ;
end
if *(Job queue ≠ 0)* **then**
  Repeat the process from the starting. ;
else
  Calculate Makespan = max $ST(PE_y)$ and $HitRatio = J_{succ}J_{sub}$ ∀ y ∈ n ;
end

Algorithm 11: Scheduling Algorithm

The PT matrix $PT(J_x, PE_y)$ is derived as

$$PT(J_x, PE_y) = \frac{Timespan_x}{ProcessingPower_y} \tag{4.1}$$

The starting time of the PE_y can complete the execution of all the jobs that have previously been assigned to it based on the PT entries for those jobs.The Starting time of the PE_y is calculated by

$$ST(J_x, PE_y) = \sum_{x=1}^{n} PT(J_x, PE_y) \tag{4.2}$$

The Finish time for a new job J_x on PE_y, $PT(J_x, PE_y)$ is the machine, projection time for pay plus the starting time of the new job J_x on PE_y, i.e.

$$FT(J_x, PE_y) = PT(J_x, PE_y) + ST(J_x, PE_y) \tag{4.3}$$

The maximum $FT(J_x, PE_y)$ value is the sub job execution time and is called the makespan.The communication time of each job x ∈ p on each $PE_y \in$ q is given by

$$CT(J_x, PE_y) = IF(J_x, PE_y) + OF(J_x, PE_y) \tag{4.4}$$

where the time taken for transfer of input file is calculated as the ratio of input file size to the bandwidth of the PE and the time taken for transfer of output file is calculated as the ratio of output file size to the bandwidth of the PE.

$$IF(J_x, PE_y) = \frac{InputFilesize_X}{PE_Bandwidth_Y} \tag{4.5}$$

$$OF(J_x, PE_y) = \frac{OutputFilesize_X}{PE_Bandwidth_Y} \tag{4.6}$$

Aggregate finish time of each job $J_x \in$ p on each $PE_y \in$ q is given by

$$AFT(J_x, PE_y) = FT(J_x, PE_y) + CT(J_x, PE_y) \tag{4.7}$$

The failure consideration for the grid is dispatched with nodes that the job does not fail. The parameter that decides this is the crash rate $CR(PE_y)$ for all y ∈ q, which is calculated by:

$$CR(PE_y) = \frac{Failed_{Gridlets}}{Job_{submit}} \tag{4.8}$$

The crash rate ranges from 0 to 1. But it is not possible to find a suitable PE with less CR and high processing power for all jobs. So, a difference HR and CHF are calculated, from which we can find the overall Health variation as:

$$H_{var} = \frac{Var(J_x) - Var_(min_x)}{2} \tag{4.9}$$

The crash rate health function is calculated as

$$CHF = \frac{CR(PE_y) - CR_{min}}{2} \tag{4.10}$$

Based on health rate, the jobs are assigned to the PE.

Makespan is one of the most important standard metric of grid scheduling to measure its performance. Makespan is defined as

$$Makespan = maxST(PE_Y) \tag{4.11}$$

Makespan is used to measure the ability of grid to accommodate Grid lets in less time. Hit Rate presents the number of jobs successfully completed without considering the failure of PEs. The hit rate is the ratio of the number of jobs successfully completed to the number of jobs submitted to grid and it is given by:

$$HitRatio = \frac{J_{succ}}{J_{sub}} \tag{4.12}$$

The complexity of the JM scheduling algorithm is O(p), where p = no of PE's.

4.2.4 Job Migration with Replication Algorithms

Replication [146] is an important technique for increasing the computer system availability. Replication stores data on multiple nodes/servers. It is an independent operation and is performed on same configuration from source to destination node. Application instance data on queues and failed events in the source node can be handled post migration by the destination node. For this purpose additional hardware and/or software resources are required to manage another production environment. This approache provide support for two parallel environments, the source and the destination. The Basic steps for replication is shown in Algorithm 12:

```
Description: Basic Replication Algorithm
Prompt all remote single resource managers that have not reported job status;
Determine the number of healthy replicas ;
if (Any replica is done) then
    Compare the correct answer with the received result;
    Inform all remote single resource managers to terminate their replica ;
    Update job table ;
else
    if (Healthy Replica > Replica Threshold value) then
        Select last replica from set to terminate ;
        Update job table ;
    else
        Pick next site from set to terminate ;
        Inform the remote SRM to execute the job ;
        Update job table ;
    end
end
```

Algorithm 12: Basic Steps for Replication Network

To understand Job Migration with replication, a case study "Load Balancing and Job Migration in Social Grid Environment" has been considered in Chapter 6.

4.3 Conclusion

This chapter discussed the proposed Job Migration Model and QoS parameter based Job Migration Policies as an outcome of the research work. The detailed requirements of Job Migration model has been analyzed. Further, the categories of JM policy in the model has been discussed. The next chapter discusses the verification and validation of Load Balancing and Job Migration algorithm.

Chapter 5

Results & Discussions

The previous chapter discussed in detail the Job Migration model and the problem formulation of Job Migration algorithms to achieve the practical constraints. It then moves to the design of Job Migration policy rules for checkpointing, scheduling and replication algorithm that have been specified in XML schema.

This chapter focuses on the verification and validation of the proposed model. The proposed Load Balancing and Job Migration algorithm has been implemented in GridSim toolkit. Statistical analysis of simulation output has been performed in order to assess the accuracy of the estimated performances by using threshold values. The algorithms have also been validated from various other perspectives similar to the existing algorithms.

At the outset, this chapter outlines the details of verification and the implementation of the proposed work through variable threshold value. 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 Job Migration technique have been validated.

In this work, the Gridlets are assumed to be computationally intensive and to arrive randomly following a Poisson process. The Gridlets are mutually independent and can be executed by any PE that satisfies the Gridlet requirements. Each PE can execute a single Gridlet at a time and no preemption is possible.The characteristics of resources and the scheduling parameters considered are given in Tables 5.1. Through this part of thesis firstly, configuration of the simulation set up is in introduced and then experimental results discussed. Each algorithm was run five consecutive times and the average of the results was taken as the final result of that algorithm.

TABLE 5.1: Simulation Parameters

S.No.	Parameters	Values
1	No of Machines	1
2	No of PEs into machines	2
3	PE Rating	5-50 MIPS
4	No of Gridlets	Min 500
5	Gridlet length	50,000 to 1,00,000 MI
6	Input File Size	50-500 MB
7	Output File Size	100-700 MB
8	Number of nodes in a cluster	25 Nodes in a Cluster
9	Number of nodes (S)	100, 500, 1000, 2000
10	Reschedule interval (s)	600
11	Number of jobs	1000
12	Requests intensity/frequency (Ri)ms	50, 100, 200, 400
13	Arrival time (ms)	200

5.1 Experimental Results of Load Balancing Algorithms

Windows 7 on an Intel i3 with 3 GB of RAM and 360 GB of hard disk has been used during simulation experiments in GridSim version 5.0 with JDK 1.5 updates 6 and JRE 1.3. The goal of this experiment is to compare the performance of a decentralized Proposed Load Balancing Algorithms (PLBA) with conventional Without Load Balancing (WLB) and Load Balancing in enhanced GridSim (LBEGS) algorithm.

Grid systems are randomized in various sizes: 100, 500, 1000, and 2000 nodes. In the experiments some of the test parameters are changed, such as the size of the Grid that is denoted by S on figures, resource requests frequency with 50, 100, 200 and 400 ms are denoted by Ri. The experiment is to randomly submit 1000 Grid requests at the starting and schedule them to the specific Grid resource based on WLB, LBGES and PLBA. The size of data carried by the Grid task agent that is denoted by D, all tasks have the same computation size D = 25KB. There are 25 Grid resource agents in the system. The arrival time of each resource request is determined randomly by an exponential distribution with mean of 200 ms, but we will change the values of arrival time when

testing effect of requests frequency on response time and resource allocation efficiency. All parameters are summarized in Table 3.3.

Initially, all nodes have zero load. Throughout the experiment, only one Grid user agent is used for the generation of Grid resource requests. For the simulated scenarios, it does not matter how many Grid user agents are there in the system. These experimental configurations are to bring up the performance of the Load Balancing algorithms.

The first experiment is to compare the performance of WLB, a LBGES algorithm with PLBA in terms of response time and resource allocation efficiency.

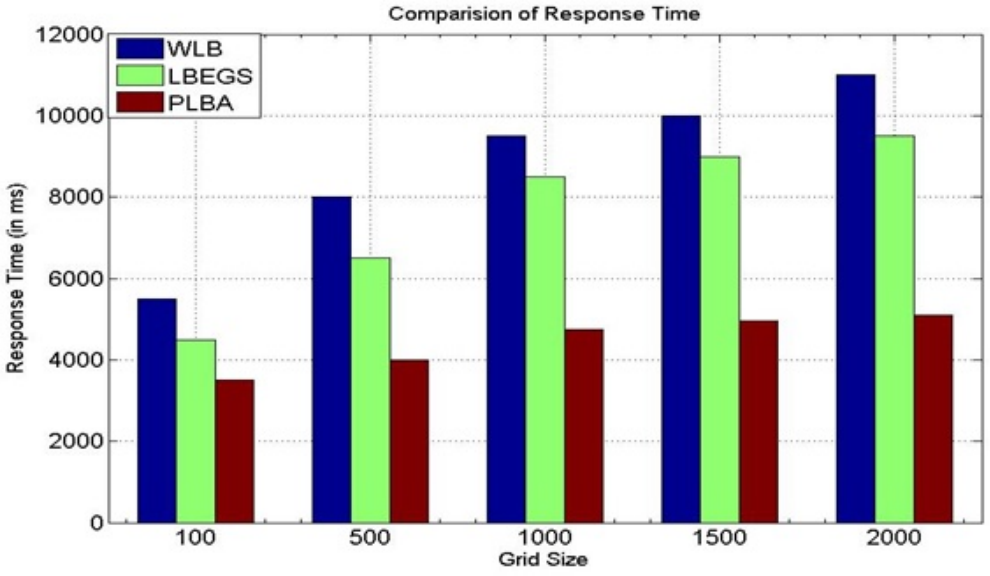


FIGURE 5.1: Comparison of Response Time

Figure 5.1 shows the Response times that are influenced by Grid size, available connections and bandwidth using Request intensity (Ri = 200 ms). For comparing different size Grid, a lower average response time is considered to be better depicted through Figure 5.1. All algorithms present the best results in the small size Grid but, when the size of the Grid is larger, PLBA is decreasing quickly. The response time using PLBA can be 50 % and 44% shorter than that using the WLB and LBGES respectively. As shown in Figure 5.2, the PLBA outperforms the conventional WLB and LBEGS for the different size grid. Secondly, we measured the resource allocation efficiency of PLBA and WLB and LBEGS using Ri = 200 ms. The WLB and LBEGS achieves to match nearly 98% of all requests in the small size Grid scenario, with PLBA closely behind. However, as Grid size increases, the WLB and LBEGS degraded performance than the PLBA. Under the large size Grid, decrease of results for WLB and LBEGS is lower than in the small size. Resource allocation efficiency using PLBA is 27% and 15% larger than that using

the WLB and LBEGS. Varying Grid size, result decreases for both methods in a similar manner.

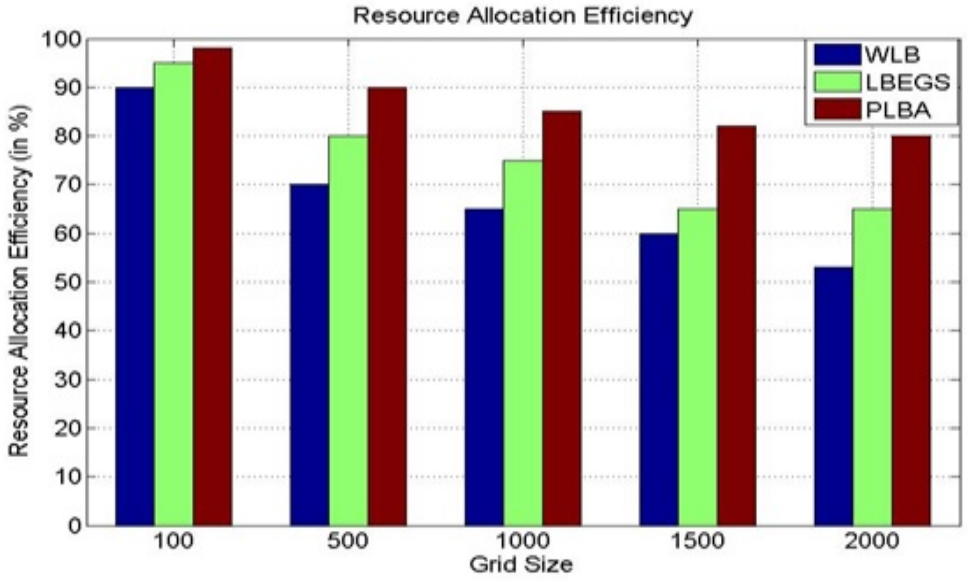


FIGURE 5.2: Resource Allocation Efficiency

The basic experiments described above have been conducted to show that the fundamental differences between the PLBA and the WLB and LBEGS. Next, a sensitivity experiments is devised in order to change the request frequency variables. The effects of request frequency to response time and resource allocation efficiency under different size of the Grid through request intensity shown in below mentioned experiments. The following figure shows the behaviour of two allocation methods when changing the resource requests frequency from 50 to 100 ms and 200 to 400 ms. For the response time, a lower request frequency leads to faster access times, whereas, X-axis shows a change in request frequency. Presented through Figure 5.3 - 5.5 the response time using PLBA can be 21% and 9% shorter than that using the WLB and LBEGS respectively, under small size Grid (S = 100) and low request frequency (Ri = 200 ms). When changing the size of Grid by S = 500 and remain request frequency as Ri = 200 ms, the response time of WLB and LBEGS is 19% & 11% & longer than that using the PLBA. When increasing the size of Grid by S = 2000 and request frequency by Ri = 50 ms, the performance of WLB and LBEGS deteriorates quickly, the response time is 79% & 29% longer than that using the PLBA. Under big size Grid (S = 2000) and high request frequency (Ri = 50, 100 ms), WLB and LBEGS take more time to allocate appropriate resources, the response time is 70% & 43% longer than that using the PLBA. This effect could be caused by an increasing synchronization of the frequency request generation with the negotiations that lead to successful resource provisioning. In other words, the large number of open requests

leads to longer queues for processing, whereas the queues get shorter with decreasing request frequency, the overall processing gets faster.

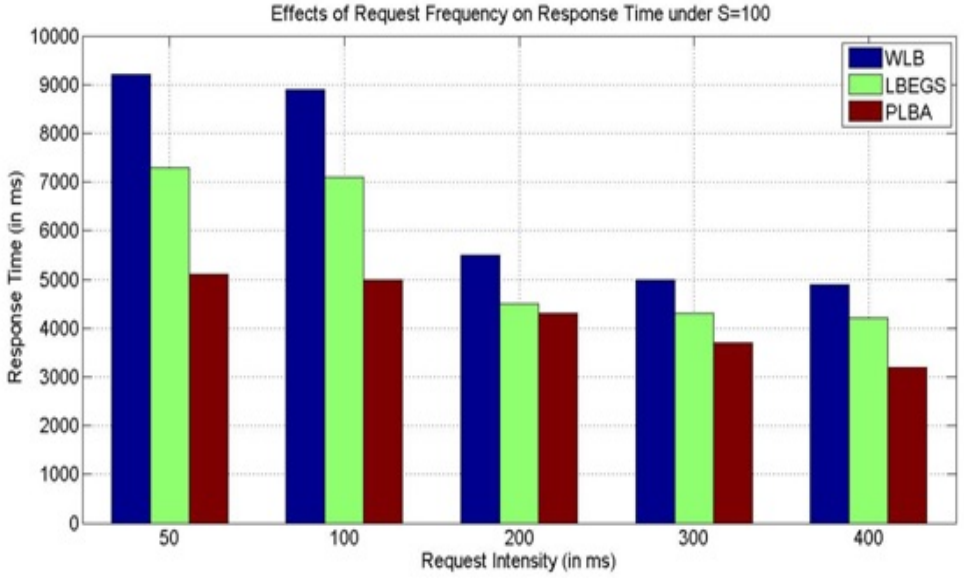


FIGURE 5.3: Effects of Request Frequency on Response Time under S = 100

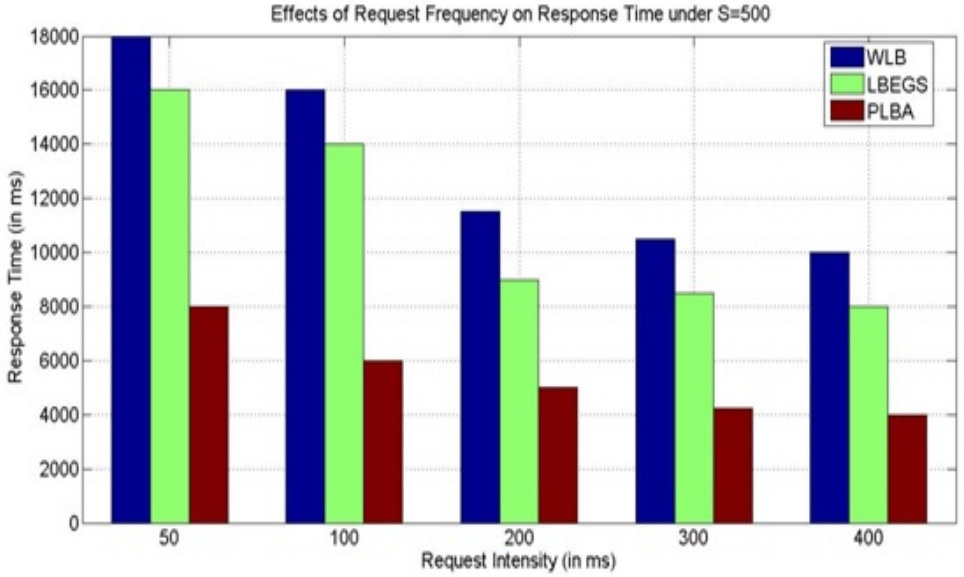


FIGURE 5.4: Effects of Request Frequency on Response Time under S = 500

Considering the resource allocation efficiency, from the results in Figures 5.6- 5.8 , under small size Grid and low request frequency (Ri = 200 ms), the resource allocation efficiency using PLBA is nearly the same as that using the WLB and LBEGS. When increasing the size of Grid by S = 100 and remain request frequency as Ri = 200 ms, the resource allocation efficiency of PLBA as much as 23% &12% larger that using the WLB and LBEGS respectively. When increasing the size of Grid by S = 500 and request frequency by Ri = 50 ms, the resource allocation efficiency of WLB and LBEGS is as much as

