Tree Based Approach For Load Balancing In Grid Environment

Er. Sahil Verma	Sh. Sandip Kumar Goyal	Er. Kavita
M-Tech (C.S.E.)	Assoc. Professor, Dept. of CSE	M-Tech (C.S.E.)
M.M.University, Mullana(Ambala), Haryana, India.	M.M.University, Mullana (Ambala), Haryana, India.	M.M.University, Mullana(Ambala), Haryana, India.

ABSTRACT

The popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components are changing the way we use computers today. These technical opportunities have led to the possibility of using geographically distributed and multi-owner resources to solve large-scale problems in science, engineering, and commerce. Recent research on these topics has led to the emergence of a new paradigm known as Grid computing.

To achieve the promising potentials of tremendous distributed resources, effective and efficient load balancing algorithms are fundamentally important. Unfortunately, load balancing algorithms in traditional parallel and distributed systems, which usually run on homogeneous and dedicated resources, cannot work well in the new circumstances. In this dissertation, the state of current research on load balancing algorithms for the new generation of computational environments will be surveyed and a new method for tree based approach for load balancing in grid environment is proposed.

KEYWORDS

Table 1: Notations used

Symbol	Definition
Avgload	Average load of grid
Ce	Computing Element
Cnts	Randomly choosen sits
Cno	Cluster Number
Cload	Average load of cluster
Clus	Cluster with less load
Load	Load of site
Noc	Number of computing elements
nos	Number of sites
qlength	Queuelength of computing elements

1. INTRODUCTION

1.1 Grid Computing

Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce. The concept of Grid computing [2] started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent.

The Grid infrastructure [3] can benefit many applications, including collaborative engineering,data exploration, high throughput computing, distributed supercomputing, and service-oriented computing.

The last decade has seen a substantial increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. Nevertheless, there are still problems in the fields of science, engineering, and business, which cannot be effectively dealt with using the current generation of supercomputers. In fact, due to their size and complexity, these problems are often resource (computational and data) intensive and consequently entail the use of a variety of heterogeneous resources that are not available in a single organization. The ubiquity of the Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components is rapidly changing the computing landscape.

These technology opportunities have led to the

possibility of using wide-area distributed computers for solving large-scale problems, leading to what is popularly known as Grid computing. The term Grid is chosen as an analogy to the electric power Grid that provides consistent, pervasive, dependable, transparent access to electricity, irrespective of its source. Such an approach to network computing is known by several names: meta computing, scalable computing, global computing, Internet computing, and more recently Peer-to-Peer (P2P) computing [4] .Moreover, due to the rapid growth of the Internet and Web, there has been a growing interest in Web-based distributed computing, and many projects have been started and aim to exploit the Web as an infrastructure for running distributed and parallel applications. In this context, the Web has the capability to act as a platform for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous Grid-based infrastructure [3]. The user essentially interacts with a resource broker that hides the complexities of Grid computing. The broker discovers resources that the user can access using information services, negotiates for access costs using trading services, maps tasks to resources (scheduling), stages the application and data for processing (deployment), starts job execution, and finally gathers the results.
More importantly both resources and end-users are geographically distributed with different time zones. In managing such complex Grid environments, traditional approaches to resource management that attempt to optimize system-wide measures of performance cannot be employed. This is because traditional approaches use centralized policies that need complete state information and a common fabric management policy, or decentralized consensus based policy. In large-scale Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy. Apart from the centralized approach, two other approaches that are used in distributed resource management are: hierarchical and decentralized scheduling or a combination of them. We note that similar heterogeneity & decentralization complexities exist in human economies where market driven economic models have been used to successfully manage them.

1.2 Load Balancing

A typical distributed system involves a large number of geographically distributed worker nodes which can be interconnected and effectively utilized in order to achieve performances not ordinarily attainable on a single node. Each worker node possesses an initial load, which represents an amount of work to be performed, and may have a different processing capacity.

To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all nodes which are based on their processing capabilities. This is why load balancing is needed. The load balancing problem is closely related to scheduling and resource allocation. It is concerned with all techniques allowing an evenly distribution of the workload among the available resources in a system. The main objective of a load balancing consists primarily to optimize the average response time of applications; this often means the maintenance the workload proportionally equivalent on the whole system resources.

