AN EFFICIENT DECENTRALIZED LOAD BALANCING ALGORITHM FOR COMPUTATIONAL GRID

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

ANAND KUMAR

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247667 (INDIA)
JUNE, 2011

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**AN EFFICIENT DECNTRALIZED LOAD BALANCING ALGORITHM FOR COMPUTATIONAL GRID**" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Computer Science and Engineering** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from July 2010 to June 2011, under the guidance of **Dr. Padam Kumar, Professor,** Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 29-6-11
Place: Roorkee

[illegible]
(ANAND KUMAR)

CERTIFICATE

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

Date: 29-6-11
Place: Roorkee

[illegible]
(Dr. Padam Kumar)
Professor
Department of Electronics and Computer Engineering
IIT Roorkee.

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor **Dr. Padam Kumar**, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for his invaluable advices, guidance, and encouragement for sharing his broad knowledge. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. He has been very generous in providing the necessary resources to carry out my research. He is an inspiring teacher, a great advisor, and most importantly a nice person.

I am thankful for the useful comments and suggestions from faculty members of our institute. I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions.

On a personal note, I owe everything to the Almighty and my parents. The support which I enjoyed from my father, mother and other family members provided me the mental support I needed.

ANANAD KUMAR

ABSTRACT

Computational grids have the potential computing power for solving large-scale scientific computing applications. To improve the global throughput of these applications, workload has to be evenly distributed among the available computational resources in the grid environment. So load balancing becomes one of the critical issues that must be considered in managing a grid computing environment. Hence we need to develop a robust and effective load balancing application which can adopt changes dynamically, because due to the distributed and heterogeneous nature of the resources in grid availability of grid resources is dynamic.

In this dissertation a decentralized load balancing algorithm for computational grid is proposed. It efficiently handles the load in grid environments with considering several other issues that are imperative to Grid environments such as handling resource heterogeneity, communication latency, and job migration from one site to other. The algorithm uses the system parameters such as the estimated completion time of task, CPU processing power, load on the resource, and predicted failure time of the resource and balance the load by migrating jobs from over loaded resources to underloaded or idle resources by taking into account the job transfer cost, resource heterogeneity, and network heterogeneity . The performance of the proposed algorithm is evaluated by using several influencing parameters such as the number of jobs, job size, data transfer rate, and migration limit. The experimental results shows that the proposed algorithm is efficient in minimizing the response time ,total execution time with maximum resource utilization and minimum communication over head.

Table of Contents

LIST OF FIGURES

Figure No. **Page No.**

LIST OF TABLES

Chapter 1

Introduction and Problem Statement

1.1 Introduction

The demand for extra and large computing power is never ending .Many research projects requires lot of CPU power, some requires a lot of memory and some projects need the ability to communicate in real time. Today super computers are not enough to solve those needs. They don't have the capacity, even if they did, it would not be economical justifiable to use these resources. Computational grids are the solution to all these problems and many more. Computational Grids provide broad access not only to massive information resources, but to massive computational resources as well. Computational grids use high performance network technology to connect hardware, software, databases, and people into a seamless web that supports a new generation of computation rich problem solving environments for users and also provides the opportunity to share a number of resources among different organizations.

Hence, grid computing is becoming a promising technology due to the collaboration opportunities it creates for organizations to work together to achieve common goals through resource sharing. With rapid progress in computing, communication, and storage technologies, grid computing has gained extensive interests in academic, industry and military. The goal of Grid computing is to create the illusion of a simple but large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. Grid is a type of a distributed system which supports the sharing and coordinated use of multi owner resources, independently from their physical types and geographical locations to solve large-scale applications like meteorological simulations, data intensive applications, etc.

As more and more critical applications shift to the grid environment, it becomes increasingly important to ensure the higher availability of efficiently managed resources. Grid Resource Management is defined as the process of identifying application requirements, matching resources to the applications, allocating resources to matching applications, scheduling and monitoring grid resources over time in order to run grid applications as efficiently as possible. Resource discovery is the first phase of resource management. Scheduling and monitoring is the next step. Scheduling process directs the job to appropriate resource and monitoring process monitors the resources. Due to uneven task arrival patterns and unequal computing capacities, some resources may be overloaded while others may be under-utilized. It is therefore desirable to dispatch tasks from highly loaded resources to idle or lightly loaded resources in a grid to achieve better resource utilization and reduce the average task response time.

So in order to fulfill the user expectations in terms of performance and efficiency, the grid system needs efficient load balancing algorithms for the distribution of tasks to the resource. A load balancing algorithm attempts to improve the response time of user's submitted applications by ensuring maximal utilization of available resources. The main goal of load balancing is to provide a distributed, low cost, scheme which prevents, if possible, the condition where some processors are overloaded with a set of tasks while others are lightly loaded or idle.

Load balancing is a mapping strategy that efficiently equilibrates the tasks load into multiple computational resources in the network based on the system status to improve performance. The essential objective of a load balancing algorithm can be, depending on the user or the system administrator, defined by: The aim for the user is to minimize the makespan of its own application, regardless the performance of other applications in the system and the main goal for administrator is to maximize the meet of tasks deadline by ensuring maximal utilization of available resources. Typically, a load balancing scheme consists of four policies: information policy, location policy, selection policy, transference policy.

The information policy is responsible to define when and how the information on the Grid resources availability is updated. Based on the information that can be used, load-balancing algorithms are classified as static, dynamic, or adaptive [2], [3], [4], [5]. In a static algorithm, the scheduling is carried out according to a predetermined policy. The state of the system at the time of the scheduling is not taken into consideration. On the other hand, a dynamic algorithm adapts its decision to the state of the system. Adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm and/or the scheduling policy itself is changed based on the global state of the system.

The location policy determines a suitable transfer partner (server or receiver) once the transference policy decided that this resource is server or receiver. Location-based policies can be broadly classified as sender initiated, receiver initiated, or symmetrically initiated [6], [7], [3]. The selection policy defines the task that should be transferred from the busiest resource to the idlest one. The transference policy classifies a resource as server or receiver of tasks according to its availability status. According to another classification, based on the degree of centralization, load scheduling algorithms could be classified as centralized or decentralized [3], [5]. In a centralized system, only a single processor does the load scheduling. Such algorithms are bound to be less reliable than decentralized algorithms, where many, if not all, processors do load scheduling in the system.

1.2 Motivation

Although many load balancing problems in conventional distributed systems has been intensively studied but new challenges in Grid computing still make it an interesting topic. This is due to the characteristics of Grid computing and to the complex nature of the problems itself. Load balancing algorithms in classical distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in Grid environment. Grids have a lot of specific characteristics [8], like Scalability, Autonomy, Adaptability, Heterogeneity, Information freshness, Considerable transfer cost, uneven job arrival pattern, and dynamicity, which remain obstacles for applications to use conventional load balancing algorithms directly. So dynamic load balancing is a key factor in achieving high performance for large scale distributed applications on grid infrastructures because Dynamic (or adaptive) policies works on recent state information and determine the tasks assignment to resources at run time [9]. However, it should be noted that load balancing process in itself is another kind of system overhead. These overheads include: the time required for computing nodes for updating their load information in a real time manner; the communication costs for sharing those load information to make a transfer decision; and the costs of job immigrations. Therefore in which conditions the load balancing process is worthy initiating and how it works should be considered.

In this report, a decentralized load balancing algorithm is propose, which uses the hierarchical strategy to balance tasks load among resources of computational Grid. Based on a tree representation of a Grid, this strategy privileges local load balancing than global one. The main objectives addressed by this neighborhood strategy are, the reduction of the average response time of tasks and the reduction of the communication cost induced by task transferring.

1.3 Problem statement

The main aim of this dissertation is to

1. Study the grid technologies in the areas concerned with load balancing in computational grid.

2. To design and simulate an efficient decentralized load balancing algorithm for Computational grid to guarantee the maximum utilization of available resources with reduction in the average response time of tasks and the reduction in the communication cost induced by task transferring.

3. Evaluate the performance of proposed approach, taking different number of resource with different capability and budget, different number of users with different number of tasks with associate budget.

1.4 Organization of the Report

This dissertation report comprises of five chapters including this chapter that introduces the topic and states the problem. The rest of the report is organized as follows.

Chapter 2 discuss about the load balancing in computational grid, different approaches for load balancing and literature review of the dynamic adaptive load balancing algorithms. It also includes the research gaps found.

Chapter 3 gives the detailed design of the proposed load balancing algorithm. This chapter includes the proposed system model, load balancing strategy, and detail about the different modules of the algorithm.

Chapter 4 gives the detail about the simulation toolkit used for the simulation of the proposed algorithm.

Chapter 5 discusses the experimental results, validation of the proposed load balancing algorithm and comparison with the previous load balancing algorithms.

Chapter 6 concludes the dissertation work and gives scope for future work.

Chapter 2

Background and Literature Review

2.1 Grid computing: An Overview

2.1.1 Evolution of Grid Computing

Distributed computing is the combination of widely spread computational machines, to solve the large computative problems like, weather forecasting, satellite launching and earthquake predetermination etc. Distributed computing is another form of parallel computing where program parts, run on different machine simultaneously, in parallel computing program parts run simultaneously on multiple processors in the same computer. Both types of processing require dividing a program into parts that can run simultaneously. Distributed programs often deal with heterogeneous environments, network links of varying latencies and unpredictable failures in the network or the computers [10].

Distributed computing connects the two remotely situated machines via a program in which it makes call to other machine taking its address space. There can be many nodes in distributed computing but a node in distributed environment does not know the hardware architecture on which the recipient is running, or the platform in which the recipient is implemented. There are many differences between local and distributed computing like throughput, memory access, concurrency and partial failure. In all these differences latency and memory access are well known and others are more difficult to explain [10]. The main problem in distributed computing is to manage all distributed resources by central resource manager along with managing issues related to performance, concurrency, communication, failure handling, deadlocks and security [10]. Moreover multiple point failure and more opportunities for unauthorized attack in distributed computing have paved way for grid computing which has thus evolved from distributed computing.

The term "*Grid*" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering [11]. The concept of *Computational Grid* has been inspired by the 'electric power Grid', in which a user could obtain electric power from any power station present on the electric Grid irrespective of its location, in an easy and reliable manner-- something that the client connects to and pays for according to the amount of use. When we require additional electricity we have to just plug into a power Grid to access additional electricity on demand, similarly for computational resources to plug into a Computational Grid to access additional computing power on demand using most economical resource. This led to coining of the term "*Grid*".

The basic idea behind the grid is sharing computing power. Now-a-days most people have more than enough computing power on their own PC. But there are number of applications which need more computing power that can be offered by a single resource or organization in order to solve them within a feasible/reasonable span of time and cost. This promoted the exploration of logically coupling geographically distributed high-end computational resources and using them for solving large-scale problems. Such emerging infrastructure is called computational grid, and led to the popularization of a field called grid computing [11].

2.1.2 Characteristics of Grid Computing

In today's complex world computers have become extremely powerful and they are capable enough to run more complex problem, still there are many complex scientific experiments, advanced modeling scenarios, genome matching, astronomical research, a wide variety of simulations, complex scientific & business modeling scenarios and real-time problems, which require huge amount of computational resources. To satisfy some of these aforementioned requirements, grid computing is being utilized. Grid computing offers seamless access to distributed data and collaborative distributed environments, for running computationally intensive applications. The following are the characteristics of grid computing:

Exploiting Underutilized Resources: In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides technique for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

Parallel CPU Capacity: The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts.