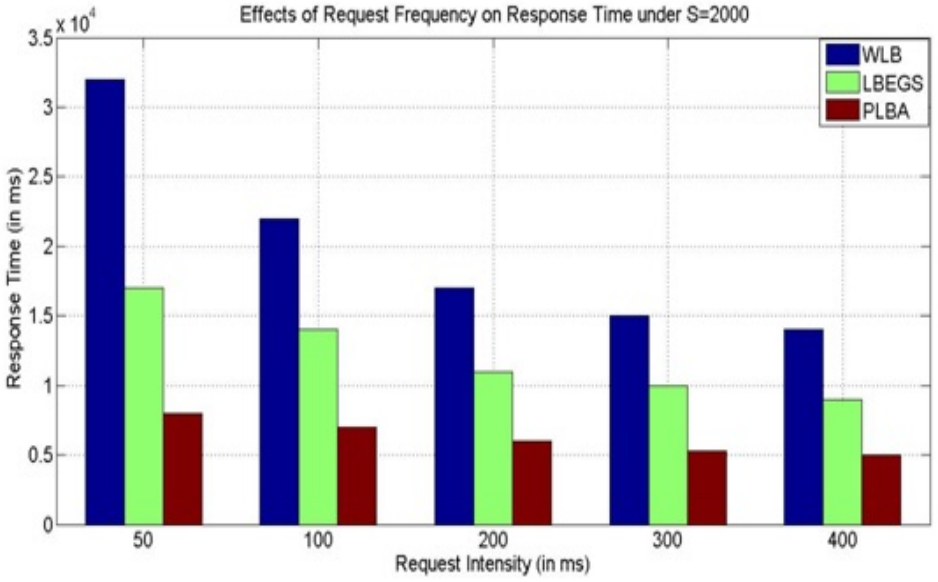


FIGURE 5.5: Effects of Request Frequency on Response Time under S = 2000

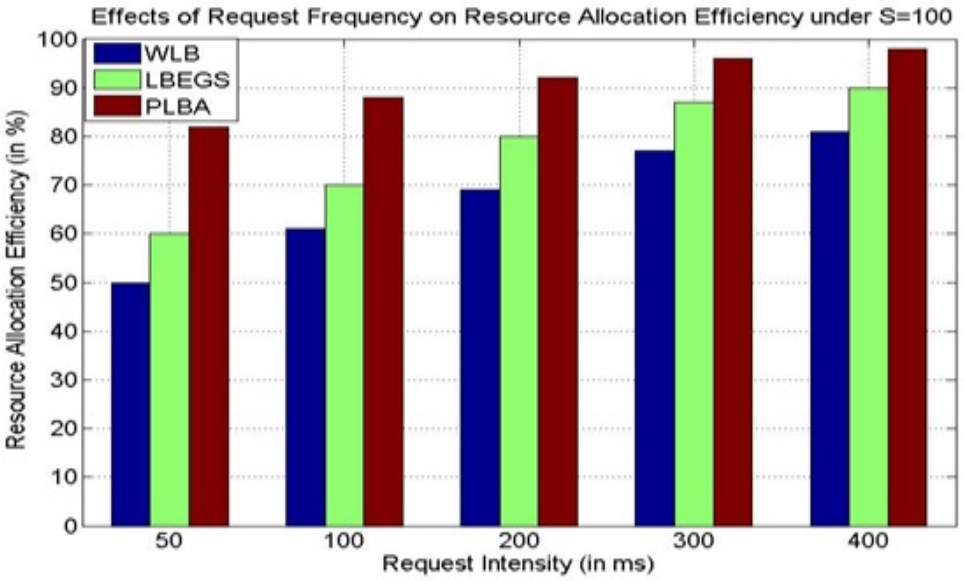


FIGURE 5.6: Effects of Request Frequency Resources on Allocation S = 100

43% & 11% less than that using the PLBA. Under big size Grid (S = 2000) and high request frequency (Ri = 50), the resource all location efficiency of WLB and LBEGS is as much as 43% less than that using the PLBA. Resource allocation efficiency of 93% can be achieved for the PLBA with a 200 ms pulse even in high request density. Complete saturation of request in WLB and LBEGS can be attained in a small size Grid/high request density scenario. Increasing the request frequency implies a high charge to the system and in case of 50 ms pulses reduce the resource allocation efficiency to nearly 50% in all scenarios. In most of the test cases, the PLBA is more efficient than the WLB and LBEGS to allocate Grid resources in the test application. When Grid size is creasing,

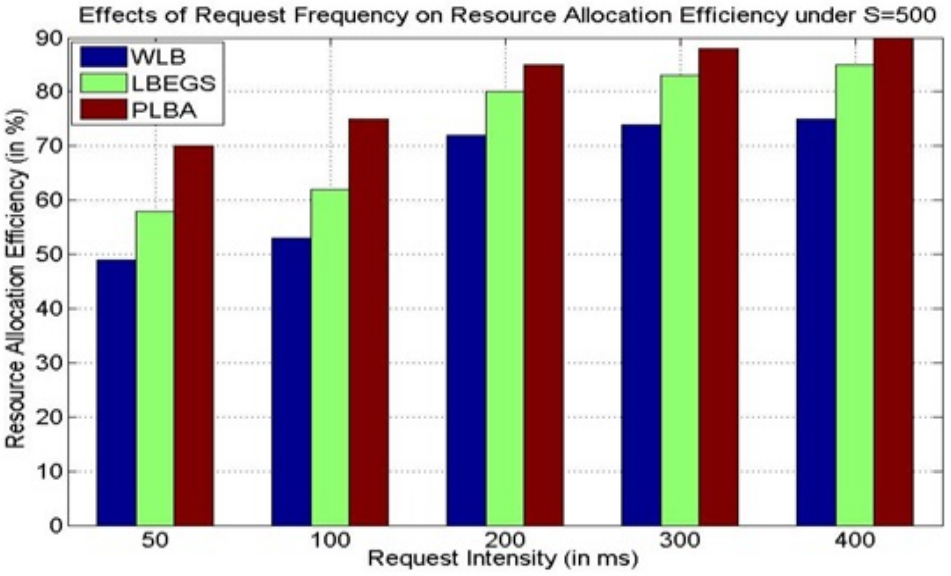


FIGURE 5.7: Frequency Effect on Resource Allocation Efficiency under S = 500

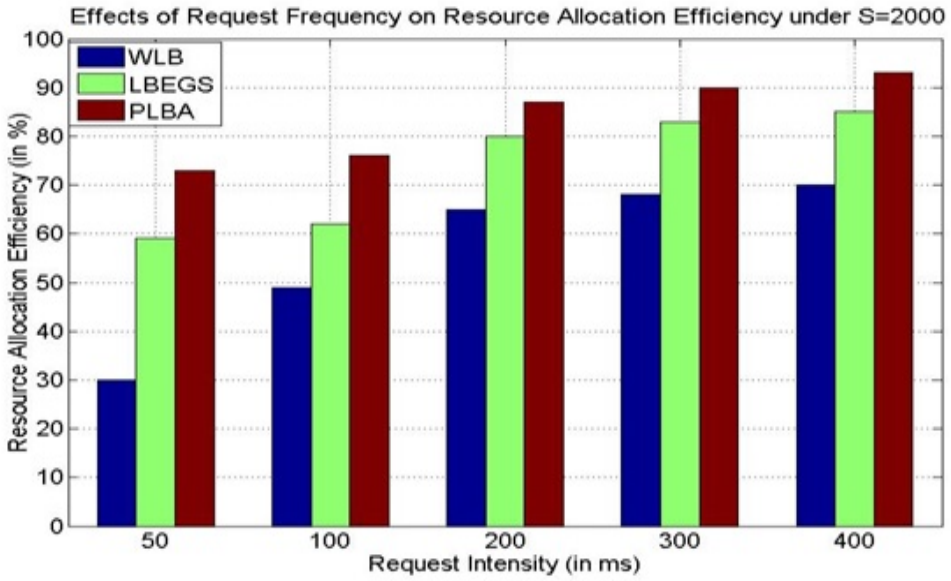


FIGURE 5.8: Frequency Effect on Resource Allocation efficiency under S = 2000

it has more merits to use the PLBA to schedule Grid resource; the PLBA has better performance than usual WLB and LBEGS.

5.1.1 Comparison of Proposed Load Balancing Algorithm with Existing Algorithms

In Section 2.2.4, an algorithm is introduced for resource selection including min-min, max-min, WLB, LBEGS, PLBA etc. In this section, the performance of WLB, min-min,

max-min, LBEGS and PLBA in a Grid will be investigated by simulation. Considering that there are ten resources that every resource has five machines, and every machine has four PEs. That is leading to 200 PEs in total. The number of Gridlets are kept constant at 500 and every Gridlet at the beginning time and without a deadline request. Every PE is between grade 1 and 10 and every Gridlet is between grade 1 and 10 too. For ease of study, the local load factor of resources is set to be zero; this does not affect the performance of the algorithms. These settings are convenient for us to analyse the effectiveness and the behaviour of the algorithms. The resource load threshold parameters and Gridlet parameters are listed in Table 3.3. The comparison of communication overhead times and makespan between above algorithms are depicted through Figure 5.9 and 5.10.

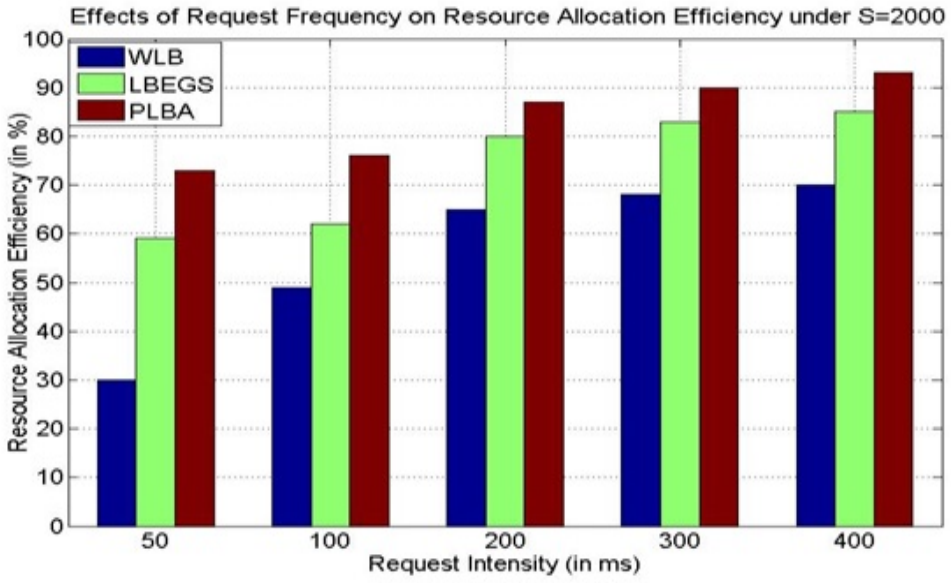


FIGURE 5.9: Communication Overhead Times vs Gridlets on PEs = 200

Figure 5.9 shows the communication overhead times of the different algorithms keeping the number of Gridlets up to 2000. It shows that the times of WLB and min-min increase quickly by increasing Gridlets; keeping max-min is in the middle. It can be observed that LBEGS and PLBA always have the minimum of the five algorithms and increase very slowly. Figure 5.9 illustrates that PLBA gives better performance by reducing communication overhead times the makespan of the different algorithms is included in Figure 5.10 which demonstrates that the makespan of all algorithms increases with the number of Gridlets increasing; PLBA is the minimum of all algorithms that gives good performance with respect to makespan. In conclusion, PLBA has better performance in reducing communication overhead times and makespan.

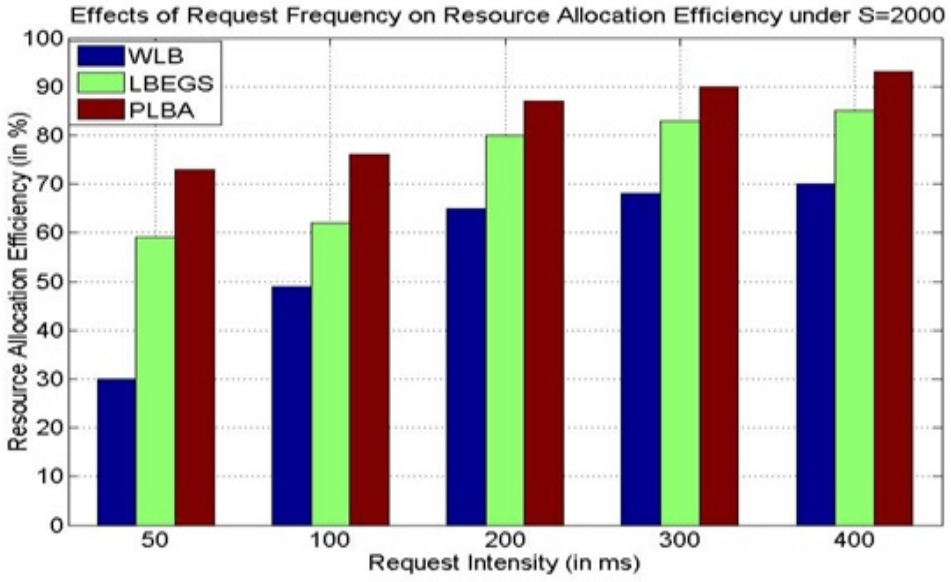


FIGURE 5.10: Makespan vs Gridlets on PEs = 200

5.2 Experimental Results of Job Migration Techniques

5.2.1 Job Migration with Checkpointing

The experiment is to compare the performance of S-K and P-K with proposed algorithm in terms of number of job migration with time and performance of processors with time.

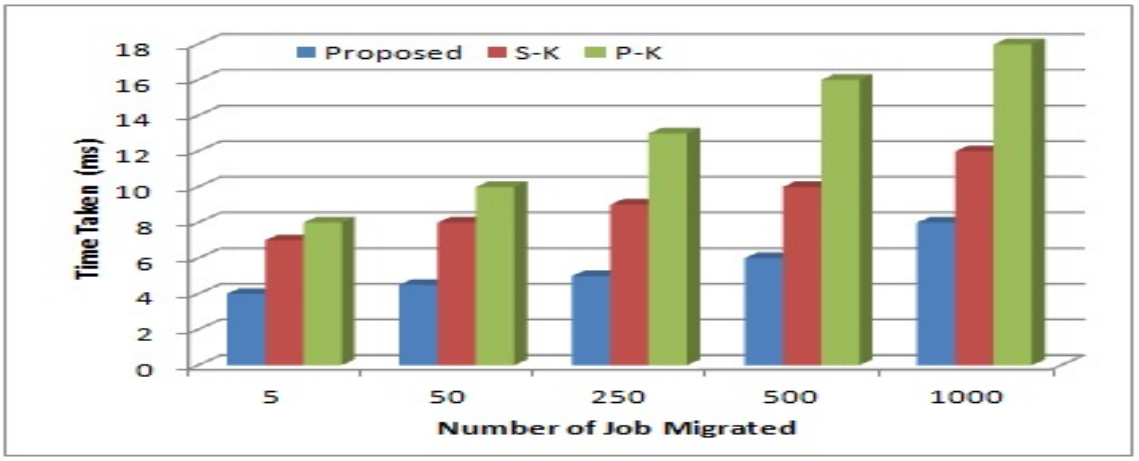


FIGURE 5.11: Time Taken with Number of Job Migrated

Figure 5.11 indicates that the differences between the performance of proposed and existing algorithms (S-K and P-K) [144] [145] are low when the migrated jobs are low. As the number of migrated jobs increases, the difference also increases. When the migrated job reaches 1000, the difference is almost twice between proposed and existing P-K algorithm. Figure 5.12 presents the performance of algorithms on 4, 6 and

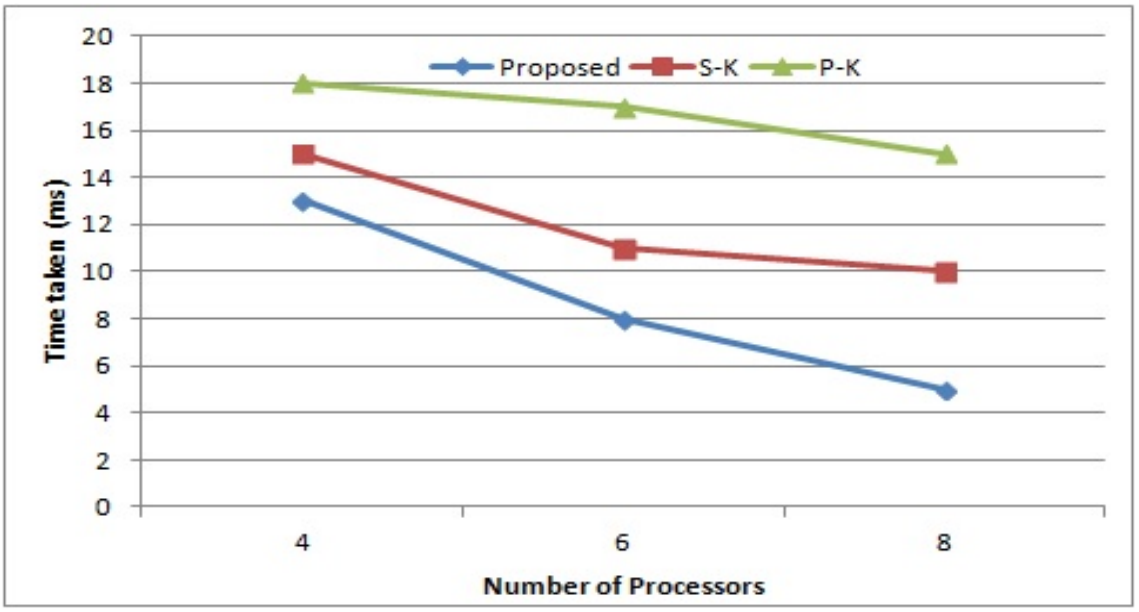


FIGURE 5.12: Time Taken with Number of Processor

8 processors. Assume that the number of migrated jobs is constant, n=1000 during checkpointing. When numbers of processors are low, the time taken between proposed and existing algorithms is low. As the number of processors increases by the time, differences also increase. This simulation result shows that proposed checkpoiniting technique has significant potential to improve performance.