Load balancing is usually described in the literature as either load balancing or load sharing. These terms are often used interchangeably, but can also attract quite distinct definitions.
In the following, we distinguish between three forms of load balancing.
Load Sharing: This is the coarsest form of load distribution. Load may only be placed on idle resources, and can be viewed as a binary problem, where a resource is either idle or busy.
Load Balancing: Where load sharing is the coarsest form of load distribution, load balancing is the finest. Load balancing attempts to ensure that the workload on each resource is within a small degree, or balance criterion, of the workload present on every other resource in the system.
Load Levelling: Load levelling introduces a third category of load balancing to describe the middle ground between the two extremes of load sharing and load balancing. Rather than trying to obtain a strictly even distribution of load across all resources, or simply utilizing idle resources, load levelling seeks to avoid congestion on any resource. A Grid load balancer receives applications from Grid users, selects feasible resources for these applications according to acquired information, and finally generates application-to-resource mappings , on the basis of objective functions and predicted resource performance. Basically , a load balancing system can be generalized into four basic steps:
(1) Monitoring resource load and state.
(2) Exchanging load and state information between resources.
(3) Calculating the new load distribution.
(4) Updating data movement.

1.3 Load Balancing Problem

This problem has been discussed in traditional distributed systems literature for more than two decades. Various strategies and algorithms have been proposed, implemented and classified in a number of studies [5], [6]. Load balancing algorithms can be classified into two categories: static or dynamic. In static algorithms, the decisions related to load balance are made at

compile time when resource requirements are estimated. A multicomputer 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. A good description of customized load balancing strategies for a network of workstations can be found in [7]. More recently, Houle and al. [8] consider algorithms for static load balancing on trees, assuming that the total load is fixed. Contrary to the traditional distributed systems for which a plethora of algorithms have been proposed, few of which were focussed on grid computing. This is due to the innovation and the specific characteristics of this infrastructure.

2. RELATED WORK

2.1 TREE-BASED BALANCING MODEL

In order to well explain as model, we first define the topological structure for a grid computing. Research works on load balancing has been focused on system-level load balancing or task scheduling [9],[10]. Their main objective is to maximize the overall throughput or average response time. Most application-level load balancing approaches are oriented on application partitioning via graph algorithms. However, it does not address the issue of reducing migration cost. That is the cost entailed by load redistribution, which can consume order of magnitude more time than the actual computation of a new decomposition. Some works have proposed a latency-tolerant algorithm that takes advantage of overlapping the internal data computation and incoming data communication to reduce data migration cost. Unfortunately, it requires applications to provide such a parallelism between data processing and migration, which restricts its applicability. Propose a dynamic load balancing approach to provide application level load balancing for individual parallel jobs in Grid computing environment. Agent-based approaches [11] have been tried to provide load balancing in cluster of machines [12]. Enhance the MPI_Scatterv primitive to support master-slave load balancing by taking into consideration the optimization of computation and data distribution using a linear programming algorithm. However, this solution is limited to static load balancing.[13] propose an optimal data migration algorithm in diffusive dynamic load balancing through the calculation of Lagrange multiplier of the Euclidean form of transferred weight. This work can effectively minimize the data movement in homogeneous environments, but it does not consider the network heterogeneity. In particular, workload migration is critical to be considered because the wide area network performance is dynamic, changing throughout execution, instable, etc., in addition to considering the resource heterogeneity. This communication aspect is neglected in traditional application-level load balancing strategies. Scheduling sets of computational tasks on distributed platforms is a key issue but a difficult problem. Although, as mentioned above, a large number of scheduling techniques and heuristics have been presented in the literature, most of them target only homogeneous resources. However, modern computing systems, such as the computational Grid, are most likely to be widely distributed and strongly heterogeneous. Therefore, it is essential to consider the impact of heterogeneity on the design and analysis of scheduling techniques. The traditional objective, when scheduling sets of computational tasks, is to minimize the overall execution time called *makespan*. However, in the context of heterogeneous distributed platforms, makespan minimization problem is in most cases *NP-complete*, sometimes even *APX-complete*. When dealing with large scale systems, an absolute minimization of the total execution time is not the only objective of a load balancing strategy. We think that the communication cost, induced by load redistribution, is also a critical issue. For this purpose, we propose, in this paper, a novel load balancing strategy to address the new challenges in Grid computing.