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 virtual organizations can then share their resources collectively as a larger grid.

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.

Reliability: Grid provides reliability in terms of failure at one location; 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.

Resource Balancing: A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid-enabled, 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.

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.

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 millions. This raises the problem of potential performance degradation as grid's size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant [12].

Dynamicity or Adaptability: 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 to dynamically adapt the current grid status so as to extract the maximum performance from the available resources and services.
Resource Coordination: Resources in a grid must be coordinated in order to provide aggregated computing capabilities [13].
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 requirement assurances that they will receive expected, continuous and often high levels of performance [12].

2.1.3 Types of Grid

Grid computing can be used in variety of ways to address various kinds of application requirements. According to the distinct application realms, grid system can be classified into two categories. But there are actually no hard boundaries between these grid categories. Real grids may be a combination of one or more of these types. The two categories of grid systems are describe as below:

Computational Grid: A computational grid system that aims at achieving higher aggregate computation power than any single constitute machine. According to how the computing power is utilized, computational grids can be further subdivided into distributed supercomputing and high throughput categories. A distributed supercomputing grid exploits the parallel execution of applications over multiple machines simultaneously to reduce the execution time. A high throughput grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as many as possible.

Data Grid: A date grid is responsible for housing and providing access to data across multiple organizations. Users are concerned with where this data is located as long as they have access to the data. For example, there can be two universities doing life science research, each with unique data. A data grid would allow them to share their data, manage the data, and manage security issues.

2.2 Load Balancing in Computational Grid

Recent years have been witness to the increasing use of distributed computing systems. This may be attributed to two main factors: the growth of the Internet, and the emergence of low-cost solutions for end-user computing devices. Computational Grid functionally combines globally distributed computers and information systems for creating a universal source of computing power and information. So in computational grid through the communication network, the resources of the system can be shared by users at different locations but the fundamental problem arises in making effective use of the total computing power of this distributed computing system. It is often the case that a certain node has very few tasks to handle at a given time, while another node has many. It is desirable to spread the total workload of the distributed system over all of its nodes so that maximum resource utilization, minimum task execution time and minimum task response time could be achieved. This form of computing power sharing for improving the performance of a distributed system by redistributing the workload among the available nodes is commonly called "load balancing".

A proper scheduling and efficient load balancing across the grid can lead to improve overall system performance and a lower turn-around time for individual jobs. To minimize the decision time is one of the objectives for load balancing which has yet not been achieved. The Job migration is the only efficient way to guarantee that submitted jobs are completed reliably and efficiently in case of process failure, resource failure, node crash, network failure, system performance degradation, communication delay Generally, load balancing mechanisms can be broadly categorized as centralized or decentralized, dynamic or static, and periodic or non-periodic [14]. All load balancing methods are designed such as, to spread the load on resources equally and maximize their utilization while minimizing the total task execution time. Selecting the optimal set of jobs for transferring from one resource to other has a significant role on the efficiency of the load balancing method as well as grid resource utilization. This problem has been neglected by researchers in most of previous contributions on load balancing, either in distributed systems or in the grid environment [15].

2.3 Load Balancing Algorithms: a Simple Classification

Many different load balancing algorithms are described in the literature. However, most of these descriptions are presented in a mixture of text, drawings and pseudo-code, using inconsistent terminology details. Reader's ability to evaluate and compare the various algorithms is severely impaired by the absence of a common reference framework. The concepts used to classify the algorithms are also useful for the methodical design and of load balancing algorithms most relevant to this research.

2.3.1 Static versus Dynamic

Load balancing could be done statically at compile-time or dynamically at run-time. Static load-balancing algorithms assume that a priori information about all of the characteristics of the jobs, the computing nodes and the communication network are known. Load-balancing decisions are made deterministically or probabilistically at compile time, and remain constant during run-time. The static approach is attractive because of its simplicity and the minimized run-time overhead. However, the static approach cannot respond to a dynamic run-time environment, and may lead to load imbalance on some nodes and significantly increase the job response time. The majority of loosely coupled distributed systems exhibit significant dynamic behavior, having load varied with time. For these systems, dynamic scheduling, in which policy decisions are made at run time based on the load-state of nodes, is required. As a result, there are fewer studies on static approaches compared with those on dynamic approaches.

Dynamic load-balancing policies attempt to dynamically balance the workload reflecting the current system state, and are therefore thought to be able to further improve system performance. Thus, compared with static ones, dynamic load-balancing policies are thought to be better able to respond to system changes and to avoid states that result in poor performance. The clear disadvantages of dynamic load-balancing policies are that these policies are more complex than their static counterparts, in the sense that they require run time information gathering overhead about the current states of the node. Due to the communication costs of load information collection and distribution, the communication cost of job transfer and processing cost of making scheduling decisions, dynamic load balancing algorithms definitely incur much run-time overhead. But a good dynamic load balancing algorithm always makes these costs minimized. Thus, it is now commonly agreed that, despite the higher run-time complexity, dynamic algorithms potentially provide better performance than do static algorithms.

Hybrid algorithms combine the advantages of both static and dynamic strategies. In hybrid algorithms, the static algorithm is considered a "coarse" adjustment, and the dynamic algorithm a "fine" adjustment [16]. When the static algorithm is used, load imbalance may result. Once this happens, the dynamic algorithm starts to work and guarantees that the jobs in the queues are balanced in the entire system.

2.3.2 Non-preemptive versus Preemptive

Dynamic load-balancing policies may be either non-preemptive or preemptive. A non-preemptive load-balancing policy assigns a newly arriving job to what appears at that moment to be the best node. Once the job execution begins, it is not moved, even if its run-time characteristics, or the run-time characteristics of any other jobs, are changed. After assigning the job in such a way causes the nodes to become much unbalanced. So an improvement in the spreading of task load is desirable, but it is accepted that this does not have to be optimal and that the load at each node need not be fully equalized. This relaxation allows these schemes to be devised that deal with a large-grain division of the workload, such as at the task level, and that use load transfers sparingly and thus do not require such high-speed communication between nodes. Non-preemptive load-balancing policies can be applied to any distributed system; however, they are particularly suited to systems, which have relatively low-speed internodes communication and tend to consist of performance heterogeneous nodes.

By contrast, a preemptive load-balancing policy allows load-balancing whenever the imbalance appears in the workloads among nodes. If a job that should be transferred to a new node is in the course of execution, it will continue at the new node. Since, in most systems, an initial distribution of jobs across nodes makes those systems appear balanced, they will become unbalanced as shorter jobs complete and leave behind an uneven distribution of longer jobs. Migration allows these imbalances to be corrected. However, to migrate a job in execution is much more complex and requires considerable overheads (caused by gathering and transferring the state of the job, resulting in performance degradation). If the preemptive policies were attempted in a loosely coupled large-scale system, the system performance would probably suffer significantly more, since there would be a large number of messages generated, which would congest the communication system. The preemptive policies are suitable for the distributed systems, in which the processing nodes are connected by a high-speed low-latency network.

2.3.3 Cluster-level versus Grid-level

When a job arrives at a cluster, the load-balancing system of the site will analyze the load situation of every node in the cluster and will select a node to run the job. Even though the cluster is heavily loaded, each job must queue in the cluster and wait to be processed. We classify this kind of load-balancing as cluster-level load-balancing, for which the objective is to optimize the system performance in a single cluster. Many traditional load-balancing algorithms fall in the category of site-level [17].

On the contrary, if a site lacks sufficient resources to complete the newly arriving tasks, or the site is heavily loaded, the load-balancing system of the site will transfer some tasks to other sites, and will increase the system throughput and resource utilization in multiple sites. We call this load-balancing as grid-level load-balancing. The focus of this dissertation is on grid-level load-balancing.

2.3.4 Centralized versus Distributed

Load-balancing policies can be classified as centralized or distributed. Centralized policies may be considered as a system with only one load-balancing decision maker. Arriving jobs to the system are sent to this 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. It appears that this policy is closely related to the overall optimal policy, in that there is only one load-balancing decision maker, which makes all of the load-balancing decisions. The major disadvantages of centralized policies are the possible performance and reliability bottleneck due to the possible heavy load on the centralized job load-balancing decision maker. For this reason, centralized approaches are inappropriate for large-scale systems; furthermore, failure of the load-balancing decision maker will make the load-balancing inoperable.

On the other hand, distributed policies delegate job distribution decisions to individual nodes. Usually, each node accepts the local job arrivals and makes decisions to send them to other nodes on the basis of its own partial or global information on the system load distribution. It appears that this policy is closely related to the individually optimal policy, in that each job (or its user) optimizes its own cost (e.g., its own expected average response time) independently of the others. The distributed load-balancing is widely used to handle imperfect system load information.

There are two kinds of hybrid models. One is a combination of fully centralized and distributed algorithms [18]. The other is a hierarchical model, which combines partially centralized and distributed algorithms to overcome some of the limits of fully centralized algorithms [19]. The first model is applicable only for small-scale distributed systems; the latter still has fault-tolerance problems, due to single point of failure in a set of manager nodes of clusters. The system is logically divided into clusters, and each cluster of nodes will have a single node that maintains the state information on the nodes within the cluster. The state information on the whole system is maintained in the form of a tree, where each tree-node maintains the state information on the set of processing nodes in the sub-tree, rooted by the tree-node. The hierarchical model can be simplified as two-level if the set of manager nodes are organized in a fully distributed style [19].

2.3.5 Partial versus Global information

How much load information on the system should be collected for load-balancing in the distributed policies is a major issue. Any dynamic load-balancing algorithms include a decision part, which may use load information from a subset of the whole system [20] or information from the whole system [21]. The former is called "partial decision base" and the latter "global decision base". For an initiating node, a subset of the whole system may be its nearest neighbors or nodes that are polled at random or formed by specific criteria. In all cases, the degree of the knowledge of the system load status and the accuracy of the redistribution decisions conflict. On one hand, more load information implies that there is a better chance of reaching a higher quality of load redistribution decisions. On the other hand, more load information also means more overhead to collect, and thus more chance for the load information to be out of date, unpredictably leading to an even worse load imbalance. Therefore, using detailed load information does not always significantly aid system performance, it is important to decide that when and how much load information should be collected.

2.3.6 Sender-initiated versus Receiver-initiated

Distributed load-balancing policies can be broadly characterized as sender-initiated and receiver-initiated. Sender-initiated algorithms let the heavily loaded sites take the initiative to request the lightly loaded sites to receive the jobs; receiver-initiated algorithms let the lightly loaded sites invite heavily loaded sites to send their jobs. Sender-initiated load-balancing algorithms perform better than receiver-initiated load-balancing algorithms at low or moderate system loads. At these loads, it is reasoned, the probability of finding a lightly loaded node is higher than that of finding a heavily loaded node; similarly, at high system loads, the receiver-initiated policy performs better since it is much easier to find a heavily loaded node.

As a result, adaptive policies have been proposed, which combine the desired features of both sender and receiver-initiated techniques, and are called symmetrically initiated [22]. They seek to find suitable receivers when senders wish to send jobs, and to find suitable senders when receivers wish to acquire jobs. Efficient symmetrical policies behave as sender-initiated under low and medium load conditions, and as receiver-initiated under heavy load conditions.