5.2.2 Job Migration with Scheduling

The simulation results assumed that each machine executes a single task at a time.

The proposed algorithm is compared with the JM algorithms Local and Global algorithm for both static and dynamic scheduling. Figure 5.13 and the makespan of the proposed JM system and the existing systems Local and Global respectively. Here, Hit ratio represents the number of jobs successfully completed within client limitation. Figure 5.14 represent the hit ratio comparison for the three algorithms. The average percentage of hit ratio, which decides the efficiency of proposed JM, represented in Figure 5.15. The average percentage improvement of five different sets of tasks and resources based on makespan over Local is 1.86 % and over Global is 14.03 %. Based on a hit ratio, the average percentage improvement is 2.69 % over Local and 10.5 % over Global.

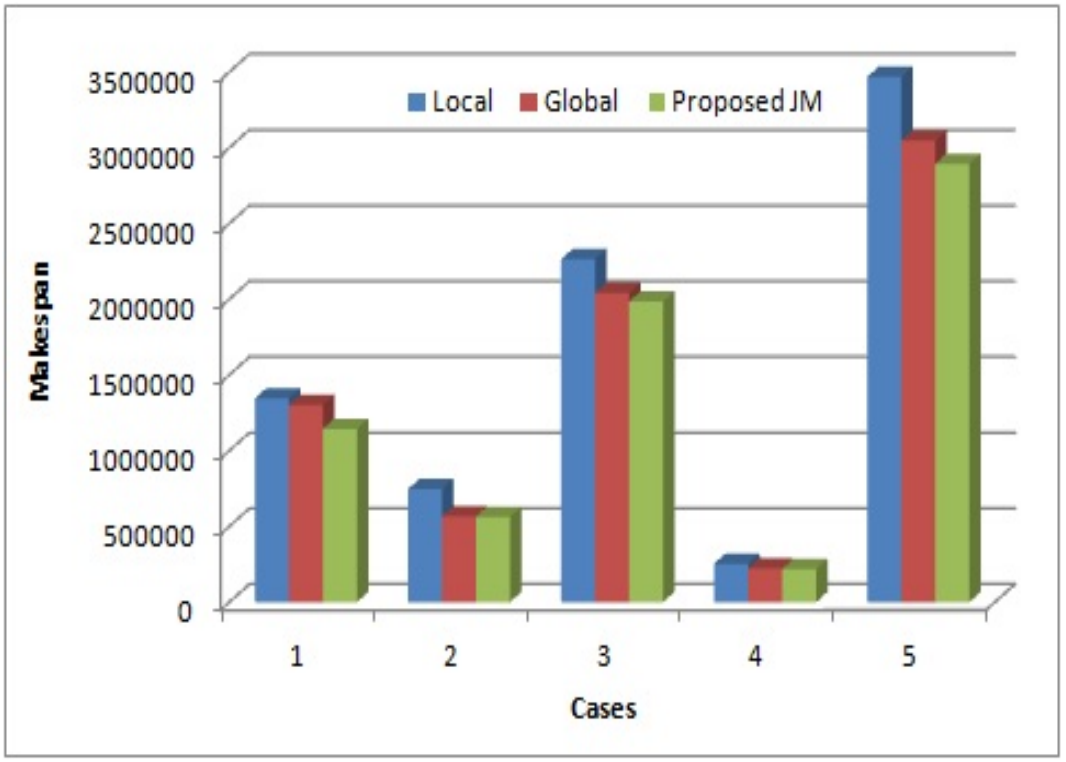


FIGURE 5.13: Job Migration Makespan

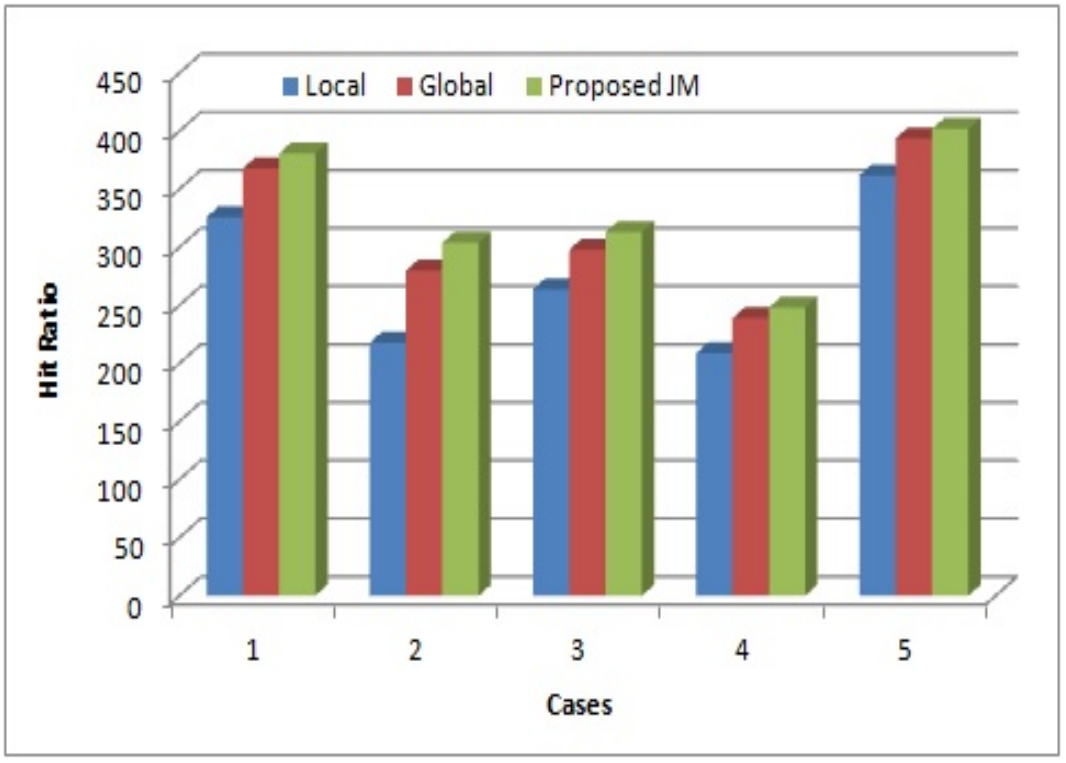


FIGURE 5.14: Job Migration Deadline Hit Ratio

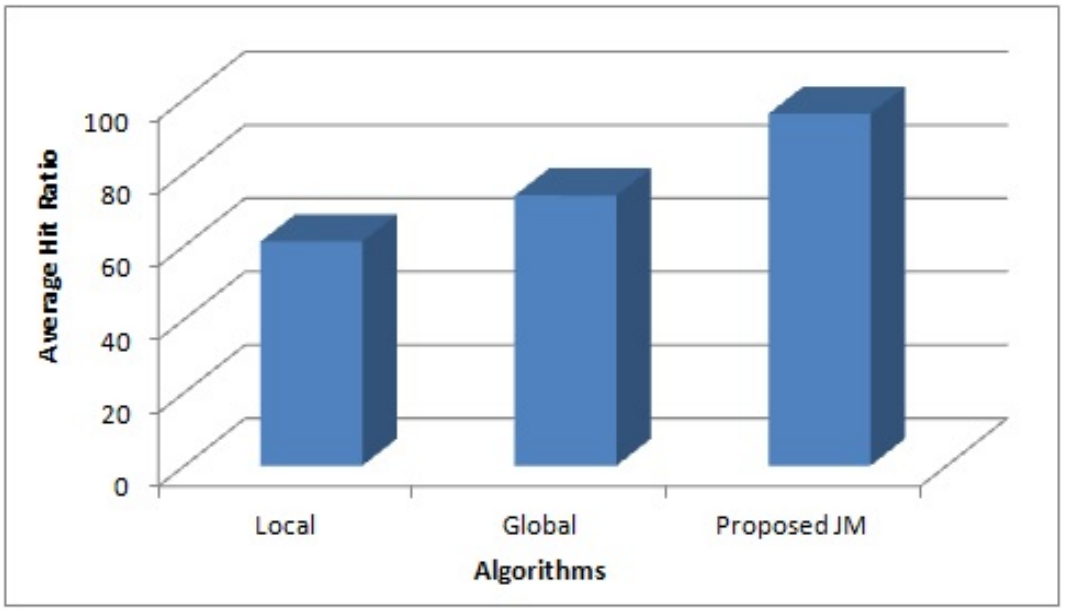


FIGURE 5.15: Job Migration Average percentage of Hit Ratio

5.3 Conclusion

Load Balancing Algorithm through variable threshold value and Job Migration technique have been implemented and simulated on GridSim toolkit. Then the experimental results have been discussed from various perspectives such as effect of resource allocation capacity, response time, heterogeneity, number of applications, number of resources, etc.

Chapter 6

Case Study: Load Balancing and Job Migration in Social Grid Environment

The previous chapter discussed the implementation of LB Algorithm and JM techniques implementation. Then the experimental results have been discussed from various perspectives such as effect of resource allocation capacity, response time, heterogeneity, number of applications, number of resources, etc.

The application of Grid Computing is wide spread to other computational areas such as communication devices and social networks. Mobile Grid computing is a computing environment that extends Grid computing to include communication devices such as laptops, net books, tablets, smart phones, etc. In this chapter, a replication based QoS scheduling using Hash Table Functionality (HTF) in Social Grid Computing (SGC) is introduced. It consists of four sub-scheduling algorithms: Unauthorized-user filtering, Grid service delivery, QoS provisioning, and Replication and load-balancing. Under the proposed scheduling, a device is used as a resource for providing Grid services, faults caused due to reason like user mobility are tolerated and user requirements for QoS are considered. The Simulation include the scheduling with and without HTF.

This chapter initially discusses various existing methods to tolerate faults and support QoS for users in many computing environments. Then, describes the system environment required for the proposed method. It also examines HTF, system architecture and data structure for SGC. Then, proposes fault tolerance based QoS scheduling algorithm for SGC consisting of four sub-algorithms is presented. Next, the simulation results in SGC are discussed. Finally, the chapter summarizes the results and discusses the various avenues for future research.

6.1 Introduction

In Grid computing, each communication device should have a suitable application for accessing the Grid. Each device can act as a server to provide Grid service or data as well as a client to request Grid service or data [13, 149, 150]. In this case, these devices can be used as a resource in Grid environments. However, devices can move freely in the network according to their properties, which may cause network disconnection. Moreover, they may run out of power, resulting in a sudden disconnection from the network and subsequent service suspension or cessation. Therefore, if devices are to displace computing servers, this inherent problem of the device Grid must be overcome and proper Job Migration is also required. A social network is a social structure composed of individuals (or organizations) called "nodes", which are connected by one or more types of interdependencies, such as friendship. Essentially, a social network is a dynamic VO with inherent trust relationships among friends called ad SGC.

The contributions in this chapter are as follows [151]:

1. A Scheduling algorithm has been proposed to support device users and share Grid services directly among the users.
2. The proposed algorithm tolerates faults in SGC. The faults may bring loss of user request to the devices and these faults may include disconnection of network, drainage of the battery, and other failures due to device movement.
3. A scheduling algorithm is proposed to provide QoS to the users in SGC. QoS metrics are first classified for the Social Grid and then apply the QoS metrics to SGC.
4. This algorithm is unique because it supports user mobility and QoS concurrently while considering device characteristics, and also supports Grid sharing services among device users without any authentication by using social networking.

6.1.1 Fault Tolerance

Studies on the methods to tolerate various faults that can occur during a service have been conducted in diverse research areas. Yi Hu et al. [152] proposed a security-aware and fault tolerant job scheduling strategy for grid computing. The scheduling strategy includes JRT (Job Retry), JMG (Job Migration without checkpointing), and JCP (Job Migration with Checkpointing). The conclusion made was that the JRT strategy has the most optimal system performance improvement for small jobs and that the JCP strategy

leads to the lowest performance improvement. Unfortunately, the checkpointing method was not described clearly. Hyunjoo Kim et al. [153] introduced server selection schemes for a service migration-based fault-tolerant streaming on P2P computing.

6.1.2 QoS Scheduling

Various classification for QoS and QoS provisioning methods have been studied to meet diverse user needs for services. QoS is defined in different ways depending on the research area. Qian Tao et al. [154] defined a `Grid_Services_QoS` including `Basic_QoS` Set and `Extended_QoS` Set and proposed a `T_QoS` (trustworthy QoS) computing algorithm. `Basic_ QoS_Set` considers time and cost, and `Extended_QoS_Set` includes reliability, availability, security, and status. Kyle Chard et al. [155] defined a social Grid as a resource and service sharing framework utilizing relationships established between members of a social network. The social Grid allows users to share heterogeneous resources with low privacy concerns and security overheads by utilizing the relationships in the computing environment.

6.1.3 HTF

Most researches focus the utilization of HTF [156–159] in P2P networks. There are only a few studies on HTF applicable to computing environments [160, 161].Anuchart and Guang [162] introduced a new framework for the self-organizing and self-healing certificate authority (CA) group in HTF that provides certificates without a centralized trusted third party in P2P networks. Henry et al. [163] introduced enhancement strategies to HTF for computational grids that provide transparent data distribution and replication. For enhancement to HTF, HTF was chosen as a basis for distributed storage algorithm because of its enormous flexibility. Roger et al. [164] documented an approach to support spatial data management over structured P2P systems. It is extending HTF virtual space to physical spatial space and introduced a new hash function for mapping spatial data objects onto nodes over a modified HTF system. The key of modified HTF were identified for each data object as a bit string. However, the proof of this has not been reported.

6.2 System Environments

6.2.1 Social Grid Computing (SGC)

SGC refers to both

i. Social networks and device Grid computing and

ii. Social Grid computing and devices.

SGC can be defined as follows:

SGC is a computing model that includes devices to support user mobility. It connects with social networks to reflect real world user relationships, and therefore provides and shares Grid services directly among the members of a social network.

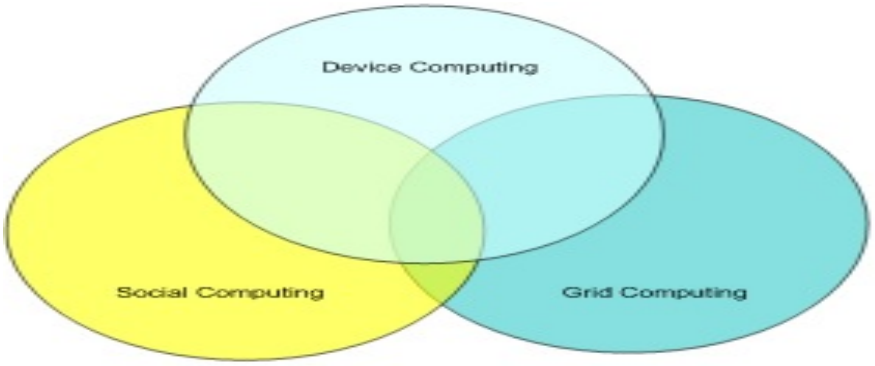


FIGURE 6.1: Global view of SGC

The SGC, assumed as the basis of this proposed work, can be described as follows:

- SGC consists of a number of devices for requesting Grid services and computing resources for providing Grid services.
- Devices in SGC create social networks based on real world human relationships among device users.
- Members of a social network share Grid services based on basic authentication on the social network without any further authentication.

Figure 6.1 presents a global view of SGC in various computing environments. Figure 6.2 depicts Grid service utilization in SGC. User1 and User2 are on the same social network and User2 has Grid service or data. When User1 requests Grid service to a Grid server, the Grid server returns User2s device information.

Ultimately, User1 and User2 are connected and share the Grid service. Therefore, User1 does not have to connect to a distant Grid server.

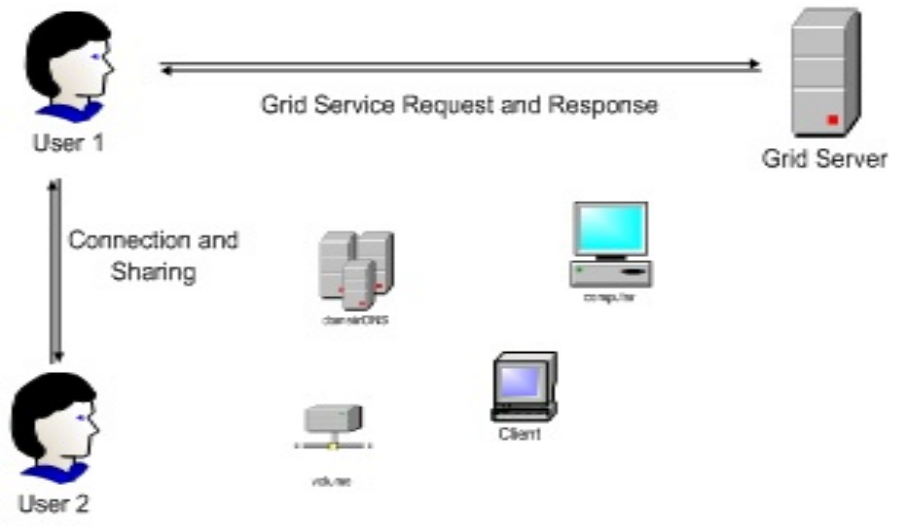


FIGURE 6.2: Grid Service Utilization in SGC

6.2.2 HTF (Hash Table Function)

HTF [156, 157] is a space partitioning mechanism that has been used as a base approach for large scale data management of frequent moving objects in various computing environments: Distributed Hash Table (DHT) and key-based routing protocol, etc. [159, 160]. In this thesis, HTF structure is used not only to improve the performance of computing environment but also to represent and manage network space consisting of devices, as well as Grid servers, for the following reasons:

- As HTF represents distributed infrastructure, it is suitable for social Grid environments consisting of multiple distributed devices and servers.
- HTF is able to represent a computing environment in which devices join and leave networks freely. A key in HTF may be a device, and HTF operations like insert and delete can be matched with joining and leaving of devices in SGC.
- HTF maintains a hash table using (key, value) pairs for lookup. These pairs can be mapped (Grid service, device) to determine the proper node for providing and sharing Grid services in SGC.
- HTF is used in peer-to-peer file sharing systems, in which data is stored in end-user devices rather than on a central server. The system is consistent with social network environments that share multimedia data with friends, family and others. Therefore, HTF is also used in SGC.

