Efficient Load Balancing Algorithm in Grid Environment

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

By:
Ratnesh Kumar Nath
(Roll No 8053118)

Under the supervision of

Ms. Inderveer Chana
Senior Lecturer, CSED
Thapar University, Patiala.

Dr. (Mrs.) Seema Bawa
Professor & Head, CSED
Thapar University, Patiala.

MAY 2007

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Efficient Load Balancing Algorithm in Grid Environment"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Ms. Inderveer Chana and Dr. (Mrs.) Seema Bawa.

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

(Ratnesh Kumar Nath*)*

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

(Dr. Seema Bawa)
Supervisor
CSED
Thapar University
Patiala

(Ms. Inderveer Chana)
Supervisor
CSED
Thapar University
Patiala

Countersigned by

(Dr. (Mrs.) Seema Bawa)
Professor & Head
Computer Science & Engineering Department
Thapar University
Patiala

(Dr. R.K. Sharma)
Dean, Academic Affairs
Thapar University
Patiala

Acknowledgement

I wish to express my deep gratitude to Ms. Inderveer Chana, Senior Lecturer and Dr. (Mrs.) Seema Bawa Professor & Head, Computer Science & Engineering Department for providing their uncanny guidance and support throughout the preparation of the thesis report.

I am also heartily thankful to Mr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department and Mrs. Shivani Goel, P.G. Coordinator, Computer Science and Engineering Department for the motivation and inspiration that triggered me for my thesis work.

I would also like to thank all the staff members, T.I.E.T. Grid Group and all my friends especially Anurag, Santosh, Puneet, Ms. Anju Sharma and Ms. Shashi who were always there at the need of the hour and provided all the help and support, which I required for the completion of the thesis.

Last but not the least, I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Ratnesh Kumar Nath

(8053118)

Abstract

Grid technology has emerged as a new way of large-scale distributed computing with high-performance orientation. Grid computing is being adopted in various areas from academic, industry research to government use. Grids are becoming platforms for high performance and distributed computing. Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. 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 Resource Management is defined as the process of identifying requirements, matching resources to applications, allocating those resources, and 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. The resources which will be heavily loaded will act as server of task and the resources which are Lightly Loaded will act as receiver of task. Task will be migrated from heavily loaded node to lightly loaded node.

Resources are dynamic in nature so the load of resources varies with change in configuration of Grid so the Load Balancing of the tasks in a Grid environment can significantly influence Grid's performance. A poor scheduling policy may leave many processors idle while a clever one may consume an unduly large portion of the total CPU cycles. The main goal of load balancing is to provide a distributed, low cost, scheme that balances the load across all the processors. To improve the global throughput of Grid resources, effective and efficient load balancing algorithms are fundamentally important. Various strategies, algorithms and policies have been proposed, implemented and classified for implementing Load balancing in Grid environment.

Focus of this thesis is on analyzing Load Balancing requirements in a Grid environment and proposing a centralized and sender initiated load balancing algorithm. A load balancing algorithm has been implemented and tested in a simulated Grid environment.

Table of Contents

LIST OF FIGURES

Chapter 1 Introduction

The rapid development in computing resources has enhanced the performance of computers and reduced their costs. This availability of low cost powerful computers coupled with the popularity of the Internet and high-speed networks has led the computing environment to be mapped from distributed to Grid environments [1]. In fact, recent researches on computing architectures are allowed the emergence of a new computing paradigm known as Grid computing. Grid is a type of distributed system which supports the sharing and coordinated use of geographically distributed and multi-owner resources, independently from their physical type and location, in dynamic virtual organizations that share the same goal of solving large-scale applications.

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. 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 is to prevent, if possible, the condition where some processors are overloaded with a set of tasks while others are lightly loaded or even idle [2]. Although load balancing problem in conventional distributed systems has been intensively studied, new challenges in Grid computing still make it an interesting topic and many research projects are under way. This is due to the characteristics of Grid computing and the complex nature of the problem itself . Load balancing algorithms in classical distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in the Grid architectures.

In this chapter we define the motivation of this research and then identify the research questions. This chapter also discusses overall organization of thesis.

1.1 Motivation

A typical distributed system will have a number of interconnected resources who can work independently or in cooperation with each other [4]. Each resource has owner workload, which represents an amount of work to be performed and every one may have a different processing capability. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed. The essential objective of a load balancing consists primarily in optimizing the average response time of applications, which often means maintaining the workload proportionally equivalent on the whole resources of a system.

This work focuses on Load Balancing in a Grid environment. Grid application performance is critical in grid computing environment so to achieve high performance we need to understand the factors that can affect the performance of an application and Load Balancing is one of most important factor which effect the overall performance of application.

Although load balancing methods in conventional parallel and distributed systems has been intensively studied, they do not work in Grid architectures because these two classes of environments are radically distinct [2]. Indeed, the schedule of tasks on multiprocessors or multi computers supposes that processors are homogeneous and linked with homogeneous and fast networks. The motivation behind this approach is that interconnection bandwidth between processing elements is high and generally resources have same capabilities.

Given the distribution of tremendous resources in a Grid environment and the size of the data to be moved, it becomes clear that this approach is not accurate because following properties.

- **Heterogeneity:** Heterogeneity exists in both of computational and networks resources.
- **Autonomy**: Because the multiple administrative domains that share Grid resources, a site are viewed as an autonomous computational entity.

- **Scalability and adaptability**: A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as the size of a Grid increases.
- **Resource selection and computation-data separation**: In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the submission of an application [2, 4]. Thus the cost for data staging can be neglected or the cost is a constant determined before execution and load balancing algorithms need not consider it. But in a Grid the computation sites of an application are usually selected by the Grid scheduler according to resource status and some performance criterion.

In a Grid the factors mentioned above play an important role in formulating Load Balancing in comparison to distributed systems.

1.2 Organization of the Dissertation

The remainder of the thesis is organized as follows:

Chapter 2 discusses history and background of Grid computing.

Chapter 3 discusses detail description of Load Balancing in Grid environment.

Chapter 4 elaborates the problem statement.

Chapter 5 presents the experimental approach, Implementation and results in detail.

Chapter 6 discusses conclusion and suggestions for future research directions.

Chapter 2 Introduction to Grid Computing

Grid computing is a model of distributed computing that uses geographically and administratively disparate resources [5]. In Grid computing, individual users can access computers and data, transparently, without having to consider location, operating system, account administration, and other details. In Grid computing, the details are abstracted, and the resources are virtualized.

Grid Computing has emerged as a new and important field and can be visualized as an enhanced form of Distributed Computing [6]. With the advent of new technology, it has been realized that parallelizing sequential applications could yield faster results and sometimes at a lower cost. Possessing multiprocessor systems was not possible for everyone. Thus, organizations started looking for a better and feasible alternative. It was found that though every organization possessed a large number of computers, which meant that they had huge processing power, but it remained underutilized. A new type of computing then came into existence, known as Distributed Computing. In Distributed Computing, the problem to be solved is divided into numerous tasks that are then distributed to various computers for processing. The computers being used are generally connected through a Local Area Network (LAN). Also the problem needs to be divided into modules that can be executed in parallel to each other.

Then the Grid computing comes into existence. Grid computing is the next generation IT infrastructure that promises to transform the way organizations and individuals compute, communicate and collaborate. Sharing in a Grid is not just a simple sharing of files but of hardware, software, data, and other resources [6]. Thus a complex yet secure sharing is at the heart of the Grid. 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.

In this chapter we have reviewed the history, requirement, features, characteristics, application areas and advantages of Grid as a powerful distributed computing platform.

2.1 History of Grid

The term "**Grid**" was coined in the mid 1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. 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. When we require additional electricity we have to just plug into a power Grid to access additional electricity on demand, similarly for computational resources plug into a Computational Grid to access additional computing power on demand using most economical resource [6].

Technology	**Year**
Networked Operating Systems	1979-81
Distributed operating systems	1988-91
Heterogeneous computing	1993-94
Parallel and distributed computing	1995-96
The Grid	1998

Table-2.1 Historical Background of the Grid

The theory behind "Grid Computing." is not to buy more resources but to borrow the power of the computational resources you need from where it's not being used". Many of the basic ideas behind the Grid have been around in one form or other throughout the history of computing. There is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented; it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rate [7]. Because of the huge improvements of the underlying hardware (typically more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor.

2.2 The Grid

Rajkumar Buyya defined the Grid as:

"Grid is type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed resources dynamically at run time depending on their availability, capability, performance, cost, user quality-of –self-service requirement"[7]

In other words we can say

- Grid is a service for sharing computer power and data storage capacity over the Internet and Intranet.
- Grid is beyond simple communication between computers but it aims ultimately to turn the global network of computer into one vast computational resource.
- Grid is to coordinate resources those are not subject to centralized control.
- Grid is to use standard, open, general-purpose protocols and interfaces.
- Grid is to deliver nontrivial Qualities of Service.

A computational Grid environment behaves like a virtual organization consisting of distributed resources. A **Virtual Organization** is a set of individuals and institutions defined by a definite set of sharing rules like what is shared, who is allowed to share, and the conditions under which the sharing takes place. A number of VOs exist like the application service providers, storage service providers, *etc*., but they do not completely satisfy the requirements of the Grid. The needs of the Grid range from client-server to peer-to-peer architecture, from single user to multi-user systems, from sharing files to sharing resources, *etc.* and all these in a dynamic, controlled, and secured manner. Current distributed computing technologies do not fulfill all these needs. As Grid computing focuses on dynamic and cross-organizational sharing, it enhances the existing distributed computing technologies. Many of the existing technologies like the World Wide Web (WWW), Internet and Peer-to-Peer computing can be considered as similar to Grid Computing, but differences do exist. Internet and Peer-to-Peer computing have much in common with Grid technologies.

2.3 The Need for Grid Technologies

Computers have been proven to be very efficient to solve complex scientific problems. They are used to model and simulate problems of a wide range of domains; for instance medicine, engineering, security control and many more. Although their computational capacity has shown greater capabilities than the human brain to solve such problems, computers are still used less than they could be. One of the most important reasons to this lack of use of computational power is that, despite the relatively powerful computing environment one can have, it is not adapted to such complicated computational purposes. The following are given the reasons for why we need grid computing.

2.3.1 Exploit Unused Resources

In most organizations, computing resources are underutilized. Most desktop machines are busy less than 25% of the time (if we consider that a normal employee works 7 hours a day and that 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usages. The easiest use of Grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy, the execution of the task would be delayed. Grid Computing should enable the job in question to be run on an idle machine elsewhere on the network [8].

2.3.2 Increase Computation

To provide users with more computational power, some crucial areas have to be considered [9]. These areas are:

- Hardware Improvement
- Periodic Computational Needs
- Capacity of Idle Machines
- Sharing of Computational Results

2.4 Characteristics of Grid

There are three main issues that characterize computational Grids [10, 11]:

- *Heterogeneity*: A Grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.
    - Resources are heterogeneous
    - Resources are administratively disparate
    - Resources are geographically disparate
    - Users do not have to worry about system details (e.g., location, operating system, accounts).
    - Resources are numerous.
    - Resources have different resource management policies.
    - Resources are owned and managed by different, potentially mutually distrustful organizations and individuals that likely have different security policies and practices.
- *Scalability:* A Grid might grow from few resources to millions. This raises the problem of potential performance degradation as a Grids size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant.
- *Dynamicity or Adaptability*: In a Grid, a resource failure is the rule, not the exception [12]. In fact, with so many resources in a Grid, the probability of some resource failing is naturally high. The resource managers or applications must tailor their behavior dynamically so as to extract the maximum performance from the available resources and services.
- *Parallel CPU execution*: One of most important feature of Grid is its scope for massive parallel CPU capacity. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive Grid application can be thought of as many smaller "subjobs," each executing on a different machine in the Grid.[12] To the extent that these subjobs do not need to communicate with each other, the

more "scalable" the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors.

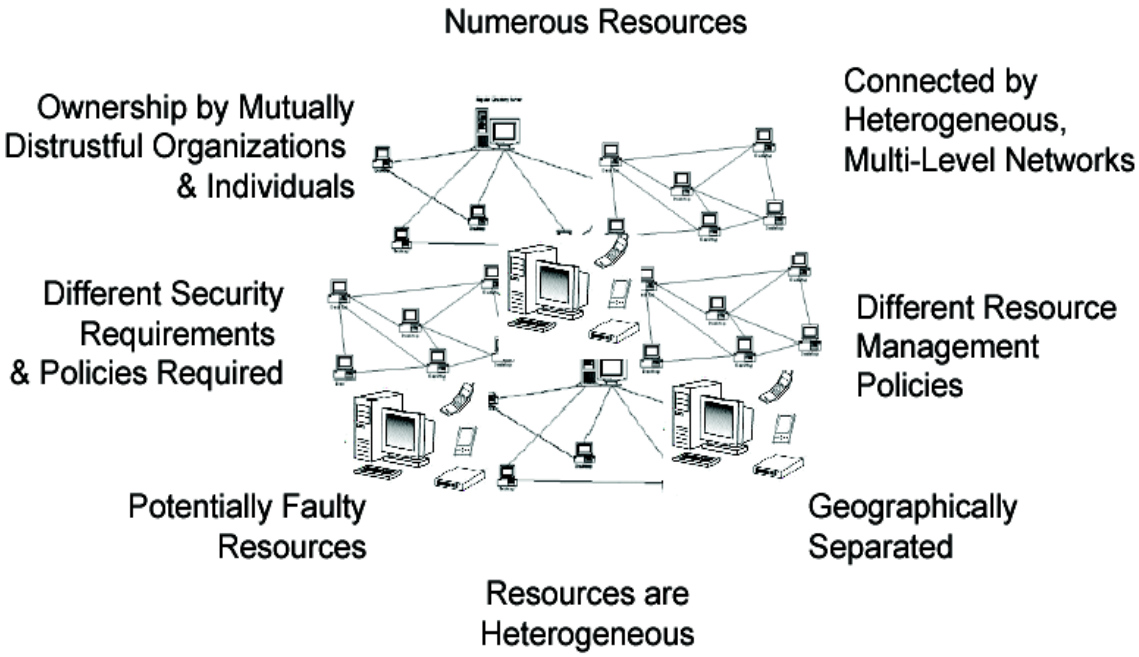


Figure 2.1: Characteristics of Grids [12]

- *Virtual organizations*: The users of the Grid can be organized dynamically into a number of virtual organizations, each with different policy requirements [13, 14]. These virtual organizations can share their resources collectively as a larger Grid.
- *Resource balancing*: A Grid contains 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 [15].
- *Reliability and Management*: High-end conventional computing systems use expensive hardware to increase reliability. A Grid is an alternate approach to reliability that relies more on software technology than expensive hardware [16]. The goal to virtualize the resources on the Grid and more uniformly handle

heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure.

2.5 Types of Grid

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, the type of solutions categorizes Grids that they best address. Of course, there are no hard boundaries between these Grid types and often Grids may be a combination of two or more of these [17 -19].

Grids can be classified on the basis of two factors:

- Scale
- Functionality

On basis of scale they can be further classified as:

- Global Grid
- Enterprise Grid
- Cluster Grid

Cluster Grid

Cluster Grid is simplest form of Grid and provides a compute service to the group or department level.