3. SYSTEM MODEL

3.1 Load balancing Generic model

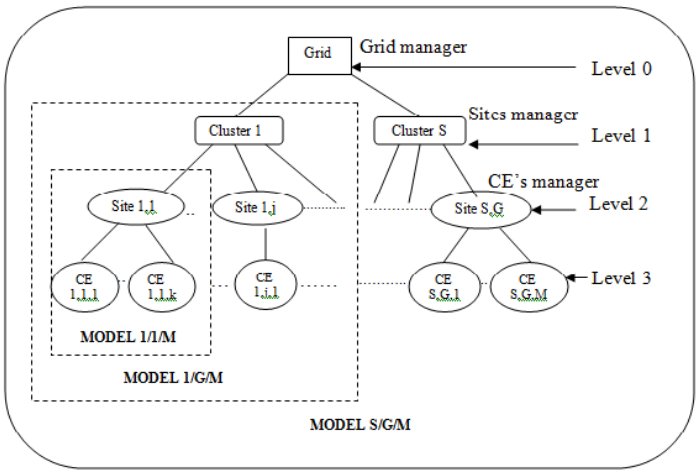

Figure 3.1 Load Balancing Generic Model

Our model is represented by an incremental tree where root of the tree is known as the grid and a software running on the grid is grid manager which is responsible to manage all cluster information of

the grid and provide fault tolerance to the grid. Leaf of the tree are known as computing elements of a site. A grid consists of various clusters and clusters are consists of various sites or we can say that various sites are aggregated to form the cluster and various computing elements are aggregated to form a site. Information about the computing elements status is stored on site. A software running on site is called computing elements manager. Each cluster have information about the load of its sites which are underlying under it. Software running on the cluster is known as the sites manager as it manages the load of the sites which are under the cluster and if any site under it will fail then it distribute its load to the less loaded site of it and prevent from failure of whole cluster due to the failure of site and hence provide fault tolerance to cluster. This model is denoted by S/G/M, where S is the number of clusters that compose a grid, G is the number of sites in each cluster and M is the number of computing elements in each sites. This model can be transformed into three specific model: S/G/M, 1/G/M, 1/1/M, depending on the values of S and G. It represents a four–level tree. Each level has its own specific function whose description is as follows.

- **Level 0:** It is the top level (root) of the tree called grid having grid manager deployed on it. Its main functions are:
  (i) To maintain or manage all clusters workload information of the grid.
  (ii) All decision making regarding the allocation of task for inter cluster load balancing are taken by it.
  (iii) It provide fault tolerance to the grid as if any cluster under it will fail then it prevent from the failure of whole grid due to the failure of cluster by taking the appropriate decision.
- **Level 1:** It contains S virtual nodes. Nodes of this level are known as clusters having sites manager deployed on it. Site manager is responsible to manage workload of sites under the cluster and provide fault tolerance to the cluster in case of failure of site.
- **Level 2:** This is the third level and nodes of this level are sites having computing elements manager deployed on it. Nodes of this level are responsible to provide fault tolerance to the site in case of failure of any computing element of site and it also manages the workload of their computing elements.
- **Level 3:** This is the last level and this is the leaves of the tree. It represents computing elements associated with the various sites. Figure 3.1 shows the load balancing generic model, with its three variants: 1/1/M, 1/G/M, S/G/M.

3.2 Characteristics of the proposed model

The main features of our proposed load balancing generic model are listed below:
1) It is *hierarchical*: This characteristic facilitate the information flow through the tree and well defines the message traffic in our strategy.
2) It supports *scalability* of grids: Adding or removing entities (computing
elements, sites or clusters) are very simple operations in our model
(adding or removing nodes, sub-trees).
3) It is totally *independent* from any physical architecture of a grid: The transformation of a grid into a tree is an univocal transformation. Each grid corresponds to one and only one tree.