2.4 Policies for Dynamic Load Balancing Algorithms

Many issues involved in dynamic load-balancing have already been addressed in load balancing algorithms, such as how to measure the load of a processing node, how much load information we should collect and where they should reside. However, the real activities happening for different algorithms on differently designed systems may differ significantly. These issues are usually grouped into several policies (or components) at a higher level. Although the grouping of the issues and the naming of the policies may differ significantly among studies, they tend to discuss in common a set of key issues. In this section, we regroup the issues, name the policies, and discuss their possible choices. The policy names may or may not mean the same as in other studies.

Information policy: this decides what, when and where information about states of other nodes is collected.
Transfer policy: this determines whether a node is in a suitable state to participate in a task transfer.
Selection policy: this decides which task should be transferred, if the node is a sender.
Location policy: this locates a suitable transfer partner.

2.4.1 Information Policy

Information policy covers most issues related to the load information necessary for making load-balancing decisions. Information policy decides what information is collected, and when information about the states of other nodes is to be collected, and from which nodes. It is also responsible for the dissemination of each node load information.

2.4.1.1 Load Measurement Rule

Measuring the load of the various nodes in the system accurately is very important for the success of a load-balancing algorithm. Measuring the load of the nodes in a distributed system is an extremely difficult task. Usually, load is measured by a metric, the "load index". A number of possible metrics have been studied in the past. These can be broadly divided into two main categories: simple and complex.

Simple indices: They consider the load on only a single resource. This approach usually focuses on the load on the CPU. A simple load index includes processor queue length, average processor queue length over a given duration, the amount of memory available, the context switch rate, the system call rate, and CPU utilization.

Complex load indices: They consist of a number of metrics, each relating to a single resource, such as CPU, disk, memory and network. The metrics that make up the load index may be combined to give a single load value or may be represented as a tuple consisting of a number of elements, one per metric.
A candidate load index should be easy to compute and correlate well with the parameter (e.g., the job response time) that is to be optimized. It has been found that simple load

indices are particularly effective and impose less overhead. One of the most effective load indices is simply the processor queue length, and this choice seems to be unanimous.

In a heterogeneous environment, the load indices from different nodes must be adjusted to make them comparable. For example, if two different nodes have different processing power, their CPU utilization may have to be divided by their processing power to compare their CPU utilization load index values. A better measurement may be the total job execution time but in most cases the execution time of a job cannot be predicted accurately, it can be estimated by parameters such as the size of the program, the type of the job, or based on past statistics and experience.

2.4.1.2 Load Information Exchange Policies

The information exchange policies can be broadly classified into three types, although hybrid versions of these types may exist.

Demand driven policies: Each node collects information when it needs it to make a load sharing decision. A poll limit is usually used. The main advantage of demand driven policy is that load information is exchanged only when it is required. This policy has the following disadvantages in practice.

- When most of the nodes are heavily loaded, they continue to poll each other for the sparse lightly loaded nodes create repeated polling, which results in wasting the processing time of the polling nodes and polled nodes. In the worst case, polling may cause system instability when all the nodes are heavily loaded.
- Repeated polling generates a large amount of network traffic. This problem becomes more significant if the network bandwidth is limited.
- As the job needs to wait for the polling result, polling will increase the response time of the waiting job. This is a problem if the communication delay is significant.

It is difficult to obtain a good value for the probe limit. The probability of a successful poll (the hit ratio) depends on the load level in the system; no predetermined number of polls can guarantee a hit. In a medium-to-heavily loaded system, if the probe limit is small, lightly loaded nodes may not be discovered. If the probe limit is large, then (i) most of the heavily loaded nodes may find the same lightly loaded nodes and dump their loads to them; and (ii) the problems caused by repeated polling will multiply.

Periodic policies: Information is disseminated or collected at regular intervals. This is simple to implement. However, it is important to determine the most appropriate dissemination period as overheads due to periodic communication increase system load and reduce scalability. Here, a fixed amount of state collection overhead will be induced in the system because each node collects and maintains state-information of other nodes at regular interval, regardless whether this information will be used. However, there is no polling delay when a task must be transferred.

State-change driven policies: Nodes issue information about their load state only when it changes by a certain amount. Determining the threshold value is problematic, because the policy must be sensitive to significant changes but not to minor fluctuations. State-change policies generally have lower communication rates than periodic policies. However, if the state at a particular node does not change for a long period of time, the information held about that node will become stale. Aged load-state information is unreliable, since there is no way of telling if the node has crashed or has just not sent a message due to a steady state. A newly joining node will not receive information concerning steady-state nodes, even if those nodes are suitable transfer partners. One way to improve the basic state-change policy is to introduce additional dissemination messages, which are sent if the load-state does not change for a long period of time. These rules differ from demand-driven rules in that each node takes the initiative for disseminating its own state instead of collecting other nodes information.

2.4.2 Transfer Policy

A transfer policy determines whether a node is in a suitable state to participate in a task transfer, either as a sender or a receiver. Many proposed transfer policies are threshold policies, which may be either based on fixed or adaptive thresholds. One way is to set one threshold value for the load imbalance (the difference between the largest and smallest loads on the nodes). If the detected load imbalance is bigger that a preset threshold value, the transfer is initiated. An equivalent method to this is to set two threshold values, T_h and T_l, by which the nodes are classified into three types, i.e., heavily loaded or sender (if loads higher than T_h), lightly loaded or receiver (if loads lower than T_l), and normally loaded otherwise. Depending on the algorithms, T_h and T_l may or may not have the same value. The choice of these thresholds is fundamental for the performance of the algorithm. Clearly, the best threshold values depend on the system load and the task transfer cost. At low loads and/or low transfer costs thresholds should favor task transfers, while at high loads and/or high transfer costs remote execution should be avoided. [22] Present a technique that efficiently and in run-time adapt the threshold to the system load.

Fixed threshold policies mean that the threshold values are not changed when system loads are changed. There are disadvantages with the fixed threshold policy. If the fixed threshold value is too small, this still causes "useless" job transfers. If the fixed threshold value is too large, the effect of using a load-balancing mechanism may be reduced. Other than using fixed threshold values, thresholds can be set in an adaptive (relative) fashion, by adjusting them when the global system load is changed. In [19], if the load of an individual node is above or below the average load over a certain domain (either the global or some local range) by a preset percentage, then load-balancing actions are initiated and load is balanced either locally or globally. In [22] adaptive approach has been used to determining proper thresholds, the average load L_{avg} is determined first. Two constant multipliers, H and L, are used in computing the heavy threshold, T_h, and light threshold, T_l. H is greater than one and L is less than one. These two values determine the flexibility and the effectiveness of a load-balancing mechanism. The heavy threshold, T_h , is computed as the product of H and L_{avg}. Similarly, the light threshold T_l is computed as the product of L and L_{avg}.

The transfer policy may be either periodic or event-triggered. The algorithm may periodically check whether the node's state qualifies itself as a candidate for a task transfer. However, the great majority of the policies proposed in the literature are event triggered. If the state of a node changes, a task transfer may be possible. The state of the node may change because either a task has ended or a new task has arrived.

Symmetrically-initiated transfer policies support load transfers initiated by both busy and low-loaded nodes. Symmetrically-initiated algorithms are more complex, but allow the advantages of both sender-initiated and receiver-initiated algorithms to be exploited. Symmetrically-initiated schemes are potentially unstable: there must be a zone between the activation thresholds for the sender and receiver parts of the algorithm so that a node cannot rapidly move between sender and receiver states.

2.4.3 Selection Policy

The role of selection policy is to select tasks for transfer. In sender-initiated schemes, busy nodes choose tasks to transfer to another node, whereas in receiver-initiated schemes, lightly loaded nodes inform potential senders of the types of task they are willing to accept. The policy determines how much load, or how many tasks, to transfer.

A task transfer may be preemptive or non-preemptive. Preemptive transfers involve transferring a partially executed task. This is generally expensive, as it involves collecting all of the task's state. Non-preemptive-task transfers involve only tasks that have not begun execution and hence do not require a transfer of the task state. A node may be overloaded and have no tasks available for non preemptive transfer if it is polled by a receiver. A selection policy should consider at least these factors.

- The overhead incurred in transferring the task should be minimized. Non-preemptive transfers and small tasks (means small amounts of information) carry less overhead.
- The execution time of the transferred task should be sufficient to justify the cost of the transfer. Even if task execution is unknown, it should be possible to classify the tasks as short or long tasks, and to consider only the long tasks for migration.

2.4.4 Location Policy

The responsibility of location policy is to find a suitable transfer partner. Location policies can be distributed, each node selecting a transfer partner on the basis of locally held information. Location policy, corresponding to information policy, specifies the balancing domain for load-balancing actions; this could be global, nearest-neighbors, a group of random polled nodes, or a set or cluster of nodes based on specified criteria. Alternatively, policies can be devised using a central information source. Busy nodes attempt to locate transfer partners that have low load levels in sender-initiated schemes. In receiver-initiated schemes, low-loaded nodes attempt to locate a busy node from which to transfer work. Five typical policies are listed below.

Random policies: A transfer partner is selected at random, and it's load-state is ignored. This can result in useless task transfers when an already-busy node receives extra work, but has been shown to provide performance improvements over no-load-distribution. The performance improvements stem from the fact that only busy nodes transmit load, while all nodes are potential receivers. Random location policies work best when there are few heavily loaded nodes and many relatively idle nodes.

Threshold policies: The node randomly selects a potential destination node for the job and probes it to determine its load index. If the load index at the proposed destination is less than or equal to a preset threshold value, that node becomes the job's receiver. Otherwise, another node is randomly selected and probed. Probing continues until a receiver is found or until the number of nodes probed is equal to a limit *Lp*. Threshold location policies are based on the result of the probing activity; if a receiver has been found, the job is sent there otherwise the job is executed locally.

Lowest policies: Like threshold policies, lowest policies employ a threshold *Lp*. However, lowest policies differ from threshold policies in that it probes a group of nodes until a node with a zero load index is found, or until exactly *Lp* nodes have been probed. The lowest location policy is to select the probed node with the lowest load index as the execution node for the incoming job, provided that the load index at that node is less than a preset threshold value.

Preferred list: Based on the topology of the system, each node orders all other nodes into a preferred list. A node is the *k*-th preferred node of one and only one other node, where *k* is an integer. If node *i* is the *k*-th preferred node of node *j*, then node *j* is also the *k*-th preferred node of node *i*. When a node is overloaded, it will contact the first node found in its preferred list, and attempts to transfer a task to that node. Although the preferred list of each node is generated statically, the actual preference of the node in transferring a task may change dynamically with the states of nodes in its preferred list. If a node's most preferred node becomes overloaded, its second preferred node will become the most preferred.

Least policies: To differentiate from the location policy lowest, we call this class of location policies "least". Least policies differ from lowest policies in that they do not need to probe nodes, and no threshold is used. The least location policy is to select the node with the smallest load index as the destination node for dispatching the jobs on the basis of the information on a specified balancing domain. In a heterogeneous environment, a node with minimal load, i.e., queue length, does not mean the best transfer partner for a certain task. Node processing power and task transfer delay incurred among the node and remote nodes should also be considered in location policy.

2.5 Literature Review

Numerous dynamic load balancing algorithms have been proposed for computational grid. The factors on the basis of which load balancing algorithms can be compared are
Communication overhead: Communication overhead is the status information which each node has to provide to other nodes in the grid.

Response time: Amount of time that elapses between the job arrival time and the time at which the job is finally accepted by a node.

Scalability: It is the ability of the algorithm to perform load balancing for a grid with any finite number of nodes.
Fault tolerance: It is the ability of the algorithm to perform uniform load balancing in spite of arbitrary node or link failure.