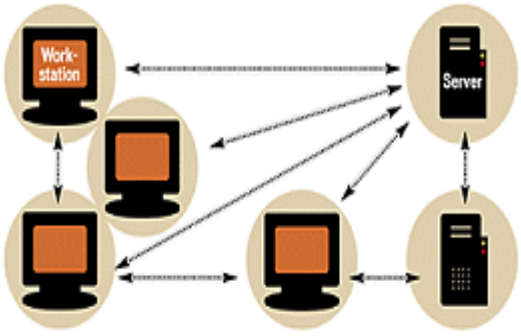


Figure 2.2: Cluster Grids [23]

Enterprise Grids

Enterprise Grids enable multiple project or department to share recourses with in enterprise or campus and not necessary have to address security and other global policy management issues associated with global Grid.

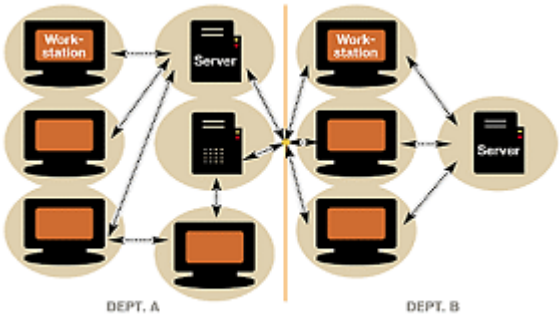


Figure 2.3: Enterprise Grids [23]

Global Grids

Global Grids are collection of enterprise and cluster Grid as well as other geographically distributed resources, all of which are agreed upon global usage policies and protocols to enable resources sharing [20].

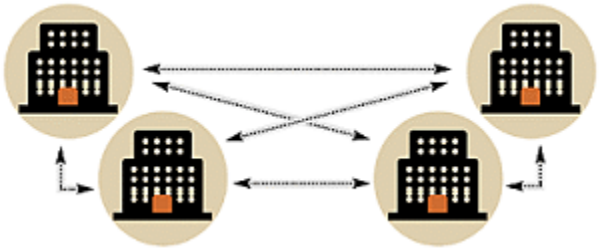


Figure 2.4: Global Grids [23]

On basis of Functionality

- Compute Grids
- Data Grids

Compute Grids

A compute Grid is essentially a collection of distributed computing resources, within or across locations that are aggregated to act as a unified processing resource or virtual supercomputer.

Data Grids

A data Grid provides wide area, secure access to current data. Data Grids enable users and applications to manage and efficiently use database information from distributed locations.

2.6 Grid Topologies

There are three topologies of grid namely, Intragrid, Intergrid and Extragrid [21]. Intragrid is comprised merely of a basic set of grid services within a single organization. Based on a single organization, the Extragrid expands on the concept by bringing together two or more Intragrids. Whereas an Intergrid [22] requires the dynamic integration of applications, resources, and services with patterns, customers, and any other authorized organizations that will obtain access to the grid via the internet/WAN.

2.7 Grid Application Areas

Applications with heavy use of computing resources (e.g., simulations, number crunching, the so-called grand challenge applications), applications using large information resources (e.g., multimedia databases), applications using special sub-applications (e.g., visualization), and applications using special devices (e.g., expensive scanners, laboratory equipment) are candidates for Grids. Especially we mention the following important application areas [23]:

- *Medical Applications:* In diagnostics huge amounts of data are generated at one place by specialized devices. These data have to be transported to the specialists, possibly located at several locations, while the patient might be at a third location. The task of a Grid in this scenario is to prepare and transport the medical data, so that they are available at the right location at the right time [24].

- *Support for multinational enterprises*: Multinational enterprises work at several locations in several time zones. Data, e.g., multimedia data from inspections, must be pre-processed and forwarded to specialists who can take decisions.
- *Multimedia Applications*: Several Multimedia Applications make use of a Grid for processing media streams Within multimedia QoS control is very important. Applications often include the handling of Digital Rights Management. E.g., multimedia data can be watermarked scrambled etc.
- *Applications from bio-informatics*, *seismology, meteorology, etc.* are data – and computing-intensive, and need often other information resources [24].

2.8 Grid Advantages

Some of the advantages of Grid Computing are listed below [25-31]:

- Seamless and secure access to large number of geographically distributed resources.
- Reduction in average job response time may occur but an overhead of limited network bandwidth and latency exists.
- Provides users around the world with dynamic and adaptive access to unparalleled levels of computing.
- With the infrastructure provided by the Grid, scientists are able to perform complex tasks, integrate their work and collaborate remotely.
- Grids can lead to savings in processing time.
- Efficient, effective, and economic utilization of available resources.
- Increased availability and reliability of resources.
- Shared access (by multiple users) to large amounts of data.
- Improved methods for collaborative work.
- Unprecedented Price-to-Performance ratio.

2.9 Grid Architecture

Architecture identifies the fundamental system components, specifies purpose and function of these components, and indicates how these components interact with each

other. Grid architecture is protocol architecture, with protocols defining the basic mechanisms by which VO [32, 33, and 34] users and resources negotiate, establish, manage and exploit sharing relationships. Grid architecture is also a services standards-based open architecture that facilitates extensibility, interoperability, portability and code sharing. The components that are necessary to form a Grid are shown in Figure 2.5 and they are briefly discussed below:

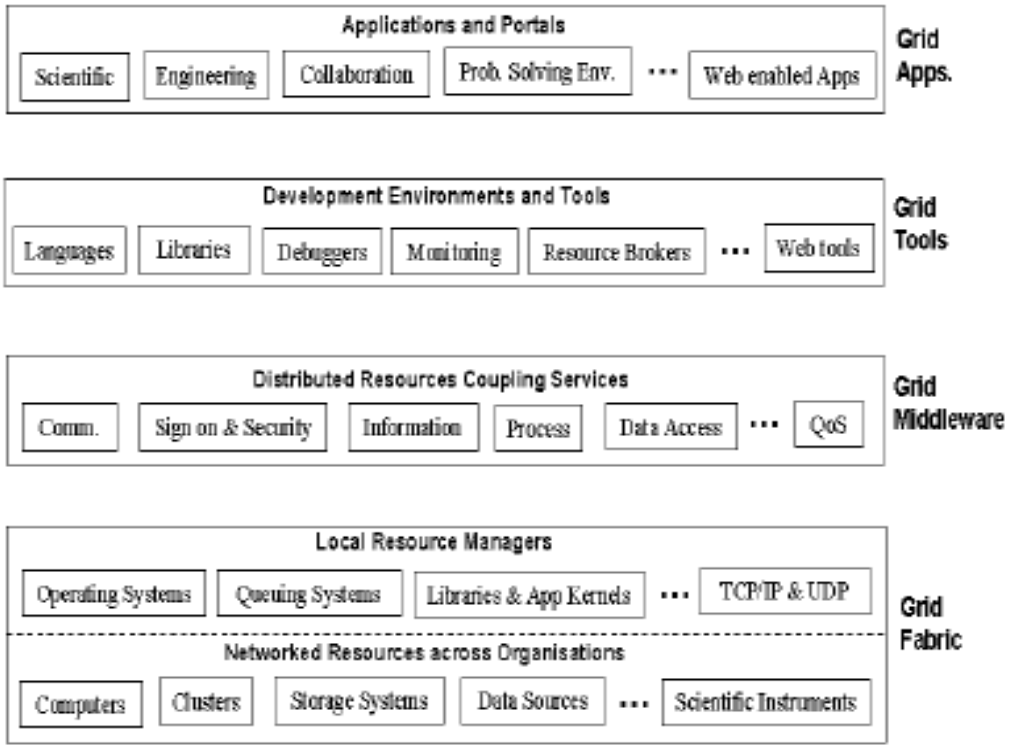


Figure 2.5: Grid Components [35]

- *Grid Fabric*: It comprises all the resources geographically distributed (across the globe) and accessible from anywhere on the Internet. They could be computers (such as PCs or Workstations running operating systems such as UNIX or NT), clusters (running cluster operating systems or resource management systems such as LSF, Condor or PBS), storage devices, databases, and special scientific instruments such as a radio telescope.
- *Grid Middleware*: It offers core services such as remote process management, co-allocation of resources, storage access, information (registry), security,

authentication, and Quality of Service (QoS) such as resource reservation and trading.

- *Grid Development Environments and Tools*: These offer high-level services that allow programmers to develop applications and brokers that act as user agents that can manage or schedule computations across global resources.
- *Grid Applications and Portals*: They are developed using Grid-enabled languages such as HPC++, and message-passing systems such as MPI. Applications, such as parameter simulations and grand-challenge problems often require considerable computational power, require access to remote data sets, and may need to interact with scientific instruments. Grid portals offer web-enabled application services — i.e., users can submit and collect results for their jobs on remote resources through a web interface.

2.10 Open Standards Platform

In order to achieve true distributed resource sharing across heterogeneous and dynamic VOs, grid-computing technologies require several improvements in alignment with other computing technologies. In the early days of grid computing, a number of custom middleware solutions were created to solve the grid problem, but this resulted in non-interoperable solutions and problematic integration among the participants. The new wave of grid computing focuses on the easier integration, security, and QoS aspects of resource sharing [36].

Foster et al described the Open Grid Services Architecture (OGSA) as a solution to the above problem. This architectural concept is a result of the alignment of existing grid standards with emerging SOAs, as well as with the Web. OGSA provides a uniform way to describe grid services and define a common pattern of behavior for these services. It defines grid-service behavior, service description mechanisms, and protocol binding information by using Web services as the technology enabler. This architecture uses the best features from both the grid-computing community and the Web-services community. "Open Grid Service Architecture" (OGSA) is the industry blueprint for standards-based grid computing. "Open" refers to both the standards-development process and the

standards themselves. OGSA [37] is “service-oriented” because it delivers functionality among loosely coupled interacting services that are aligned with industry-accepted Web service standards. “Architecture” defines the components, their organizations and interactions, and the overall design philosophy.

OGSA is the responsibility of the Global Grid Forum (GGF), working in conjunction with other leading standards organizations engaged in delivering industry-standard distributed computing. GGF is the community of users, developers, and vendors leading the global standardization effort for grid computing.

2.10.1 OGSA Architecture

OGSA is a layered architecture, as shown in Figure 2.6, with clear separation of the functionalities at each layer. As seen in the Figure, the core architecture layers are the Open Grid Services Infrastructure (OGSI) and OGSA platform services. The platform services establish a set of standard services including policy, logging, service level management, and other networking services. High-level applications and services use these lower-layer platform core components to become a part of a resource-sharing grid.

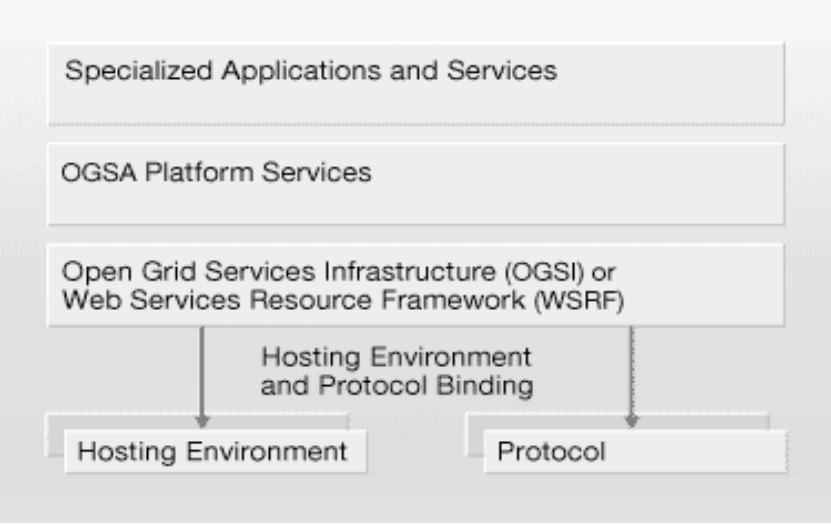


Figure 2.6: OGSA Platform Architecture [37]

OGSA specifies following eight high-level categories of services:

- *Infrastructure Services* enable communication between disparate resources (computer, storage, applications, etc.), removing barriers associated with shared utilization.
- *Resource Management Services* enable the monitoring, reservation, deployment, and configuration of grid resources based on quality of service requirements.
- *Data Services* enable the movement of data where it is needed – managing replicated copies, query execution and updates, and transforming data into new formats if required.
- *Context Services* describe the required resources and usage policies for each customer that utilizes the grid – enabling resource optimization based on service requirements.
- *Information Services* provide efficient production of, and access to, information about the grid and its resources, including status and availability of a particular resource.
- *Self-Management Services* support the attainment of stated levels of service with as much automation as possible, to reduce the costs and complexity of managing the system.
- *Security Services* enforce security policies within a virtual organization, promoting safe resource-sharing and appropriate authentication and authorization of users.

Execution Management Services enable both simple and more complex workflow actions to be executed, including placement, provisioning, and management of the task lifecycle.

This chapter mainly discusses generically about Grid. In the forthcoming chapter we are going to discuss relevance of load balancing in Grid and the intricacies involved.

Chapter 3 Load Balancing in Grid Environment

Grids functionally combine globally distributed computers and information systems for creating a universal source of computing power and information [4]. A key characteristic of Grids is that resources (e.g., CPU cycles and network capacities) are shared among numerous applications, and therefore, the amount of resources available to any given application highly fluctuates over time. In this scenario load balancing plays key role. For applications that are Grid enabled, the Grid can offer a resource balancing effect by scheduling Grid jobs on machines with low utilization. A proper scheduling and efficient load balancing across the Grid can lead to improved overall system performance and a lower turn-around time for individual jobs.

Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to cut response time through an appropriate distribution of the application [38]. To minimize the decision time is one of the objectives for load balancing which has yet not been achieved.

As illustrated in Figure 3.1 load balancing feature can prove invaluable for handling occasional peak loads of activity in parts of a larger organization. These are important issues in Load Balancing:

- An unexpected peak can be routed to relatively idle machines in the Grid.
- If the Grid is already fully utilized, the lowest priority work being performed on the Grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work.

In this chapter we have discussed Load balancing algorithms, strategies and policies in relevance to Grid. A brief comparison has been outlined between Load Balancing in Grid and other Distributed Computing environments. Herein, a generic model of Load balancing has also been discussed.

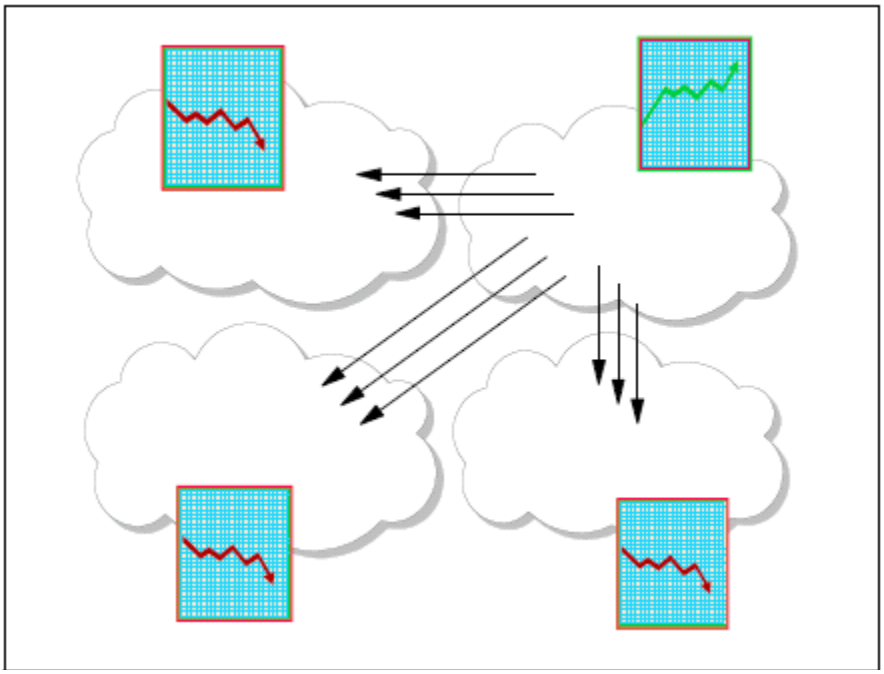


Figure 3.1: Job Migration [39]

3.1 Comparison of Load Balancing in Grid environment and DCE

Often, Grid environments are seen as the successors of distributed computing environments (DCEs). Nevertheless, these two environments are fundamentally different. A DCE environment is rather predictable: the nodes are usually homogeneous, the availability of resources is based on reservation, and the processing speeds are static and known beforehand. A Grid environment, however, is highly unpredictable in many respects: resources have different and usually unknown capacities, they can be added and removed at any time, and the processing speeds fluctuate over time. In this context, it is challenging to realize good performance of parallel applications running in a Grid environment. Nature of Grid makes Load Balancing important in case of Grid environment compare to DCE.

