A Fault Tolerant Approach For Load Balancing In Grid Environment

Er. Kavita	Sh. Sandip Kumar Goyal	Er. Sahil Verma
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 a fault tolerant approach for load balancing in grid environment is proposed.

KEYWORDS

Table 1: Notations used

Symbol	Definition
Avgload	Load of site
Ce	Computing elements
CESO	Overloaded computing elements set
CERU	Underloaded computing elements Set
CENB	Balanced computing elements set
CEijk	ith CE of jth site of kth Cluster
CEsjk	Computing element most overloaded
CErjk	Computing element most lightly overloaded
cntce	Number of computing elements in site cnts
cnts	Randomly chosen site
cno	Cluster number
clus	Cluster whose load is less
cload	Average load of cluster
eload	Extra load
Lijk	Actual Workload of Ceijk
noc	Number of computing elements
nos	Number of sites
nloadc	Load to be shared for computing element
nloads	Load to be shared for each site in Inter-cluster load balancing
qlen	Queue length of computing element
qlength	Queue length of computing elements
share	Load taken by the underloaded site
site 1	Total number of sites in cluster
Sjk	jth Site of kth Cluster

1. INTRODUCTION

1.1 Grid Computing

"A Grid is a collection of distributed computing resources available over a local or wide area network that appears to an end user or application as one large virtual computing system."-IBM

Grid computing [3] can mean different things to different individuals. The grand vision is often presented as an analogy to power grids where users (or electrical appliances) get access to electricity through wall sockets with no care or consideration for where or how the electricity is actually generated. In this view of grid computing, individual users (or client applications) gain access to computing resources (processors, storage, data, applications, and so on) as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on are [4].

1.2 Data Grids

A Data Grid provides services [2] that help users discover, transfer and manipulate large datasets stored in distributed repositories and also, create and manage copies of these datasets. At the minimum, a Data Grid provides two basic functionalities: a high-performance, reliable data transfer mechanism, and a scalable replica discovery and management mechanism.

1.3 Fault Tolerance

A characteristic feature of distributed systems that distinguishes them from single-machine systems is the notion of partial failure. A partial failure may happen when one component in a distributed system fails. This failure may affect the proper operation of other components, while at the same time leaving yet other components totally unaffected. In contrast, a failure in non-distributed systems is often total in the sense that it affects all components, and may easily bring down the entire system. An important goal in distributed systems design is to construct the system in such a way that it can automatically recover from partial failures without seriously affecting the overall performance. In particular, whenever a failure occurs, the distributed system should continue to operate in an acceptable way while repairs are being made, that is, it should tolerate faults and continue to operate to some extent even in their presence. The key technique for handling failures is redundancy. A system is said to fail when it cannot meet its promises. Being fault tolerant is strongly related to what are called dependable systems. Dependability is a term that covers a number of Useful requirements for distributed systems including the following:

1. Availability
2. Reliability
3. Safety
4. Maintainability

1.4 Fault and Its Types

The cause of an error is called a fault. Clearly, finding out what caused an error is important. For example, a wrong or bad transmission medium may easily cause packets to be damaged. In this case it is relatively easy to remove the fault. However, transmission errors may also be caused by bad weather conditions such as in wireless networks.

Building dependable systems closely relates to controlling faults. A distinction can be made between preventing, removing, and forecasting faults. For our purposes, the most important issue is fault tolerance, meaning that a system can provide its services even in the presence of faults. In other words, the system can tolerate faults and continue to operate normally.

Faults are generally classified as transient, intermittent, or permanent.

- **Transient faults** occur once and then disappear. If the operation is repeated, the fault goes away. A bird flying through the beam of a microwave transmitter may cause lost bits on some network (not to mention a roasted bird). If the transmission times out and is retried, it will probably work the second time.
- **Intermittent fault** occurs, then vanishes of its own accord, then reappears, and so on. A loose contact on a connector will often cause an intermittent fault. Intermittent faults cause a great deal of aggravation because they are difficult to diagnose. Typically, when the fault doctor shows up, the system works fine.
- **Permanent fault** is one that continues to exist until the faulty component is replaced. Burnt-out chips, software bugs, and disk head crashes are examples of permanent faults.

2 LITERATURE SURVEY

2.1 Introduction

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 software environments, workloads have to be evenly scheduled among the available resources. To realize this goal several load balancing strategies and algorithms have been proposed. Most strategies were developed in mind, assuming homogeneous set of sites linked with homogeneous and fast networks. However for computational grids we must address main new issues, namely: *heterogeneity*, *scalability* and *adaptability*.

The development of computational grids and the associated middleware has been actively pursued in recent

years to deal with the emergence of greedy applications of large computing tasks and amounts of data. Managing such applications leads to some complex problems for which traditional architectures are insufficient. There are many potential advantages of using grid architectures, including the ability to solve large scale advanced scientific and engineering applications whose computational requirements exceed local resources, and the reduction of job turnaround time through workload balancing across multiple computing facilities. In his reference book, Foster defined a computational grid as an emerging computing infrastructure that enables effective access to high performance computing resources. An important issue of such systems is the efficient assignment of tasks and utilization of resources, commonly referred to as load balancing problem. We seek to achieve load balancing that privilege workload neighborhood, to reduce amount of messages. Our strategy deals with a three layers algorithms (intra-site, intra-cluster and intra-grid).