3.3 Proposed Algorithm

Step 1: Cluster Creation algorithm:

1. Initialize cno, site, ce, qlength, nos.
2. For cno=1 to 10.
i) Generate number of sites between 1 to 5 randomly.
ii) Generate computing element ce of each site between 1 to 5 randomly.
iii) Generate queue length of each computing element ce between 1 to 50 randomly.
end For.

Step 2: Load Calculation for 10 cluster's sites algorithm:

1. Initialize site, cno.
2. For cno=1 to 10.
Call calculateLoad(cno,site).
end For
calculateLoad(cno,site)
1. Initialize avgload=0, load=0.
2. Calculate avgload of site s and cluster cno into variable load.
3. Calculate avgload of grid into variable avgload.
4. Calculate newload=Math.abs((load-avgload)/load.
5. If(newload<=threshold)
then no need to balance.
else
load balancing is required.
end If

Step 3: Intrasite load balancing algorithm:

1. Initialize cno, cnts, ce.
2. Input cluster number into cno.

3. Choose site cnts and computing element ce of cluster number cno randomly.
4. Call intraSite(cno,cnts,ce).
5. If site cnts of cluster number cno is overloaded then
Call intraSite(cno,cnts,ce).
else
No need to balance the site.
end If

intrasite(cno,cnts,ce)

1. Initialize noc=0, load=0, avgload=0, share=0, cntce=0.
2. Calculate number of computing element noc in cluster cno.
3. If noc==1 then
Intra site load balancing is not possible.
Else
- Choose new Computing element newce of site cnts of cluster cno randomly.
- Calculate load of Computing element ce into load and average of grid into avgload.
- Calculate share=Math.abs(avgload-load).
- Add share to the load of new Computing element newce of site cnts of cluster cno.
- Subtract share from the load of Computing element ce of site cnts of cluster cno.
end If

Step 4: Intersite load balancing algorithm:

1. Initialize cno, cnts.
2. Input cluster number into cno.
3. Choose site cnts of cluster number cno randomly.
4. Calculate no of computing element into variable noc.
5. Change or update qlength of computing element ce in site cnts in cluster number cno b/w 40 to 80.
6. Call calculateLoad(cno,cnts).
7. After increasing load if site become overloaded
- Call InterSite(cno,cnts).
- After 20 attempt if the site is overloaded then
- Call intercluster(cno).
else
No need of balancing.
end If.

InterSite(cno,cnts)

1. Initialize nos=0, nos1=0, load=0, avgload=0, share=0, cntce=0.
2. Calculate no of sites nos in cluster number cno.
3. If(nos==1) then
Number of site is 1 , intersite load balancing is not possible.
Else
- Calculate newsite=generateRandomSite (nos1,site).\\ For balancing it randomly chosen site is newsite.
- Calculate load of site into load and average of grid into avgload.
- Calculate share=Math.abs(avgload-load).
- Add share to the qlength q of computing element ce of new site newsite of cluster cno.
- Subtract share from the load of site site of cluster cno.
- Call calculateLoad(cno,newsite).
- Call calculateLoad(cno,site).
end If.

Step 5: Intercluster load balancing algorithm:

1. Initialize cno, lo.
2. Input cluster number into cno.
3. For each site s in cluster cno
- Increase load or qlength of each computing element of site s
- Call calculateload(cno,s).
end For
4. calculate lo=checkLoad(cno).
5. If(lo==1)
Call intercluster(cno).
end If

intercluster(cno)

1. Initialize cload, clus, nos, nloads, noc.
2. Calculate avgload of cluster cno into variable cload.
3. Choose cluster with less load into variable clus and its load into
variable load.
4. Calculate share=Math.abs(avg-load).
5. Choose site s1 of cluster clus randomly.
6. Choose computing element into variable ce1 of site s1 randomly.
7. Update qlength of computing element ce1 of site s1 of cluster clus
- qlength=qlength+share.
- calculateload(clus,s1).
8. Choose site s2 of overloaded cluster cno.
9. Choose computing element ce2 of site s2 of cluster cno.
10. Update qlength of computing element ce2 of site s2 of cluster cno
qlength=qlength-share.
11. Call calculateload(cno,s2).

checkLoad(cno)

1. Initialize chk=0, avgload=0, lo=0.
2. Calculate load and avgload of cluster cno.
3.Calculate newload=Math.abs((chk-avgload) /chk).
4. If(newload<=threshold) then
No need to balance.
lo=0.
else
lo=1.
load balancing is required.
end If.