In order to make optimal balancing and work distribution decisions in Grid environment, a load balancer needs to take some or all of the following information into consideration:

- Available capacity on each node (CPU, memory, disk space)
- Current load of each node
- Required capacity for each task
- Network connectivity and capacity
- Communication pattern for each task (if applicable)

Choice of load index to use when measuring the performance of a node has a considerable effect on the performance of the load balancer as a whole.

3.2 Load Balancing Approaches

Load balancing problem has been discussed in traditional distributed systems literature for more than two decades. Various algorithms, strategies and policies have been proposed, implemented and classified [40].

3.2.1 Load Balancing Algorithms

Algorithms can be classified into two categories: static or dynamic.

(a) Static Load Balancing Algorithm

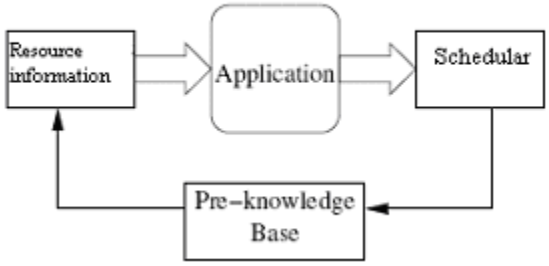


Figure 3.2: Static Load Balancing [40]

Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of our workstation cluster. The decisions related to load balance are made at compile time when resource requirements are estimated. The advantage in this sort of

algorithm is the simplicity in terms of both implementation as well as overhead, since there is no need to constantly monitor the workstations for performance statistics. However, static algorithms only work well when there is not much variation in the load on the workstations. Clearly, static load balancing algorithms aren't well suited to a Grid environment, where loads may vary significantly at various times. A few static load balancing techniques are:

- Round robin algorithm - the tasks are passed to processes in a sequential order; when the last process has received a task the schedule continues with the first process (a new round)
- Randomized algorithm: the allocation of tasks to processes is random
- Simulated annealing or genetic algorithms: mixture allocation procedure including optimization techniques

Even when a good mathematical solution exists, static load balancing still have several flaws:

- it is very difficult to estimate a-priori [in an accurate way] the execution time of various parts of a program
- sometimes there are communication delays that vary in an uncontrollable way
- for some problems the number of steps to reach a solution is not known in advance

(b) Dynamic Load Balancing Algorithm

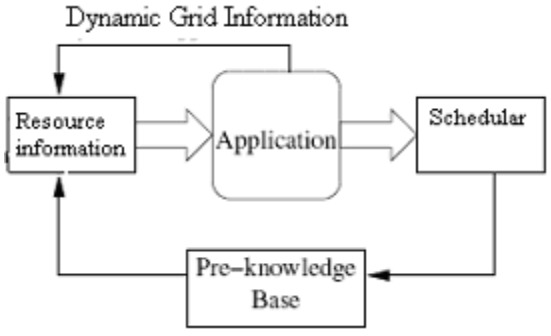


Figure 3.3: Dynamic Load Balancing [40]

Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions. Multicomputers with dynamic load balancing allocate/reallocate resources at runtime based on no a priori task information, which may determine when and whose tasks can be migrated.

As a result, dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms. However, this comes at the additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits

3.2.2 Load Balancing Strategies

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ [9]. These three parameters answer three important questions:

- who makes the load balancing decision
- what information is used to make the load balancing decision, and
- Where the load balancing decision is made.

(a) Sender-Initiated vs. Receiver-Initiated Strategies

The question of who makes the load balancing decision is answered based on whether a sender-initiated or receiver-initiated policy is employed [41]. In sender-initiated policies, congested nodes attempt to move work to lightly-loaded nodes. In receiver-initiated policies, lightly-loaded nodes look for heavily-loaded nodes from which work may be received.

Figure 3.4 shows the relative performance of a sender-initiated and receiver-initiated load balancing algorithm. As can be seen, both the sender-initiated and receiver-initiated policies perform substantially better than a system which has no load sharing.

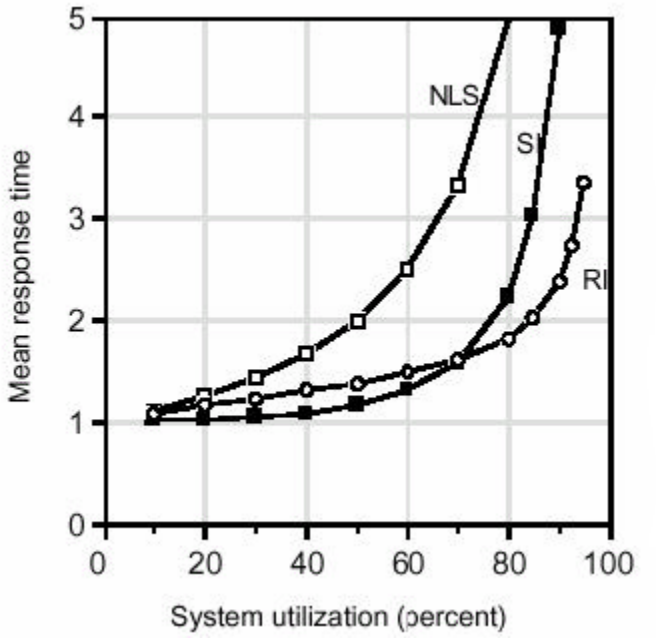


Figure 3.4: Performance of Sender-initiated Vs Receiver-initiated Strategies

NLS – No Load Sharing, SI – Sender Initiated, RI – Receiver Initiated.

The sender-initiated policy performing better than the receiver-initiated policy at low to moderate system loads. Reasons are that at these loads, 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 behave like sender-initiated policies at low to moderate system loads, while at high system loads they behave like receiver-initiated policies.

(b) Global vs. Local Strategies

Global or local policies answer the question of what information will be used to make a load balancing decision in global policies, the load balancer uses the performance profiles of all available workstations. In local policies workstations are partitioned into different groups. The benefit in a local scheme is that performance profile information is only exchanged within the group. The choice of a global or local policy depends on the behavior an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time.

However, this requires additional communication and synchronization between the various workstations; the local schemes minimize this extra overhead. But the reduced synchronization between workstations is also a downfall of the local schemes if the various groups exhibit major differences in performance. If one group has processors with poor performance (high load), and another group has very fast processors (little or no load), the latter will finish quite early while the former group is overloaded.

(c) Centralized vs. De-centralized Strategies

A load balancer is categorized as either centralized or distributed, both of which define where load balancing decisions are made [44-46]. In a centralized scheme, the load balancer is located on one master workstation node and all decisions are made there.
Basic features of centralized approach are:

- a master node holds the collection of tasks to be performed,
- tasks are sent to the execution node
- when a execution process completes one task, it requests another task from the master node

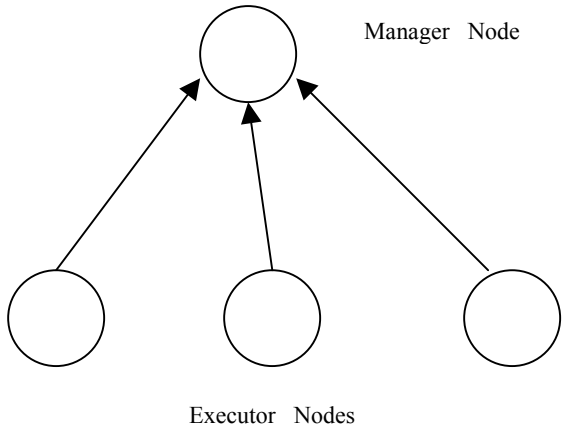


Figure 3.5: Centralized Strategies

In a de-centralized scheme, the load balancer is replicated on all workstations. There are different algorithms used in de-centralized scheme for job selection. These algorithms are

Round robin algorithm, Random polling algorithm etc

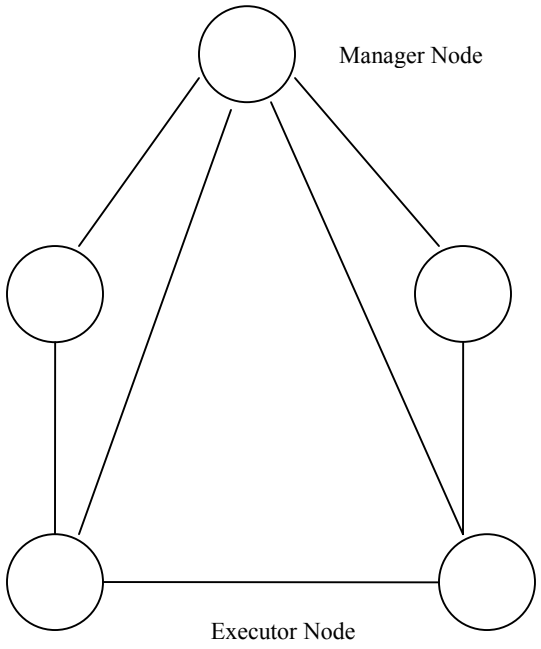


Figure 3.6: Decentralized Strategies

Once again, there are tradeoffs associated with choosing one location scheme over the other. For centralized schemes, the reliance on one central point of balancing control could limit future scalability. Additionally, the central scheme also requires an "all-to-one" exchange of profile information from workstations to the balancer as well as a "one-to-all" exchange of distribution instructions from the balancer to the workstations. The distributed scheme helps solve the scalability problems, but at the expense of an "all-to-all" broadcast of profile information between workstations. However, the distributed scheme avoids the "one-to-all" distribution exchange since the distribution decisions are made on each workstation.

3.2.3 Load Balancing Policies

Load balancing algorithms can be defined by their implementation of the following policies [42-45]:

- Information policy: specifies what workload information to be collected, when it is to be collected and from where.

- Triggering policy: determines the appropriate period to start a load balancing operation.
- Resource type policy: classifies a resource as server or receiver of tasks according to its availability status.
- Location policy: uses the results of the resource type policy to find a suitable partner for a server or receiver.
- Selection policy: defines the tasks that should be migrated from overloaded resources (source) to most idle resources (receiver).

The main objective of load balancing methods is to speed up the execution of applications on resources whose workload varies at run time in unpredictable way. Hence, it is significant to define metrics to measure the resource workload. Every dynamic load balancing method must estimate the timely workload information of each resource. This is key information in a load balancing system where responses are given to following questions:

- How to measure resource workload?
- What criteria are retaining to define this workload?
- How to avoid the negative effects of resources dynamicity on the workload; and,
- How to take into account the resources heterogeneity in order to obtain an instantaneous average workload representative of the system?

Several load indices have been proposed in the literature, like CPU queue length, average CPU queue length, CPU utilization, etc. The success of a load balancing algorithm depends from stability of the number of messages (small overhead), support environment, low cost update of the workload, and short mean response time which is a significant measurement for a user. It is also essential to measure the communication cost induced by a load balancing operation.

3.3 Scheduling and Load Balancing

Two important aspects of any wide area network scheduler are its transfer and location policies [46].The transfer policy decides if there is a need to initiate load balancing across

the system and is typically threshold based. Using workload information, it determines when a node becomes eligible to act as a sender (transfer a job to another node) or as a receiver (retrieve a job from another node).the location policy selects a partner node for a job transfer transaction. In other words, it locates complementary nodes to/from which a node can send/receive workload to improve overall system performance.

Location policies can be broadly classified as sender-initiated, receiver-initiated, or symmetrically-initiated. Sender initiated policies are those where heavily-loaded nodes search for lightly-loaded nodes while receiver-initiated policies are those where lightly-loaded nodes search for suitable senders. Symmetrically-initiated policies combine the advantages of these two by requiring both senders ad receivers o look for appropriate partners.

Load balancing policies can also be classified on the basis of how up-to-date each node's knowledge is about the state of the system. Dynamic policies make decisions based on the current system state and can rapidly adapt to workload fluctuations. On the other hand, polices that that use static information and are not amenable to changes in the workload are known as static policies. However, dynamic policies incur the overhead of communicating among the system nodes to keep them informed about the state of the system.

3.3.1 Concept of Queues

There are two types of queues in Grid environment [46]. The internal queue of node consists of the ready jobs which would be executed by this particular node only. Instead, external queues of a node consists of jobs which have been initially submitted to this node by a user ,but are yet to be mapped and scheduled for execution.

3.3.2 Rescheduling

Launch-time scheduling can at best start the application with a good schedule [47]. Over time, other applications may introduce load in the system or application requirements may change. To sustain good performance for longer running applications, the schedule may need to be modified during application execution. This process, called rescheduling,

can include changing the machines on which the application is executing (migration) or changing the mapping of data and/or processes to those machines (dynamic load balancing). Rescheduling involves a number of complexities not seen in launch-time scheduling. First, while nearly all parallel applications support some form of launch-time scheduling (selection of machines at a minimum), very few applications have built-in mechanisms to support migration or dynamic load balancing. Second, for resource monitoring we have to differentiate between processors on which the application is running and processors on which the application is not running. Measurements from resource monitors such as NWS CPU sensors can not be directly compared between active and inactive processors. Third, the overheads of rescheduling can be high: monitoring for the need to reschedule is an ongoing process and, when a rescheduling event is initiated, migration of application processes or reallocation of data can be very expensive operations. Without careful design, rescheduling can in fact hurt application performance.