2.2 Motivation for Load Balancing

Load balancing algorithms [1] are essentially designed to spread the resources load equally thus maximizing their utilization while minimizing the total task execution time. This is crucial in a computational grid where the most pressing issue is to fairly assign jobs to resources. Thus, the difference between the heaviest and the lightest resource load is minimized. A flexible load sharing algorithm is required to be general, adaptable, stable, scalable, fault tolerant, transparent to the application and to also induce minimum overhead to the system. The properties listed above are interdependent.

For example, a lengthy delay in processing and communication can affect the algorithm overhead significantly, result in instability and indicate that the algorithm is not scalable. The load balancing process can be defined in three rules: the location, distribution and selection rule.

-The **location rule** determines which resource domain will be included in the balancing operation. The domain may be local, i.e. inside the node, or global, i.e. between different nodes.

-The **distribution rule** establishes the redistribution of the workload among available resources in the domain,

-The **selection rule** decides whether the load balancing operation can be performed preemptively or not.

2.3 Typically, a load balancing scheme consists of four policies:

1) The ***information policy*** is responsible to define when and how the information on the Grid resources availability is updated.

2) The ***location policy*** determines a suitable transfer partner (server or receiver) once the transference policy decided that this resource is server or receiver.

3) The ***selection policy*** defines the task that should be transferred from the busiest resource to the idlest one.

4) The ***transference policy*** classifies a resource as server or receiver of tasks according to its availability status.

2.4 Load Balancing Strategy

In accordance with the structure of proposed model, the load balancing strategy [1] is also hierarchical. Hence, we distinguish between three load balancing levels:

1) ***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.

2) ***Intra-cluster load balancing*****:** In this second level, load balance concerns only one cluster, among the clusters of a grid. This kind of load balance is achieved only if some sites fail to load balance its workload among their respective computing elements. In this case, they need the assistance of its direct parent, namely cluster and balance load to the less loaded site within the cluster.

3) ***Intra-grid load balancing*****:** The load balance at this level is used only if some clusters 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)

.

3 SYSTEM MODEL

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

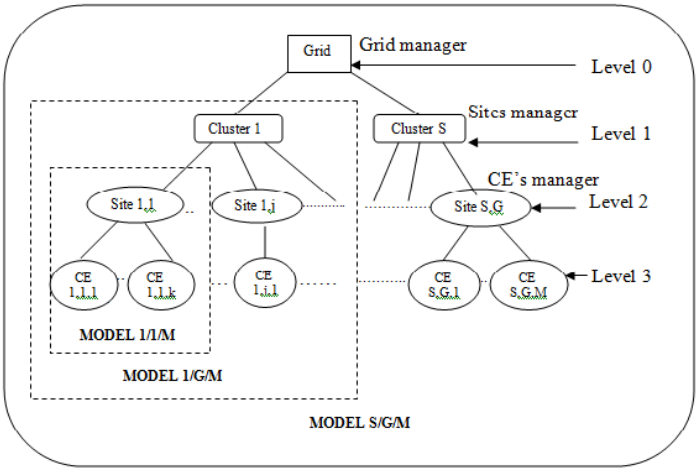

Figure 3.1 Load Balancing Generic Model

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 ***heterogeneity*** and ***scalability*** of grids: Adding or removing entities (computing elements, sites or clusters) are very simple operations in our model (adding or removing nodes, subtrees).
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 no of sites between 1 to 5 randomly.
ii) Generate computing element ce of each site between 1 to 5 randomly.
iii)Generate qlength 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: 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

Step 4: Balancing Algorithm:

1. Initialize cno, site1.
2. Input cluster number into cno.
3. For each site site1 in cluster cno
Call balance(cno,site1).
end For
balance(cno,site1)
1. Initialize avg=0, k=0, l=0, m=0, n=0, qlen.
2. Take 4 strings CESO, CERU, CENB, ce.
3. For i =1 to site.
4. Initialize k=0, l=0, CESO="", CERU="", CENB="".
5. For Every CEijk of Sjk do
6. Switch
- Lijk > Avrg + T
CESO <- CESO U {CEijk}
- Lijk < Avrg
CERU <- CERU U {CEijk}
- Avrg = Lijk = Avrg + T
CENB <- CENB U {CEijk}
end Switch
end For
7. While (CESO !="" AND CERU !="") do
- Sort CESO by descending order of Lijk
- Sort CERU by ascending order of Lijk
- CEsjk <- CE most overloaded
- CErjk <- CE most lightly overloaded
- Load offered by CErjk = Avrg - Lrjk
- Tasks migration stage from CEsjk to CErjk
- Update current workload of CEsjk, CErjk.