6.2.3 Architecture of SGC

SGC, based on wired servers, includes communication devices to support device users requesting Grid services in the network. Device users use devices such as Smartphones, laptops, PDAs, etc. that utilize device and wireless networks. These devices can communicate with wired computers and other devices through an AP (Access Point). There are multiple Grid servers in SGC and each device can access a Grid server. Grid servers can provide services to several devices. Every device has a main Grid server with which the device is registered after joining the network. Devices transmit their information to the main Grid server periodically.

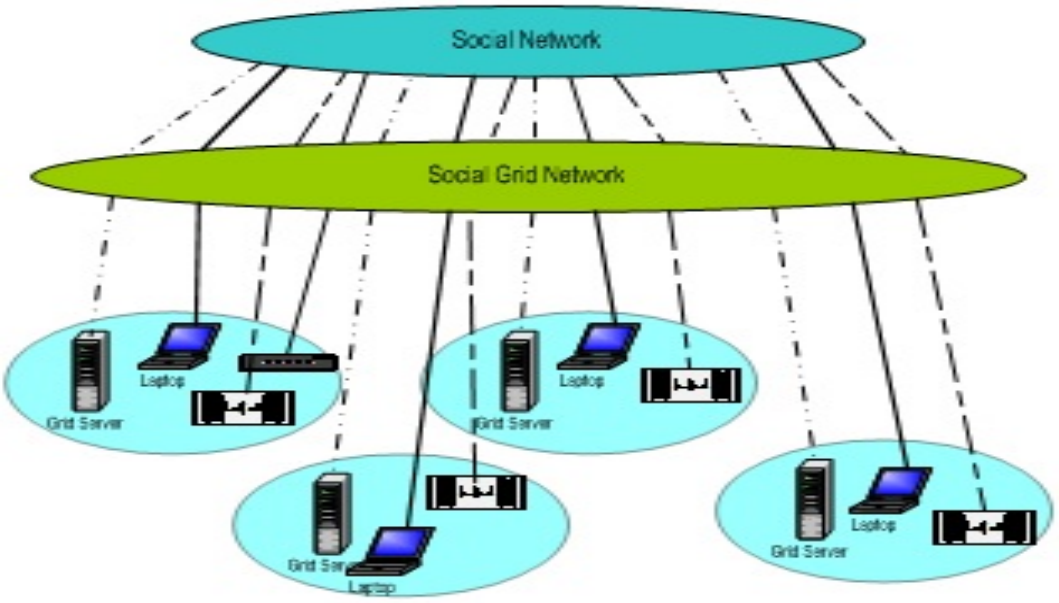


FIGURE 6.3: Architecture of SGC

Device users can be a member of a social network by using real world relationships and the members of a social network can share Grid service or data without further authentication. Grid servers as well as devices provide Grid services to the other members of the social network, so devices act as the resources.

Grid servers and devices have their positions determined by GPS in the network and can recognize the other's position through a main Grid server. The knowledge of, who is a member of their social network, is stored in the main Grid server. Therefore, a device requests Grid services to the closest Grid server or device that is also a member of a social network. This should result in improved service response time as it allows the sharing of Grid services without any further authentication.

Figure 6.3 depicts the architecture of an SGC environment. There are some Grid servers and devices connecting to an AP devices to form a social network with other devices

and Grid servers. Grid server can be a member of every social network to provide Grid services. Every Grid server has a HTF structure to manage the devices. Every device is registered on HTF in the Grid server and is mapped on a point of HTF having a virtual logical address, namely HTF coordinates, for HTF routing.

6.2.4 Preliminary

In SGC, a device can be used as a resource, so the device requests and provides Grid service using a resource ID. Resource ID is managed by the resource along with GPS-based coordinates (x, y), AP_id, current and neighboring zone IDs of HTF, social network ID, and Grid service IDs. GPS-based coordinates are used to search for a resource in a neighboring Grid server and compute the distance between the resource and a requested resource. Zones of HTF are used to look up any resource on Grid servers.

TABLE 6.1: Information Stored in a Resource

Information/Function	**Variables**
`Id_resource`	Axis_x, Axis_y
`Id_accesspoint`	
`Id_current_zone`	`Id_neighboring_zone`
`Id_socialnetwork`	`Grid_services`

TABLE 6.2: Stored Information on Grid Server

Information/Function	**Variables**	**Sub-Variable**
Resource	`Id_resource`	
	Status	
	`Res_capability`	
	Cost	
	`Id_socialnetwork`	
`Grid_services`	`Id_service`	
	Time	
	Frequency	`Id_providers`
Socialnetworks	`Id_socialnetwork`	Nodes

Grid service ID refers to the ID of a Grid service that the resource can provide to other resources. This information is periodically sent to the main Grid server. Table 6.1 shows information stored in a resource. Each Grid server stores and manages resource information, Grid Services and Social network information. The information stored in a Grid server is summarized in Table 6.2. When a Grid server receives a request for Grid service, the server looks for the most proper resource to provide the service from. For Grid service delivery, the algorithm to discover the most proper resource is introduced in the next Section 6.3.

6.3 Proposed Replication based Job Migration Algorithms

Replication based Job Migration Algorithms for SGC have been presented below:

6.3.1 Stop Unauthorized User

Requesting and retrieving of the Grid information should be transparent for all users in the network. For an efficient system, mutual communication should be there among users but for unilateral service or data flow of the network, i is important. The unauthorized user only uses Grid services from other users or is restricted from providing Grid services to others. Because unauthorized users reduce Grid service quality, thus, they were not considered from SGC network configurations. To do this, device status is used, which depends on someone's behaviour in the sharing of Grid services previously. If his behaviour is unauthorized, then that person will have a low status. The status of a device is defined as follows [165]:

6.3.1.1 Fitness Function for Device Status

Status is the probability of a device that can be assessed by other communication device on the basis of feedback. Status in an SGC network is used to determine whether a user of the device is unauthorized or not.

An SGC network is represented by HTF. Devices join the network by communicating with a randomly selected node in HTF and then managed by a main Grid server. The main Grid server manages the device after the device joins HTF. The main Grid server sends a unique resource identifier (`Id_resource`) and tests application to a newly joined device. After some time, the main Grid server requests the result of the test application to the device. Unauthorized users may not return the result, return an error result, or return part of the result. Namely, the test application is used to calculate the status value of a device and determine whether the user is unauthorized or not. After the device transfers the result of the test application to the main Grid server, the main Grid server calculates the status value of the device by comparing the original answer of the test application and the result received from the device. If the status value of the device is below a specified threshold, the main Grid server evaluates the device as unauthorized.

A series of random matrices is used for the test applications. The initial status, $status_0$, of a device can be attained as shown in Equation 6.1, in which, m is the number of matrices in a series and n is the number of matrix element in a matrix. The expression Solution Status SS_{ij} represents a value of the Answer Matrix AM_{ij} element for the test

application in a main Grid server, and RM_{ij} represents a value of the Result Matrix element that is calculated in the device and submitted to the main Grid server.

$$Status_0 = \sum_{i=1}^{m} \sum_{j=1}^{n} SS_{ij} \tag{6.1}$$

$SS_{ij} = 0 (AM_{ij} \neq RM_{ij})$
$SS_{ij} = 1 (AM_{ij} = RM_{ij})$

If the device is reliable, the status value is high. Conversely, lower the status value, the higher the probability of unauthorized user. If the initial status, $status_0$ of a device is below a certain threshold, the device may not be connected to that particular SGC. That means the device can neither request nor provide Grid service in the network. Because the status of a device constantly changes, status is recalculated periodically by Equation 6.2 to consider changes. The notation w is a weighted value for current status ranging from 0 to 1. Also, if the status, $status_i$ of a device is below the threshold, the device can neither request nor provide Grid service.

$$\left.\begin{array}{ll} status_{i+1} = status_i & (if \quad i = 0) \\ status_i * w + ststus_{i-1} * (1 - w) & (if \quad i >= 1) \end{array}\right\} \tag{6.2}$$

The average status of a device for some duration is calculated by Equation 6.3, where n is the number of calculations in the duration. The average status is used for applying device QoS to select a proper resource.

$$avg\_status_{RM} = \sum_{i=0}^{n-1} \frac{status_i}{n} \tag{6.3}$$

6.3.1.2 Network Assembly for Algorithm

Through status, a device that has just joined the network can be evaluated as to whether it may construct the network or not. If the initial status is less than the threshold, the device is excluded from the construction of the network for a particular time unit, which means the device can neither obtain nor provide Grid service. If the main Grid server does not receive a result from the device, the main Grid server sets the status of the device to the minimum value, labeling the device as unauthorized. After some time unit, the main Grid server recalculates the status of the device. Algorithm 13 shows the algorithm for SGC network construction in a main Grid server.

Description: Send a unique id_resource and test application to the newly joined device;

for *(a unit time)* **do**
  Request result of the test application to the device;
  if *(receive results from the device)* **then**
    Compare the correct answer with the received result;
    Compute an initial status of the device using Equation 6.1;
  else
    Set the status of the device to minimum value;
  end
  for *(some unit time)* **do**
    Recalculate the status of the device using Equation 6.2;
  end
end

Algorithm 13: Construction of Network

6.3.2 Basic Algorithm for Grid Service Delivery with Social Network

When a Grid server receives a request for Grid service from a device, the server first checks whether the device and the Grid server are in the same network area, namely the same AP area. If so, the Grid server directly provides Grid service to the device. If not, the server searches for a proper resource that has the requested Grid service and is managed by the Grid server's HTF. The Grid server puts searched resources into the first candidate group. Then, the Grid server checks the social network of the device and selects resources belonging to the social network of the device, that is, the same social network among the first candidate group.

These selected resources are included in the second candidate group. If there are no proper resources in the second candidate group, the server may then transfer the request to the closest neighbouring Grid server with the id_resource of the device. HTF based-coordinates are used to search the proper resource in a Grid server, and GPS based coordinates used to identify the closest neighboring Grid server.

In sequence, the Grid server selects a proper resource by calculating the Euclidean distance between the two resources, device a requesting Grid service and resource B included in the second candidate group, in Equation 6.4 If the coordinates of the device A is A (X1, X2), then the coordinate of resource B is B (Y1, Y2).

$$Distance_{XY} = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2} \tag{6.4}$$

After searching for the most appropriate resource, the Grid server returns its id_resource to the device requesting Grid service. When the requested Grid service is completed,

```
Description:Delivery_service (id_service)
if (receive a request for grid service (id_service)) then
    id_frequencyservice= id_frequencyservice +1;
    Check the location of the device requesting a grid service;
    if (access point area is same) then
        id_properresource= id_server;
    else
        id_properresource=resorcediscover (id_device, id_service);
    end
    Return id_properresource;
else
    for (a unit time) do
        id_Frequencyservice=id_frequencyservice-1;
    end
end
if (receive a grid_service_finish_message from the device (id_resource)) then
    Compute id_statusresource;
end
```

Algorithm 14: Grid service delivery

the device informs the Grid server. Finally, the Grid server updates the status of the proper resource and the frequency of the requested Grid service. If the resource does not provide Grid service to the device, the status of the resource will decline.

```
Description: Search resource managed by the grid server and having id_service;
Put resources into the first candidate group; Discover_resource (id_device,
id_service)
for each resource into the first candidate group do
    Check their id_socialnetwork
    if same social network with the device then
        Put searched resource into the second candidate group;
        for each resource in the 2nd candidate group do
            Compute distance between the device and the resource using Equation
            6.4;
        end
    end
end
Select one proper resource with minimum distance from the second candidate
group;
Return id_resource of the proper resource;
```

Algorithm 15: Grid service discovery

In addition, if the frequency of a Grid service is greater than the threshold, the Grid server replicates the Grid service to another neighboring Grid server according to the replication algorithm in Section 6.3.4.

Algorithm 14 shows the basic algorithm for Grid service delivery in a Grid server and Algorithm 15 shows the basic algorithm for Grid service discovery in a Grid server.

6.3.3 QoS Provisioning

The QoS metrics for SGC are proposed to consider the user needs and the device characteristics.

6.3.3.1 QoS Metrics for SGC

QoS metrics are classified into common QoS and device QoS. Common QoS refers to general QoS parameters for most of the users and includes time and cost. Time is the task processing time for executing a task, and cost is task processing cost for using a Grid resource. Device QoS denotes specialized QoS on a device and includes status and resourceability. Figure 6.4 shows QoS metrics for SGC. Status is already described in Section 6.3.1 and is subjected to a device usage patterns based on historical information. Average status is used by Equation 6.3 for QoS.

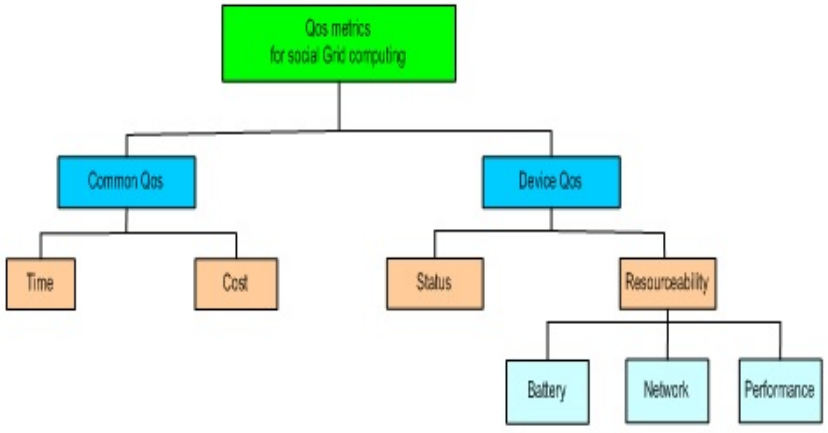


FIGURE 6.4: QoS Metrics for SGC

6.3.3.2 Calculation of Resourceability

Remaining battery power, network status, and performance of a device are the metrics of Resourceability. It prevents faults that can arise from the movement or other error of devices during execution. Resourceability, used as a criterion to select the most proper device, consists of battery power, network status and performance. These three variables are independent and affect resourceability; therefore, multiple regression analysis is used. The resourceability value, ranging from 0 to 1, of a device is computed through Equation

6.5. Variables x1, x2, and x3 represents remaining battery power, network status, and performance of a device, respectively. Parameters $\beta_0, \beta_1, \beta_2 and \beta_3$ mean the regression coefficients and Îţ means the error rate of multiple regressions.

$$Resourceability = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3 + \in \tag{6.5}$$

The higher the resourceability value, the higher will be the probability of the device to be the most proper resource, because a device with low remaining battery power or poor network status cannot be used as a resource, if the value of x1 and x2 is less than or equal to the threshold, then the resourceability value of the device is set to zero.

6.3.3.3 QoS Provisioning Algorithm

Description:Sevice_delivery (id_service)
if *(Receive a request for grid service (id_service))* **then**
  $Frequency_{id\_service} = Frequency_{id\_service}$ +1;
  Check the location of the device requesting a grid service;
  if *(Access point area is same)* **then**
    id_properresource= id_server;
  else
    id_properresource=discover_resource (id_device, id_service);
  end
  Return id_properresource;
else
  for *(A unit time)* **do**
    $Frequency_{id\_service} = Frequency_{id\_service}$ -1;
  end
end
if *(Receive a grid_service_finish_message from the device (id_resource))* **then**
  Compute status_id_resource;
end

Algorithm 16: QoS algorithm in a Grid server for service delivery

If a user wants to use QoS for Grid service, the user requests Grid service through the QoS metrics. Namely, a device user requesting a Grid service can set the QoS metrics. If QoS metrics including common QoS and device QoS are set, the QoS algorithm is applied instead of the basic algorithm for Grid service delivery, as seen in Section 6.3.2.

When a Grid server receives QoS metrics from a device, the Grid server applies the QoS metrics to SGC. Firstly, the Grid server checks whether the device and the Grid server are in the same network area, as in Section 6.3.2. If so, then the Grid server provides the Grid service to the device directly. The reason is that a Grid server is unaffected

by advice's remaining battery power, network status and performance. If these device parameters are good, the Grid server is included in every social network. If the two are not in the same network area, the server searches for resources, along the lines of Section 6.3.2.

```
Description:Search resources managed by the Grid server and having id_service;
Put resources into the 1st candidate group;
Discover_resource (id_device, id_service)
for (Each resource into the first candidate group) do
    Check their id_socialnetwork ;
    if (Same social network with the device) then
        Put searched resource into the second candidate group;
        Apply common Qos to the 2nd candidate group;
        for (Each resource satisfying common QoS) do
            Put resources into resource group R_tc ;
            Apply device QoS to R_tc ;
            for (Each resource satisfying device QoS) do
                Put resources into resource group R_rr ;
                Rank resources in group R_rr and select one proper resource;
                Return id_resource of the proper resource;
            end
        end
    end
end
```

Algorithm 17: QoS Algorithm in a Grid Server for Resource Discovery

After obtaining the second candidate group, the Grid server applies common QoS to the second candidate group and obtains resource group R_{tc} as a result. Namely, resource group R_{tc} is a set of resources satisfying time and cost among the second candidate group.

$$R_{tc} = devices\ satisfying * \frac{(QoS_{time}\ *\ QoS_{cost})}{second\ candidate\ group} \tag{6.6}$$

If resource group R_{tc} has only one resource, the Grid server returns the id_resource of the resource to the device. If resource group R_{tc} includes more than one resource, the Grid server applies device QoS to resource group Rtc and obtains resource group R_{rr} as a result. QoS_{status} and $QoS_{resourceability}$ are calculated by Equations 6.3 and 6.5, respectively.

$$R_{rr} = devices\ satisfying * \frac{(QoS_{status}\ *\ QoS_{resourceability})}{R_{tc}} \tag{6.7}$$

If resource group Rrr has only one resource, it becomes the proper resource. If resource group Rrr includes more than one resource, the Grid server ranks the resources in group

Rrr by calculating the distance between the mobile device requesting Grid service and a resource included in Rrr using Equation 6.4. The Grid server selects the highest ranked resource as the most proper resource.