Following is the rescheduling architecture and it depicts an application already executing on n processors. While the application is executing, application sensors are co-located with application processes to monitor application progress. Progress can be measured, for example, by generic metrics such as flop rate or by application-intrinsic measures such as number of iterations completed. In order that these sensors have access to internal application state, we embed them in the application itself. It works to minimize the impact of these sensors on the application's footprint and execution progress. Meanwhile, resource sensors are located on the machines on which the application is executing, as well as the other machines available to the user for rescheduling. For evaluating schedules, it is important that measurements taken on the application's current execution machines can be compared against measurements taken on unused machines in the testbed. Application sensor data and the performance contract are passed to the contract monitor, which compares achieved application performance against expectations. When performance falls below expectations, the contract monitor signals a violation and contacts the rescheduler. An important aspect of a well designed contract monitor is the ability to differentiate transient performance problems from longer-term issues that warrant modification of the application's execution. Upon a contract violation, the

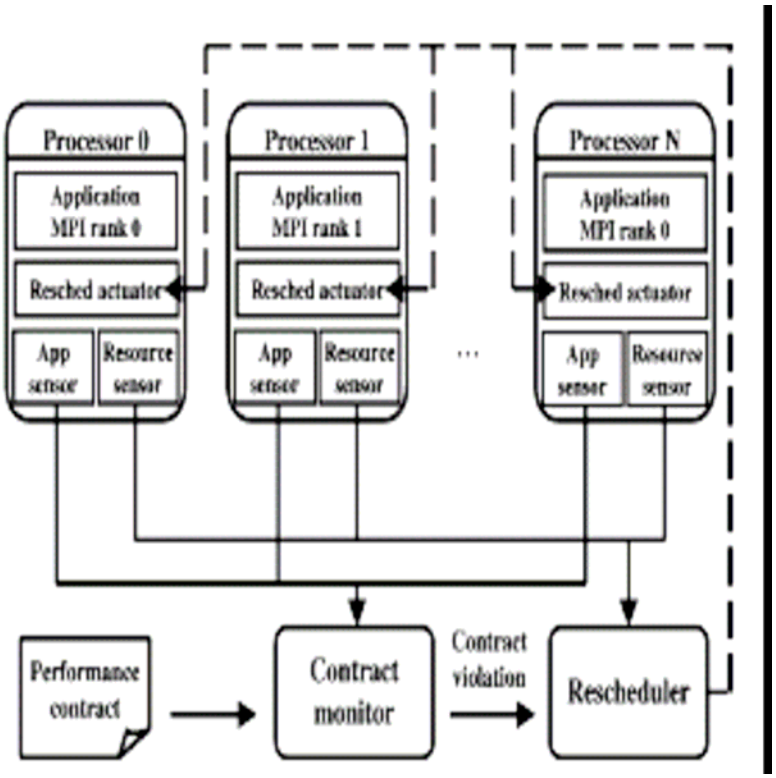


Figure 3.7: Rescheduling Architecture [47]

rescheduler must determine whether rescheduling is profitable, and if so, what new schedule should be used. Data from the resource sensors can be used to evaluate various schedules, but the rescheduler must also consider the cost of moving the application to a new execution schedule and the amount of work remaining in the application that can benefit from a new schedule. To initiate schedule modifications, the rescheduler contacts the rescheduling actuators located on each processor. These actuators use some mechanism to initiate the actual migration or load balancing. Application support for migration or load balancing is the most important part of the rescheduling system. The most transparent migration solution would involve an external migratory that, without application knowledge, freezes execution, records important state such as register values and message queues, and restarts the execution on a new processor.

When the owner of a workstation claims control of his/her node in the network, migrating the job to another workstation is desirable to simply restarting it on another workstation.

This implies that the state of a task should be captured, after which it is started on a target machine and initialized with the captured state. Correct migration is difficult since the interactions of the task with its environment need to be taken into account. Issues such as dealing with open files, handling communications with other tasks, and checkpointing overhead all need to be taken into consideration by a job migration system. From the perspective of a load balancer, migration is an important issue since the balancer needs to consider the overhead associated with job migration in order to measure the potential cost when making a balancing decision.

3.4 Generic Model for Load Balancing

This is topological structure for a Grid computing.

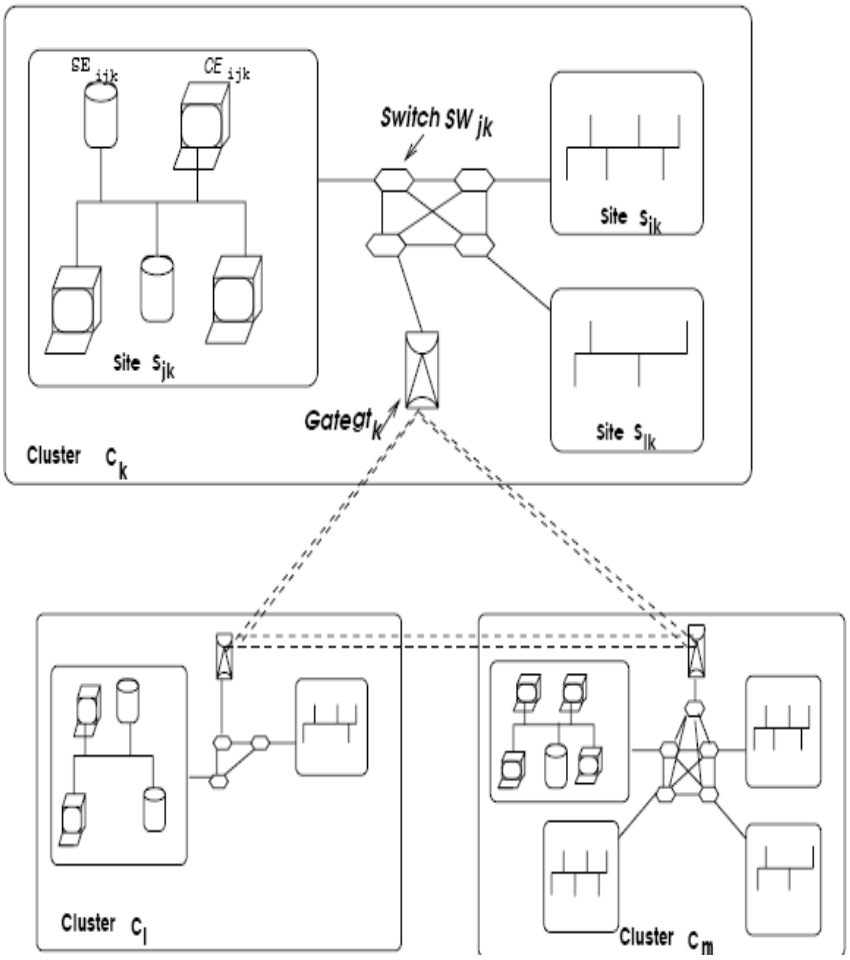


Figure 3.8: Grid Topology [48]

Grid computing is a finite set of G clusters C_k, interconnected by gates gt_k, k belongs to {0, ...,G − 1}, where each cluster contains one or more sites S_{jk} interconnected by switches SW_{jk} and every site contains some Computing Elements CE_{ijk} and some Storage Elements SE_{ijk}, interconnected by a local area network[48].

This model is based on an incremental tree. First, for each site it creates a two-level subtree. The leaves of this subtree correspond to the computing elements of a site, and the root is a virtual node associated to the site. These subtrees, that correspond to sites of a cluster, are then aggregated to form a three-level sub-tree. Finally, these sub-trees are connected together to generate a four-level tree called load balancing generic model. This model is denoted by G/S/M, where G is the number of clusters that compose a Grid, S the number of sites in the Grid and M the number of Computing Elements. This model can be transformed into three specific models: G/S/M, 1/S/M and 1/1/M, depending on the values of G and S. The generic model is a non cyclic connected graph where each level has specific functions [49].

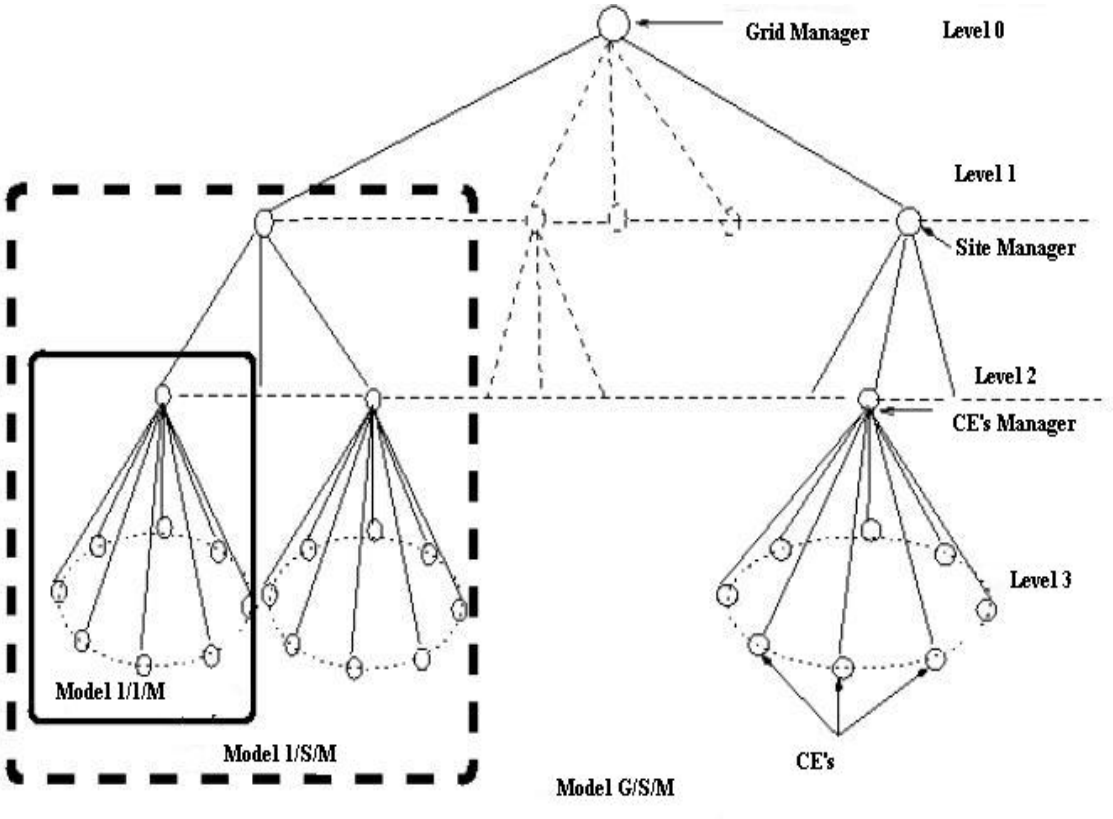


Figure 3.9: Load Balancing Generic Model [42, 48]

Level 0: In this first level (top level of the tree), it has a virtual node that corresponds to the root of the tree. It is associated to the Grid and performs two main functions: (i) manage the workload information of the Grid; (ii) decides, upon receiving tasks from users, where these tasks can be launched, based on the user requirements and the current load of the Grid.

Level 1: This level contains G virtual nodes, each one associated to a physical cluster of the Grid. In this load balancing strategy, this virtual node is responsible to manage workload of its sites.

Level 2: In this third level, it finds S nodes associated to physical sites of all clusters of the Grid. The main function of these nodes is to manage the workload of their physical computing elements.

Level 3: At this last level (leaves of the tree), it finds the M Computing Elements of the Grid linked to their respective sites and clusters.

3.4.1 Characteristics of Model

The main features of load balancing generic model are listed below [50]:

- It is hierarchical: this characteristic facilitates the information flow through the tree and well defines the message traffic in our strategy;
- It supports heterogeneity and scalability of Grids: adding or removing entities (computing elements, sites or clusters) are very simple operations in this model (adding or removing nodes, sub-trees);
- It is totally independent from any physical architecture of a Grid: the transformation of a Grid into a tree is a univocal transformation. Each Grid corresponds to one and only one tree.

In accordance with the structure of model, the load balancing strategy is also hierarchical. Hence, we distinguish between three load balancing levels:

- **Intra-site load balancing**: In this first level, depending on its current load, each site decides to start a load balancing operation. In this case, the site tries, in priority, to load balance its workload among its computing elements.

- **Intra-cluster load balancing**: In this second level, load balance concerns only one cluster, C_k , among the clusters of a Grid. This kind of load balance is achieved only if some sites S_{jk} fail to load balance its workload among their respective computing elements. In this case, they need the assistance of its direct parent, namely cluster C_k.
- **Intra-Grid load balancing**: The load balance at this level is used only if some clusters C_k's fail to load balance their load among their associated sites. The main advantage of this strategy is to prioritize local load balancing first (within a site, then within a cluster and finally on the whole Grid). The goal of this neighborhood strategy is to decrease the amount of messages between computing elements.

This chapter mainly focuses Load balancing strategies, algorithms and policies in a Grid environment. The next chapter discusses about problem formulation and approaches used to implement the same.

Chapter 4 Problem Formulation

Grid computing enables sharing, selection and aggregation of large collections of geographically and organizationally distributed heterogeneous resources for solving large-scale data and compute intensive problems. Resources are dynamic in nature so the Load of resources varies with change in configuration of Grid and it makes Load Balancing more important in case of Grid environment. This chapter presents detailed description of thesis problem and approach used to solve it.

4.1 Problem Statement

In grid environments, the shared resources are dynamic in nature, which in turn affects application performance. Workload and resource management are two essential functions provided at the service level of the Grid software infrastructure. To improve the global throughput of these environments, effective and efficient load balancing algorithms are fundamentally important. The focus of our study is to consider factors which can be used as characteristics for decision making to initiate Load Balancing. Load Balancing is one of the most important factors which can affect the performance of the grid application.

This thesis work analyzes the existing Load Balancing modules and tries to find out performance bottlenecks in it. All Load Balancing algorithms implement five policies [7]. The efficient implementation of these policies decides overall performance of Load Balancing algorithm. The main objective of this thesis is to propose an efficient Load Balancing Algorithm for Grid environment. Main difference between existing Load Balancing algorithm and proposed Load Balancing is in implementation of three policies: Information Policy, Triggering Policy and Selection Policy. For implementation of Information Policy all existing Load Balancing algorithm use periodic approach, which is time consuming. The proposed approach uses activity based approach for implementing Information policy. For Triggering Load Balancing proposed algorithm uses two parameters which decide Load Index. On the basis of Load Index Load Balancer decide to activate Load Balancing process. For implementation of Selection Policy Proposed

algorithm uses Job length as a parameter, which can be used more reliably to make decision about selection of job for migration from heavily loaded node to lightly loaded node.

Following table discusses the main differences between the proposed algorithm and Condor Load Balancing algorithm.

	Information Policy	**Triggering Policy**	**Selection Policy**
Condor Load Balancer (existing)	Load Balancing information is collected using periodic approach	Load Balancer is triggered based on Queue Length	Task is selected for migration using Job Length as criteria.
Proposed Load Balancer	Load Balancing information is collected using Activity based approach	Load Balancer is triggered based on Queue Length and current CPU Load	Task is selected for migration based upon CPU consumption of tasks

Table 4.1: Comparison between Condor LB Module and Proposed LB Module

As a result the following things have been addressed in this thesis:

- Study of existing Load Balancing algorithm for a Grid environment
- A Load Balancing algorithm has been proposed and executed in the simulated Grid environment.

This thesis aims is to design and development of a performance efficient Load Balancing algorithm which overcomes the shortcomings of the current state of the art in the context.

Chapter 5 Proposed Load Balancing Algorithm

Load balancing is defined as the allocation of the work of a single application to processors at run-time so that the execution time of the application is minimized. This chapter is going to discuss the design of proposed Load Balancing algorithm.

5.1 Background

The choice of a load balancing algorithm for a Grid environment is not always an easy task. Various algorithms have been proposed in the literature, and each of them varies based on some specific application domain. Some load balancing strategies work well for applications with large parallel jobs, while others work well for short, quick jobs. Some strategies are focused towards handling data-heavy tasks, while others are more suited to parallel tasks that are computation heavy. While many different load balancing algorithms have been proposed, there are four basic steps that nearly all algorithms have in common:

- Monitoring workstation performance (load monitoring)
- Exchanging this information between workstations (synchronization)
- Calculating new distributions and making the work movement decision (rebalancing criteria)
- Actual data movement (job migration)

Efficient Load Balancing algorithm makes Grid Middleware efficient and which will ultimately leads to fast execution of application in Grid environment. In this work, an attempt has been made to formulate a decentralized, sender-initiated load balancing algorithm for Grid environments which is based on different parameters. One of the important characteristics of this algorithm is to estimate system parameters such as queue length and CPU utilization of each participating nodes and to perform job migration if required.