Step 5: Failure of Computing Element algorithm:

1. Initialize cno, cnts, qlen, uqlen.
2. Input cluster number into cno.
3. Choose site cnts from cluster number cno randomly.
4. Calculate number of computing element into variable cntce in randomly choosen site cnts of cluster number cno.
5. If(cntce>1) then
- Choose computing element ce in site cnts in cluster number cno randomly to fail.
- Calculate load of computing element ce of site cnts of cluster cno into variable qlen.
- Calculate share uqlen=qlen/(cntce-1).
- For each computing element ce in site cnts of cluster cno.
i) update qlenth=qlength+uqlen.
ii) calculateLoad(cno,cnts).
end For
- After calculating load if site become overloaded then
Call intersite(cno,cnts).
else
- intersite load balancing is required.
Call intersiteLB(cno,cnts).
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 cno.
3. If(nos==1) then
- Number of site is 1 , inter site load balancing is not possible.
else
- newsite = generateRandomSite(nos1,site).\\ For balancing it randomly choosen 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
intersiteLB(cno,cnts)
1. Initialize load=0, i=0, eload=0, noc=0, qlen=0, maxc=0, newload=0.
2. Calculate load of site cnts of cluster cno into variable load.
3. Delete or fail site cnts of cluster cno.
4. Choose site with less load into variable s of cluster cno.
5. Calculate number of computing element noc of (site with less load) s in cluster cno.
6. Calculate newload=load/noc.
7. Calculate extra load eload.
8. eload=load-(newload*noc).
9. For each computing element ce in site s of cluster cno
- Update qlength=qlength+newload.
End For
10. If(eload>0) then
- Choose computing element el in site s in cluster number cno randomly.
- Update qlength of computing element e1.
qlength = qlength+eload.
end If

Step 6: Failure of Site algorithm:

1. Initialize cno, cnts.
2. Input cluster number into cno.
3. Choose site cnts from cluster number cno randomly.
4. Calculate no of site into variable site1 of cluster number cno.
5. If(site1>1) then
- Choose site into variable cnts of cluster cno randomly.
- Call intersiteLB(cno,cnts)
else
- Call interCluster(cno)
end If

Step 7: Failure of Cluster algorithm:

1. Initialize cno.
2. Choose cluster randomly into variable cno.
3. Call interCluster(cno).

interCluster(cno)

1. Initialize cload, clus, nos, nloads, noc.
2. Calculate avgload of cluster cno into variable cload.
3. Delete or fail cluster number cno.
4. Choose cluster with less load into variable clus.
5. Choose number of site into variable nos of cluster clus.
6. Calculate share for each site into variable nloads.

nloads= cload/nos.

7. For each site in cluster clus.

i) Calculate number of computing element into variable noc.

ii) Calculate share for computing element ce.

nloadc = nloads/noc.

iii) Update qlength of each computing element

qlength=qlength+nloadc.

iv) Call calculateLoad(clus,site).

end For.

4 CONCLUSION AND FUTURE WORK

In this dissertation, simulator of Fault Tolerant Approach for Load Balancing in Grid Environment is build. In this 10 clusters are created. Each cluster have 1 to 5 sites and each sites have 1 to 5 computing elements and queue length of each computing element is from 1 to 50. When a computing element is failed then the load of the failed computing element is distributed to the other computing elements within the site equally. And after within site load balancing if site is overloaded then load is distributed to the randomly selected site of the cluster. When a site is failed then load of failed site is equally distributed among the computing elements of the randomly selected site within the cluster and when a cluster is failed then load of failed cluster is distributed to the less loaded cluster of the grid and in this way load balancing is done and fault tolerance is achieved.

In this way we have implemented various scenarios showing the intra-site, inter-site and inter-cluster 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.

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] M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide area distributed computing," International Journal of Software: Practice and Experience (SPE), vol. 32(15), 2002.

[5]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.

[6] 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.

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

[8] K. Lu, and A. Zomaya, "A Hybrid Policy for Job Scheduling and Load Balancing in Heterogeneous Computational Grids," Proceeding of 6th International Symposium on Parallel and Distributed Computing, pp.19-26, 5 July 2007.

[9] P. K. Suri, and Manpreet Singh, "An Efficient Decentralized Load Balancing Algorithm For Grid," IEEE 2nd International Advance Computing Conference, pp. 10-13, February 2010.

[10] Bin Lu, and Hongbin Zhang, "Grid Load Balancing Scheduling Algorithm Based on Statistics Thinking," IEEE 9th International Conference, pp. 288-292, 2008.

[11] Gabor Vincze, Zoltan Novak, Zoltan Pap, and Rolland Vida, "RESERV: A Distributed, Load Balanced Information System for Grid Applications," 8th IEEE International Symposium on Cluster Computing and the Grid, pp. 596-601, 2008.

[12] Han Zhao, Xinxin Liu, and Xiaolin Li, "DLBEM: Dynamic Load Balancing Using Expectation-Maximization," IEEE, August 2008.