After selecting the appropriate resource, the Grid server returns its id_resource to the device requesting Grid service. When the requested Grid service is completed, the device informs the Grid server about the same. Finally, the Grid server updates the status of the proper resource and the frequency of the requested Grid service, as in Section 6.3.2. Algorithm 16 presents the QoS algorithm in a Grid service delivery and Algorithm 17 shows the QoS algorithm in a Grid service discovery.

6.3.4 Replication and Load Balancing

6.3.4.1 Algorithm for Replication of Service Request

Replication helps to improve the service availability, service completion rate and Load Balancing of the entire network because the request for Grid service can be distributed to replicas.

```
if (Receive a request for grid service (id_service)) then
    Frequency_id_service = Frequency_id_service + 1 ;
    Check the location of the device requesting a grid service;
    if (Access point area is same) then
        id_properresource= id_server;
    else
        id1_properresource=discover_resource (id_device, id_service);
        while (Id1_properresource= Id2_properresource) do
            Return (Id1_properresource, Id2_properresource);
        end
        repeat
        until (Id2_properresource= discover_resource (id_device, id_service));
    end
else
    for (A unit time) do
        Frequency_id_service= Frequency_id_service -1;
    end
end
if (Receive a grid_service_finish_message from the device (id_resource)) then
    Compute status_id_resource;
end
```

Algorithm 18: Algorithm for Replication in a Grid Server

Therefore, Grid service replication is used to minimize waiting time for the request and improve SGC performance. When a Grid server receives a request for Grid service, the

Description:Request for grid service (id_service) to grid service;
if *(Receive 2 id_properresources from grid server)* **then**
  Select one resource (id1_properresource);
  Connect with the resource;
  Ask the resource to provide grid service (id_service);
  if *(There is any fault over grid service)* **then**
    Connect to another resource (id2_properresource);
    Ask the resource to provide grid service (id_service);
  end
end
if *(Finish grid service (id_service))* **then**
  Inform the grid server of finishing of the service;
end

Algorithm 19: Service_replication_request_in_D (id_service)

Grid server looks for two more proper resources and returns their id_resource using the Grid service delivery algorithm or QoS algorithm.

A device requests the Grid service to the resources having the id_resource that have been received from the Grid server. If the device cannot connect with one resource or any fault occurs over the Grid service, the device asks the other resource to provide Grid service. Algorithms 18, 19 and 20 present the algorithms for replication of service request in a Grid server and a device, respectively.

1. When a neighboring Grid server receives the notification of replication, it stores the Grid server and set the frequency value of the Grid service to one.
2. After some time unit, if there is no request for a Grid service and the frequency is zero, a Grid server deletes the Grid service.

Algorithm 20: Algorithm for Replication in a Device

6.3.4.2 Algorithm for Replication with Load Balancing

Grid servers check the Grid service frequency periodically. The service is issued and decreases whenever a time unit passes without any request for Grid service. If the frequency of a Grid service exceeds the threshold, the Grid service is deemed to be popular and the Grid server replicates the Grid service to other neighboring Grid servers to balance the load in the Grid. For replication, a Grid server informs a neighboring Grid server to replicate a Grid service and transfers the Grid service.

For replication, a Grid server informs a neighboring Grid server to replicate a Grid service and transfers the Grid service. Algorithms 21, 22, 23 present the algorithm for

```
Description:Replication_with_loadbalancing (id_service)
if (Receive a request for grid service (id_service)) then
    Frequency_id_service = Frequency_id_service + 1 ;
    Check the location of the device requesting a grid service;
    if (Access point area is same) then
        id_properresource= id_server;
    else
        id_properresource=discover_resource (id_device, id_service);
        while (Id1_properresource= Id2_properresource) do
            Return (Id1_properresource, Id2_properresource);
        end
        repeat
        until (Id2_properresource= discover_resource (id_device, id_service));
    end
else
end
for (A unit time) do
    Frequency_id_service= Frequency_id_service -1;
end
if (Receive a grid_service_finish_message from the device (id_resource)) then
    Compute status_id_resource;
    if (Frequency_id_service >=threshold) then
        Id_neighbor_server=discover_neighbor_server (id_service, id_service);
        Replication (Id_neighbor_server, id_service);
    end
    for (A some time unit) do
        if (No request of id_service and frequency_id_service=0) then
            Delete the grid service (id_service);
        end
    end
end
```

Algorithm 21: Algorithm for Replication with Load Balancing Grid Server

```
Description:Discover the closest neighboring server from grid server (id_server);
if (A neighboring server does not have the grid service (id_service)) then
    Return the neighboring server ID;
else
    Discover another neighboring server;
end
```

Algorithm 22: Discover Neighbour Server

replication supporting Load Balancing in a Grid server and the neighboring Grid server, respectively.

```
Description:Replication_with_loadbalancing_in_neighbor_server (id_server)
if (Receive replication message from a server (id_server)) then
    Store the grid service (id_service);
    Frequency_id_service=1;
end
for (Some unit time) do
    if (No request for id_service and frequency_id_service=0) then
        Delete the grid service (id_service);
    end
end
```

Algorithm 23: Algorithm for Replication with Load Balancing in Neighboring Server

6.4 Performance Evaluation

GridSim [17] [166] is used to simulate a SGC environment, which is defined as a framework for modelling and simulation of Grid computing infrastructures and services.

6.4.1 Experimental Setup

In order to build a network that includes devices, a network topology generator is used that generates realistic Internet topologies. Each result value corresponds to the geographical location of a device. The assumption made here is that there is no data transfer delay between devices belonging to the same AP. In addition, next assumption is that there is no usage error for Grid services between SNS members in the same SNS Group. For the simulation, the configuration for the simulation is set as in Table 6.3

TABLE 6.3: Configurations for Simulation

S.No.	Parameters	Values
1	Number of devices	100
2	Number of VM(virtual machine)s	30
3	Number of Grid services	50
4	Number of AP(access point)s	4
5	Number of SNS(social network service)s	3

The simulation environments are classified into four cases according to: the filtering of unauthorized users, use of SNS, use of user QoS, and use of service replication. According to the algorithm for QoS Provisioning in Section 6.3.3, cases supporting the user QoS but not supporting the SNS concurrently among them are not considered in this paper.

TABLE 6.4: Simulation Cases

CASE	Filtering of unauthorized users	SNS User	QoS	Service replication
Case1	No	No	No	No
Case2	No	No	No	Yes
Case3	No	Yes	No	No
Case4	No	Yes	No	Yes
Case5	No	Yes	Yes	No
Case6	No	Yes	Yes	Yes
Case7	Yes	No	No	No
Case8	Yes	No	No	Yes
Case9	Yes	Yes	No	No
Case10	Yes	Yes	No	Yes
Case11	Yes	Yes	Yes	No
Case12	Yes	Yes	Yes	Yes

That is, these four cases are excluded for the simulation. Table 6.4 shows the twelve cases for simulation according to the fault tolerance [4] and the QoS scheduling. We simulated 30 times for each case.

6.4.2 Evaluation Results and Analysis

To evaluate the scheduling algorithm performance of 50 Grid services, the following criteria is used: average execution time, finish time, reliability and error rate of Grid services. Replication factor is also included, i.e., the number of replicated services is set to 1, for the replication and load-balancing algorithm (Results are shown with or without using HTF).

TABLE 6.5: Comparison of Evaluation Results

CASE	Execution Time		Finish Time		Reliability		Error rate	
	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF
1	602.07	627.5	2097.8	2175.8	0.55	0.52	0.48	0.45
2	873.5	837.1	1637.0	1580.5	0.63	0.64	0.36	0.37
3	506.5	498.5	2166.5	2066.9	0.74	0.76	0.24	0.26
4	612.2	581.8	1602.8	1675.2	0.74	0.73	0.27	0.26
5	357.3	353.3	1423.3	1480.5	0.78	0.77	0.23	0.21
6	402.7	404.7	1262.8	1332.2	0.75	0.76	0.20	0.21
7	536.5	517.6	2275.8	2114.8	0.76	0.74	0.26	0.24
8	586.0	646.4	1683.0	1775.3	0.73	0.74	0.26	0.27
9	508.5	492.4	2169.2	2121.6	0.92	0.93	0.06	0.07
10	407.6	415.7	1649.1	1642.8	0.94	0.91	0.08	0.06
11	341.8	344.9	1456.0	1501.6	0.92	0.97	0.02	0.03
12	313.7	327.8	1359.5	1340.0	0.93	0.92	0.00	0.00

HTF structure does not affect either fault tolerance or QoS scheduling performance and is only used to make a logical structure of the devices. Table 6.5 compares each case of evaluation criteria. Table 6.6 compares each case of evaluation criteria through a ranking

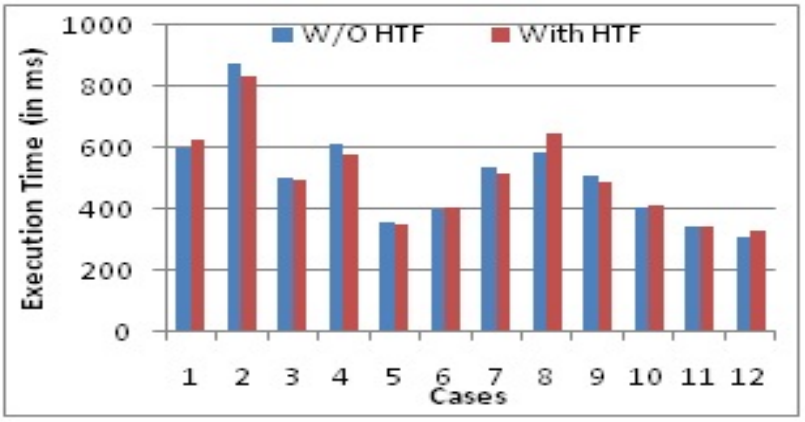


FIGURE 6.5: Evalution Results for Execution Time

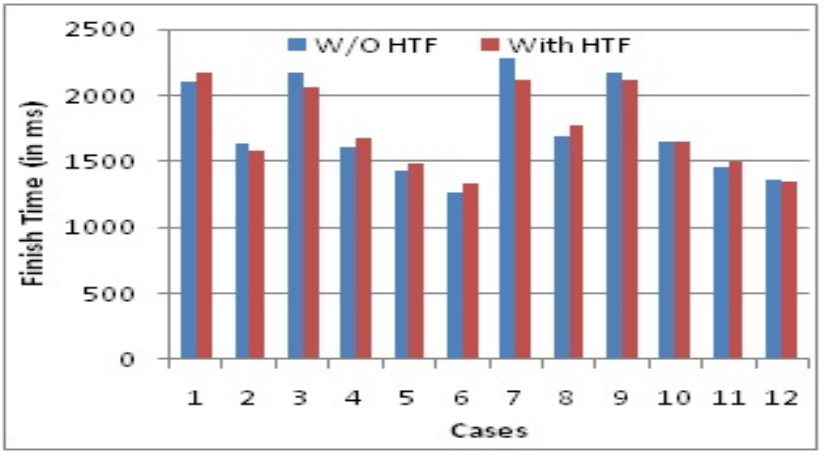


FIGURE 6.6: Evalution Results for Finish Time

TABLE 6.6: Comparison of Evaluation Results in Ranks

CASE	Execution Time		Finish Time		Reliability		Error rate	
	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF
1	10	10	12	9	12	12	12	12
2	12	12	5	6	11	11	11	11
3	7	6	9	10	7	8	8	7
4	9	11	7	5	10	8	9	10
5	3	3	3	3	5	5	5	6
6	4	4	1	1	6	7	6	5
7	8	8	10	12	9	6	7	8
8	11	9	8	8	8	10	10	8
9	6	7	11	11	2	3	4	3
10	5	5	6	7	4	1	3	4
11	2	2	4	4	1	3	1	1
12	1	1	2	2	3	2	1	1

TABLE 6.7: Comparison of Evaluation Result in Grid Services

	Execution Time		Finish Time		Reliability		Error rate	
	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF	W/OHTF	WithHTF
Filtering of unauthorized users	457.47	550.48	1749.35	1718.52	0.87	0.69	0.11	0.29
Social Networking Services	427.38	657.16	1645.10	1957.04	0.84	0.66	0.13	0.33
Users Qos	357.65	577.13	1413.57	1894.12	0.85	0.75	0.10	0.25
Service replication	535.59	472.35	1557.67	1910.2	0.78	0.78	0.19	0.20

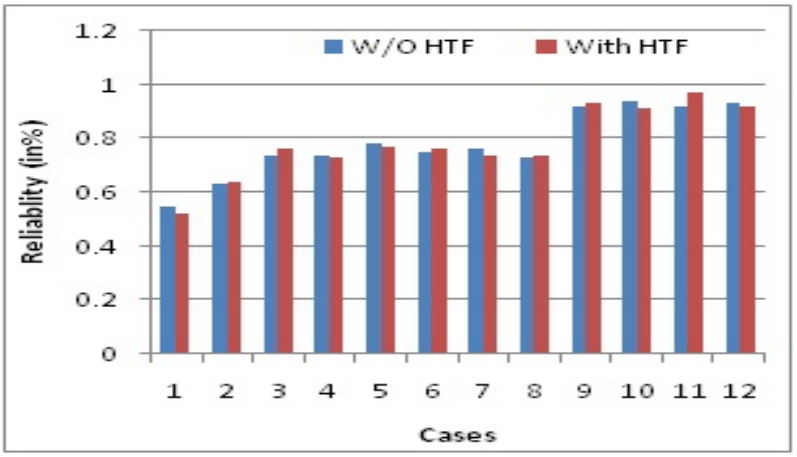


FIGURE 6.7: Evalution Results for Reliablity

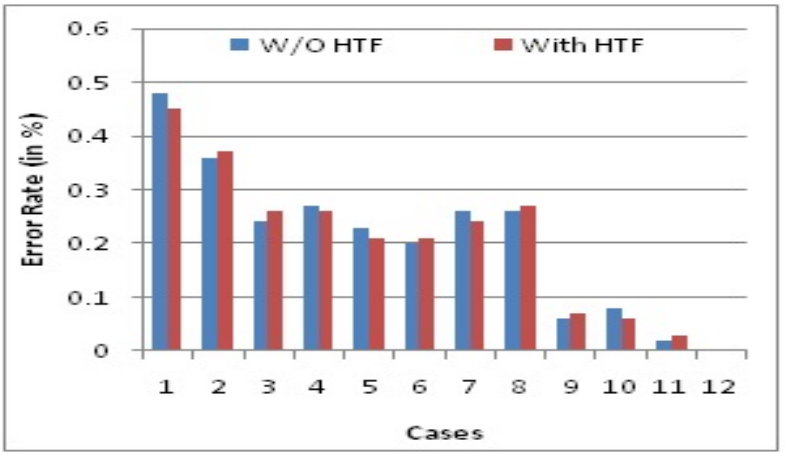


FIGURE 6.8: Evalution Results for Error Rate

from 1 to 12. With the HTF structure, for an average execution time, case 12 is the fastest and case 2 is the slowest. For finish time, case 6 is the fastest.

For reliability, case 10 is the most reliable. For error rate, case 12 is the most reliable. Without the HTF structure, for an average execution time, case 12 is the fastest. For finish time, case 6 is the fastest. For reliability, case 11 is the most reliable. For error rate, case 12 is the most reliable. From here on, because there is little difference between with and without HTF, we consider cases only of evaluation criteria with HTF.

6.4.2.1 Grid Service Execution Time

Under the same cases, conditions that filter unauthorized users and use SNS and user QoS are faster and those replicating Grid services are slower. It seems that filtering unauthorized users using SNS and by providing desired, reliable Grid service among SNS members is given. As there are fewer errors during Grid service, there is no need to re-provide Grid service and therefore average execution time decreases. Conversely,

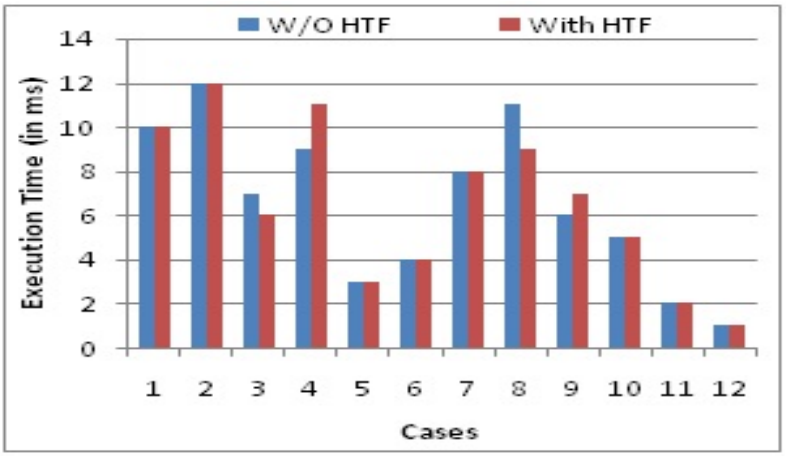


FIGURE 6.9: Evaluation Result in Ranks of Execution Time

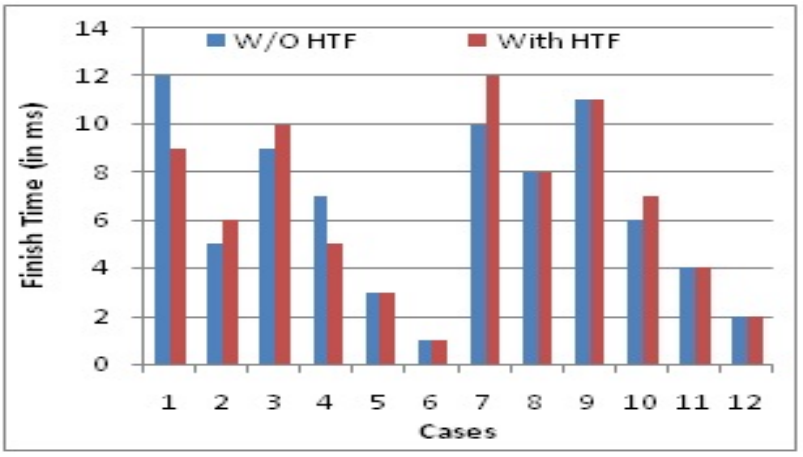


FIGURE 6.10: Evaluation Result in Ranks of Finish Time

service replication is slower because extra overhead, such as selecting a resource and determining better Grid service results, is necessary for replication. Table 6.7 shows the average execution time for each scheduling algorithm according to the use of HTF or not. Figure 6.13 shows the average execution time for each case.