Reliability: It is the ability of the algorithm to schedule job in predetermined amount of time.

Stability: It is defined as the maximum job arrival rate which the load balancing algorithm. In [17] an efficient desirability-aware load balancing algorithms has been presented .it is novel approach taking two factors for the desirability of sites, which are processing power and communication delay, respectively. For each site si in the grid, this algorithm uses desirability of other sites to si to form k number of partners and p number of neighbors for the site si. A new job arriving at a site si is immediately distributed to the site si or its partner sites. Continuous load adjustment is employed among neighbor sites. In order to reduce/minimize the state-collection overhead in this LB algorithm, state information exchange is done via mutual information feedback. So in this case when a node failure occurs the information belonging to that node will not change because there will be no feedback from the failed site, resulting in sending the task from other sites to this site which no longer existed. So creating unnecessary network congestion and communication overhead and increasing the completion time of the jobs.

Belabbas et el., [19] proposed a task load balancing model in Grid environment. It is a tree-based model to represent Grid architecture in order to manage workload. This model is characterized by three main features: (i) it is hierarchical; (ii) it supports heterogeneity and scalability; and, (iii) it is totally independent from any Grid physical architecture. It uses a hierarchical load balancing strategy to balance tasks among Grid resources. The main characteristics of the proposed strategy are: (i) it uses a task-level load balancing; (ii) it privileges local tasks transfer to reduce communication cost; and, (iii) it is a distributed strategy with local decision making. The proposed strategy works very well for small jobs but in case of big jobs this approach will result in communication overhead , increased response time and finish time of the task.

Abed et al., [23] proposed a bidding approach. In it controller first tries to distribute the jobs uniformly among its own nodes. It sends the jobs to some other controller if it is unable to perform load balancing uniformly. The controller broadcasts the load balancing request to other controllers in the network. Each controller node submits a bid. The controller assigns the set of jobs to that controller node which submits the highest bid.

This algorithm is a mix of sender initiated and receiver initiated load sharing algorithms. It is a receiver initiated in the sense that a node can bid high price to discourage the load from other controllers. It is a sender initiated, as the controllers have node load information, so it can form its bidding set accordingly. The performance improves as the number of nodes in the system increases, but when the number of nodes exceeds the number of jobs, no further improvement results.

Shah et al., [24] overcome the drawbacks posed by the bidding approach by proposing that each node in the grid calculates its status parameters during the status exchange interval. At each periodic interval of time called the status exchange interval, each node in the system calculates its status parameters, which are the estimated arrival rate, service rate, and load on the node. Each node in the system exchanges its status information with the nodes in its buddy set. Each status exchange period is further divided into equal subintervals called estimation interval. As each processor balances the load within its buddy set, every processor estimates the load in the processors belonging to its buddy set at each estimation instant. The status exchange instants and the estimation instants together constitute the set of transfer instants. At the transfer instants, the rescheduling of jobs is carried out. Thus, the decision to transfer jobs and the actual transfer of jobs is done at the transfer instants. Although this is a decentralized load balancing algorithm which considers job migration cost, communication delay is large in it. This becomes a disadvantage when this algorithm is applied to small-scale grids. With these approaches, jobs need to wait till the next transfer instant for migration, and due to the random arrival rate and service rate at each node, it is possible that the load does not get distributed evenly across all nodes. In this case, there can be large waiting times at highly loaded nodes, whereas lightly loaded nodes continue to remain idle.

Acker et al., [25] proposed a new decentralized dynamic job dispersion algorithm that is capable of dynamically adapting to changing operating parameters. This distributed load-balancing algorithm is dynamic, decentralized, and it handles systems that are heterogeneous in terms of node speed, architecture and networking speed. The algorithm allows individual nodes to leave and join the network at any time and have jobs assigned to them as they become available. Each node saves information about its neighbors including the network bandwidth available between the local resource and its neighbor; the current CPU utilization; and the current I/O utilization. This status information is exchanged periodically. Knowing the status information each node can choose when to send jobs to its neighbors. This algorithm is fault tolerant and takes job size into account. But it will decrease the performance when nodes failure increases because the job on that node has to be moved on some other node whenever a failure occurs. So it will increase the job completion time as well as communication overhead.

Bin Lu et al., [26] proposed a new task scheduling and resource allocation algorithm, which can not only increase the utilization of resources and system throughput, but also realize the load balancing within Grid systems. This algorithm consists of three main modules, they are load tracking module, job distributing module and load monitoring module. It considers the task number and the performance of every DRM (domain resource manager), as well as the situation of current load. This algorithm will increase the task response time and also balance the load but create the bottleneck on GRM (grid resource manager) because this algorithm uses the hierarchical model of grid and the user requests are coming from top to bottom way.

Saravanakumar et al., [27] proposed an algorithm, Load Balancing on Arrival (LBA) for small-scale grid systems. It is efficient in minimizing the response time for small-scale grid environment. When a job arrives LBA computes system parameters and expected finish time on buddy processors and the job is migrated immediately. This algorithm estimates system parameters such as job arrival rate, CPU processing rate and load on each processor to make migration decision. This algorithm also considers job transfer cost, resource heterogeneity and network heterogeneity while making migration decision but it has not taken the task failure into account and also didn't work well for large scale grid environment.

An enhanced ant algorithm for load balancing in grid is proposed in [28]. This algorithm determines the best resource to be allocated to the jobs based on job characteristics and resource capacity, and at the same time to balance the entire resources. It focuses on local pheromone trail update and trail limit. This is a technique to control the value of pheromone updated on each resource. The local pheromone trail update will reduce the amount of pheromone in visited resource, so the resource they have visited is less desirable for other ants while the trail limit, which is the allowed range of the pheromone strength, is limited to maximum and minimum trail strength. This algorithm does not provide a way about the preemption of jobs from the resources and results in many resources being idle and many other being overloaded most of the time.

P. K. et al., [18] proposed a decentralized grid model, as a collection of clusters. It introduces a Dynamic Load Balancing Algorithm (DLBA) which performs intra cluster and inter cluster (grid) load balancing. DLBA considers load index as well as other conventional influential parameters at each node for scheduling of tasks. But this approach didn't consider the communication cost, task time parameters and failure nature of nodes. On grid level this algorithm will be decentralized but on cluster level it will become centralized because the load balancing decisions will depend on cluster server. So congestion on the cluster server or failure of this server will result in performance digression. The Decentralized Recent Neighbor Load Balancing Algorithm for Computational Grid presented in [29] has the same drawbacks

Mohsen et el., [30] proposed a probabilistic scheduling algorithm for load balancing purpose, in this algorithm cost, deadline and resources behavior have been considered. Probabilistic algorithm chooses the resources that have better past and least completion time. But in this approach the resources that have greater completion time always get ideal and a bottle neck may be created on the resources that have least completion time.

Jingyi [31] proposed a a novel heuristic genetic load balancing algorithm which takes advantages of genetic algorithm and applied to solve grid computing load balancing problems. it uses the Elistim selection procedure for the experiment . Elitism is to copy best solutions in present population to next generation. After selecting chromosomes with respect to the evaluation function, genetic operators such as, crossover and mutation, are applied to these individuals. Crossover refers to information exchange between individuals in a population in order to produce new individuals. The idea behind a crossover operation is as follows. It takes as input two individuals, selects a random point, and exchanges the sub-individuals behind the selected point. On the other hand, mutation is an operation that defines a local or global variation in an individual. Mutation is traditionally performed in order to increase the diversity of the genetic information. This algorithm works well for large scale heterogeneous grid system as long as flow of the coming tasks over grid is in a continuous manner. But when the tasks start arriving on an node in a unpredictable manner like at a moment large no of tasks arrived on the system and the next moment very less , results in performance digression and system overload.

Above load balancing algorithms take the dynamic runtime environment of the system into consideration before assigning jobs to the node. Some of them are adaptive to fault tolerance in the sense that the nodes will only have the status information about the other nodes which are currently available. Therefore jobs will not be sent to such node which is not available due to failure. Above algorithms are also scalable but at a certain cost in terms of load balancing time or communication overhead.

Most load-balancing policies execute two activities that require communications: distribute its own load information and collect other nodes information and transfer tasks. If each node is required to interact with other nodes, it will have to use mechanisms such as broadcast, global gathering, long-distance communication; which are not scalable and create intolerable overhead or congestion in systems with a large number of nodes.
To reduce this overhead, in many policies, a node only exchange information and transfer tasks to its physical and/or logical neighbors. These are usually called "neighbor-based" load-balancing algorithms. Clustering is another technique to tackle the problem. The nodes can be partitioned into clusters based on network transfer delay, where load balancing operates on two-level: intra-cluster and inter-cluster via cluster managers or brokers. These are usually called "cluster-based" load-balancing algorithms.

2.6 Research Gaps

- All Existing load balancing models for computational grid are either the combination of fully centralized and distributed algorithms or a hierarchical model, which combines partially centralized and distributed algorithms. So there is a need to develop a fully decentralized load balancing model to overcome the limits of existing load balancing models.

- The time taken in the migration of a job from one node to other over Grid network should not be greater than the execution time of the job. Because it is not desirable if a job spend more time in migration rather that execution.So there should be a limit on the number of count, a job will migrate from one node to other and should be proportional to the job size.
- The priority of the different type of jobs and their deadline time has not been taken into account. Such that if there is a higher priority job in the queue then it must be allowed to execute first by the processor and we also have to take into consideration that a low priority job does not just spend its time waiting in queue, so preventing starvation.
- Each node collects information when it needs to make load sharing decision. The main advantage is that load information is exchanged only when it is required. But when the system load gets heavy, they continue to poll each other for the sparse lightly loaded site results in repeated polling. It increases the chance of system instability, communication overhead, response time of the job and congestion over the network. So it should be avoided.
- In most of the existing load balancing algorithms faulty nature of the resources in grid has not been taken into account. So there is need to use a predictive approach which submitted the task on the resources by analyzing it's past so that the probability of re-execution of partially executed jobs after the failure of resource will be less. The best way to avoid it to submit only small tasks on the resources which have bad past.

To fill the research gaps discussed above a decentralized load balancing algorithm has been proposed in this report. It uses the hierarchical load balancing model with adaptive load balancing approach according the current state of the nodes. And uses the demand based load sharing information between nodes.

Chapter 3

Proposed Load Balancing Algorithm

The various research gaps mentioned in the previous chapter are addressed by the proposed load balancing algorithm. In this section we first discuss the proposed system model which is used by proposed decentralized load balancing algorithm. After then we will discuss about the design of the algorithm with it's various modules.