Step 6: Display cluster algorithm:

1. Initialize cno.
2. Input cluster number into cno.
3. Call display(cno).
4. Call displayLoad(cno).

display(cno)

1. Display sites of each cluster and computing element in each sites with its queue length.

displayLoad(cno)

1. Calculate load of site1 in cluster number cno.
2. If(loaded.equals("1")) then
load balancing is required.
else
No need to balance.
end If.

4. CONCLUSION AND FUTURE WORK

In this dissertation, simulator of Tree Based Approach for Load Balancing in Grid Environment is build. In this, 10 clusters are created. Each Cluster has 1 to 5 sites and each site have 1 to 5 computing elements. For intra-site load balancing we have queue length of computing elements from 10 to 40. For inter-site load balancing queue length of computing element is increased from 40 to 80 and then intersite load balancing is done. For intercluster load of cluster is increased by assigning queue length of all computing elements from 40 to 80 and intercluster load balancing is done by choosing the cluster which have less load. In this way, we have implemented various scenarios showing the intrasite, inter-site and intercluster load balancing in grid environment.

Future work may include following points:

i) Resources can be taken as heterogeneous.
ii) Random arrival of jobs can be considered.

5. REFERENCES

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

[2] M. Baker, R.Buyya, and D. Laforenza, "Grids and grid technologies
for wide-area distributed computing," International Journal of Software: Practice and Experience (SPE), 2002.

[3] F. Berman, G. Fox, and Y. Hey, "Grid Computing: Making the Global Infrastructure a Reality," Wiley Series in Communications Networking & Distributed Systems, 2003.

[4] C. Chen, K.C. Tsai., "The server reassignment problem for load balancing in structured P2P systems," IEEE Transaction on Parallel Distributed Systems, 19(2) : 234–246, 2008.

[5] W. Leinberger, G. Karypis, V. Kumar, and R. Biswas, "Load balancing across near-homogeneous multi-resource servers," In 9th Heterogeneous Computing Workshop, pp 60–71, 2000.

[6] C.Z. Xu and F.C.M. Lau, "Load Balancing in Parallel Computers: Theory and Practice," Kluwer, Boston, MA, 1997.

[7] M.J. Zaki, W. Li, and S. Parthasarathy, " Customized dynamic load balancing for a network of workstations," In Proceeding of the 5th IEEE International Symposium HDPC, pp 282–291, 1996.

[8] M. Houle, A. Symnovis, and D. Wood, "Dimension-exchange algorithms for load balancing on trees," In Proceedings of 9th International Colloquium on Structural Information and Communication Complexity, pp 181–196, Andros, Greece, June 2002.

[9] B. Yagoubi, and M. Medebber, "A load balancing model for grid environment," In Proceeding of 22nd International Symposium on Computer and Information Sciences (ISCISC 2007), pp. 1-7, 7 November 2007.

[10] B.Yagoubi, "Dynamic load balancing for beowulf clusters,"In Proceeding of the 2005 International Arab Conference On information Technology,pp 394–401, Israa University, Jordan, December 2005.

[11] J. Cao, D.P. Spooner, S. A. Jarvi, and G.R. Nudd, "Grid load balancing using intelligent agents," Future Generation Computer Systems, 135-149, January 2005.

[12] Rodrigo Fernandes de Mello, and Luciano Jos´e Senger, " A Routing Load Balancing Policy for Grid Computing Environments," In Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06), IEEE, 2006.

[13] H. Shan, L. Oliker, R. Biswas, and W. Smith, "Scheduling in heterogeneous grid environments: The effects of data migration," In Proceedings of (ADCOM2004) International Conference on Advanced Computing and Communication, India, December 2004.

[14] M. Dobber, R. Mei, and G. Koole, " Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison," IEEE Transaction on Parallel and Distributed Systems, 20(2): 207- 218, February 2009.