Performance analysis of IO Intensive task allocation strategies in heterogeneous web servers

A

Thesis Report

Submitted in

Partial fulfillment of the requirement for the degree of

Bachelor of Technology in Computer Science

By

D.Ansuman Acharya and Swapnil Kothe

Department of Computer Science Engineering
NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA
ROURKELA-769008 (ORISSA)

May 2009

National Institute of Technology Rourkela
Rourkela-769008 (Orissa)

Certificate

This is to certify that the work in this Thesis Report entitled "***Performance analysis of IO Intensive task allocation strategies in heterogeneous web servers***" by D.Ansuman Acharya and Swapnil Kothe has been carried out under my supervision in partial fulfillment of the requirements for the degree of ***Bachelor of Technology*** in Computer Science during session 2005-2009 in the Department of Computer Science and Engineering, National Institute of Technology Rourkela, and this work has not been submitted elsewhere for a degree.

Place: Rourkela
Date: May 11, 2009

(Bibhudatta Sahoo)
Sr. Lecturer
Department of CSE

Acknowledgements

No thesis is created entirely by an individual or a group of individuals for that matter, and this is no exception. Many people have helped to create this thesis and each of their contribution has been valuable. We express our sincerest gratitude and heartfelt thanks to our project supervisor, *Bibhu D. Sahoo*, Sr. *Lecturer, CSE,* for his kind and able guidance for the completion of the thesis work. His consistent support and intellectual guidance pushed us to the limits of our capabilities and inspired to us to work towards the completion of this project. The work of other researchers that we have referred to and modified in certain ways has proved to be of immense help too.

We are grateful to *Dr. B.Majhi*, *Professor and Head, CSE* for his excellent support during my work. Thank are due to all my classmates for their love and support in hard times.

Last, but not least we would like to thank all professors and lecturers, and members of the department of Computer Science, Engineering, N.I.T. Rourkela for their generous help in various ways for the completion of this thesis.

(D.Ansuman Acharya)

10506064

(Swapnil Kothe)

10506004

B.Tech.(Comp. Sc.), 2005-2009

Abstract

The current rate of growth of the World Wide Web has led to an explosion in internet traffic for many popular websites. To overcome the problem of falling quality of service for its customers an efficient approach would be to use a heterogeneous cluster of nodes which replicate the entire site data. In a centralized system, a master node would load balance the user requests and allocate them to the appropriate node. A web application which mainly provides file sharing services to its users offers a system where the tasks are basically of retrieval based nature and hence more IO intensive. In order to address the allocation problem of these tasks, several IO aware policies have been designed and compared with respect to certain standard performance metrics. The study shows that considering the IO nature of tasks yields significantly better results than other existing algorithms.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 The Need for Load balancing in a web application

Internet has experienced a near exponential growth in user base, infrastructure, content size and resources like low-latency, high throughput network links. This explosive increase means that high traffic sites offering e-commerce, community and other resource intensive services like a file sharing web site face an enormous challenge when it comes to ensuring high availability and fault tolerance for their services.

This problem of congestion and slow user-request processing speeds due to heavy loads can be solved in various ways. The most obvious solution would seem the use of a single large powerful Server. However, this solution soon fails because of the gargantuan extent of this web traffic. The next approach could be replicating the server information over many geographically separated independent servers, called as 'mirrored-server' architecture. This approach provides us with a list of independent URL sites that have to be manually selected by the user. Although this should and does solve the congestion problem but with a number of disadvantages, including not user-transparent architecture, lack of control on the request distribution by the Web-server system and a huge loss in terms of resources. The next solution, a promising and efficient approach, is the development of a distributed architecture where the user-requests can be routed among several server nodes in a user-transparent way.

It is in this regard that a technique called **load balancing** that aims to spread work between two or more computers or web servers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, and minimize

response time comes into picture. Using multiple components with load balancing, instead of a single component, may also increase reliability through redundancy. The balancing service is usually provided by a dedicated program or hardware device (such as a dispatcher or switch). We have used a centralized system which can broadcast mechanism to handle load distribution in a server farm.

We have extensively surveyed the current state of art in this area. A web server load balancer coordinates the allocation of several information retrieval requests for a distributed web based application within a set of homogenous web servers that host the application. It helps to select the best web server for servicing the request and tries to balance their overall utilization. A dispatcher based model has been proposed in the paper which routes requests to the appropriate web server based on an IO workload policy. The reason behind choosing such a policy is that in the type of web application or web site that we have considered tasks of a retrieval nature are dominant.