3.1 Proposed System Model

The Grid system consists of large number of heterogeneous computational resources, connected via communication channels through an arbitrary topology (Fig. 3.1). The differences among resources may be in the hardware architecture, operating systems, processing power and resource capacity. In this study, heterogeneity only refers to the

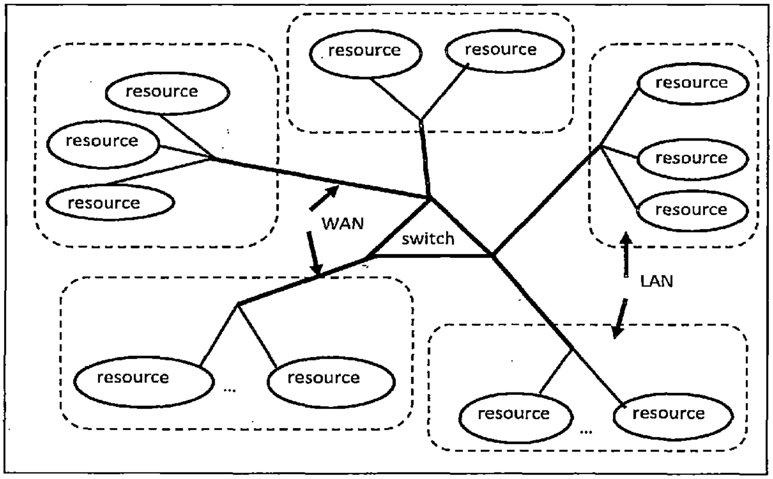


Figure 3.1 Grid Topology

processing power of the computational resources. Since in a Grid environment, the network topology is varying, the proposed model captures this constraint by considering an arbitrary topology in which the data transfer rate is not fixed and varies from link to link. The resources that are directly connected to other resources under the same domain forms a set, we can consider is as site. It may be noted that two neighboring sites may have few resources common to each site. So logically, grid architecture can be divided into three levels: Grid-Level, Site-Level and the Resource-Level as shown in figure 3.2.

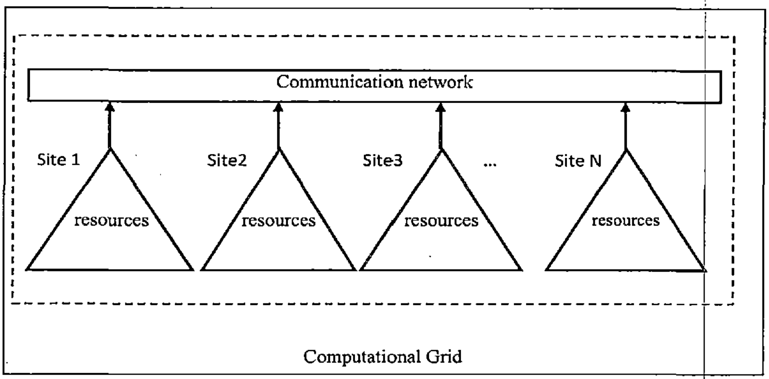


Figure 3.2 Grid Architecture

The processing power of the grid is measured by the combination of processing power of the sites and the processing power of a site is measured by the combination of processing power of resources under that site. It is assumed that the sites in the grid are fully interconnected, meaning that there exists at least one communication path between any two sites in the grid. Inter-site communication is achieved through message passing. There is a non-trivial communication delay on the communication network between the sites and that delay is different between different pair of sites. The underlying network protocol guarantees that messages sent across the network are received in the order sent.

For any computing resource in the grid, there are tasks arriving at the resource for execution. It is assumed that each processor has an infinite capacity buffer to store jobs waiting for execution. This assumption eliminates the possibility of dropping a job due to unavailability of buffer space. The tasks submitted by the user are assumed to be computationally intensive, mutually independent and can be executed at any resource which satisfies the QoS (quality of service) requirement of user, such as cast of execution. No deterministic or priori information about the task arrival is available. The tasks are assumed to arrive randomly at the resources, the inter-arrival time being randomly. As soon as the task arrives, it must be assigned to exactly one resource for processing, satisfying the QoS requirement. When a task is completed, the executing computing resource will return the results to the originating user of the task. Each task is assigned a timer when it's generated if the timer reaches a threshold and the task is not processed, the task is given the highest priority for execution.

In a computing environment, task migration is the only efficient way to guarantee that the submitted tasks are completed reliably and efficiently even though a computing resource failure occurs. But in case of resource failure the partially executed job have to be re-executed on some other resource, it results in increased communication cost and finishing time of the task. So it is desirable to send only small tasks on the recourses on which fault is more likely to occur. Lightly balanced nodes are determined using the symmetrically initiated load balancing strategy. It is desirable to transfer the load from highly loaded recourses to lightly loaded resources in such a manner, so that it will result in increased utilization of resources, reduced response time and finish time of the tasks with less communication overhead.

3.1.1 *Communication Model*

Components of the proposed communication model are:

Resource manager (RM): Every resource in the system runs a resource manager and it is local to that resource. Resource manager is responsible for calculating the current load on the resource, arranging the tasks in the queue on the basis of priority. And sending the information about the current load and number of processes currently available (waiting in the queue and running on the processor) to the Site

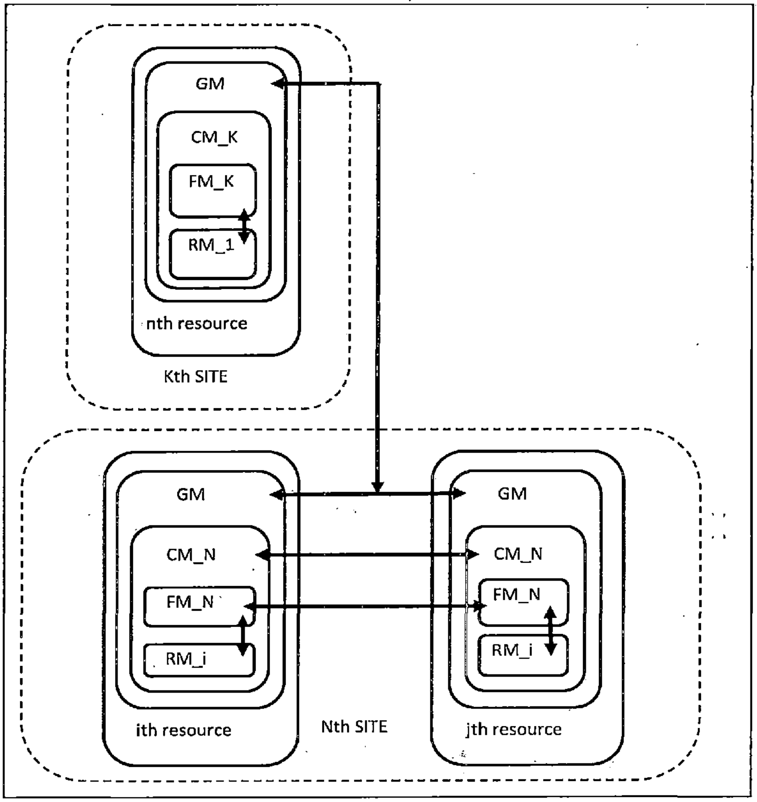


Figure 3.3 Communication Model

manager (SM) associated with it. Considering the heterogeneity of resources the workload index has been taken as a ratio of the total numbers of processes currently residing in a resource running or waiting in queue with it's processing power, so

$$RL = \frac{M}{PC} \tag{1}$$

Where *RL* shows the current load on the resource, *M* number of total tasks on the resource and *PC* processing power of the resource. It also manages the current load balancing status (*LS*).

Fault manager (FM): It works at site level. Each site has their separate fault manager. Fault manager runs on each resource under a site and is global to all resources under that site. It detects the occurrence and type of resource failure by analyzing the information about state of a resource and transfers the information about the failure to Site manager (SM). When FM receives the information about failure, it tries to resolve failures. Resource failures can be the node crash or network failure. Fault manager guarantees that the tasks submitted are completely executed using available resources. The fault manager is responsible for check pointing.

Site manager (SM): site manager also works at site level. Each site has their separate site manager. SM runs on each resource under a site locally and is global to all resources under that site.SM can fully control the resources of the site to which it belongs, but cannot operate the resources of other sites directly. It is responsible for sending the information to the Grid Manager (GM) about its average site load and average processing power at different time intervals. SM of a site maintains the information about current load(*RL*), processor capacity(*PC*), communication bandwidth(*CB*) , availability status(*S*), fault index (*FI*), fault time (*FT*) along with registration information of resources within that site. The availability status shows that the particular resource is currently available or not, fault index shows the number of time failure has been occurred and the fault time shows the amount of time after which the probability of failure of a resource is high. Fault time (FT) is calculated by using the exponential moving average.

$$FT_{new} = FT_{prev} + \frac{2}{(1+FI)}(RFT - FT_{prev}) \tag{2}$$

Where FT_{prev} shows the previous value of FT and RFT shows the amount of time for which the resource was available and FT_{new} shows the new estimated value of FT. If a resource is not available for a time period which is twice of FT, then site manager deletes the registration entry for that resource.

Grid manager (GM): Grid manager work at grid level. GM runs globally on each resource which are part of the grid. It has the information about the average load on sites (SL), current site status(under loaded or overloaded), communication bandwidth and each site will have the information about it's neighbor sites (the sites which are directly connected to each other).

3.1.2 Task Model

For task *ti* that belongs to a global task set T the following functions have been defined:
BornNode(ti): denotes the originating computing node for the task ti.

ExeNode(ti): denotes the executing computing node for the task ti.

ArrivalTime(ti): denotes the arrival time of task ti, which is the time when the task is generated at BornNode(ti) and submitted to the grid for execution.

Sub*Time(ti)*:denotes *the* time *of submission of task on a resource on which it is currently* submitted for execution.

FinishTime(ti): denotes the time when task ti has completed and the results has been send from ExeNode(ti) to BornNode(ti).

RespTime(ti): denotes the completion time of ti. It is the difference between the finishTime(ti) and RespTime(ti).

DeadlineTime(ti): It is the expected time for a task ti under which it is supposed to be completed. DeadlineTime(ti) of a task can be decide by the knowledge of type of task it is and from the predictions of it's previous results that how much time these type of tasks take in completion .

WaitTime(ti): Denotes the total amount of time which task has spend waiting in the queue.

probCount(ti): Denotes the number of time a job has been migrated from one resource to other for load balancing.

3.1.3 Load Balancing Model

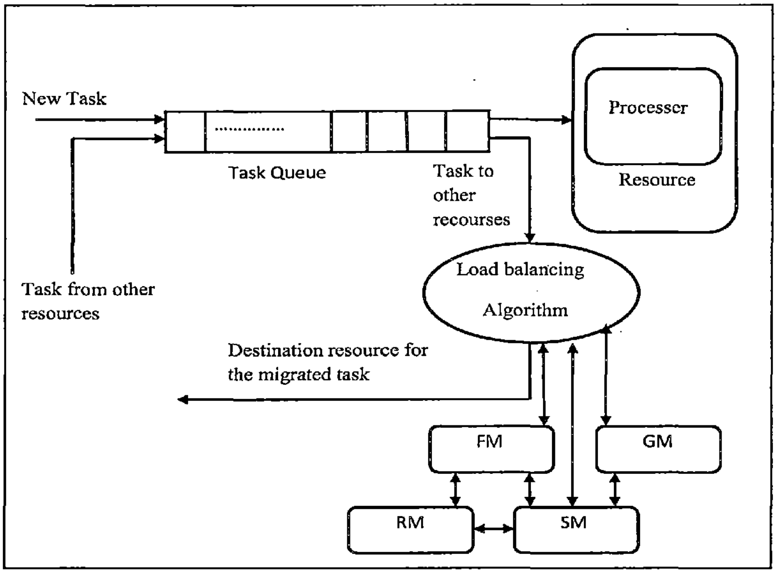


Figure 3.4 Load Balancing Model

The proposed load balancing model supports heterogeneity and scalability of grid, resources can connect and disconnect the grid at any time. It is totally independ from any physical architecture of a grid. The model works in a hierarchical way at three levels resource level, site level, grid level.

Level 0: In this first level, we have the GM, which realizes the following functions:

(i) Maintains the workload information about the site and the information about the neighbors of the site to which the sites are directly connected.