5.2 Design of Load Balancing Algorithm

Load balancing should take place when the load situation has changed. There are some particular activities which change the load configuration in Grid environment. The activities can be categorized as following:

- Arrival of any new job and queuing of that job to any particular node.
- Completion of execution of any job.
- Arrival of any new resource
- Withdrawal of any existing resource.

Whenever any of these four activities happens activity is communicated to master node then load information is collected and load balancing condition is checked. If load balancing condition is fulfilled then actual load balancing activity is performed.
Following is the proposed algorithm for Load Balancing:

```
Loop
      wait for load change
                        // depends on happening of any of four defined activities
 if (activity_happens ())
   If (LoadBalancing_start ())
        while HeavilyLoaded_list is not empty
            Determine tasks which can be migratable using criteria of CPU consumed by
            each job which has least CPU consumption selected for being migrated.
              Selected job = j;
              If  LightlyLoaded_list is empty
                   PendingJob_list = PendingJob_list + j;
              Else
                   Migrate (LightlyLoaded_list [first], HeavilyLoaded_list[n], j);
                              // update the database
         End while
End Loop
```

Figure 5.1: Outline of Proposed Load Balancing Algorithm

Following are some functions used in the above algorithm:

Activity_happens (): this function return Boolean value. If any of above defined activity occurs it returns true otherwise it returns false.

LoadBalancing_start (): this function also return Boolean value. If on the basis of given parameters (CPU utilization and queue length) load balancing will be required it will return true else it will return false. This function also updates two lists: HeavilyLoaded_list and LightlyLoaded_list

Threshold heavy load and threshold light load is defined initially which depends on the traffic of application on the Grid.

```
Function: LoadBalancing_start
Return Type: Boolean
Start:
            If (Standard Deviation of Load of nodes < SD_Threshold)
          If (Load of any node is greater then average Load value of nodes)
       HeavilyLoaded_list= HeavilyLoaded_list + l (new selected node);
          End if
  Else (Load of any node is greater then threshold heavy load value)
          HeavilyLoaded_list= HeavilyLoaded_list + l (new selected node);
  Else if (Load of any node is less then threshold light load value)
End
```

Figure 5.2: Outline of LoadBalancing_start Function

Here actual load distribution is performed at a centralized controller or manager node. The central controller polls each workstation and collects state information consisting of a node's current load as well as the number of jobs in the node's queue. The polling is done on basis of occurrence of some defined activity. It is not done periodically. Periodic checking approach is used in Condor. In case of periodic approach Load Balancer collects Load sample periodically which is not required and infect creates an overhead

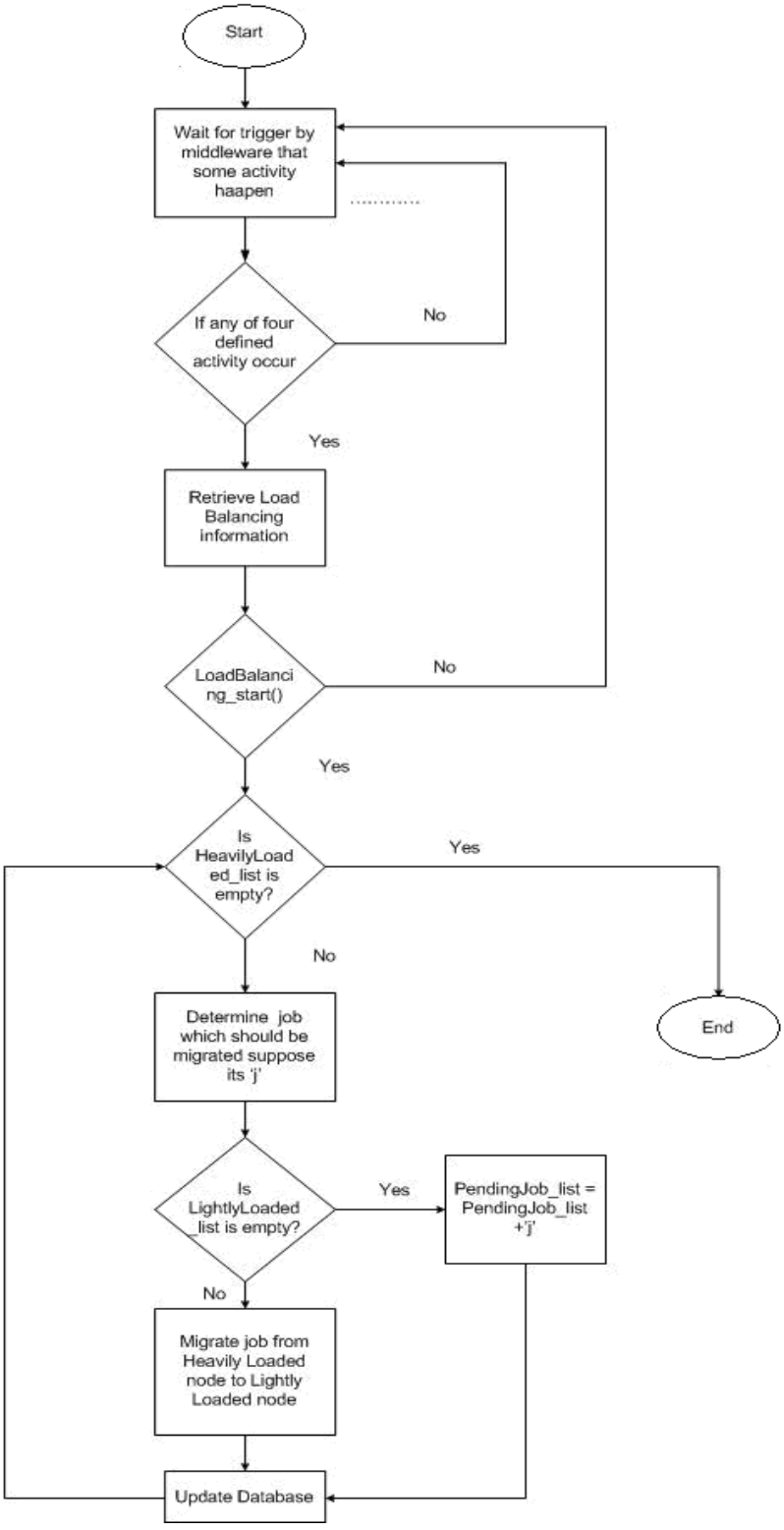


Figure 5.3: Flow Chart of Algorithm

also. In the proposed algorithm information is collected only if there is a change in configuration of Grid. This information is used to perform load balancing. Above is the flow diagram of algorithm. First of all it initializes different parameters. Whenever any of four activities which are required to start information policy of load balancing occurs, it starts collecting load balancing information. Once information has been gathered then it is decided that load balancing is required or not. For this purpose application uses CPU utilization and queue length parameters. With help of these parameters we decide which resource is heavily loaded and which resource is lightly loaded. After selection of resource the application selects job out of n-jobs running on that resource. This selection is based upon on CPU consumption of different jobs. Least CPU consumed job will be selected for migration. When job is selected, application checks for available lightly loaded resource. If lightly loaded resource is available then migrate selected job from heavily loaded resource to lightly loaded resource. If no lightly loaded resource is available then add selected job to pending job list. This job will be executed later when some lightly loaded resource will be available. Finally all the value will be updated in database.

Additionally, it supports check pointing, whereby a disk image of an active process is stored on the workstation in order to facilitate both fault tolerance and process migration. Check pointing time is fixed initially.

Next chapter will discuss technology used and implementation detail for the proposed solution

Chapter 6 Implementation Details and Experimental Results

To implement the proposed Load Balancing Algorithm, an application has been developed, which is executed in simulated grid environment. The application has been developed using J2EE and MySQL database server.

6.1 Used Technologies

In this section the summary of technologies used in developing the project has been discussed.

6.1.1 Java

Java language offers several features that facilitate with easiness the development and deployment of a software environment for Grid computing. As Grid is network based java's network based features are very useful to develop Grid application. Java's network-centric approach and its built-in support for mobile code enable the distribution of computational tasks to different computer platforms.

In particular the various benefits of Java this seems ideal for multiparadigm communication environments. Java's platform-independent bytecode can be executed securely on many platforms, making Java an attractive basis for portable Grid computing. In addition, Java's performance on sequential codes, which is a strong prerequisite for the development of such Grid applications, has increased substantially over the past years. Java provides a sophisticated graphical user interface framework, as well as a paradigm to invoke methods on remote objects. These features are of particular interest for Grid application.

Java runtime systems are available for most hardware platforms and operating systems. Because of the heterogeneity of the hardware and of operating systems employed by Internet users, it is crucial that a platform for large-scale Grid computing be available for a large variety of different computer systems. Consequently, a Java- based platform potentially allows every computer in the Internet to be exploited for distributed, large-

scale computations, while at the same time the maintenance costs for the platform are minimal (”write once, run everywhere”). Apart from its portability and compatibility, language safety and a sophisticated security model with flexible access control are further frequently cited advantages of Java. As security is of paramount importance for the acceptance of a platform for Grid computing, the security and safety features of Java are highly appreciated in this context.

6.1.2 GridSim: Grid Modeling and Simulation Toolkit

The GridSim 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, managing resources such as clusters in a single administrative domain, 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.

6.1.2.1 Features

Salient features of the GridSim toolkit include the following:

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled operating under space- or time-shared mode.
- Resource capability can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark.
- Resources can be located in any time zone.

- Weekends and holidays can be mapped depending on resource's local time to model on-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- 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. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- 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.

6.1.2.1 Use Case Study – Grid Computing Environment Simulation Using GridSim

This section describes how a simulated Grid computing environment is created using GridSim. First, the Grid users and resources for the simulated Grid environment have to be created. This can be done easily using the wizard dialog as shown in Figure 6.1.

Figure 6.1: Wizard Dialog to Create Grid Users and Resources

The GridSim user only needs to specify the required number of users and resources to be created. Random properties can also be automatically generated for these users and resources. The GridSim user can then view and modify the properties of these Grid users and resources by activating their respective property dialog.

Figure 6.2 shows the property dialog of a sample Grid resource. GridSim creates Grid resources similar to those present in any other testbed. Resources of different capabilities and configurations can be simulated, by setting properties such as cost of using this resource, allocation policy of resource managers (time/space-shared) and number of machines in the resource (with Processing Elements (PEs) in each machine and their Million Instructions per Second (MIPS) rating).

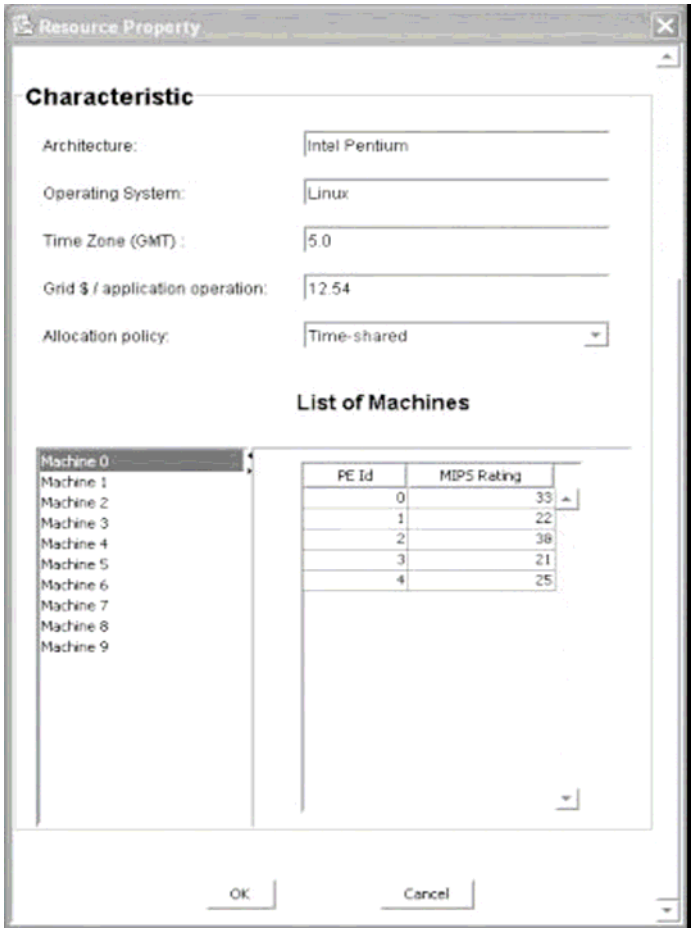


Figure 6.2: Resource Dialog to View Grid Resource Properties

Figure 6.3 shows the property dialog of a sample Grid user. Users can be created with different requirements (application and quality of service requirements). These requirements include the baud rate of the network (connection speed), maximum time to run the simulation, time delay between each simulation, and scheduling strategy such as cost and/or time optimization for running the application jobs. The application jobs are modeled as Gridlets. The parameters of Gridlets that can be defined includes number of Gridlets, job length of Gridlets (in Million Instructions (MI)), and length of input and output data (in bytes). GridSim provides a useful feature that supports random distribution of these parameter values within the specified derivation range. Each Grid user has its own economic requirements (deadline and budget) that constrain the running of application jobs. GridSim supports the flexibility of defining deadline and budget based on factors or values.

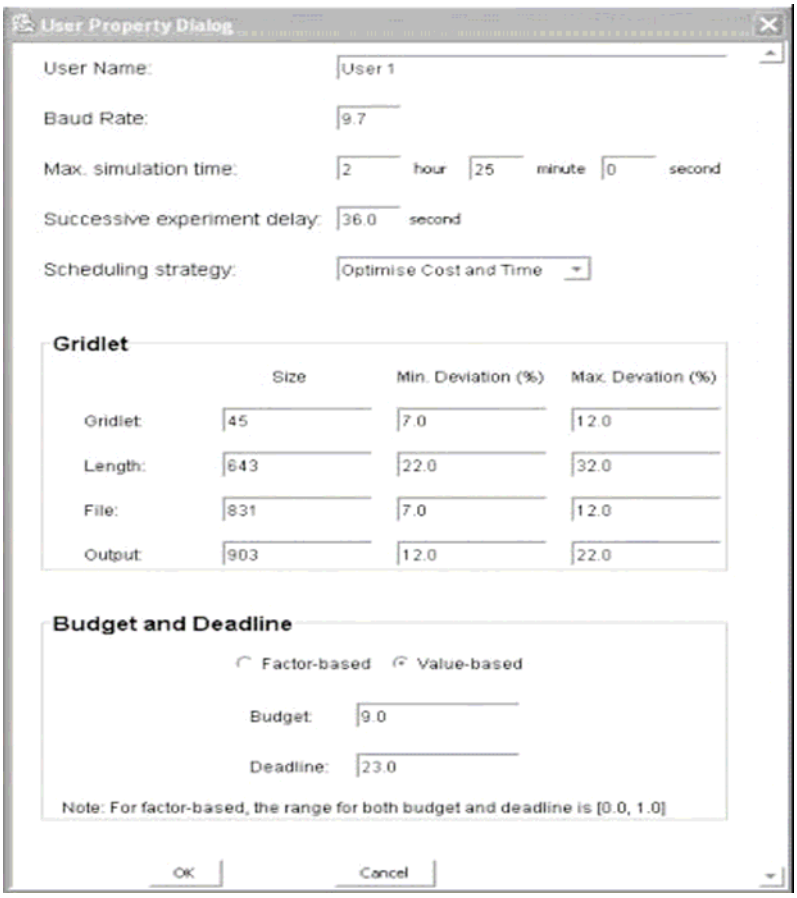


Figure 6.3: User Dialog to View Grid User Properties

If it is factor-based (between 0.0 and 1.0), a budget factor close to 1.0 signifies the Grid user's willingness to spend as much money as required. The Grid user can have the exact cost amount that it is willing to spend for the value-based option. GridSim will automatically generate Java code for running the Grid simulation .This file can then be compiled and run with the GridSim toolkit packages to simulate the required Grid computing environment.