A few algorithms that implement the policies have been simulated and compared with respect to their performance across of a range of performance parameters and results interpreted which show that considering IO load for tasks in a file sharing web application based on a network of heterogeneous web servers leads to a much better performance than general task allocation policies.

1.2 Related Work

The issue of task allocation by load balancing for CPU and memory resources has been extensively studied and reported in the literature in recent years. Harchol-Balter et al. [8]

proposed a CPU-based preemptive migration policy that was more effective than non-preemptive migration policies. Zhang et al. [9] focused on load sharing policies that consider both CPU and memory services among the nodes. The experimental results show that their policies not only improve performance of memory-intensive jobs, but also maintain the same load sharing quality of the CPU-based policies for CPU intensive jobs.

A large body of work can be found in the literature that addresses the issue of balancing the load of disk I/O Lee et al. [10] proposed two file assignment algorithms that balance the load across all disks. The I/O load balancing policies in these studies have been shown to be effective in improving overall system performance by fully utilizing the available hard drives. Zhang et al. proposed three I/O-aware scheduling schemes that are aware of the job's spatial preferences [11]. While the above approaches address the issue of load balancing for explicit I/O load, our technique tackles the problem by considering both explicit I/O invoked by application programs and implicit I/O induced by page faults.

Cho et al. [12] have developed heuristics to choose the number of I/O servers and place them on physical processors. We have studied dynamic scheduling algorithms to improve the read and write performance of a parallel system by balancing the global workload. The above techniques can improve system performance by fully utilizing the available hard drives. However, these approaches become less effective under a complex workload where I/O-intensive tasks share resources with many memory- and CPU-intensive tasks. We have not considered the effect of memory and CPU in this paper but only focused on the IO load on servers due to the tasks. For simplicity, we have not considered the sharing of resources between tasks and neglected communication overhead during the allocation of jobs in the network.

1.3 Problem Formulation

A file sharing web application needs to store huge amounts of data. It also needs to service many simultaneous requests, download or upload from several users who are connected to it from across the globe. In order to maintain the service level agreement for the web application, we must use multiple servers for the same web application. Using multiple servers which have the entire data or parts of it replicated not only improves the response time for the user requests but also helps to enhance reliability and fault tolerance [13]. The routing of requests to the appropriate server in a balanced way presents a major challenge to most web architects. Usually either a centralized or distributed scheme is followed. In this study, we have accepted a centralized scheme with a master server whose only responsibility is to allocate tasks or requests to the different web servers. An advantage of this kind of architecture is the relatively lesser communication overheads over any kind of distributed scheme A major drawback of this scheme is however, the bottleneck presented by the master server upon increase in traffic and the chance of single point failure.

The popularity of the web application may increase by a large amount over the years. In order to accommodate the growing needs of its customers and to maintain its quality of service, the design must be scalable. Change also implies the introduction of newer hardware of configurations different from the original setup thus favoring a heterogeneous model of the system which we have assumed in our study.

Allocation of each user request or task to a server presents a decision for the master server which is a NP complete problem [14]. In this kind of web application most of the tasks are of an IO intensive nature because the uploading and downloading of data items mainly involves disk access from the server.

The problem basically is to distribute the load consisting of several tasks amongst the servers in a balanced manner and try to minimize the total time in which a set of retrieval requests for a set of users can be allocated and serviced.

1.4 Approaches of Solving the Problem

There are several different ways of load balancing a set of tasks on a heterogeneous network of servers. The existing techniques are concerned with the effective usage of CPU and memory resources. Due to imbalance of disk IO under IO intensive workloads, the previous memory or CPU aware algorithms suffer a significant performance drop. The use of IO loads information while load balancing thus proves to be a remedy to this deficiency. In this study, we have proposed a few variants of existing and well proven algorithms which aim at maintaining high range of resource utilization on the web servers under a wide range of workload conditions. Using memory, CPU and IO loads at once as per the varying request traffic in a system is a highly innovative approach which can also be followed for solving this problem. There also exist methods which consider the communication overheads and task breakdown and simultaneous execution.

The rest of the thesis focuses on solving this problem by dominantly using IO aware policies and considering IO centric work loads without network communication overheads or task dependence due to resource sharing.