6.4.2.2 Grid Service Finish Time

Under the same conditions, in the cases that use SNS, the user QoS and service replication are faster than those not using SNS. Regarding the filtering of unauthorized users, there is no apparent difference between use and non-use HTF. Using SNS, the user QoS is faster as in Section 6.4.2.1. The service replication using the algorithm is more optimal as during the run, between the original and replication Grid services, one Grid service finishes earlier than the other. Its execution time is set to the Grid service execution time. Table 6.7 shows the average finish time for each algorithm according to the use or not HTF option and Figure 6.14 shows the average finish time for each case.

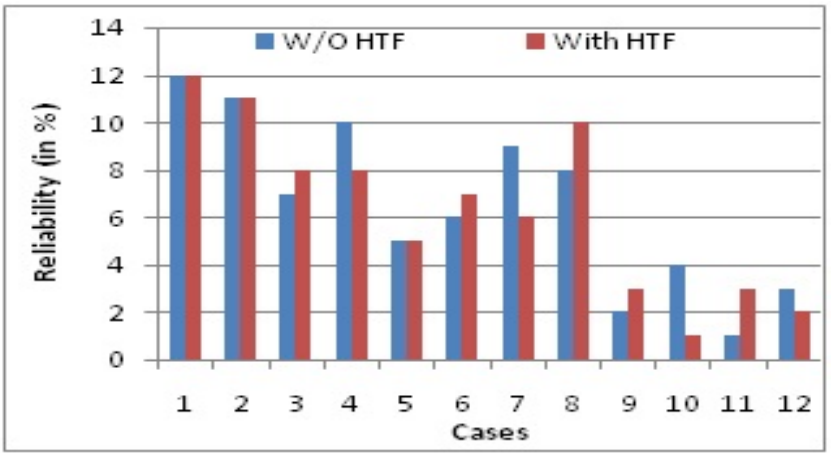


FIGURE 6.11: Evaluation Result in Ranks of Reliability

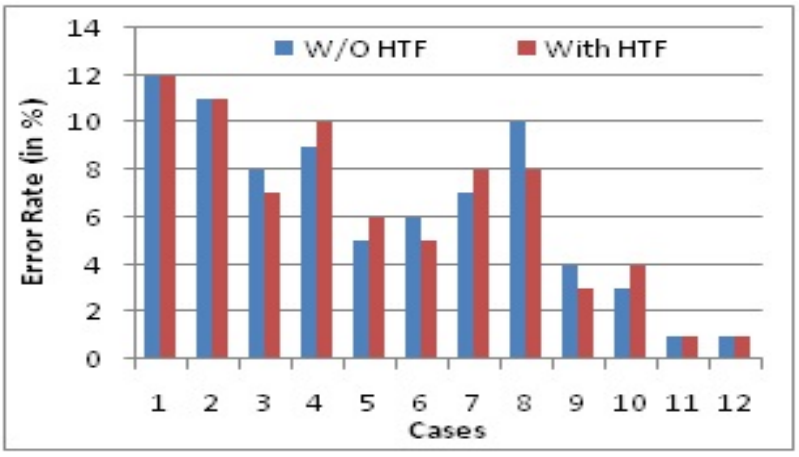


FIGURE 6.12: Evaluation Result in Ranks of Error Rate

6.4.2.3 Grid Service Reliability and Error Rate

Under the same conditions, all cases for each algorithm produce a more optimal result than when the algorithm is not used. In all the cases using each scheduling algorithm, Grid service reliability is improved and simultaneously the Grid service error rate is reduced. Using service replication does not seem to have much effect on either Grid service reliability or the error rate because the replication factor is 1. If the replication factor is increased, the service replication will affect the results. Table 6.7 exhibits average Grid service reliability and the error rate for each scheduling algorithm according to use or non-use HTF. Figures 6.15- 6.16 show the average Grid service reliability and error rate for each case.

6.4.2.4 Evaluation Results with HTF

When considering the four scheduling algorithms for each case, cases 11 and 12 produce the most optimal performance. These cases filter unauthorized users and use SNS and

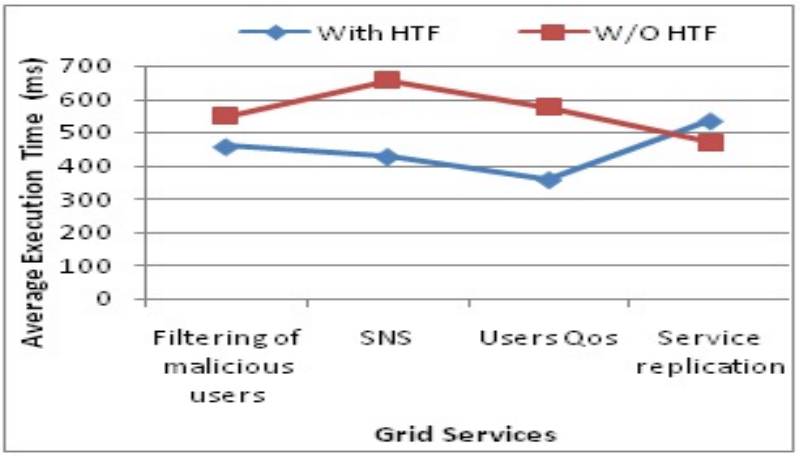


FIGURE 6.13: Average Execution Time for Grid Services

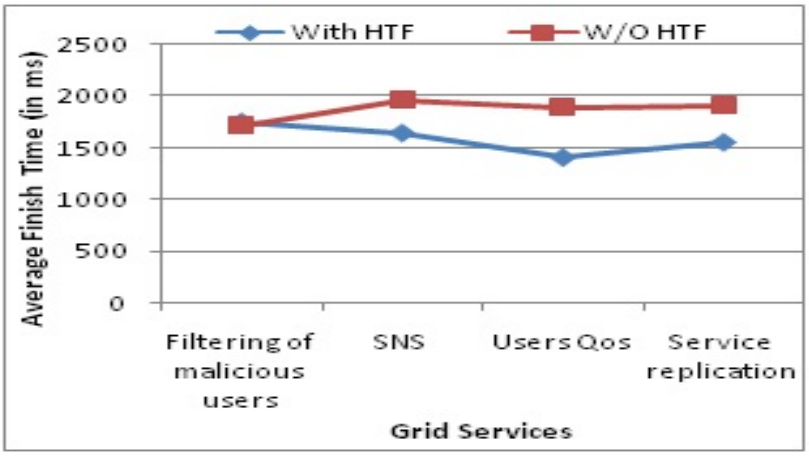


FIGURE 6.14: Average Finish Time for Grid Services

the user QoS. Conversely, the worst performing cases, 1 and 2, neither filter unauthorized users nor use SNS and the user QoS. In other words, filtering unauthorized users and using SNS and the user QoS improve Grid service performance, including Grid service execution time, finish time, reliability, and error rate. Service replication with the high replication factor improves reliability and reduces the error rate.

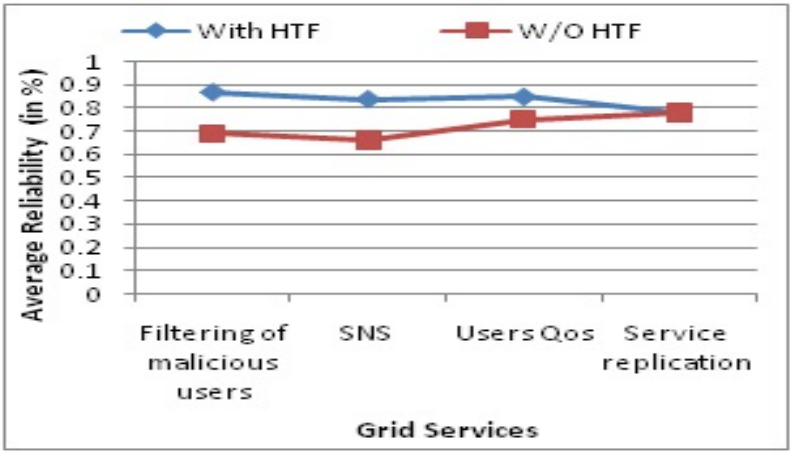


FIGURE 6.15: Average Reliability for Grid Services

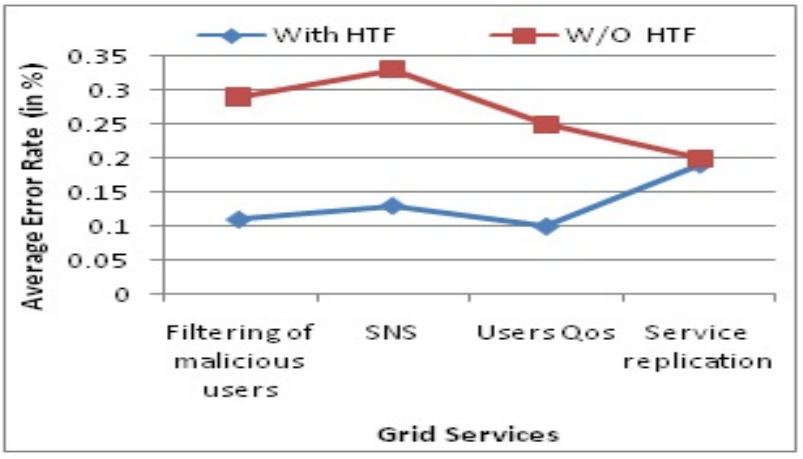


FIGURE 6.16: Average Error Rate for Grid Services

6.5 Conclusion

In this chapter a fault tolerant and QoS based scheduling is proposed using HTF in SGC. SGC is a social network-based Grid computing environment supporting user mobility, user QoS, and sharing of Grid services. A HTF structure, one of distributed network structures, to manage the devices in the computing environment has been used. Fault tolerant QoS scheduling consists of four sub-scheduling algorithms: unauthorized user filtering, Grid service delivery, QoS provisioning, replication and Load Balancing. By using fault tolerance and QoS scheduling, faults arising from the device are tolerated, user QoS needs are considered, and members of a social network share Grid service with other members without further authentication.

The simulation shows that using a SNS improves Grid service execution time and service reliability because members of a social network do not act maliciously. Also, filtering unauthorized users increases Grid service reliability and the service replication increases Grid service execution time and reliability. Therefore, the proposed scheduling algorithms can be applied to improve LB and JM through replication and execution time and reliability in a wide range of computing environments that provide shared services.

Chapter 7

Conclusions and Future Directions

This chapter gives concluding remarks on the thesis by highlighting the main contributions of this research work. Based on the analysis, it can be concluded that with the possible involvement of extremely large number of heterogeneous resources, Load Balancing and Job Migration remains to be a major challenge for Grid Computing. A number of issues related to this have been identified and addressed in this thesis.

A dynamic and decentralized hierarchical Load Balancing algorithm based on variable threshold value has been presented. In this algorithm load is divided into different categories, like, lightly loaded, under-lightly loaded, overloaded, & normally loaded. A threshold value, which can be found out using load deviation, is responsible for transferring the task and flow of workload information. Next, Job Migration techniques and Job Migration selection policy has been proposed and implemented for transferred the job from overloaded node to the underloaded node. Job Migration can be achieved by three procedures such as Check-pointing, Scheduling and Replication. Further, the performance of the algorithms is measured using GridSim toolkit based on variable metrics including response time, resource allocation capacity, makespan, hit ratio etc. Finally, the model, algorithms and techniques have been designed, developed and validated in this thesis.

This chapter concludes the thesis by highlighting the main contributions of this research work. It commences with explaining of the outcome of each chapter and then discusses the contributions of the Load Balancing and Job Migration for Grids. Later, it focuses upon the future scope of the work.

7.1 Conclusions

The thesis, "An Efficient dynamic and Decentralized Load Balancing Technique for Grid" addresses Load Balancing and Job Migration challenges in Grid computing.

The work reported in the thesis is organized in seven chapters. The details are given below:

Chapter 2 presents a literature survey on LB and JM techniques. A taxonomy of the Grid Load Balancing components, policies, strategies, attributes, algorithm's types, classification and hierarchical approach have been presented. A detailed description and comparison of the existing Grid LB algorithm in different environments has been included. The analysis of Job Migration techniques for different environments, Job Migration steps, policies and features has also been discussed. The existing methods involving fault tolerance & user defined QoS fulfillment parameters in several computing & system environments has been depicted. Chapter 3 presents design of hierarchal Load Balancing algorithms based on Variable threshold value. The load is divided into different categories, like, lightly loaded, under-lightly loaded, overloaded, and normally loaded. A threshold value, which can be found out using load deviation, is responsible for transferring the task and flow of workload information. In order to improve response time and to increase throughput of the Grid, a random policy has been introduced to reduce the resource allocation capacity etc.In order to overcome the scalability, resource allocation efficiency and response time a dynamic threshold value has been used.

Chapter 4 presents the design of the Job Migration model and algorithms. For the Job Migration model, decision making Job Migration policy has been proposed for resource selection. Job Migration can be made via checkpointing, scheduling and replication techniques. For choosing one of the three available techniques, Job Migration selection policy takes decision at the time of migration of the load. Chapter 5 explaines the deployment of the model. The experimental results have been illustrated with the help of the test cases. Further, the model has been validated with the existing systems. Validation of the Load Balancing model has been done through variable threshold based dynamic Load Balancing algorithm with the random policy for better response time, reduced communication overhead and improved resource allocation efficiency etc. Job Migration decision has been chosen at the time of migration according to the scenario. This approach actually minimizes the searching time to find out the receiver machine in order to transfer the jobs knows as Gridlets. A rigorous examination of the proposed algorithm on the GridSim simulator has been carried out in order to judge the effectiveness of the algorithm. The simulation results show significant advantages of proposed algorithm over other existing approaches, such as, centralized Load Balancing, no Load Balancing, etc.

Chapter 6 describes the case study based on Job Migration, A JM based replication algorithm has been proposed to support device users and share Grid services directly between users and provide fault tolerance based QoS to the users in the SGC. SGC is a computing model that includes devices to support user mobility. It connects with social networks to reflect real world user relationships, and therefore provides and shares Grid services directly among the members of a social network. To demonstrate the usability of the proposed scheduling algorithm, its implementation and validation has been performed using scheduling with and without HTF. HTF is a space partitioning mechanism that has been used as a base approach for large scale data management of frequent moving objects in various computing environments. The simulation results show that by using all the four algorithms the results have been improved. Therefore, the proposed scheduling algorithms can be applied to improve the execution time and reliability in a wide range of computing environments offering shared services. Chapter 7 finally concludes the thesis and discusses the future scope of the work.

The contributions of this thesis are as follows:

- A detailed analysis of the existing work in the area of Grid Load Balancing and Job Migration has been done along with a detailed study of the scheduling algorithms.
- A dynamic, hierarchical Load Balancing algorithm based on the variable threshold value has been proposed and implemented. Load Balancing threshold parameters have been used for three level Load Balancing model that offers equal workload distribution and faster response time.
- The Job Migration technique with the selection based policy has been proposed to transfer the load from the overloaded the under loaded node, while monitoring the job and pool status. JM selection can be made via check-pointing, scheduling and replication.
- The performance of the algorithms is evaluated using GridSim toolkit, which shows that the proposed LB & JM algorithm outperforms as compared to the existing Load Balancing algorithms in all respects such as effect of resource allocation capacity, response time, Grid service execution time, service reliability, error rate, makespan, complexity etc.
- A replication based QoS scheduling using Hash Table Functionality (HTF) in Social Grid Computing (SGC) has been introduced. It consists of four sub-scheduling algorithms: Unauthorized-user filtering, Grid service delivery, QoS provisioning, and Replication and load-balancing.

7.2 Future Directions

In this thesis, the problem of Load Balancing and Job Migration in the Grid Computing has been addressed.A JM policy for Grid environment considering the QoS parameters has been developed in XML. The proposed techniques are useful in terms of response time, resource allocation capacity, communication cost, makespan etc. However, the work can be further enhanced and extended in future. Some of the future directions are:

- Research into predictive measures of load would be useful. It may be possible to determine future load by examining past requirements and current process properties.
- New methods of dynamic allocation of resources to services by means of migration in order to facilitate the offering of services as an infrastructure available on the internet would be a research area.
- Load Balancing and Job Migration can be considered with migrations in a heterogeneous system consisting of multiple clusters would be an another logical extension in the future.
- It would be interesting to study a larger scale grid where different strategies on the selection of a cluster as a destination for migrating jobs could be examined.
- Job Migration with checkpointing can be enhanced for parallel nodes in the Grid environment.
- Job Migration mechanisms need to be extended to multi-node/Intranet Grid environments on the one hand and to distributed running multi-node jobs on the other hand.
- Job Migration with combination of all three technique Checkpointing & replication & Scheduling is also a direction of research.
- To study the replication factor for improving Grid service reliability and reducing the error rate when using service replications can also be planned for future work.
- In addition, more QoS metrics and additional factors in SGC can be considered.