(ii) Decides to start a grid level load balancing between the sites of the Grid, which is intra-Grid load balancing.

(iii) Sends the load balancing decisions to the SM on level 1 for execution.

Level 1: The SM and FM of a site work at this level. In this load balancing model, they are responsible to:

(i) SM Maintain the current *RL,S, FI, FT* of it's each resource.

(ii) Estimates the load of associated site and send this information to the GM.

(iii) Decides when to start a local load balancing, which we will call intra-site load balancing.

(iv) Send the load balancing decisions to the resources which it manages, for execution.

(v) FM sends information about the failure of a resource to SM and also responsible for the check pointing of the tasks currently running on a resources.

Level 2: At this last level, we find resources of the Grid linked to their respective sites. RM works at this level, it is responsible to:

(i) Maintain workload information of the resource and load balancing status (*LS*).

(ii) Send this information to SM of the resource.

(iii) Arrange the task queue in a priority manner taking WaitTime(ti) as key value.

(iv) Perform the load balancing decided by its SM or GM manager.

3.2 Performance Metrics

In this work three performances metrics have been considered which relevance at three different levels. At the job level, we consider the ART (average response time) of the jobs processed in the system as the performance metric. If n jobs are processed by the system, then

$$ART = \frac{1}{n}\sum_{k=0}^{n}(finish_k - arrival_k) \quad (3)$$

Where $Arrival_k$ is the time at which the kth job arrives, and $Finish_k$ is the time at which it leaves the system. The delay due to the job transfer, waiting time in the queue, and processing time together constitute the response time. At the system level, we consider the total execution time (*TET*) as the performance metric to measure the algorithm's efficiency. It indicates the time at which all n jobs get executed. At the resource level, we consider CPU utilization as the performance metric. It is the ratio between the processor's busy times to the total (ideal+busy) time of the processor.

$$U_k = \frac{Busy_k}{Busy_k + Ideal_k} \qquad (4)$$

Where $Busy_k$ indicates the amount of time processor P_k remains busy, and $Idle_k$ indicates the amount of time P_k remains idle during the total execution time of N jobs.

3.3 Design of Load Balancing Algorithm

In accordance with the hierarchical structure of the proposed model, we distinguish two load balancing levels: Intra-sites (or Inter-resources) and Intra-Grid (or Inter-sites):

Intra-site load balancing: In this first level, depending on current workload on its own resources, each site manager decides whether to start or not a local load balancing operation. If it decides to start a load balancing operation, then it tries, in priority, to balance its workload among its resources. Hence, grid can perform N parallel local load balancing, where N is the number of sites.

Intra-Grid load balancing: The load balancing at this level is performed only if some sites get over loaded while other are lightly loaded. In this case, tasks of overloaded sites are transferred to underloaded ones regarding the communication cost and according to the selection criteria. The chosen underloaded sites are those which needs minimal communication cost for transferring tasks from overloaded sites. The main advantage of this strategy is to privilege local load balancing in first (within a cluster and then on the entire Grid). The goal of this neighborhood approach is to decrease the amount of messages exchanged between sites. As a consequence, the communication overhead induced by tasks transfer and flow information is reduced.

3.3.1 Load Balancing Strategy

The proposed load balancing algorithm uses four steps to balance the load. As the description will be done in a generic way, here i will use the concept of *group* and *element*. Depending on cases, a group designs either a site or the Grid. The steps of our strategy can be summarized as follows:

Step 1: Start load balancing and Information update

Whenever a new task arrives for grid or finish on a resource, performs these actions:

(i) If the task arrives or finish on a resource and load balancing status (LS) of resource is 0 then performs steps which are below else set LS to 1.

(ii) Calculate the current workload on the resource and send this to SM.

(iii) Calculate the current average workload on site and send this to GM.

Step 2: Estimation of the current load deviation value LD_S (for Site) and LD_G (for grid)

(i) Estimates current average workload of the site based on workload information received from its resources.

(ii) Send this site workload information to *GM.*

(iii) Estimates current average workload of the grid based on workload information received from its sites.

(iv) On grid level compute LD_G and on site level computes LD_S by using the standard deviation over the workload index under it's elements in order to measure the deviations between them.

Step 3: Decision making

In this step the load balancing algorithm decides whether it is necessary to perform a site level load balancing or a grid level load balancing operation or not. For this purpose it executes the following two actions:

(i) *Defining the balance/imbalance state of the group (site or grid):* If we consider that the standard deviation measures the average deviation between the workload of elements of their group, then we can say that this group is in balance state when this deviation is small. In practice, we define a *balance* *threshold*, denoted as ε, from which we can say that the standard deviation tends to zero and hence the group is balanced., thus can write the following expression:

If (($LD_{Si} \leq \varepsilon$*) Then* the site S_i is balanced *Else* It is unbalanced.

If (($LD_G \leq \varepsilon$*) Then* the grid is balanced *Else* It is unbalanced.

Where *LD* shows load deviation.

(ii) *Group partitioning*: For an imbalance case, we determine the overloaded elements (*sources*) and the underloaded ones (*receivers*), depending on the load index of every element relatively. The elements which have their load index less than the average load index of the group are under loaded and others are overloaded.

Step 4: ***Tasks transferring***

In order to transfer tasks from overloaded elements to under loaded ones, the proposed load balancing algorithm uses the following rules:
In case of intra site load balancing:

(i) Transfer the task with ProbCount(ti) less than the migration threshold, to underloaded resources taking communication cost and task priority into consideration.
(ii) Arrange the tasks in the task queue in priority order using Wait Time (ti) as priority key.
(iii) Set LS status of the resources to 1.

In case of intra grid load balancing

(i) Transfer task with ProbCount(ti) less than threshold value from overloaded sites to under loaded sites. In this case it is best to transfer the task from resources which have higher LI in overloaded sites to resources which have lower LI in underloaded sites with taking communication cost and task priority in consideration.

(ii) Arrange the tasks in the task queue in priority order using WaitTime (ti) as priority key.

(iii) Set load balancing status of the resources to 1.

Above in both cases before transferring the task to a resource, it is important to take into consideration the communication cast and failure time of that resource, if fault index value of the resource is greater than one. Taking failure time of a resource into consideration before transferring a task on it will increase the probability of completion of that task before resource failure occurs. And for task selection always select the last task from the queue which has been already shorted on priority basis.

3.3.2 Modules of the Algorithm

```
Procedure: Main

    For a recourse R When a new task arrive or finish on it do
    1. if ( it's LS is 0)                          // load balance status of resource R
      {
      1. Sends its workload $LI_R$ to its SM.
       2. SM calculates the current $ALI_{Si}$ and $APC_{Si}$ of the site $S_i$ where $SM \in S_i$.
       { $ALI_{Si} = 0$; $APC_{Si} = 0$;                 // $ALI_{Si}$ is average load of a site
         For all $Rj \in Si$ do        // $APC_S$ is average processing power of a site
         $ALI_{Si} = ALI_{Si} + LI_{Ri}$;   $APC_{Si} = APC_{Si} + PC_{Ri}$; // PCR processing power
                                                              //  of resource R
         End For
         $ALI_{Si} = ALI_{Si}/N$;   $APC_{Si} = APC_{Si}/N$; //where N is number of resources
                                                                    //under Si
       }
     3. Send information about current $ALI_{Si}$ and $APC_{Si}$ to GM.
     4. GM calculates the $ALI_G$ // $ALI_G$ average load of grid
// Procedure main continue
        For all sites Si ∈ G do
        $ALI_G = ALI_G + ALI_{Si}$;
      End For
        $ALI_G = ALI_G/M$; //where M is number of sites grid.
        }
    5. GM calculates the $LD_G$.  // $LD_G$ load deviation of grid
       {Initially set $LD_G = 0$;
         For all Si ∈ G do
        $LD_G = LD_G + (ALI_G - ALI_{Si})^2$;
       End For
         $LD_G = (LD_G /M)^{1/2}$;
        }
    6. If ($LD_G > \varepsilon$)
        Perform grid level load balancing.
    7. SM Calculates The $LD_S$.    // LDS is load deviation of a site s.
        {
        Set $LD_S = 0$;
        For all Rj ∈ Si do
        $LD_S = LD_S + (ALI_{Si}-LI_{Rj})^2$
        End For
        $LD_S=(LD_S/N)^{1/2}$;
        }
    8. If ($LD_S > \varepsilon$)
        Perform site level load balancing.
        }
   Else set load balancing status to 0.
End for
```

Procedure: Site load balancing for a site S

```
1. Partition the recourses of S in underloaded and overloaded resources.
   {
   For all Rj ϵ S do
   If (((($LI_{Rj}$ * $PC_{Rj}$) -1) / $PC_{Rj}$ ) > $ALI_S$)
   Then Rj ϵ OR          // where OR is overloaded resource set
   Else If (((($LI_{Rj}$ *$PC_{Rj}$) +1) / $PC_{Rj}$ ) < $ALI_S$)
   Then Rj ϵ UR          // where UR is underloaded resource set
   }
2. Short the elements of UR in Descending order of bandwidth and elements of
   OR in descending order of load.
3. While ($LD_S > \varepsilon$)
   {
   1. Set j=0;
   2. While (J< sizeof OR && j < sizeof UR)
   {
   1. Take Rj from OR.
          1. Take task t from the end of task queue which has lowest priority
             and and ProbCount(t) is less than migration threshold ;
          2. Increment ProbCount (t);
   2. Set i=0
   3. While ( i < sizeof UR )
      {
      1. Take Ri from UR.
      2. If ( ($ET_{Ri}$ + ( DeadlineTime(t) – currentclocktime)) <= $FT_{Ri}$)
         {
         Assign t to this resource Ri; End while;
         }i++;     // $ET_R$ estimated execution time of the tasks on R
//Procedure site load balancing continue

        4. If the task t has not assign to a resource then assign it to the Ri ∈ UR for
        which ($ET_{Ri}$ + ( DeadlineTime(t) – currentclocktime)) was minimum

        5. Increment j.

        }

        3. Update the workload on resources and short the tasks using WaitTime(ti)
        as key under site S.

        4. Update the set OR, US and value of $LD_S$.

        5. Short the elements of OR in ascending order of bandwidth and elements
        of UR in descending order of bandwidth.

        }

        4. Site is in balance state call return.
```

```
Procedure: Grid load balancing

    1. Partition the grid into underloaded and overloaded sites

      {

        For all Sj ∈ G do
        If (((($LI_{Sj}$ *$PC_{Sj}$) -1) / $PC_{Sj}$) > $ALI_G$)&( total tasks on S> number of
        resources in S)
        Then Rj ∈ OS            // where OS is overloaded site set
        Else If (((($LI_{Sj}$ *$PC_{Sj}$) +1) / $PC_{Sj}$ ) < $ALI_G$)
        Then Rj ∈ US            // where US is underloaded site set

    2. Short the elements of UR in descending order of bandwidth and elements of
    OS in descending order of load.
```

```
// procedure for grid load balancing continue

  3.  While ($LD_G > \varepsilon$)

      {

      1. Set j=0;

      2. While (J< sizeof OS && j < sizeof US)

       {
        1.  Take Sj from OS
        2.  Short the resources of Sj in decreasing order of their load
              1.Take task t from the end of task queue of resource R $\in$ Sj
                 having highest load and probe (t) of task is less than migration
                 thresold
              2.  Increment Prob (t);
        3.  Set i=0
        4.  While ( i < sizeof US )
            {

            1. Take Si from US.
            2. Short the resources of Si in increasing order of their load set k=0;
            3. While (k < number of resources in Si)
            {
              1. Take the $R_k$ resource from shorted list of site Si.
              2.  If ( ($ET_{Rk}$ + ( DeadlineTime(t) – currentclocktime)) <= $FT_{Rk}$)

                {
                 Assign t to this resource Ri;
                End while;
                } K++;
            }
// procedure for grid load balancing continue

            4. If the task t has not assign to a resource then assign it to the Rk ϵ
            Si for which ($ET_{Rk}$ + ( DeadlineTime(t) – currentclocktime))
            was minimum
             5. i++;
            }

        5. Increment j;

        }

       3. Update the workload on resources and short the tasks using
       WaitTime(ti) as key under site S.

       4. Update the set OS, US and value of $LD_G$.

       5. Short the elements of OS in descending order of bandwidth and
       elements of UR in descending order of bandwidth.

       }

       4. Grid is in balance state call return
```

Chapter 4
Simulation Tool

4.1 GridSim: Grid Modeling and Simulation Toolkit

The GridSim[32] toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users. GridSim is better for simulating the grid based algorithms because:

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled in two modes: space shared and time shared.
- Resource capability can be defined in the form of MIPS (Million Instructions per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark.
- Advance reservation of resources can be done.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.

- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

4.2 GridSim Entities

GridSim supports entities for simulation of single processor and multiprocessor, heterogeneous resources that can be configured as time or space shared systems. It allows setting their clock to different time zones to simulate geographic distribution of resources. It supports entities that simulate networks used for communication among resources. During simulation, GridSim creates a number of multi-threaded entities, each of which runs in parallel in its own thread. An entity's behavior needs to be simulated within its body () method, as dictated by SimJava. GridSim based simulations contain entities for the users, brokers, resources, information service, statistics, and network based I/O.

(1) User: Each instance of the User entity represents a Grid user. Each user may differ from the rest of the users with respect to the following characteristics:

- Types of job created e.g., job execution time, number of parametric replications, etc.
- Scheduling optimization strategy e.g., minimization of cost, tim
- Activity rate e.g., how often it creates new job,
- Time zone, and
- Absolute deadline and budget, or

- D-and B-factors, deadline and budget relaxation parameters, measured in the range [0, 1] express deadline and budget affordability of the user relative to the application processing requirements and available resources

(2) Broker: Each user is connected to an instance of the Broker entity. Every job of a user is first submitted to its broker and the broker then schedules the parametric tasks according to the user's scheduling policy. Before scheduling the tasks, the broker dynamically gets a list of available resources from the global directory entity. Every broker tries to optimize the policy of its user and therefore, brokers are expected to face extreme competition while gaining access to resources. The scheduling algorithms used by the brokers must be highly adaptable to the market's supply and demand situation

(3) Resource: Each instance of the Resource entity represents a Grid resource. Each resource may differ from the rest of resources with respect to the following characteristics:

- Number of processors;
- Cost of processing;
- Speed of processing;
- Internal process scheduling policy e.g., time shared or space shared;
- Local load factor; and
- Time zone

The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC. They can also be defined with respect to the standard machine. Upon obtaining the resource contact details from the Grid information service, brokers can query resources directly for their static and dynamic properties.

iv) Grid Information Service: It provides resource registration services and maintains a list of resources available in the Grid. This service can be used by brokers to discover resource contact, configuration, and status information.

v) Input and Output: - The flow of information among the GridSim entities happens via their Input and Output entities. Every networked GridSim entity has I/O channels, which are used for establishing a link between the entity and its own Input and Output entities. Note that the GridSim entity and its Input and Output entities are threaded entities i.e., they have their own execution thread with body () method that handle the events. The use of separate entities for input and output enables a networked entity to model full duplex and multi-user parallel communications. The support for buffered input and output channels associated with every GridSim entity provides a simple mechanism for an entity to communicate with other entities and at the same time enables the modeling of a communication delay transparently.

4.3 Application Model

GridSim does not explicitly define any specific application model. It is up to the developers (of schedulers and resource brokers) to define them. In GridSim, each independent task may require varying processing time and input files size. Such tasks can be created and their requirements are defined through Gridlet objects. A Gridlet is a package that contains all the information related to the job and its execution management details such as the job length expressed in MI (million instructions), disk I/O operations, the size of input and output files, and the job originator. These basic parameters help in determining execution time, the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the Originator along with the results. The GridSim toolkit supports a wide range of Gridlet management protocols and services that allow schedulers to map a Gridlet to a resource and manage it throughout the life cycle.

4.4 Resource Model

In the GridSim toolkit, we can create Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Then, one or more PEs can be put together to create a machine. Similarly, one or more machines can be put together to create a Grid resource. Thus, the resulting Grid resource can be a single processor, shared memory multiprocessors (SMP), or a distributed memory cluster of computers. These Grid resources can simulate time- or space-shared scheduling depending on the allocation policy. The tasks execution queue on more than one PEs or SMPs under a Grid resource is typically managed by using time-shared policy that use round-robin scheduling policy for multitasking. And the tasks in wait queue on a grid Grid resource is managed by using space-shared policy such as first-come-first-served (FCFS), back filling, shortest-job-first served (SJFS), and so on.

Chapter 5
Simulation results and discussion

5.1 Results and Discussion

In order to evaluate the efficiency of proposed decentralized load balancing algorithm, I have implemented it on the GridSim simulator [32], which I extended to support the simulation of varying Grid load balancing problems. The experiments were performed, based on the variation of several performance parameters in a Grid, mainly the large number recourses with varying processing capacity and large number of tasks of varying length. The experiments focus on the work load of different resources, utilization of resources, average response time of the tasks, and total finish time of the tasks submitted during a given period. To evaluate the benefit of this algorithm, different graph has been computed from the output of this load balancing algorithm. All experiments have been performed on 2.4 GHz Intel Dual Core with 4 GB main memory, running on windows7. The grid topology I am taking for experiments is heterogeneous which has five sites connect with WAN (wide area network) and each site having twenty five resources connected with LAN (ethernet or wireless). The communication bandwidth of different sites and resources is different and the resources are heterogeneous in terms of their processing power. In order to obtain reliable results, same experimentations have been performed several times. After many evaluation tests the threshold value ε has been set to ε=0.045 and migration limit of a task has been set to 4.

Experiment 1: The first set of experiments has focused on the relative load of different resources under the same site. For this experiment I have taken 20 users having variable number of tasks, which is between 60 to 100. The task length varies between 10000 to 150000 MIPS. And the resources are of different power having number of machines varying between 1 to 4, each machine with number of PE varying between 1 to 4 and each PE having power varying between 900 to 1500 MIPS. Figure 5.1 shows the load graph of different resources under the same site. From this graph we can see that the load on all the three resources is increasing and decreasing almost in the same manner. It shows the distribution of load on resources by using load balancing algorithm is almost equal.

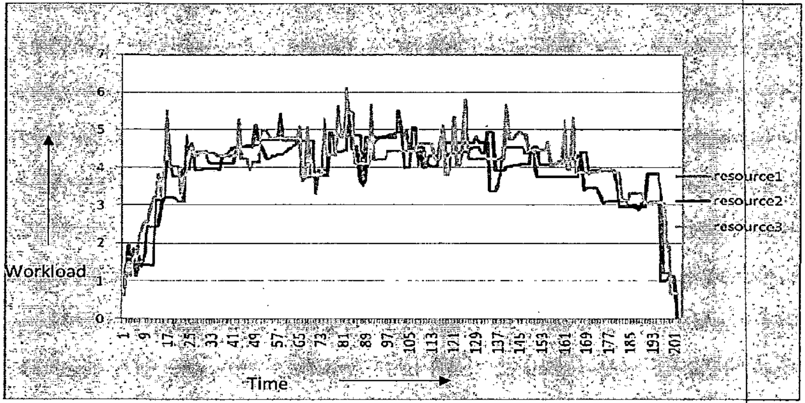


Figure.5.1 Comparison of Uneven Load on Different Resources

Experiment 2: The second set of experiments has focused on the total execution time of the system. For this experiment the number of jobs from 10000 to 100000 is submitted to the grid. The task length varies between 10000 to 150000 MIPS. The processing power of resources is same as in experiment.1.

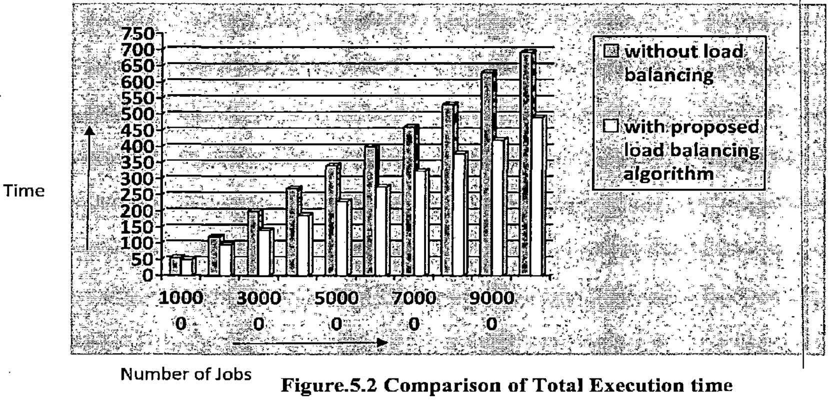


Figure.5.2 Comparison of Total Execution time

Figure 5.2 shows the total execution time (TET) of the system for different number of jobs. TET of the system without load balancing is very high and with load balancing it is low. Because in case of no load balancing many resources get overloaded while other may be idle .This increases the waiting time of resources, hence increases the execution time of a task. So this delay in task execution increases the total execution time of the system. While in case of using load balancing algorithm, it tries to utilize the resources fully transferring the task from under loaded resources to overloaded resources.

Experiment.3: This experiment focused on the average response time of the tasks submitted on the grid. The data for tasks and resources are same as in example 2. Figure 5.3 shows the gain in average response time using load balancing algorithm.

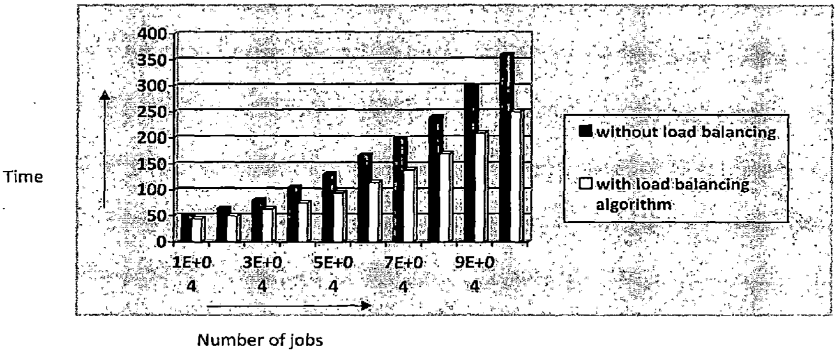
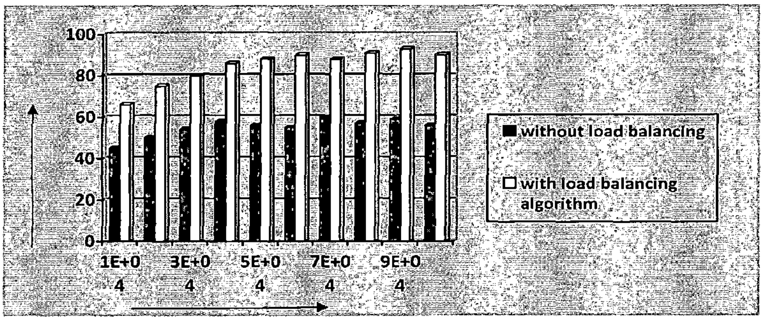


Figure.5.3 Average Response Time

Experiment.4: This experiment focused on the average resource utilization in case of uneven load distribution on the resources. The data for tasks and resources are same as in example 2. Figure 5.4 shows the comparison between resources utilization, using proposed decentralized load balancing algorithm and without load balancing.

ization



Number of jobs

Figure.5.4 Resource Utilization Comparison For Uneven Load Distribution

5.2 Performance Evaluation

Effect of the Status Exchange Period: In the proposed load balancing algorithm information exchange takes place, only when it predicts that there is a need to balance the load. This makes the information exchange period to be adaptive, so less number of status messages exchanges results in less communication overheads.