1.5 Outline of the Thesis

The thesis is divided into five chapters. In this chapter which is the first, we have provided the reader with a brief overview and idea about the need of load balancing in a modern day web application and the problem, whose solution has been explored in this work. The second

chapter deals with the system model and architecture in detail and also describes the IO aware policy along with the performance metrics that are used to evaluate the algorithms that implement various flavors of the policy. The third chapter discusses the many algorithms in detail and their steps. The fourth chapter describes the framework used for simulation and compares the various algorithms with respect to the performance metrics. The fifth chapter concludes the entire thesis and the sixth proposes certain enhancements to the work done in newer viable directions.

1.6 Conclusion

There are several policies for allocation of tasks in a heterogeneous web server based system. The choice of a centralized or a distributed architecture is currently a topic of great research. We have considered a centralized approach and taken advantage of the IO intensive nature of tasks in our system to propose variations of certain known algorithms and compared them with respect to certain performance metrics like standard deviation of utilization and total makespan, in the process observing the results and providing suitable interpretations and conclusions on basis of these results.

Chapter 2 System Model

In this chapter, we consider a model of our overall system and the several policies addressed therein. The web application which we have modeled through our system is basically a file sharing website that lets its users upload and download files. Now as the number of users who access the website increases, the data hosted on the site must be replicated in order to service these requests with adherence to the service level agreements.

2.1 System Architecture

In this study, we consider a collection of nodes connected by a high-speed LAN network in a star topology as shown in Figure 2.1. The file sharing application essentially runs on the Master Server but the data required by it are present on the servers in a completely redundant form. Tasks arrive at the master load balancer server which allocates these requests to one of the many web servers on basis of the task allocation policy in use. Each node maintains its individual task queue where newly arrived tasks are stored before the retrieval involved in each task starts. It is implemented as a centralized system although a distributed solution could be also possible.

The main advantage of centralized systems is that they are simple to implement, and the search mechanism is fast and efficient. Yet, they have the same disadvantage of any centralized system: they have a single point of failure, and so are vulnerable to attacks to the server, censorship, technical failures, etc. Furthermore, these solutions are inherently non-scalable, and limited by the capacity of the central server. The system described above is a heterogeneous system and since the nature of our tasks is predominantly of IO intensive kind, we chose to keep the

heterogeneity limited to the difference in disk speeds. The heterogeneity of each web server can be characterized by its CPU speed, memory capacity and disk performance [1]. We characterize each node i by its CPU speed C_i, memory capacity M_i, and disk performance D_i. The disk performance can be measured as

$$\mathbf{D_i = 1 / (S_i + R_i + d / B_i^{disk})} \quad (2.1)$$

where d is the average size of data stored or retrieved by I/O requests, B_i^{disk}, S_i, and R_i denote the disk bandwidth, average seek time, and average rotation time of the disk in node i.

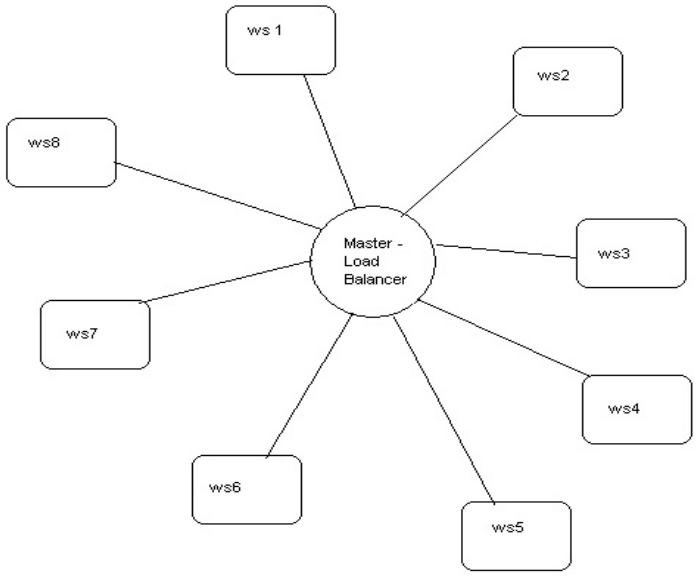

Figure 2.1 – Topology of Web servers in a centralized load balancing for a file sharing application