6.2 Implementation Details

A Load Balancing Module has been developed which executes in simulated grid environment. This application has been developed using J2EE and MySQL database server.

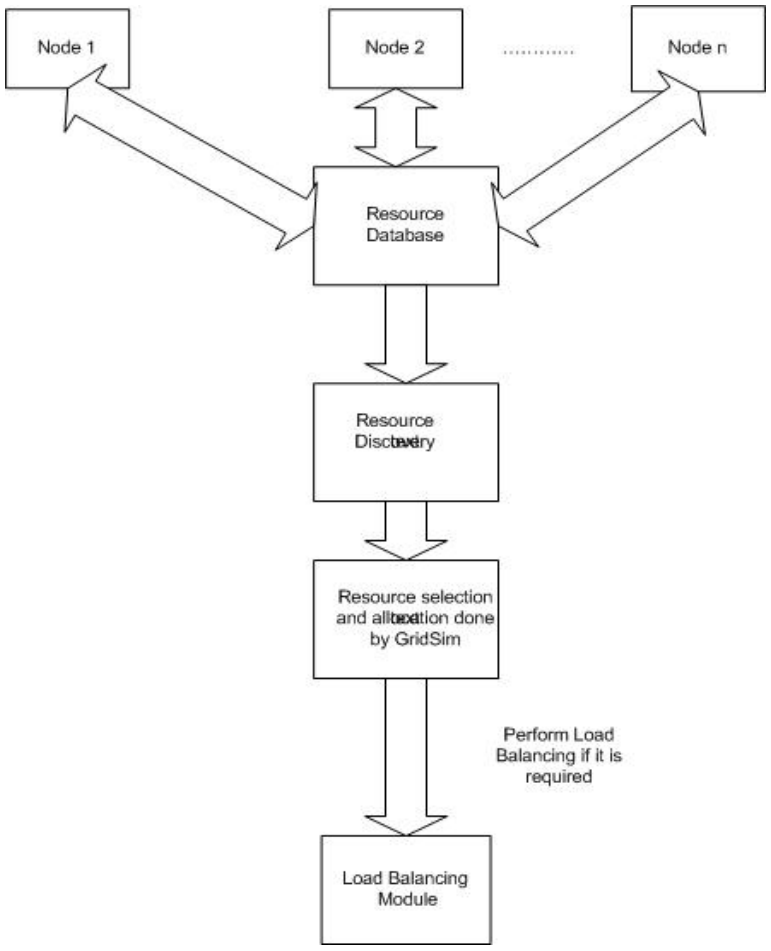
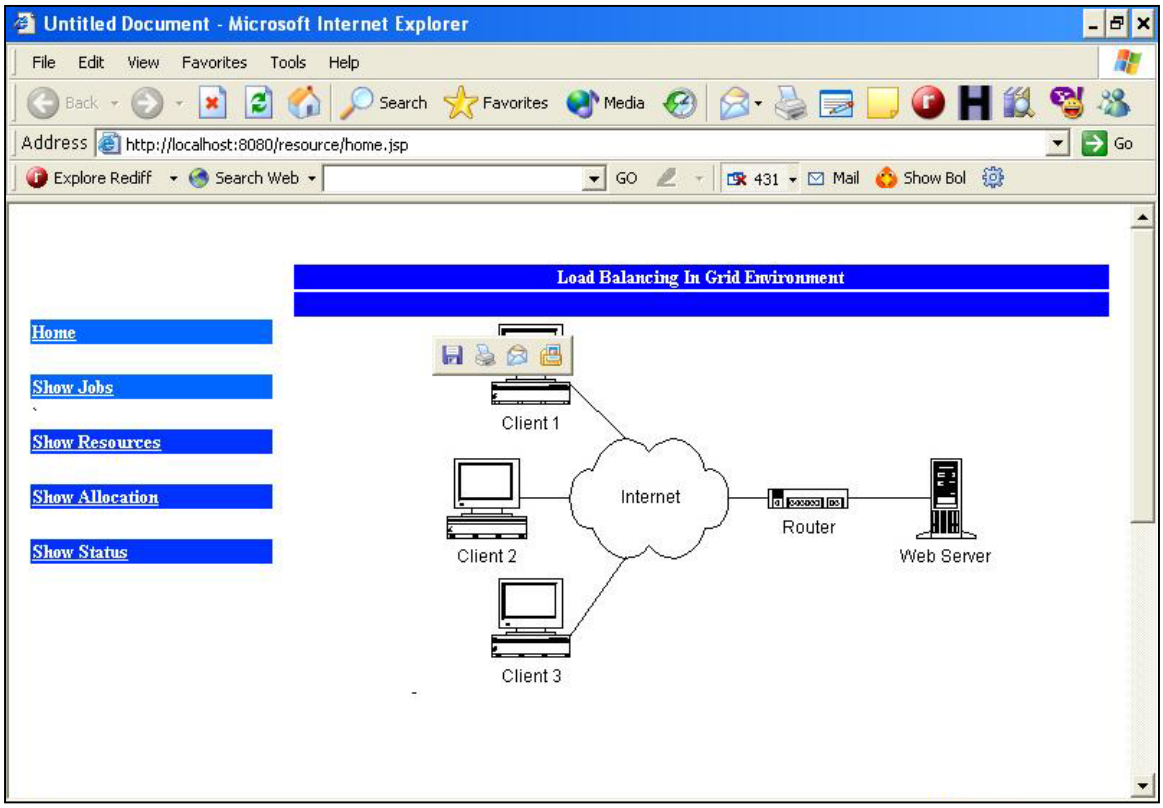


Figure 6.4: Overall System Architecture

Above is the overall architecture of the application developed. Information about all resources is stored in resource database. Resources are generated by GridSim. Resource discovery process use resource database to discover all possible match to the resource query. Next process is resource selection and allocation. This process is also done by GridSim. Once resource allocation is done then Load Balancing process come in existence. Execution of Load Balancing depends on condition specified.

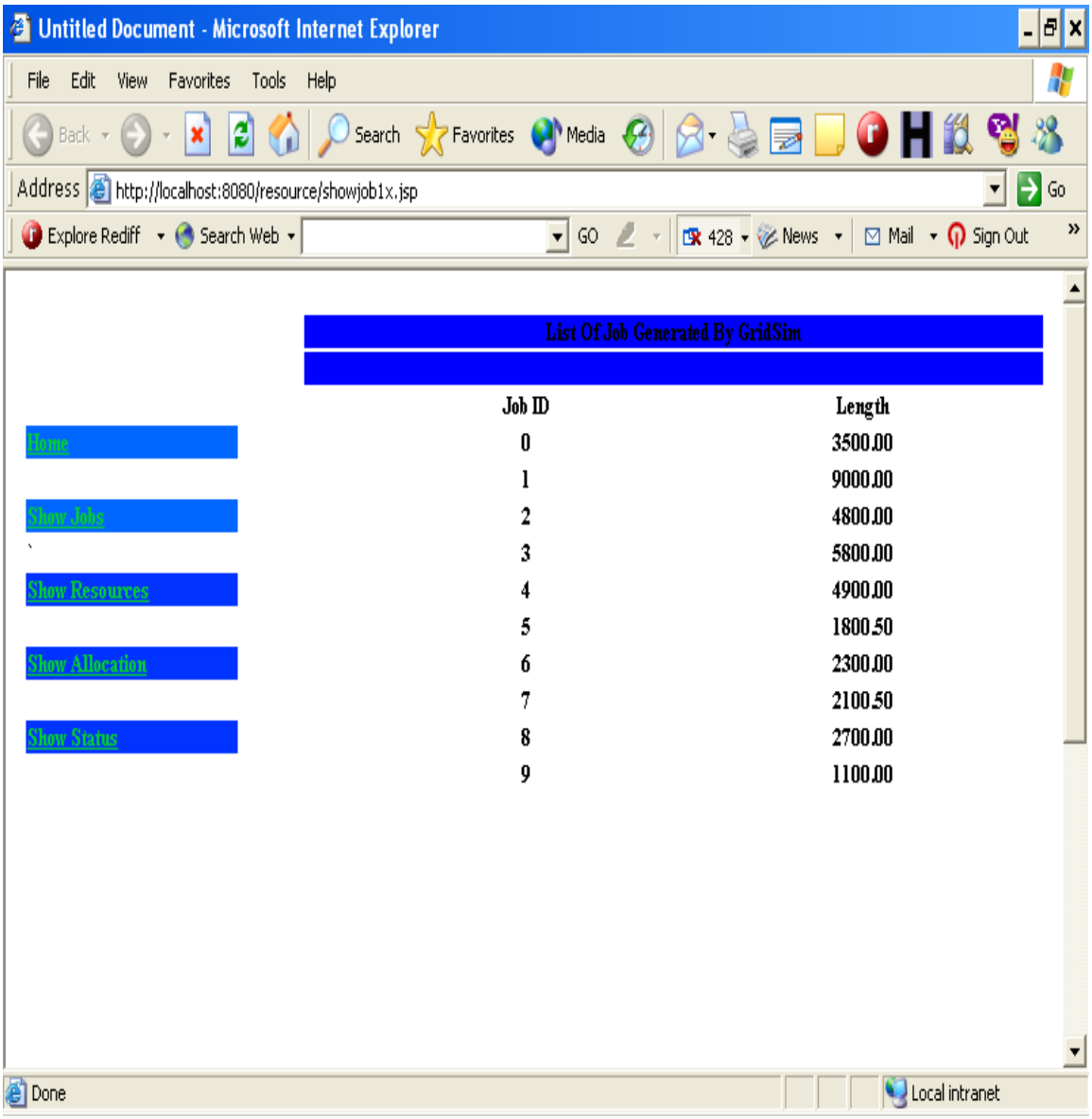
6.3 Experimental Results

As discussed earlier the application is developed in java. Following is the image of the index page.



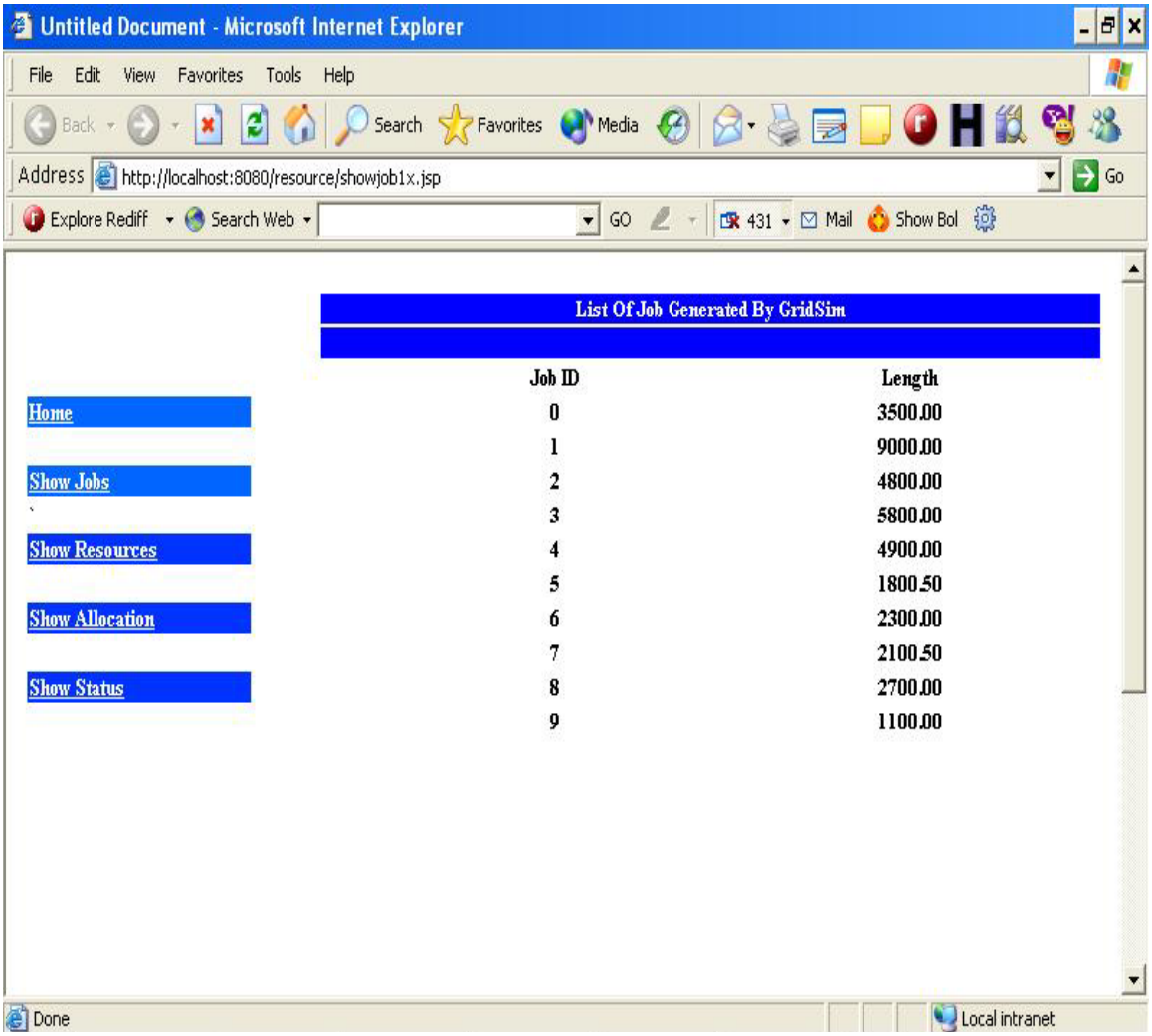
Screen Shot 6.5: Image of Home Page

It contains link to different pages: show jobs, show resources, show allocation and show status. A click on one of the links will link to the referenced JSP page.