Bibliography

[1] Krauter K, Buyya R and Maheswaran M, "A taxonomy and survey of grid resource management systems for distributed computing", Softw Pract Exp 32:135-164, 2002.

[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] 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. Rienho, 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.

[4] Rathore N. K., & Chan, I., "Fault tolerance checkpointing algorithm in Alchemi.NET Middleware", In National conference on education & research (ConFR10),Third CSINational conference of CSI Division V, Bhopal Chapter, IEEE Bombay, and MPCST Bhopal, organized by JUIT, Guna (MP) India, 2010.

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] Rathore N. and Chana I., "A cogitative analysis of load balancing technique with job migration in grid environment" In IEEE proceedings paper (pp 77-82). World Congress on Information and Communication Technology (WICT), Mumbai, ISBN -978-1-4673-0127-5, 2011.

[14] 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.

[15] http://www.gridcafe.org/

[16] I. Foster, "What is the Grid? A Three Point CheckList,GRID Today", http://www.mcs.anl.gov/itf/ Articles/WhatIsTheGrid.pdf.2002.

[17] Rathore N. and Chana I., "Report on hierarchal load balancing technique in grid environment", International Journal of Scientific and Innovative Technology, i-manager's Journal on Information Technology, 2(4), 21-35. ISSN Print: 2277-5110, ISSN Online: 2277-5250, 2013.

[18] N Rathore and I Chana, "Load Balancing and Job Migration Algorithm : A Survey of Recent Trends", Wireless Personal Communication, Springer Publication, ISSN print 0929-6212, ISSN online 1572-834X, DOI-10.1007/s11277-014-1975-9, July 2014.

[19] Belabbas Yagoubi, Hadj Tayedlilia and Halima Si Moussa, "Load Balancing in Grid Computing", Asian Journal of Information Tech. 5(10): 1095-1103, 2006.

[20] Belabbas Yagoubi and Yahya Slimani, "Dynamic Load Balancing Strategy for Grid Computing", World Academy of Science, Engineering and Tech. 2006.

[21] Rathore N. K., & Chana I. "Checkpointing algorithm in Alchemi.NET", Pragyaan: Journal of Information Technology, 8(1), 32-38. IMS Dehradun. ISSN No.: 0974-5513, IEEE, CSI and MPCET, Dehradun, 2010.

[22] Rathore, N. K., & Chana, I., "Checkpointing algorithm in Alchemi.NET.", Lambert Academic Publication House (LBA), Germany ISBN-10: 3843361371, ISBN-13: 978-3843361378, 2010.

[23] Z. Zeng, B. Veeravalli, "Rate-based and queue-based dynamic load balancing algorithms in distributed systems", In: Proceedings of the 10th IEEE international conference on parallel & distributed systems (pp. 349-356), 2004.

[24] H.C. Lin, C.S. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC)", IEEE Transaction on Software Engineering, 18(2), 148-158,1992.

[25] M. Amini, H. Deldari, "Grid Load Balancing Using an Echo System of Ants", Proc. Of 24th IASTED Int. Conf, Innsbruck, pp. 47-52, 2006.

[26] Y. Lan, T. Yu. A, "Dynamic Central Scheduler Load-Balancing Mechanism", Proc. 14th IEEE Conf. on Computers and Communication, Tokyo, Japan, pp.734-740, 1995.

[27] Kun-Ming V. Yu, Chih-Hsun Chou, Yao-Tien Wang, "A fuzzy-based dynamic load-balancing algorithm", Journal of Information, Technology and Society, 4(2), 55-63, 2004.

[28] Shih-Jie Lin, Min-Chun Huang, Kuan-Chou Lai and Kuo-Chan Huang, "Design and implementation of job migration policies in P2P grid system", In Asia-Pacific services computing conference IEEE, 2008.

[29] M. Backschat, A. Pfaffinger, "Economic Based Dynamic Load Distribution in Large Workstation Networks", In Euro-Par'96, no. 1124, pp: 631-634, 1996.

[30] Alaoui, Frieder and El-hzawi. A parallel genetic algorithm for task mapping on parallel machine. Job scheduling stragies for parallel processing, IPPS'95 Workshop, LNCS, 1999.

[31] Franco Travostino, Paul Daspit, Leon Gommans, Chetan Jog, Cees de Laat, Joe Mambretti, Inder Monga, Bas van Oudenaarde, Satish Raghunath, Phil Wang. Seamless Live Migration of Virtual Machines over the MAN/WAN, Elsevier Future Generation Computer Systems 2006.

[32] Kai Lu. Decentralized Load Balancing In Heterogeneous Computational Grids, ISBN: 978-1-4244-5166-1, 545-550, 2009.

[33] Abraham A., R. Buyya and B. Nath, "Nature's heuristic for scheduling jobs on computational grids", In proc. of 8th IEEE Intl. Conf. Adv. Computer. Communicate. (ADCOM 2000), Cochin, India, pp: 45-52, 2000.

[34] Gery Schneider, Hugo Kohmann, Hakon Bugge. Fault Tolerant Checkpointing Solution for Clusters and Grid Systems, Checkpoint White paper V 1.0 Copyright HPC4U 2007-2008.

[35] Jasma Balasangameshwara and Nedunchezhian Raju, "A fault tolerance optimal neighbor load balancing algorithm for grid environment", In International conference on computational intelligence and communication networks, 2010.

[36] Y. Zhang, H.Franke, J. E. Moreira and A. Sivasubramaniam, "The impact of migration on parallel job scheduling for distributed systems", In Euro-Par 2000 Parallel Processing, Lecture Notes in Computer Science (Vol. 1900, pp. 242-251), 2000.

[37] Nazir B., Qureshi K.,& Manuel P. "Adaptive checkpointing strategy to tolerate faults in economy based grid", The Journal of Supercomputing, 50(1), 1-18, 2009.

[38] Qureshi K., & Manuel P., "Adaptive pre-task scheduling strategy for heterogeneous distributed ray-tracing system", International Journal of Computer and Electrical Engineering, 33, 70-78, 2007.

[39] Shakti Mishra, Dharmender Singh.Kushwaha, Arun Kumar Misra, "An efficient job scheduling technique in trusted clusters for load balancing. In The first international conference on cloud computing, GRIDs, and virtualization: Cloud computing", 2010.

[40] Rathore N. K., & Chana I., "Comparative analysis of checkpointing", PIMR third national IT conference, IT enabled practices and emerging management paradigm book and category is communication technologies and security issues (pp. 32-35), Topic No/Name-46, Prestige Management and Research, Indore, MP, India, 2008.

[41] Said Fathy El-Zoghdy, "A hierarchical load balancing policy for grid computing environment", International Journal of Computer Network and Information Security, 5, 1-12, 2012.

[42] Rathore N. and Chana I., "Variable threshold based hierarchical load balancing technique in grid", In Engineering with computers. Springer, Berlin. ISSN: 0177-0667 (print version) ISSN: 1435-5663 (electronic version). doi:10.1007/s00366-014-0364-z, 2014.

[43] S Kalaiselvi, V Rajaraman, "A survey of check-pointing algorithms for parallel and distributed computers", (vol. 25, Part 5, pp. 489-510), 2000.

[44] Leandro dos Santos Coelho, Diego Luis De Andrade Bernert, "A modified ant colony optimization algorithm based on differential evolution for chaotic synchronization", Elsevier 2010.

[45] Bansal J.C., Sharma H., Arya K.V. & Nagar A, "Memetic search in artificial bee colony algorithm", Soft Computing, vol. 17, no. 10, pp. 1911-1928, 2013.

[46] Joshua Samuel Raj, Rex Fiona, "Load balancing technique in grid environment: A survey", In ICCCI-2013, India, 2013.

[47] S. Prakash, D. P. Vidyarthi, D. P., "Load balancing in computational grid using genetic algorithm", Advances in Computing: 2011, 1(1), 8-17. doi:10.5923/j.ac.20110101.02, 2011.

[48] Kouider Benmohammed-Mahieddine K., "An evaluation of load balancing algorithms for distributed systems", Doctor of Philosophy The University of Leeds, 1991.

[49] PENG Limin, XIAO Wenjun "A binary-tree based hierarchical load balancing algorithm in structured peer-to-peer systems", Journal of Convergence Information Technology, 6(4), 42-49, 2011.

[50] Kai Lu and Albert Y. Zomaya, "A hybrid policy for job scheduling and load balancing in heterogeneous computational grids" In Sixth international symposium on parallel and distributed computing (ISPDC̀07), IEEE, 2007.

[51] Hemant Kumar Mehta, Manohar Chandwani and Priyesh Kanungo, "A modified delay strategy for dynamic load balancing in cluster and grid environment", IEEE, 2010.

[52] Kamlesh Gupta, Sanjay Silakari, "Novel Approach for fast Compressed Hybrid color image Cryptosystem", Advances in Engineering Software, Elsevier Vol. 49, pp. 29-42, 2012.

[53] Gorthi Praksh, R Bajpai, V. Anand, S. NFPA, "A New Model to Estimate Response-Times of Software Applications", in the proceedings of Software Engineering and Applications, Dallas, USA.DEC , 2011.

[54] H.Shan, L. Oliker, R. Biswas and W. Smith, "Scheduling in heterogeneous grid environments: The effects of data migration", In Proceedings of the ADCOM2004: International conference advanced computing and communication, Ahmedabad Gujarat, India, 2004.

[55] Jasma Balasangameshwara, Nedunchezhian Raju, "A hybrid policy for fault tolerant load balancing in grid computing environments", Journal of Network and Computer Applications, 35, 412-422, 2012.

[56] Parveen Kumar, Richa Setiya and Poonam Gahlan, "Checkpointing algorithms for distributed systems", In TECHNIA-international journal of computing science and communication technologies (vol. 2, no.1), ISSN 0974-3375, 2009.

[57] Yunhua Deng and Rynson W. H. Lau, "Dynamic load balancing in distributed virtual environments using heat diffusion. ACMTransactions on Multimedia Computing, Communications, and Applications", 10(2), Article 16, 2014.

[58] Jun Zhang, "Flexible distributed computing with volunteered resources" PhD Thesis, University of London. 2010.

[59] R. Srinivasan, P. Dasgupta, T. Gohad, "Software Based Remote Attestation for OS Kernel and User Applications", Proceedings - IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCompp. 1048-1055, 2011.

[60] Ramon Lawrence, "A Survey of Process Migration Mechanisms", University of Manitoba, project report, umlawren@cs.umanitoba.ca, May 29, 1998.

[61] Amit Agarwal, Saloni Jain, "Efficient optimal algorithm of task scheduling in cloud computing environment", International Journal of Computer Trends and Technology (IJCTT), 9(7), 344, ISSN: 2231-280, 2014.

[62] Katja Gilly, Nigel Thomas, Carlos Juiz and Ramon Puigjaner, "Scalable QoS content-aware load balancing algorithm for a Web Switch based on classical", AINA 2008, 934-941. doi:10.1109/AINA.2008.75, 2008.

[63] Gracjan Jankowski, Radoslaw Januszewski, Rafal Mikolaj czak", Grid Checkpointing Architecture - a revised proposal", Core GRID TR-0036, 2006.

[64] Ruchir Shah, Bhardwaj Veeravalli, Manoj Misra, "On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation for Computational Grid Environments", IEEE transactions on parallel and distributed systems, vol. 18, no. 12, pp 1675-1686, December 2007.

[65] Foster I. and C. Kesselman, "The grid: blueprint for a new computing infrastructure", Morgan Kaufman, 1998.

[66] L. Anand, D. Ghose, and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems", Int'l J. Computers and Math with Applications, vol. 37, no. 8, pp. 57-85, Apr. 1999.

[67] T. Altameem, "Fault Tolerance Techniques in Grid Computing Systems", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 4 (6), 858-862, 2013.

[68] Prasenjit Kumar Patra, Harshpreet Singh, Gurpreet Singh, "Fault Tolerance Techniques and Comparative Implementation in Cloud Computing", International Journal of Computer Applications (0975-8887) Volume 64-No.14, February 2013.

[69] Shuangquan Yang, "Online scheduling with migration cost", IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012.

[70] Susanne Albers, Matthias Hellwig, "On the Value of Job Migration in Online Makespan Minimization", arXiv:1111.0773v2 [cs.DS] 8 Mar 2012.

[71] Zafer Altug Sayar and Prof. Dr. Nadia Erdogan, "Network Load Balancing with strong migration in an agent based grid using CSP Approach", International Journal of Grid Computing & Applications (IJGCA) Vol.3, No.4, pp 43-53, December 2012.

[72] Tariq Alwada'n, Hamza Aldabbas, Helge Janicke,Thair Khdour, Omer Aldabba, "Dynamic Policy Management In Mobile Grid Environments", International Journal of Computer Networks & Communications (IJCNC) Vol.4, No.2,pp-35-51,March 2012.

[73] Said Fathy El-Zoghdy, "A Capacity-Based Load Balancing And Job Migration Algorithm For Heterogeneous Computational Grids", International Journal of Computer Networks & Communications (IJCNC) Vol.4, No.1,pp(113-125), January 2012.

[74] Vasinee Siripoonya Kasidit Chanchio, "Thread-Based Live Checkpointing of Virtual Machines", 2011 IEEE International Symposium on Network Computing and Applications,2011.

[75] Federico Calzolari and Silvia Volpe, "A new job migration algorithm to improve data center efficiency, proceeding on science", The International Symposium on Grids and Clouds and the Open Grid Forum Academia Sinica, Taipei, Taiwan March 19 - 25, 2011.

[76] DaeWon Lee, "IP-Paging base Resource Management and Task Migration in Mobile Grid Environments, International Journal of Grid and Distributed Computing", Vol. 3, No. 3, pp 29-40, September, 2010.

[77] Shakti Mishra, D.S.Kushwaha, A.K.Misra, "An Optimized Scheduling Algorithm for Migrated Jobs in Trusted Distributed Systems", Int'l Conf. on Computer & Communication Technology, 2010.

[78] David Cuesta, Jos'e L. Ayala, Jos'e I. Hidalgo, David Atienza, Andrea Acquaviva and Enrico Macii, "Adaptive Task Migration Policies for Thermal control in MP-SoCs", IEEE Annual Symposium on VLSI, 2010.

[79] John Mehnert-Spahn and Michael Schoettner, "Checkpointing and Migration of Communication Channels in Heterogeneous Grid Environments", In Algorithms and Architectures for Parallel Processing, Lecture Notes in Computer Science,Vol. 6081, pp 254-265, 2010.

[80] Remi Busseuil, Gabriel Marchesan Almeida, Sameer Varyani, Gilles Sassatelli, Michel Robert, "Exploration of task migration techniques for distributed memory MultiProcessor Systems on Chips", In 18th IEEE/IFIP International Conference on VLSI and Systems-on-Chip (VLSI-SoC2010), Madrid, Spain: IEEE Computer Society, 2010.

[81] A. Satheesh,K. Vimal Kumar, and S. Krishnaveni, "Generalized Load Sharing for Homogeneous Networks of Distributed Environment, Hindawi Publishing Corporation Journal of Computer Systems, Networks, and Communications", Article ID 294106, 2008.

[82] Zhirou Zhang, Wengang Cheng, "Model and Optimization of Mobile Agent's Migration in Grid", IEEE, 2007.

[83] Vipin Chaudhary John Paul Walters and Hai Jiang, "An Adaptive Heterogeneous Software DSM" Proceedings of International Conference on Parallel Processing Workshops ICPPW- 2006.

[84] Abdullah Azfar, Md. Sarwar Morshed,, Md. Zahidul Islam, Abu Saleh Shah Muhammad Barkat Ullah, Al-Mukaddim Khan Pathan, "A Simplified Process for Grid Job Migration", In International Conference of Computer and Information Technology (ICCIT-2005) (pp. 706-709). Bangladesh: Islamic University of Technology (IUT),2005.

[85] HwaMin Lee, KwangSik Chung, SungHo Chin, JongHyuk Lee, DaeWon Lee, Seongbin Parka, HeonChang Yua, "Aresource management and fault tolerance services in grid computing", J. Parallel Distrib. Comput. 65, 1305-1317, 2005.

[86] Sathish S. Vadhiyar, Jack J. Dongarra, "A Performance Oriented Migration Framework for The Grid, In Cluster Computing and the Grid, Proceedings", CCGrid 2003 (pp. 130-137). 3rd IEEE/ACM International Symposium on IEEE, 2003.

[87] Sriram Satish Tadepalli, "GEMS: A Fault Tolerant Grid Job Management System", Thesis, Master of Science, Blacksburg, Virginia, December 19, 2003.

[88] Po-Cheng Chen, Cheng-I Lin, Sheng-Wei Huang, Jyh-Biau Chang, Ce-Kuen Shieh, Tyng-Yeu Liang, "A Performance Study of Virtual Machine Migration vs. Thread Migration for Grid Systems", 22nd International Conference on Advanced Information Networking and Applications-Workshops.

[89] T. N. Ellahi, B. Hudzia, L. McDermott and T. Kechadi, "Transparent Migration Of Multi-Threaded Applications On A Java Based Grid", University College Dublin Dublin-Ireland.

[90] P. Emerald Chung, Yennun Huang, Shalini Yajnik, Glenn Fowler, Kiem-Phong Vo and Yi-Min Wang, "Checkpointing in CosMiC: a User-level Process Migration Environment", IEEE, 1997.

[91] Simone A. Ludwig, Azin Moallem, "Swarm Intelligence Approaches for Grid Load Balancing", J Grid Computing, Springer Science Business Media B.V. 9, 279-301. doi:10.1007/s10723-011-9180-5, 2011.

[92] Qingqi Long, Jie Lin, Zhixun Sun, "Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations", Tongji University, Shanghai , China Simulation Modelling Practice and Theory, pp. 1021-1034 Science Direct Elsevier, 2011.

[93] Sotirios Kontogiannis, Stavros Valsamidis, and Alexandros Karakos, "ALBL, ALBL/HSC algorithms: Towards more scalable, more adaptive and fully utilized balancing systems", Journal of Computing, Volume 3, Issue 2, Issn 2151-9617, February 2011.