A measure of the heterogeneity of the system is provided by H_d which is known as the disk heterogeneity factor. For every disk there is a parameter called as W_{disk} which is the ratio between its performance and the fastest disk in the cluster. Thus we have

$$W_i^{disk} = D_i \; / \; (\max_{j=1-n}(D_j)) \quad (2.2)$$

And then, we have

$$H_D = \sqrt{\frac{\sum_{i=1}^{n} (W_{avg}^{disk} - W_i^{disk})^2}{n}} \tag{2.3}$$

Now we consider the nature of a task. The task which is actually a request for resource retrieval is modeled on basis of its three components. Each task consists basically of three parts.

$$\mathbf{T(j) = C(j) + IO(j)+M(j)} \tag{2.4}$$

Where T(j) denotes the strength of the j^{th} task, C(j) denotes the CPU load due to it or the time it spends executing in the CPU, M(j) denotes the memory requirements of the task or the amount of primary memory it will be using while it is in execution and IO(j) be the explicit IO requirements or the amount of data retrieval it has to perform for the task [1]. The tasks arrive in a Poisson process with arrival rate of λ at the central load balancer from several users [2] as shown in Figure 2.2.

2.2 Task Specifications

Each task is of uploading or downloading or a transaction based nature. Most of the tasks in our system are usually downloads. Most users look forward to downloading when they are searching for some kind of file. The upload usually takes place only once compared to the several times a file is downloaded by multiple users. So we consider only download based tasks that involve retrieval of a file from one of the servers.

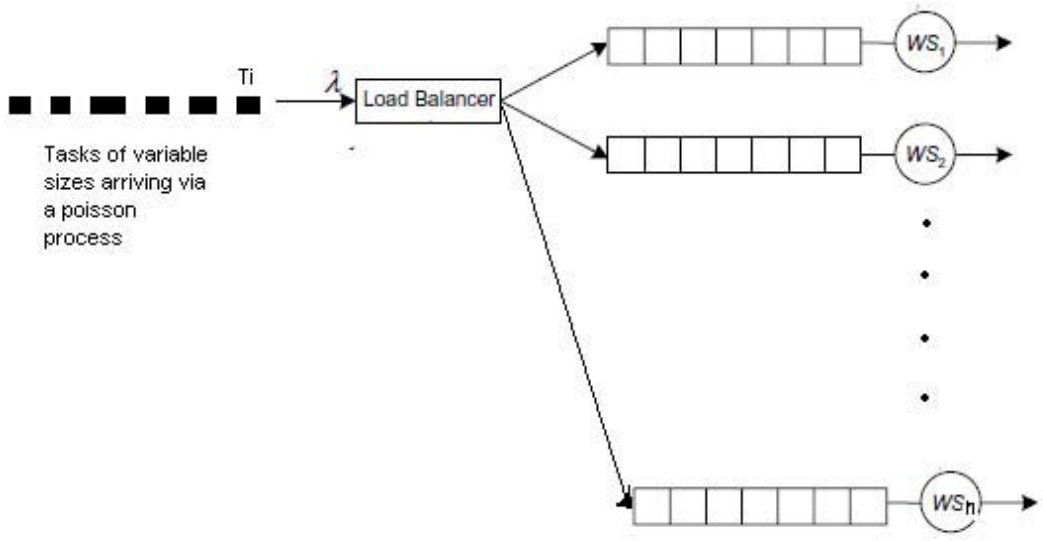

Figure 2.2 – Queuing model for centralized load balancing in a file sharing application

The size of each request which actually refers to the size of the file included is represented by a random variable that follows the Weibull distribution [3]. It is obviously impractical to involve thousands of users to generate a realistic web workload. Thus we need to generate the workload by means of software. A trace client takes as input a web access log file and then makes a replay of the get-requests contained in the web access log file. Each get-request (entry) in a web access log contains a time stamp, specifying when the requests are to be made, and a specification of the document requested. The log file is analyzed to reveal information about the distribution of file sizes. The cumulative probability distribution model is given as:

$$F(x) = 1 - 1.2393187 * e^{-0.024458885x^{0.475}} \quad (2.5)$$

The probability density function is given by

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.6}$$

For our model, we have chosen the value of k = 5 and λ = 1. The cumulative distribution function looks as in Figure 2.3.