Effect of an Uneven Load Distribution: One of the major advantages of the proposed algorithm is that it attempts to balance the load on each processor "as soon as possible." Whenever a task arrives at a resource, that resource will determine whether there is a need to balance the load .If it finds out that there is a need then load balancing take place and task is migrated from overloaded resources to underloaded resources, and hence the is balanced as soon as possible.

Effect of Migration Limit: One of the important parameters for proposed load balancing algorithm is the migration limit. that is, how many hops we should allow a task to migrate before execution. Obviously, this decision depends on the network topology considered. By setting the migration limit value to the maximum path length of the graph, we can obtain almost the same result as when the migration limit is very large. Therefore by restricting the value of migration limit to a finite value, we can reduce the task migration cost by reducing the total number of task migrations. I conducted a set of experiments by varying the migration limit and I observed that, by setting the value of migration limit around the maximum path length (but less than maximum path length) of the topology gives a better performance.

Effect of task Size: For the load balancing algorithm, the task migration cost is also one of the most important factors. An increase in task size would lead to an increase in the task migration cost factors for load balancing. For this purpose the proposed algorithm uses the migration limit to decide how many times a task can be transfer from one resource to other for load balancing.

Comparison with other load balancing algorithms

Metrics	MELISA	Dynamic Job Dispersion Algorithm	Decentralized Recent Neighbor Load Balancing algorithm	Proposed decentralized load balancing algorithm
Communication overhead	More; Due to periodic status exchange process	More; As each node obtains status information of its neighbors periodically	More; As each node obtains status information of its neighbors periodically	Less; ass each node obtains status infor-mation only when it feel that it will used in recently in future.
Load balancing time	More; Sensitive to the status exchange interval	Less; As each node has status information of all other nodes. This status information is collected periodically	Less; As each node has status information of all other nodes. This status information is collected periodically	Less; As each node has status information of all other nodes.
Scalability	Scalable; The status information is exchanged every time a node is added or deleted	Scalable; System allows a node to join or leave at any time.	Partially scalable; Work best for small size grid systems	Fully scalable; Node can join and leave at any time
Fault tolerance	Partially Incorporated; Faulty node will not provide status information during the status exchange interval	Partially Incorporated; If a multicast message is not received from the node in stipulated time the node is assumed to be faulty	Partially incorporated;	Incorporated; Both in case of job migration and in collecting status information
Reliability	Incorporated; Job will be scheduled during every status exchange interval.	Incorporated ; Since each node has status information of all the other nodes, the job will definitely be scheduled on one of the nodes	Incorporated; Since each node has status information of all the other nodes, the job will definitely be scheduled on one	Incorporated; since each node has status information of other nodes job will be definitely schedule.

Stability	Incorporated; Overall system is stable	Incorporated; The algorithm is tested for varying job arrival rates	Not Incorporated;	Incorporated; Algorithm works for varying task arrival
task migration cost	Partially Incorporated; jobs need to wait till the next transfer instant for migration	Partially Incorporated; decrease the performance when nodes failure increases because tasks has to move	Partially Incorporated; Create bottleneck on hierarchical level servers	Incorporated; Use predictive approach and allow those task to execute on a node which fit best with that prediction

Table.5.1 Comparison With Other Load Balancing Algorithms

Chapter 6
Conclusions and Scope for Future Work

6.1 Conclusions

In this dissertation, an efficient decentralized load balancing algorithm has been proposed to balance load across resources for data-intensive computations on Grid environments. The objective of the algorithm is to minimize average response time and the total execution time for jobs that arrive at a Grid system for processing with minimum communication overhead and task migration cost. The algorithm minimizes the communication overhead and task migration cost by privileging local load balancing to avoid the WAN communication. Several constraints such as communication delays due to the underlying network, processing delays at the processors, and an arbitrary topology for a Grid system are explicitly considered in the problem formulation. The proposed algorithm is adaptive in the sense that it uses the information about current status of resources to balance the load among them. The experimental results shows that the proposed algorithm minimizes the average response time and total execution time of the tasks submitted to the grid for execution and the gain in time increase with number of jobs submitted to the grid. The proposed algorithm has been compared with other algorithms and it seems to be more adaptive to grid environment than others.

6.2 Scope for Future Work

In the future, this work can be extended in following ways:

(i) This work can be implemented in actual middle ware like Globus.

(ii) And the impact of communication delay on the model under varying load conditions need to be studied.

REFERENCES

[1] C. Xu and F. Lau, "Load Balancing in Parallel Computers," *Theory and Practice, Kluwer academic publicers, Boston*, PP(225-260), May, 1997.

[2] G. Manimaran and C. Siva, "An EfficientDynamic Scheduling Algorithm for Multiprocessor RealTime Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 312-319, Mar. 1998.

[3] N. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, vol. 25, no. 12, pp. 33-44, Dec. 1992.

[4] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 235-248, Mar. 1998.

[5] M.J. Zaki and W.L.S. Parthasarathy, "Customized Dynamic Load Balancing for a Network of Workstations," *J. Parallel and Distributed Computing*, vol. 43, no. 2, pp. 156-162, June 1997.

[6] Y. Feng, D. Li, H. Wu, and Y. Zhang, "A Dynamic Load Balancing Algorithm Based on Distributed Database System," *Proc. Fourth Internatiol Conf. High-Performance Computing in the Asia-Pacific Region*, pp. 949-952, May 2000.

[7] H. Lin and C. Raghavendra, "A Dynamic LoadBalancing Policy with a Central Job Dispatcher (LBC)," *IEEE Trans. Software Eng.*, vol. 18, no. 2, pp. 148-158, Feb. 1992.

[8] M. Baker, R.Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing," *International Journal of Software: Practice and Experience (SPE)*, vol. 32, no. 15, 2002.

[9] K.G. Shin and C.J. Hou, "Analytic models of adaptive load sharing schemes in distributed real-time systems, " *IEEE Transactions on Parallel and Distributed Systems,* Vol. 4, No. 9, pp. 740-761, July 1993.

[10] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall, "A Note on Distributed Computing", *Sun Microsystems Laboratories 2550 Garcia Avenue Mountain View*, CA 94043.

[11] Ian Foster, Carl Kesselman, and Steve Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High-performance Computing Applications*, vol.15, no.3, pp. 200-222, june 2001.

[12] Thilo Kielmann, Vrije Universiteit and Amsterdam, "Scalability in Grid". PPT Core GRID, *Bridging Global Computing with Grid (BIGG)*, Sophia Antipolis, France, Nov. 29,2006.

[13] L. Bote and Yannis S´anchez, "Grid Characteristics and Uses: a Grid Definition" Postproc. the First European Across Grids Conference (ACG'03), *Springer Verlag LNCS 2970*, pp. 291-298, Santiago de Compostela, Spain, Feb. 2004.

[14] Y.Lan, T.Yu () "A Dynamic Central Scheduler Load-Balancing Mechanism", *Proc. 14th IEEE Conf. on Computers and Communication*, pp.734- 40, Tokyo, Japan, July 1995.

[15] S.Rips "Load Balancing Support for Grid-enabled Applications" *NIC Series*, Vol. 33, ISBN 3-00- 017352-8, pp. 97-104, 2006.

[16] R. Tong and X. Zhu A," Load Balancing Strategy Based on the Combination of Static and Dynamic," *IEEE 2nd international workshop on database technology and applications*,pp1-4 ,Nov-2010.

[17] K. Subrata, R. Zomaya, "An Efficient Load Balancing Algorithm for Heterogeneous Grid Systems Considering Desirability of Grid Sites," *25 th IEEE international conference on performance, computing and communications* ,pp.320, 2006.

[18] Suri, P.K., and manpreet,"An Efficient Decentralized Load balancing Algorithm for Grid ,"*IEEE 2nd International Conference on Advance Computing*, 2010 .

[19] B. yagoubi and M. medebber,"A Load Balancing Algorithm For Computational Grid",*IEEE 22nd International Conference On Computer and Information Technology*, 2007.

[20] M. Arora, S.K. Das, R. Biswas, "A de-centralized scheduling and load balancing algorithm for heterogeneous Grid environments", *In: Proceedings of the International Conference on Parallel Processing Workshops*, pp. 499–505, August 2002.

[21] D. Z. Gu, L. Yang and L. R. Welch, A Predictive," Decentralized Load Balancing Approach," *In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Denver, Colorado, 04-08 April 2005.

[22]. H.Shan, L.Oliker, and R.Biswas, "Job superscheduler architecture and performance in computational grid environments", *Proceedings of the ACM/IEEE conference on Supercomputing*, 15-21 November 2003.

[23] A. Abed, G. Oz, A. Kostin," Competition-Based Load Balancing for Distributed Systems", *Proceedings of the Seventh IEEE International Symposium on Computer Networks* (ISCN' 06),pp 230 – 235,2006.

[24] R. Shah, B. Veeravalli, M. Misra," On the Design of Adaptive and Decentralized Load-Balancing Algorithms with Load Estimation forComputational Grid Environments," *IEEE Transactions on Paralleland Distributed systems*, Vol. 18, pp 1675 – 1687, Dec.2007.

[25] D. Acker, S. Kulkarni, "A Dynamic Load Dispersion Algorithm for Load-Balancing in a Heterogeneous Grid System", *Sarnoff Symposium IEEE*, pp 1- 5, May 2007.

[26] Bin lu and H. Zhang, "Grid Load Balancing Scheduling Algorithm Based on Statistics Thinking,"*IEEE 9th international conference*, pp.288-292, 2008.

[27] Prof. E. Saravanakumar and Gomathy Prathima. E," A Novel Load Balancing Algorithm for Computational Grid",*IEEE*, 2009.

[28] Nashir ,H.J." Load Balancing Using Enhanced Ant Algorithm in Grid Computing,"*IEEE Second International Conference*" pp.160-165, 2010.

[29] J. balasangameshwara, N. raju, "A Decentralized Recent Neighbour Load Balancing Algorithm for Computational Grid," *IEEE international conference*, pp.428-433, 2010.

[30] M. Moradi and M. Abbasi ,"A New Time Optimizing Probabilistic Load Balancing Algorithm in Grid Computing", *IEEE*, 2010.

[31] Jingyi," A Novel Heuristic Genetic Load Balancing Algorithm in Grid Computing,"*IEEE 2nd international conference*, vol.2, pp.166-169, 2010.

[32] GridSim toolkit available at : http://www.gridbus.org/gridsim/