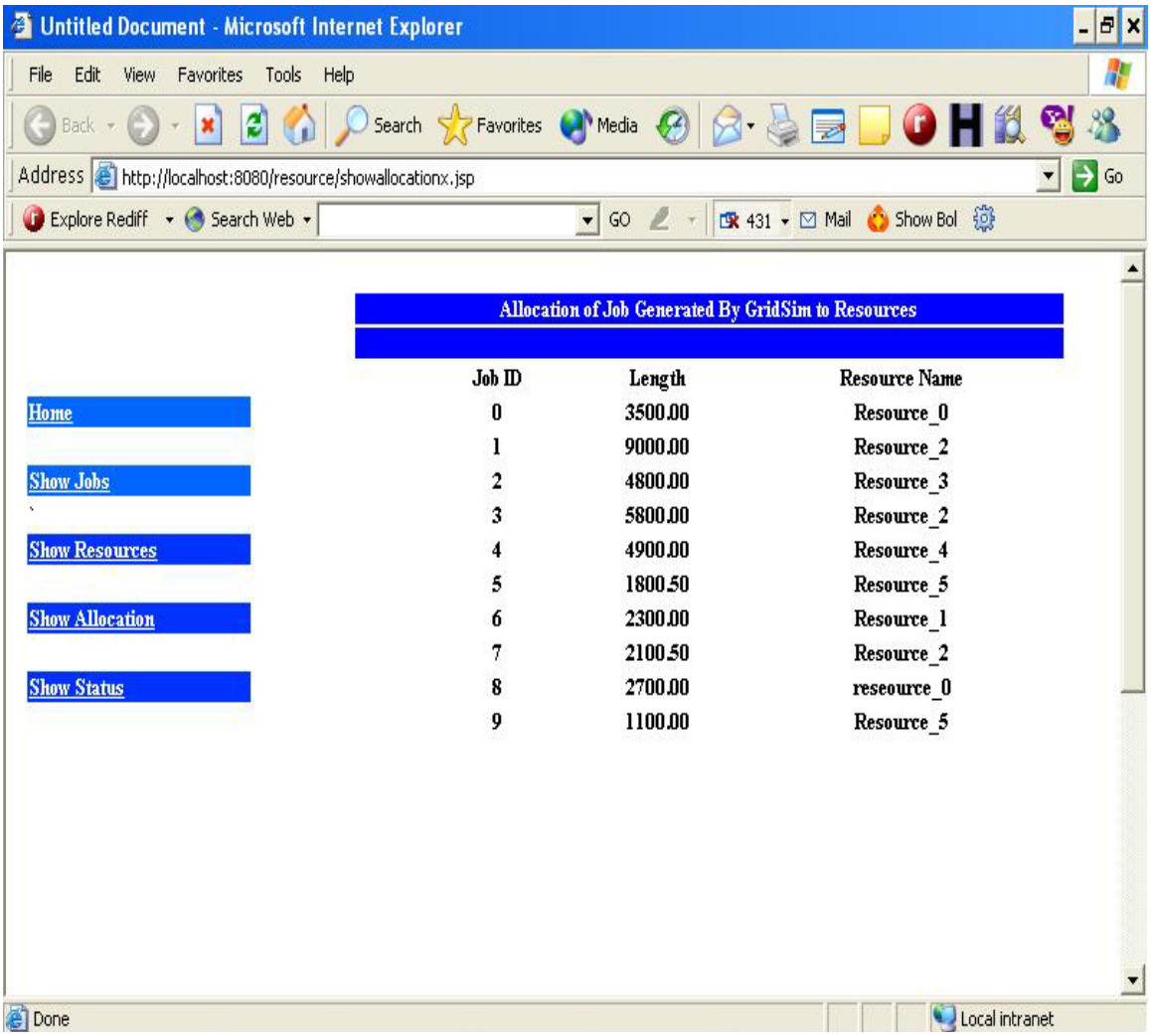
Screen Shot 6.6: Image of Show Job Page

Above is the image of show job page. This image shows two columns: first column is Job ID and second is job length. Job ID is unique for each job submitted to Grid. Length is the expected job time length given at the time of submission. These jobs are created by GridSim other characteristics of job (Gridlet) are maintained automatically by GridSim. These fields are submission time, Job execution priority etc.



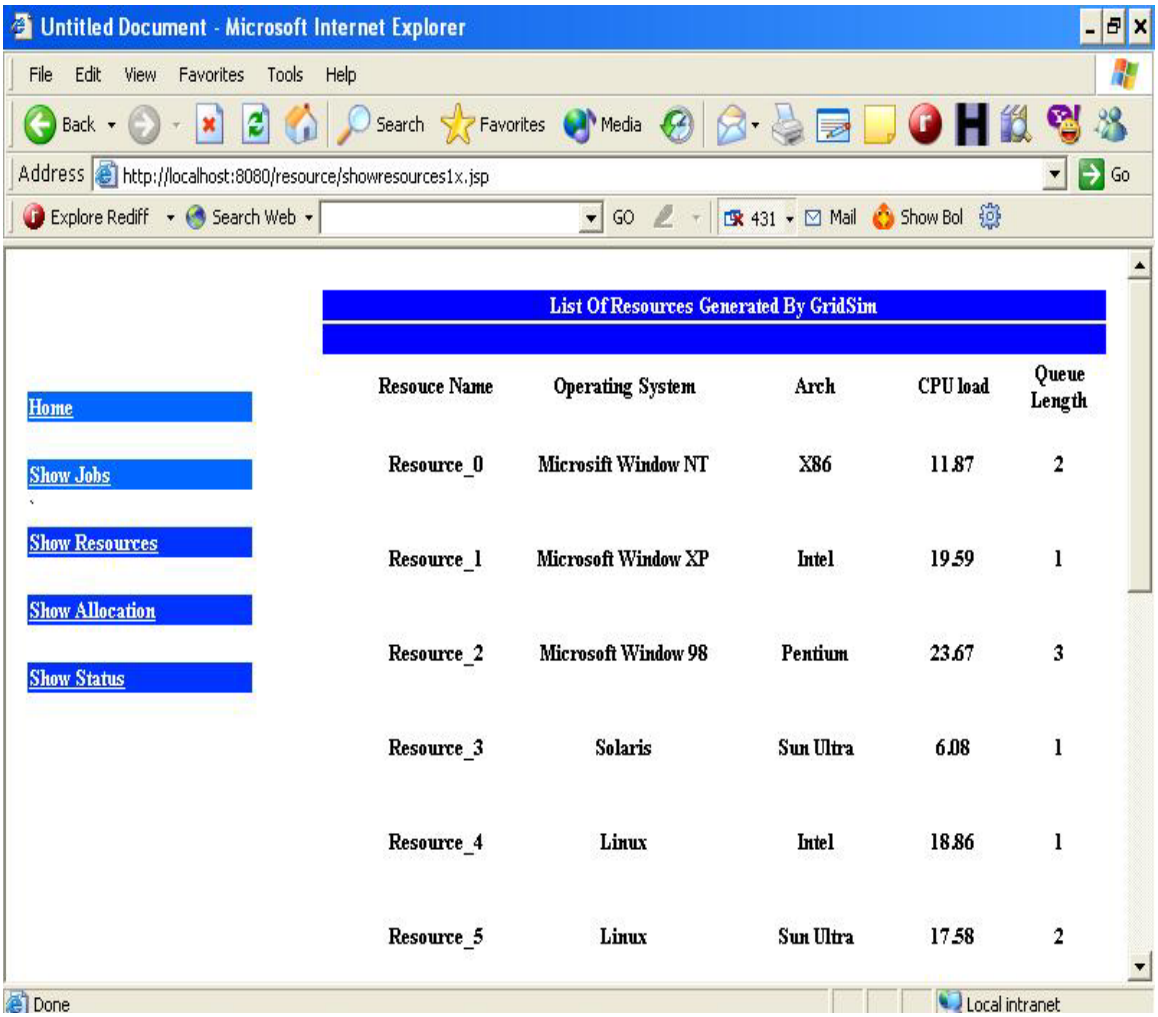
Screen Shot 6.7: Image of Show Resources Page

Above is the image of show resource page. This image shows five fields: resource name (Resname), operating system, architecture (arch), and CPU load and queue length. These resources are created by GridSim. Last two fields are important because these fields are used to decide that particular resource is heavily loaded or lightly loaded resources. If any resource is heavily loaded then the actual load balancing starts.



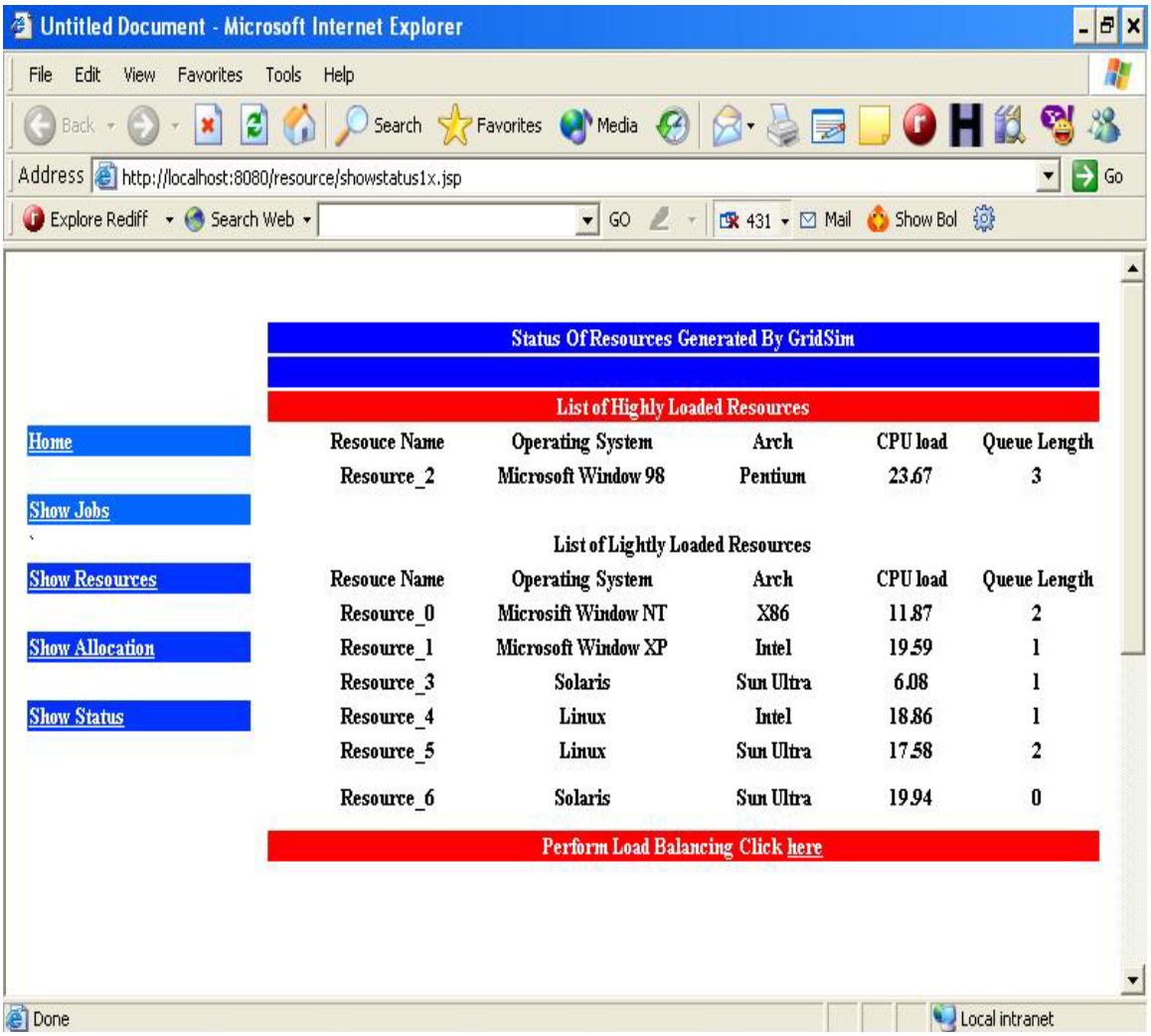
Screen Shot 6.8: Image of Show Allocation Page

Above is the image of show allocation page. This image shows three fields. This is generated after allocation of job (Gridlet) to resource. This allocation is done by GridSim. For allocation GridSim uses time shared allocation policy. This allocation maximizes the resource utilization. In this table corresponding to each Job ID, there is a resource name.



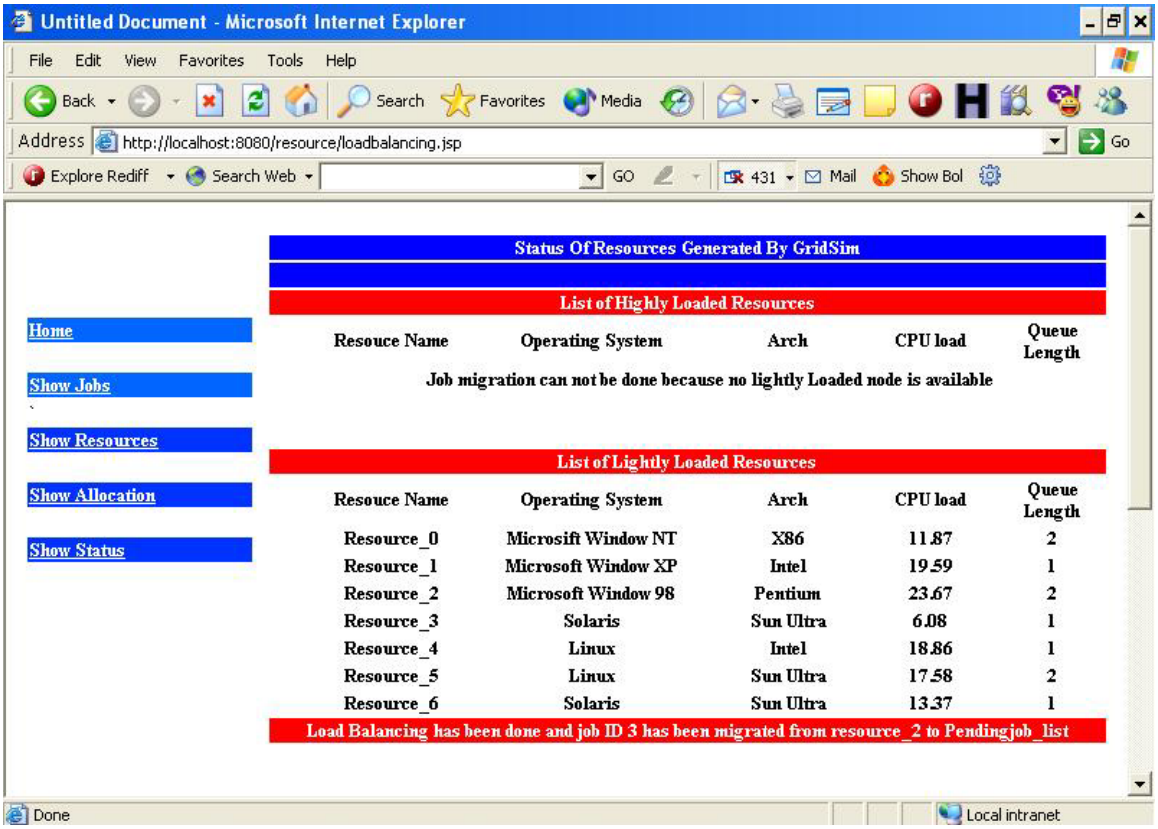
Screen shot 6.9: Image of Show Status Page

Above is the image of show status page. This image shows information about gathered about Load Balancing. This page contains two tables. First table contain name of those resource which is heavily loaded and second table contain those resources which are lightly loaded. These two tables are used to decide which resource will act sender and which resource will act as receiver. Job will be migrated from sender resource to receiver resource.



Screen Shot 6.10: Image of Load Balancing (1) Page

Above is the image of Load Balancing (1) page. This window appears after Load Balancing has been performed. In normal scenario if sufficient lightly loaded resources are available then after load balancing no heavily loaded resource will be available. Job from all heavily loaded resource will be migrated to lightly loaded resource. This page also gives information about which Job ID is migrated from which resource to which resource.