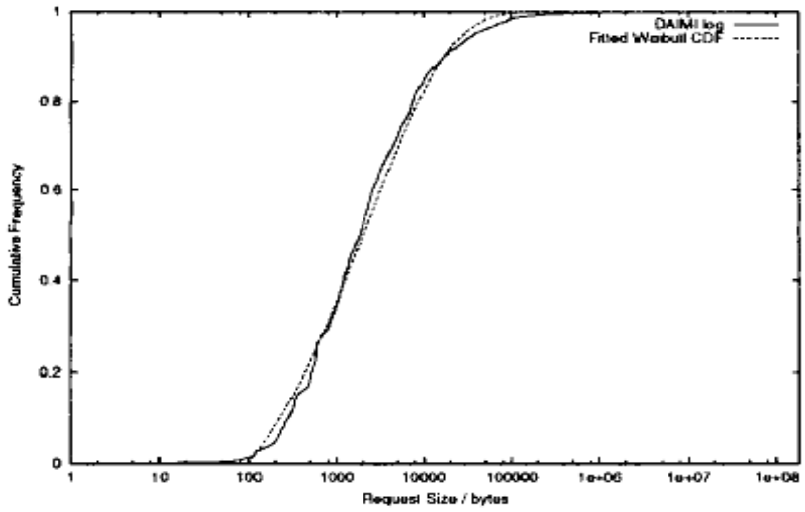

Figure 2.3 – Distribution of standard File Sizes

2.3 Load Balancing Model

The load balancing is implemented using a dispatcher. It follows a Virtual IP with direct reply technique [4]. The load balancer only owns a public IP address on the internet. The IP addresses of all the web servers are private which are inaccessible from the internet (like 192.168.*.* or 10.*.*.*). The only way to access the site is through the load balancer's IP address thus confines security management to the load balancer only. The dispatcher redirects each request to one of the web servers by either modifying the destination IP address contained in the packet header

or establishing an additional connection between the dispatcher and the web server. The selected web server sends response to the request to the client directly or indirectly. Although the dispatcher-based architecture could yield excellent performance of load balancing, the dispatcher may become the bottleneck and restrict the scalability of the system [5].

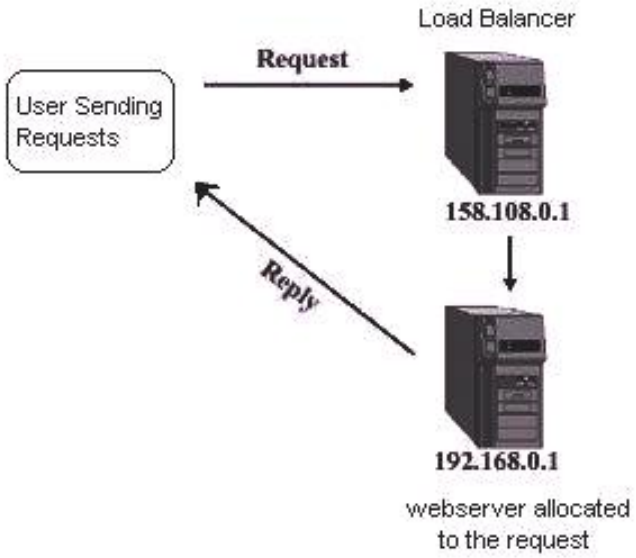

Figure 2.4 – Virtual IP with direct reply

2.4 Policies for Load balancing

Load balancing of retrieval based tasks can be done in several approaches that deal with consideration of the CPU utilization, IO utilization and memory utilization of a given task. The following is generalized classification of the different types of algorithms that are generally used.

1. CPU based Load balancing which considers the load created by a task on the CPU for determining a threshold value beyond which a server is not allocated any further task.[6]

2. CPU- Memory based Load balancing takes both CPU and memory resources into account. This is then used to determine a threshold value beyond which any server is not allocated any further task. [6]

3. IO aware Load balancing only takes into account the IO load caused by a task on the server.[1]

Since the tasks we expect to get in this system also have some CPU bound parts, following alternate policies may lead to better performance but in this paper, we only focus on the third kind of policy.

2.5 I/O-aware load balancing in heterogeneous environment

I/O-aware load balancing policy (IO-RE), policy relies on an I/O load index to measure two types of I/O access: the implicit I/O load induced by page faults and the explicit I/O requests resulting from tasks accessing disks. A node i's I/O load index is given by

$$\mathbf{load_{IO}(i)} = \sum_{j \epsilon Ni}^{n} \mathbf{page(i,j)} + \sum_{j \epsilon Ni}^{n} \mathbf{IO(j)} \qquad (2.7)$$

where page(i; j) is the implicit I/O load of task j on node i, and IO(j) is the explicit I/O requirement of task j.