[94] Hemant Kumar Mehta, Priyesh Kanungo and Manohar Chandwani, "Performance Enhancement of Scheduling Algorithms in Clusters and Grids using Improved Dynamic Load Balancing Techniques", WWW 2011-Ph. D. Symposium, March 28 April, 2011, Hyderabad, India.

[95] Robson E. De Grande, Azzedine Boukerche, "Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems", J. Parallel Distrib. Comput. 40-52, 2011.

[96] Deepti Sharma, Archana B.Saxena, "Framework to Solve Load Balancing Problem in Heterogeneous Web Servers", International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, 50-63 Feb 2011.

[97] Hung-Chang Hsiao, Hao Liao, Ssu-Ta Chen, and Kuo-Chan Huang, "Load Balance with Imperfect Information in Structured Peer-to-Peer Systems", IEEE Transactions on Parallel and Distributed Systems, VOL. 22(4), 634-649, 2011.

[98] T.R.V. Anandharajan, Dr. M.A. Bhagyaveni, "Co-operative Scheduled Energy Aware Load-Balancing technique for an Efficient Computational Cloud", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2, 571-576, March 2011.

[99] Jasma Balasangameshwara, Nedunchezhian Raju, "A Decentralized Recent Neighbour Load Balancing Algorithm for Computational Grid", International Journal of ACM Jordan (ISSN 2078-7952), Vol. 1, No. 3, September 2010.

[100] Yuan Rao, Ru-chuan Wang, "Agent-based load balancing routing for LEO satellite networks", Computer Networks,pp. 3187-3195, ELSEVIER 2010.

[101] Safa Khalouli, Fatima Ghedjati, Abdelaziz Hamzaoui, "A Meta-Heuristic approach to solve a JIT scheduling problem in hybrid flowshop", Engineering Applications of Artificial Intelligence, pp.765-771, Elsevier, 2010.

[102] M.H. Afshar, "A parameter free Continuous Ant Colony Optimization Algorithm for the optimal design of storm sewer networks: Constrained and unconstrained approach", Advances in Engineering Software 41, 188-195, 2010.

[103] Al-Dahoud Ali, Mohamed A. Belal and Moh'd Belal Al-Zoubi, "Load Balancing of Distributed Systems Based on Multiple Ant Colonies Optimization", American Journal of Applied Sciences 7 (3): 433-438, Science Publications, 2010.

[104] Liang Bai, Yan-Li Hu, Song-Yang Lao and Wei-Ming Zhang, "Task Scheduling with Load Balancing using Multiple Ant Colonies Optimization in Grid Computing", Sixth International Conference on Natural Computation (ICNC 2010)

[105] Husna Jamal Abdul Nasir, Ku Ruhana and Ku-Mahamud, "Grid Load Balancing Using Ant Colony Optimization", Second International Conference on Computer and Network Technology, IEEE ICCNT-2010.

[106] Antony Lidya Therasa.S, Sumathi.G and Antony Dalya.S, "Dynamic Adaptation of Checkpoints and Rescheduling in Grid Computing", International Journal of Computer Applications (0975-8887) Volume 2-No.3, May 2010.

[107] B. Cosenza, G. Cordasco, R. De Chiara and V. Scarano, "Distributed Load Balancing for Parallel Agent-based Simulations",http://www.isislab.it/projects/DistrSteer/Dist Steer: Parallel Distributed Agent-Based Simulations, 2010.

[108] Magdy Saeb and Cherine Fathy, "Performance Evaluation of Mobile Agent-based Dynamic Load Balancing Algorithm", In 9th International Conference on Distributed Multimedia Systems, DMS Conference, Data Communication Security, Miami: Mobile Agent Paradigm, 2010.

[109] Ghada F. El Kabbany, Nayer M. Wanas, Nadia H. Hegazi, Samir I. Shaheen, "A Dynamic Load Balancing Framework for Real-time Applications in Message Passing Systems", Springer Science Business Media, LLC 28 April 2010.

[110] P. K. Suri, Manpreet Singh, "An Efficient Decentralized Load Balancing Algorithm For Grid", IEEE, 2010.

[111] Weiwei Lin,Wuyao Shen, "Tree-Based Task Scheduling Model and Dynamic Load-Balancing Algorithm for P2P Computing", South China University of Technology, Guangzhou, China, 10th IEEE International Conference on Computer and Information Technology (CIT 2010), pp-2903-2907, 2010.

[112] Somayeh Abdi and Somayeh Mohamadi, "The Impact of Data Replication on Job Scheduling Performance in Hierarchical Data Grid", International journal on applications of graph theory in wireless ad hoc networks and sensor networks (GRAPH-HOC), Vol.2, No.3, 177-185, September 2010.

[113] Malarvizhi Nandagopal and Rhymend V Uthariaraj, "Hierarchical Status Information Exchange Scheduling and Load Balancing For Computational Grid Environments", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.2, February 2010.

[114] Malarvizhi Nandagopal, K.Gokulnath and V.Rhymend Uthariaraj, "Sender Initiated Decentralized Dynamic Load Balancing for Multi Cluster Computational Grid Environment", A2CWiC 2010.

[115] Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kale, "Periodic Hierarchical Load Balancing for Large Supercomputers", 25(4), 371-385,2010.

[116] Bin Wang and Qing-guo Shen, "ID Management and Allocation Algorithm for P2P Load Balancing", IEEE, 2010.

[117] Robson E. De Grande, Azzedine Boukerche, "Dynamic balancing of communication and computation load for HLA- based simulations on large-scale distributed systems", J. Parallel Distrib. Computer, 71(1), 40-52, 2010.

[118] Xiao Qin, Hong Jiang, Adam Manzanares, Xiaojun Ruan and Shu Yin, "Communication-Aware Load Balancing for Parallel Applications on Clusters", IEEE Transactions on computers, vol. 59, no. 1, 42-52, january 2010.

[119] Shoukat Ali, Behdis Eslamnour, Zehra Shah, "A Case for on-machine load balancing", Elsevier 2010.

[120] Chao-Chin Ren-Yi Sun, "An integrated security-aware job scheduling strategy for large-scale computational grids", Future Generation Computer Systems,26(2), 198-206, 2010.

[121] P. V. Prasad, S. Sivanagaraju and N. Sreenivasulu, "Network Reconfiguration for Load Balancing in Radial Distribution Systems Using Genetic Algorithm", Electric Power Components and Systems, Publication details, including instructions for authors and subscription information, 2010.

[122] Wenmin Miao, Dongni Li and Wei Zhang, "A Load-Balancing Dynamic Scheduling Algorithm under Machine Failure Conditions", International Conference on Intelligent Computation Technology and Automation, IEEE, 2010.

[123] Jie Chang, Wen'an Zhou, Junde Song and Zhiqi Lin, "Scheduling Algorithm of Load Balancing Based on Dynamic Policies", Sixth International Conference on Networking and Services, 2010.

[124] Youngjoon Jo and Milind Kulkarni, "Brief Announcement: Locality-aware Load Balancing for Speculatively-parallelized Irregular Applications", Purdue University, SPAA'10, Thira, Santorini, Greece.ACM 978-1-4503-0079-7/10/06, June 13-15, 2010.

[125] Jaehwan Lee, Pete Keleher and Alan Sussman, "Decentralized Dynamic Scheduling across Heterogeneous Multi-core Desktop Grids", IEEE-2010.

[126] Shakti Mishra, D.S.Kushwaha, A.K.Misra, "Hybrid and reliable load balancing with MOSIX as middleware and its formal verification using process algebra", Future generation of computer science,27(5), 506-526, 2010.

[127] Mohsen Moradi, Mashaala Abbasi Dezfuli, Mohammad Hasan Safavi, "A New Time Optimizing Probabilistic Load Balancing Algorithm in Grid Computing", IEEE,2010.

[128] Dongliang Zhang, Changjun Jiang and Shu Li, "A fast adaptive load balancing method for parallel particle-based simulations", Simulation Modelling Practice and Theory, pp 1032-1042, 2009.

[129] Yong Hee Kim, Seungwok Han, Chang Hun Lyu and Hee Yong Youn, "An Efficient Dynamic Load Balancing Scheme for Multi-Agent System Reflecting Agent Workload", International Conference on Computational Science and Engineering, 2009.

[130] Chang Hui, Lei Xiaoyong, Dai Shuling, "A Dynamic Load Balancing Algorithm for Sort-first Rendering Clusters", IEEE, 2009.

[131] Azzedine Boukerche and Robson Eduardo De Grande, "Dynamic Load Balancing Using Grid Services for HLA-Based Simulations on Large-Scale Distributed Systems", 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications 2009.

[132] Abhinav Bhatele, Laxmikant V. Kale and Sameer Kumar, "Dynamic Topology Aware Load Balancing Algorithms for Molecular Dynamics Applications", ICS'09, New York town Heights, New York, USA, ACM, 2009.

[133] Xiao Qin,Hong Jiang,Adam Manzanares, Xiaojun Ruan, and Shu Yin, "Dynamic Load Balancing for I/O-Intensive Applications on Clusters", ACM Transactions on Storage 5:3, Article 9, November 2009.

[134] P Visalakshi and S N Sivanandam, "Dynamic Task Scheduling with Load Balancing using Hybrid Particle Swarm Optimization", Int. J. Open Problems Compt. Math., Vol. 2, No. 3, Copyright ICSRS Publication, September 2009.

[135] Yajun Li, Yuhang Yanga, Maode Mab and Liang Zhou, "A hybrid load balancing strategy of sequential tasks for grid computing environments", Future Generation Computer Systems,25(8), 819-828, 2009.

[136] Thein Thein Aye and Htway Htway Hlaing, "An Efficient Dynamic Load Balancing Policy in Cluster Computing System", GMSARN International Conference on Sustainable Development: Issues and Prospects for the GMS 12-14 Nov. 2008.

[137] Minglong Zhang, Boqin Feng, "A Novel Migration Algorithm Based-on "States-balancing" in a Distributed Multimedia Services System", International Conference on Multimedia and Ubiquitous Engineering, IEEE 2008.

[138] Zhengping Qian, Ming , Deyu Qi, Kefu Xu., "A Dynamic Scheduling Algorithm for Distributed Kahn Process Networks in a Cluster Environment", DOI 10.1109/PACIIA.2008.190, IEEE 2008.

[139] K. Saruladha, G. Santhi, "Behavior of Agent Based Dynamic Load Balancing Algorithm for Heterogeneous P2P Systems", Pondicherry Engineering College, Pondicherry, International Conference on Computational Intelligence and Multimedia Applications, 2007.

[140] Ana Cortes Fite, "A new distributed diffusion algorithm for dynamic load balancing in parallel systems", Barcelona (Spain) September 2000.

[141] Ching-Jung Liao, Yeh-Ching Chung, "Tree-Based Parallel Load-Balancing Methods for Solution-Adaptive Finite Element Graphs on Distributed Memory Multicomputers", IEEE Transactions On Parallel And Distributed Systems, VOL. 10, NO. 4, 36-370, APRIL 1999.

[142] Mohammed Javeed Zaki, Wei Li and Srinivasan Parthasarathy, "Customized Dynamic Load Balancing for a network workstations", jour. of parallel and distributed computing, 43, 156-162, 1997.

[143] Qureshi K., Rehman A. and Manuel P.,"Enhanced gridsim architecture with load balancing", The Journal of Supercomputing, 57(3):265-275, September 2011.

[144] R. prakash, M.singhal, "Maximum global snapshot with concurrent intiators", in: proceding in sixth IEEE symposium of parallel and distributed Systems 7(10), pp 1035-1048, 1996.

[145] M. Spezialetti, P. Kearns, "Efficient distributed snapshots", in: Proceedings of the Sixth ICDCS, pp. 382-388, 1986.

[146] S. Siva Sathya, K. Syam Babu, "Survey of fault tolerant techniques for grid", computer science view, pp 101-120, 2010.

[147] Keerthika P., and N. Kasthuri, "An Efficient Grid Scheduling Algorithm with Fault Tolerance and User Satisfaction", Mathematical Problems in Engineering, 2013.

[148] F Xhafa, A Abraham, "Computational models and heuristic methods for Grid scheduling problems", Future generation computer systems 26 (4), 608-621.

[149] Rathore N. and Chana I., "A sender initiate based hierarchical load balancing technique for grid using variable threshold value", pages 1-6, September 2013.

[150] Rathore N., "An efficient hierarchical load balancing technique for grid", In 29th M.P. Young Scientist congress (p. 55), Bhopal, MP, 2014.

[151] Rathore, N. K., & Chana, I., "Job migration mechanism with fault tolerance and QoS scheduling using hash table functionality in social grid computing", Journal of Intelligent & Fuzzy Systems, 26, (6), 21-35. IOS Press publication, ISSN print 1064-1246, ISSN online 1875-8967,DOI- 10.3233/IFS-141243, June 2014.

[152] Hu Y. and Gong B.and Wang F., "Cloud model-based security-aware and fault-tolerant job scheduling for computing grid", pages 25-30, July 2010.

[153] Kim H., Kang S., and Yeom H.Y., "Server selection schemes considering node status for a fault-tolerant streaming service on a peer-to-peer network", March 2006.

[154] Tao Q., Chang Hu.and Yang Yi, and Chunqin Gu., "A trustworthy management approach for cloud services qos data", 4:1626-1631, July 2010.

[155] Chard K., Bubendorfer K., Caton S., and Rana Omer F., "Social cloud computing: A vision for socially motivated resource sharing", IEEE Transactions on Services Computing, 5(4):551-563, 2012.

[156] Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S., "A scalable content-addressable network", In IN PROC. ACM SIGCOMM 2001, pages 161-172. ACM, 2001.

[157] Alexandru P., David E., Markus F., and Demetres K., "Routing in content addressable networks: Algorithms and performance", In 20th ITC- Specialist Seminar. IEEE, 2009.

[158] Ali M.E., Tanin E., Zhang R., and Kulik L., "Load balancing for moving object management in a p2p network", pages 251-266, March 2008.

[159] Sahin O. D., Gupta A., Agrawal D., and El Abbadi A., "A peer-to-peer framework for caching range queries",In ICDE, page 165, 2004.

[160] Amir H. Basirat, Anang H. Mohammad Amin, and Asad I. Khan, "Under the cloud: A novel content addressable data framework for cloud parallelization to create and virtualize new breeds of cloud applications", IEEE 12th International Symposium on Network Computing and Applications, pages 168-173, 2010.

[161] Zhang S., Bai W., Gengyu W., and Chao X., "Web qos management model based on can", pages 143-146. IEEE, 2011.

[162] Tassanaviboon A. and Gong G., "A framework toward a self-organizing and self-healing certificate authority group in a content addressable network", pages 614-621, October 2010.

[163] Ristau H., D. Versick, and D. Tavangarian, "Enhancements to can for the application as distributed data storage system in grids", pages 1355-1361 Vol. 2, October 2005.

[164] Roger Z.and Wei-Shinn K. and Haojun W., "Spatial data query support in peer-to-peer systems", In COMPSAC Workshops, pages 82-85. IEEE Computer Society, 2004.

[165] Zhang Y. and Fang Y., "A fine-grained reputation system for reliable service selection in peer-to-peer networks", IEEE Transactions on Parallel and Distributed Systems, 18(8):1134-1145, 2007.

[166] Calheiros R. N., R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya., "Cloudsim: A toolkit for the modeling and simulation of cloud resource management and application provisioning techniques". Software: Practice and Experience, 41(1):23-50, January 2011.

List of Publications

International Journals (SCI Indexed)

1. **N Rathore** and I Chana, "Variable Threshold Based Hierarchical Load Balancing Technique in Grid", Engineering with Computers, **Springer publication**, ISSN: 0177-0667 (print version) ISSN: 1435-5663 (electronic version), DOI-10.1007/s00366-014-0364-z ,IF- 1.088, June 2014.

2. **N Rathore** and I Chana, "Job Migration with Fault Tolerance and QoS Scheduling using Hash Table Functionality in Social Grid Computing", Journal of Intelligent & Fuzzy Systems, **IOS Press publication**, ISSN print 1064-1246, ISSN online 1875-8967, DOI- 10.3233/IFS-141243 , IF- 0.936, June 2014.

3. **N Rathore** and I Chana, "Load Balancing and Job Migration Algorithm : A Survey of Recent Trends", Wireless Personal Communication, **Springer Publication**, ISSN print 0929-6212, ISSN online 1572-834X, IF - 0.979, DOI-10.1007/s11277-014-1975-9, July 2014.

International Journal

1. **Neeraj Kumar Rathore** and Inderveer Chana, "Report on Hierarchal Load Balancing Technique in Grid Environment", International Journal of Scientific and Innovative Technology, i-manager's Journal on Information Technology, Vol. 2, No. 4, ISSN Print: 2277-5110, pp-21-35, Sep - Nov 2013.

International Conferences

1. **N Rathore** and I Chana, "A Cognitative Analysis of Load Balancing Technique with Job Migration in Grid Environment", World Congress on Information and Communication Technology (WICT), Mumbai, IEEE proceedings paper, ISBN - 978-1-4673-0127-5 pp- 77-82, December 2011

2. **N Rathore** and I Chana, "A Sender Initiate Based Hierarchical Load Balancing Technique for Grid Using Variable Threshold Value" in International conference IEEE-ISPC, ISBN- 978-1-4673-6188-0, pp.1-6, 26-28 Sept. 2013.

3. **Neeraj Kumar Rathore** and Inderveer Chana, "An Efficient Hierarchical Load Balancing Technique for Grid" in 29th M.P. Young Scientist congress, Bhopal, M.P., pp. 55, Feb 28, 2014.

Paper Communicated

1. **N Rathore** and Inderveer Chana, "Performance of Load Balancing Algorithm in Complex Networks", Wireless Personal Communication, **Springer Publication**, SCI indexed, ISSN print 0929-6212, ISSN online 1572-834X, impact factor -0.979, July 2014 (Communicated)

2. **N Rathore** and I Chana, "Job Migration Policies for a Grid Enviornment", Chiang Mai Journal of Sciences, SCI indexed, Print ISSN: 0125-2526, Impact Factor: 0.516, June 2014. (Communicated)