Screen Shot 6.11: Image of Load Balancing (2) Page

Above is the image of Load Balancing (2) page. This image shows after Load Balancing has been performed but job is not migrated. This is one particular case when heavily loaded resource has been finalize and job which should be migrated has been finalize but there is no corresponding lightly loaded resource is available. In this case job is put in to PendingJob list. When ever any lightly loaded resource will be available this job will be migrated to the lightly loaded resource.

This chapter mainly focused on proposing design and implementation of decentralized load balancing algorithm. In the next chapter the emphasis has been laid upon the overall summary of thesis and future direction in enhancement of the algorithm and its integration with other Middlewares such as Condor etc.

Chapter 7 .Conclusion and Scope of Future Work

This chapter discusses the conclusions of the work presented in this thesis. The chapter ends with a discussion of the future direction which this work will take.

Through this thesis, we have described multiple aspects of Grid Computing and introduced numerous concepts which illustrate its broad capabilities. Grid Computing is definitely a promising tendency to solve high demanding applications and all kinds of problems. Objective of the grid environment is to achieve high performance computing by optimal usage of geographically distributed and heterogeneous resources.

But grid application performance remains a challenge in dynamic grid environment. Resources can be submitted to Grid and can be withdrawn from Grid at any moment. This characteristic of Grid makes Load Balancing one of the critical features of Grid infrastructure.

There are a number of factors, which can affect the grid application performance like load balancing, heterogeneity of resources and resource sharing in the Grid environment. In this thesis we have focused on Load Balancing and tried to present the impacts of Load Balancing on grid application performance and finally proposed a efficient Load Balancing algorithm for Grid environment.

Every Load Balancing algorithm implements five policies. The efficient implementation of these policies decides overall performance of Load Balancing algorithm. In this work we analyzed existing Load Balancing algorithm and proposed an enhanced algorithm which more efficiently implements three out of five policies implemented in existing Load Balancing algorithm. These three policies are: Information Policy, Triggering Policy and Selection Policy. Proposed algorithm is executed in simulated Grid environment.

Load Balancing is one of most important features of Grid Middleware for efficient execution of compute intensive applications. The efficiency of Load Balancing Module overall decides the efficiency of Grid Middleware.

7.1 Future Directions

- The work performed in this thesis can be used as the basis for an improved load balancing module in Condor.
- This not only improves the performance of grid application but also makes it more powerful, reliable and capable of handling more complex and large problems in Grid environment.
- A further extension to this work would be in making this Load balancing Module a middleware independent module.

References

[1] Krishnaram Kenthapadi, Stanford University , kngk@cs.stanford.edu and Gurmeet Singh Mankuy , Google Inc., manku@google.com,Decentralized Algorithms using both Local and Random Probes for P2P Load Balancing

[2] B. Yagoubi , Department of Computer Science, Faculty of Sciences, University of Oran and Y. Slimani , Department of Computer Science, Faculty of Sciences of Tunis, Task Load Balancing Strategy for Grid Computing .

[3] Rajkumar Buyya , Grid Computing and Distributed Systems (GRIDS) Lab., Department of Computer Science and Software Engineering, University of Melbourne, Australia and Manzur Murshed ,Gippsland School of comp and IT, Monash University, Gippsland Campus , GridSim: a toolkit for the modeling and simulation of distributed resource mgnt and scheduling for Grid computing,

[4] Dazhang Gu, Lin Yang, Lonnie R. Welch ,Center for Intelligent, Distributed and Dependable Systems ,School of Electrical Engineering & Computer Science ,Ohio University, A Predictive, Decentralized Load Balancing Approach.

[5] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", GRIDS Laboratory, The University of Melbourne, Australia

[6] Ian Foster , Carl Kesselman Steven Tuecke , The Anatomy of the Grid Enabling Scalable Virtual Organizations , Intl J. Supercomputer Applications, 2001

[7] Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "Brief History of Grid", http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html.

[8] Rajkumar Buyya and S Venugopal, "A Gentle Introduction to Grid Computing and Technologies", http://www.buyya.com/papers/GridIntroCSI2005.pdf.

[9] Gregor von laszewaski, Ian Foster, Argonne National Laboratory, Designing Grid Based Problem solving Environments www-unix.mcs.anl.gov/~laszewsk/papers/cog-pse-final.pdf.

[10] Hans-Ulrich Heiss and Michael Schmitz, Decentralized Dynamic Load Balancing: The Particles Approach.

[11]Junwei Cao1, Daniel P. Spooner, Stephen A. Jarvis, and Graham R. Nudd, Grid Load Balancing Using Intelligent Agents.

[12]Ian Foster, Argonne National Laboratory & University of Chicago, What is the Grid? A Three Point Checklist.

[13]Jennifer M. Schopf, Mathematics and Computer Science Division, Argonne National Lab, Department of Computer Science, Northwestern University, Grids: The Top Ten Questions.

[14]Karl Czajkowski, Ian Foster and Carl Kesselman, Resource Co-Allocation in Computational Grids.

[15]Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury and Steven Tuecke, The Data Grid:Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets

[16]Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, A Taxonomy and Survey of Grid Resource Management Systems.

[17]Arie Shoshani, Alex Sim and Junmin Gu, Lawrence Berkeley National aboratory, Storage Resource Managers: Middleware Components for Grid Storage.

[18]Hai Zhuge, Xiaoping Sun, Jie Liu, Erlin Yao, and Xue Chen , A Scalable P2P Platform for the Knowledge Grid .

[19]Scheduling and Resource Management in Computational Mini-Grids,July 1, 2002 .

[20]Yih-Jiun Lee and Peter Henderson, DSSE research group,.Electronics and Computer Science ,University of Southampton , A Modelling Notation for Grid Computing.

[21]Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., “Introduction to Grid Computing with Globus,” Redbook, IBM Corp., http://www.redbooks.ibm.com /redbooks/pdfs/sg246895.pdf

[22]Rob V. van Nieuwpoort, Thilo Kielmann and Henri E. Bal ,Faculty of Sciences, Division of Mathematics and Computer Science, Vrije Universiteit , Efficient Load Balancing for WideArea Divide and Conquer Applications .

[23]Thierry Prioi, “Grid Middleware”, Advanced Grid Research Workshops through European and Asian Co-operation.

[24]Ian Foster, Carl Kesselman, Jeffrey M. Nick and Steven Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.

[25]Liang Fang, Aleksander Slominski, and Dennis Gannon ,Computer Science Dept, Indiana University , Web Services Security and Load Balancing in Grid Environment.

[26]Stéphane Genaud, Arnaud Giersch and Frédéric Vivien, Load-Balancing Scatter Operations for Grid Computing, March 2003

[27]Belabbas Yagoubi and Yahya Slimani, Dynamic Load Balancing Strategy for Grid Computing

[28]Stéphane Genaud and Arnaud Giersch1 Frédéric Vivien , Load-Balancing Scatter Operations for Grid Computing.

[29]Menno Dobbera, Ger Koole and Rob van der Mei, Dynamic Load Balancing Experiments in a Grid.

[30]Reinefed Alexander, Position Paper, www.egrid.org/ec_initiatives/reinefeld.html.

[31]www.entropia.com/pdf/DCGridBrouchre0402.pdf

[32]Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". International Journal of Supercomputer Applications, 2001.

[33]Jean-Christophe Durand, "Grid Computing a Conceptual and Practical Study", November 8, 2004

[34]Clovis Chapman1, Paul Wilson2, "Condor services for the Global Grid: Interoperability between Condor and OGSA", Proceedings of the 2004 UK e-Science All Hands Meeting, ISBN 1-904425-21-6, pages 870-877, Nottingham, UK, August 2004 http://www.cs.wisc.edu/condor/doc/condor-ogsa-2004.pdf

[35]Jarek Nabrzyski ,Pozna´n Supercomputing and Networking Center ,Jennifer Mschopf Argonne National Laboratory, Institute of Computing Science, Pozna´n University of Technology, Grid Resource Management ,State of the Art and Future Trends

[36]Douglas Thain, Todd Tannenbaum, and Miron Livny, University of Wisconsin-Madison, Condor and the Grid.

[37]M. Surridge “A rough Guide to Grid Security”. Issue 1.1a, IT-Innovation centre, 2002.

[38]Javier Bustos Jimenez, Robin Hood: An Active Objects Load Balancing Mechanism for Intranet.

[39]IBM .Redbooks paper

[40]Javier Bustos Jimenez, Robin Hood: An Active Objects Load Balancing Mechanism for Intranet.

[41]Shahzad Malik, Dynamic Load Balancing in a Network of Workstations, 95.515F Research Report, November 29, 2000.

[42]Menno Dobber, Ger Koole, and Rob van der Mei, Dynamic Load Balancing for a Grid Application, http://www.cs.vu.nl/~amdobber

[43]Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp and Ion Stoica, Load Balancing in Dynamic Structured P2P Systems

[44]Kai Lu , Riky Subrata and Albert Y. Zomaya, Networks & Systems Lab, School of Information Technologies , University of Sydney , An Efficient Load Balancing Algorithm for Heterogeneous Grid Systems considering Desirability of Grid Sites .

[45]Guy Bernard, A Decentralized and Efficient Algorithm for Load Sharing in Networks of Workstations.

[46]Manish Arora and Sajal K. Das and Rupak Biswas, A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environment

[47]Jarek Nabrzyski, Jenniffer M. Schopf and Jan Weglarz, Grid Resource Management: State of the Art and Future Trends

[48] Belabbas Yagoubi and Yahya Slimani, Dynamic Load Balancing Strategy for Grid Computing

[49]Marcin Bienkowski, Miroslaw Korzeniowski, Friedhelm Meyer aud der Heide, Dynamic Load Balancing in Distributed Hash Tables.

[50]Giuseppe Di Fatta and Michael R. Berthold, Department of Computer and Information Science, University of Konstanz, Decentralized Load Balancing for Highly Irregular Search Problems.

[51]Anthony Sulistio, Chee Shin Yeo, and Rajkumar Buyya, Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit.

List of Publications

1 Ratnesh Kumar Nath, Seema Bawa, Inderveer Chana "Load Balancing Issues in Grid Environment", National Seminar on Recent Advancement in Information Technology (RAIT-2007) organized by Indian School of Mines, Dhanbad .

2 Ratnesh Kumar Nath, Seema Bawa, Inderveer Chana "Load Balancing in Grid Environment: Strategies, Algorithms and Policies", National Symposium on Security and Soft Computing (NSSC-2007) organized by National Institute of Technology, Surat.

Introduction to GridSim Appendix A

The GridSim toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers [3]. 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, managing resources such as clusters in a single administrative domain, 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.

System Architecture of GridSim

It employed a layered and modular architecture for Grid simulation to leverage existing technologies and manage them as separate components [51]. A multi-layer architecture and abstraction for the development of GridSim platform and its applications is shown in Figure A-1.

The first layer is concerned with the scalable Java interface and the runtime machinery, called JVM (Java Virtual Machine), whose implementation is available for single and multiprocessor systems including clusters. The second layer is concerned with a basic discrete-event infrastructure built using the interfaces provided by the first layer. One of the popular discrete-event infrastructure implementations available in Java is SimJava. Recently, a distributed implementation of SimJava was also made available. The third layer is concerned with modeling and simulation of core Grid entities such as resources, information services, and so on; application model, uniform access interface, and

primitives application modeling and framework for creating higher level entities. The GridSim toolkit focuses on this layer that simulates system entities using the discrete-event services offered by the lower-level infrastructure. The fourth layer is concerned with the simulation of resource aggregators called Grid resource brokers or schedulers. The final layer is focused on application and resource modeling with different scenarios using the services provided by the two lower-level layers for evaluating scheduling and resource management policies, heuristics, and algorithms. In this section, we briefly discuss the SimJava model for discrete events (a second-layer component) and focus mainly on the GridSim (the third layer) design and implementation. Resource broker simulation and performance evaluation are highlighted in the next two sections.

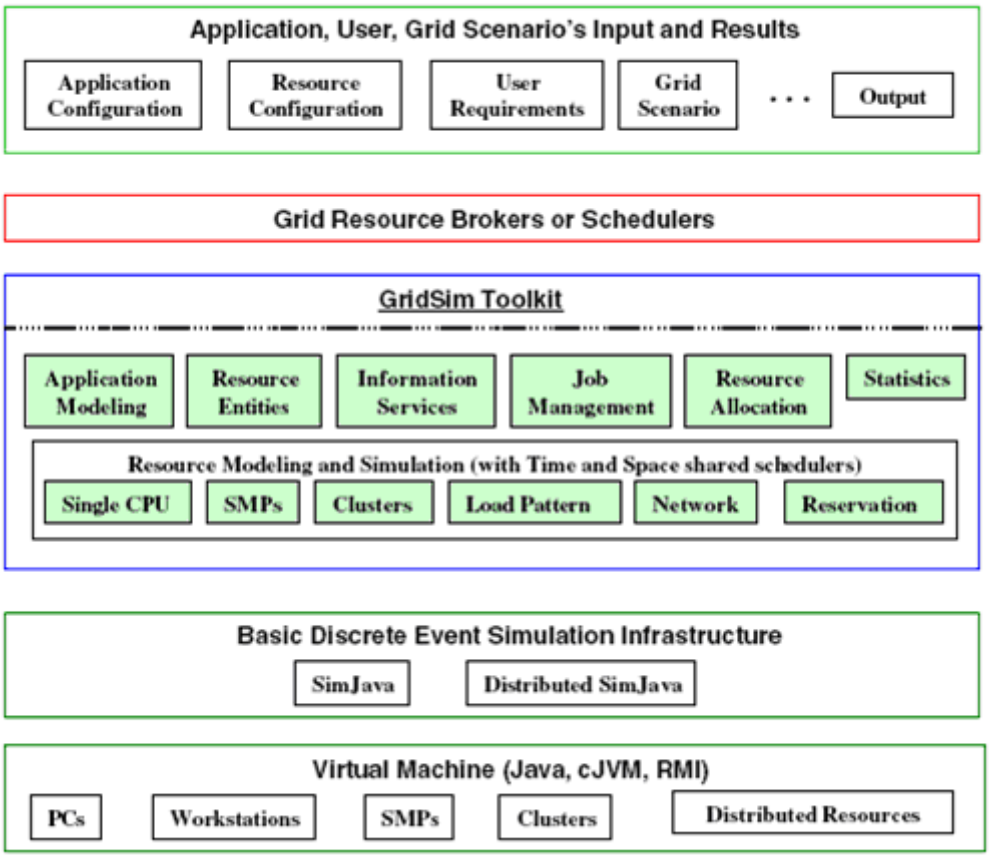


Figure A-1: A Modular Architecture of GridSim Platform and Components [51].

Installation Steps for GridSim

Step 1

The GridSim toolkit software with source code can be downloaded from the project Web site:http://www.buyya.com/gridsim/.

Step 2

To Import GridSim libraries into an Eclipse project. Some instruction will be followed in order to import the GridSim library (gridsim.jar file) in Eclipse project. First, right click on the project name, and select “Properties”.

Step 3

Window will be depicted. Under the “Libraries” tab all the libraries already imported into this project will be displayed. Now import the GridSim library (the filegridsim.jar), just click on “Add external JARs”.

Step 4

After that, you just have to browse to the gridsim.jar file, wherever it is.

Step 5

Click “Ok”, and then the gridsim.jar file will be imported into Eclipse project

Step 6

Now, make application and use GridSim.

Learning GridSim

To understand on how to use GridSim, please go through the examples provided in the $GRIDSIM/examples/ directory. Examples 1 - 6 are mainly for beginners, whereas the rest describes more complex GridSim functionalities.