An I/O threshold, $threshold_{IO}(i)$, is introduced to identify whether node i's I/O resource is overloaded. Node i's I/O resource is considered overloaded if $load_{IO}(i)$ is higher than $threshold_{IO}(i)$. The I/O threshold is given as

$$\mathbf{threshold_{IO}(i) = Di \,/\, (\sum_{j=1}^{n} Dj) * \sum_{j=1}^{n} loadIO(j)} \qquad (2.8)$$

Implicit I/O depends on M_i, $load_{mem}(i)$ and page fault rate pr_i.

$$page(i,j) = \begin{cases} 0 & \text{if } load_{mem}(i) \leq M_i, \\ \frac{pr_i \times load_{mem}(i)}{M_i} & \text{otherwise.} \end{cases} \tag{2.9}$$

Explicit I/O load IO(i,j) is proportional to I/O access rate ar(j) and inversely proportional to I/O buffer hit rate hr(i, j). The buffer hit rate for task j on node i is approximated by the following formula:

$$hr(i,j) = \begin{cases} \frac{r}{r+1} & \text{if } buf(i,j) \geq d(j), \\ \frac{r \times buf(i,j)}{(r+1) \times d(j)} & \text{otherwise,} \end{cases} \tag{2.10}$$

Where r is the data re-access rate (defined to be the number of times the same data is accessed by a task), buf(i, j) denotes the buffer size allocated to task j, and d(j) is the amount of data accessed by task j, given a buffer with infinite size.

2.6 Performance Metrics

The system that we have modeled needs to be measured in terms of mathematical parameters. We have used three parameters which we use as performance metrics to evaluate the system during simulation which shall be covered in the next chapter.

2.6.1. Standard Deviation of Utilization of servers

$$SD = \sqrt{\sum_{i=1}^{K} (U_i - U_{avg})^2 / K} \tag{2.11}$$

Here SD stands for the standard deviation of all the server utilization values of the system. U_i stands for the utilization of the i^{th} server and u_{avg} stands for the average utilization of all servers. There are K servers in the system to which tasks are allocated. Utilization of each server is determined by considered the number of tasks that have completed their execution at it. To measure this we use the formula below –

$$\mathbf{U_i = d_i/T_a} \quad (2.12)$$

Where d_i is the total data transferred on the disks of the server due to tasks on it and T_a is the time for which it has been active and executing retrieval tasks. A higher value of standard deviation implies that the utilization of the system is skewed and the load balancing is not effective and vice versa [5].

2.6.2 Makespan

A central problem in scheduling theory is to design a schedule such that the last finishing time of the given jobs (also called makespan) is minimized [15]. This problem is called the minimum makespan scheduling. The makespan of a job allocation problem refers to the time by which the last task in a given set of tasks is completed. In our problem, tasks arrive continuously and so we find the makespan metric only for a finite set of tasks and assume it reflects upon the performance of the entire system. A smaller makespan means the algorithm will be more efficient at allocating jobs and provide lesser over response time and better throughput.

In the next chapter, we shall discuss the different algorithms in detail that are used to implement the policies discussed above and compare their performances across these performance parameters and interpret the results.

Chapter 3 I/O intensive Load Balancing Policies

In this chapter we divert our focus to presenting the several algorithms that implement the policies discussed in the previous chapter. In all, four algorithms have been discussed – IORE-M, DBBS, WAL and RAT. These are variants of the algorithms suggested two existing load balancing policies – CPU based load balancing and CPU-memory based load balancing that have been discussed earlier [1]. Since our tasks are I/O intensive, we consider I/O aware load balancing techniques. We have also proposed an algorithm that also takes into account the CPU load for a task. In the sections below we describe each of these algorithms.

3.1 Algorithms for IO intensive task allocation

3.1.1. IO-aware Load Balancing

For a task j arriving at a local node i, the IORE-M (IO aware)[1] scheme attempts to balance I/O resources in the following four main steps. This algorithm uses the IO load attribute of the tasks to calculate the IO load on each server using equation 2.7. It then finds the threshold due to IO load on each server using equation 2.8.

This algorithm has two stages. In the first stage the decision making criteria is not used and the tasks are simply allocated to the servers on a FCFS basis till all servers are loaded. In the second stage the decision making of allocation is done using the IO threshold mechanism. This has an advantage that the overhead for the entire algorithm is reduced because the initial stage does not incur the server selection overhead.

Algorithm 1 - IORE-M

Input:

- S_i , R_i and B_i^{disk} values for all the nodes used to calculate D_i for each node
- The set of incoming tasks having Poisson distributed inter-arrival time and their I/O explicit I/O requirements

Output: A valid schedule for the task set.

Steps of execution –

1. For the first n tasks (where n is the number of servers) in the system allocate the first n tasks to the n nodes in a FCFS sequence.

2. Calculate I/O load of node i by adding task j's explicit and implicit I/O load as per the formula 2.5.

3. Calculate the I/O threshold of each node based on Equation 2.8.

For all the subsequent tasks,

4. Pre-allocate to each server in turn and re-compute the load and threshold for it.

5. If all the nodes are overloaded, allocate the task to the server which has the lowest absolute difference between the current load and the current threshold.

6. Else allocate the task to the server which has the highest absolute difference between the current load and the current threshold.

7. Repeat step 4-6 till all tasks are allocated.

3.1.2. Disk Performance based Best Server algorithm

In this section, we present a centralized task allocation algorithm that utilizes the disk heterogeneity of the servers to allocate tasks in balanced manner. In this algorithm, the selection

criteria for the server to which the task will be allocated is based on its disk performance. This algorithm is based on the fact that the most powerful node, in terms of its disk IO capability should handle the highest IO load. The load then contributes to changing the disk performance parameter which leads to a dynamic adjustment of task handling capabilities of each node.

Algorithm 2 - DBBS

Input:

- S_i , R_i and B_i^{disk} values for all the nodes
- The set of incoming tasks having Poisson distributed inter-arrival time and their I/O explicit I/O requirements

Output: A valid schedule for the set of tasks

The steps of execution are as follows:

1. Calculate the value of D_i for each node, taking d_{avg}=0 as given in formula 2.1
2. Construct a max-heap of the nodes based on the values of D_i
3. For each task present in the queue on the master server, perform the following steps

a. Extract the node present on top of the heap and allocate the task to that node. The disk performance threshold D_{th} which is defined as –

$$\mathbf{D_{th} = (Di / \sum_{f=0}^{n} D_f) \times Load_{totposs}} \quad (3.1)$$

Where the $Load_{totposs}$ parameter represents the total static load that can be produced to the entire set of tasks on the system irrespective of what kind of load it is.

If the selected node is found overloaded after allocation of the task, consider the next element on the heap.

b. If it is already servicing a task, add the new task in the node's own queue.

c. Increment the value of d_{total} for that node by the amount of I/O data to be retrieved in the task where d_{total} is the total amount of I/O data transferred at the node due to all the tasks on that node.

d. Compute d_{avg} for the node and recalculate its D_i

e. Re-compute the heap

3.1.3. CPU and IO based Best Server algorithm (Weighted Average Load)

For every node *i*, the weighted load index defined in WAL is the weighted average of the required resource load including both CPU and IO based resources:

$$\mathbf{Load\ (i) = W_{io} \times Load_{io}(i) + W_{cpu} \times Load_{cpu}(i)} \quad (3.2)$$

WAL dispatches the job to a node with the smallest value of the load index. In our experiments, the W_{IO} and W_{CPU} parameters are computed dynamically on basis of the tasks that are allocated by finding the average ratio between their CPU and IO strengths. We know that I/O and CPU are not equally important in the workload[7] since our system model deals mainly with IO intensive tasks of a retrieval based nature so the first weight is generally smaller than the second. The formulae to calculate $Load_{io}(i)$ is mentioned in chapter 2 under equation 2.7. For calculating $Load_{cpu}(i)$, we shall use the formula –

$$\mathbf{Load_{cpu}(i) = L_i \times (max_{j=1->n}(C_j))/C_i} \quad (3.3)$$

Where L_i is the no. of tasks on node i and C_i is the cpu performance of node i

Algorithm 3 - WAL

Input:

- The set of incoming tasks having Poisson distributed inter-arrival time and their I/O explicit I/O requirements

- The CPU / IO breakdown of task strength of each task

Output:

- A valid schedule for the set of scheduled tasks.

The Steps of execution are follows:

1. For each task in the system perform the following steps:
    a. Scan all the nodes to determine which has the least load value
    b. If the IO load on the server exceeds the threshold, select the next least loaded server.
    c. Assign the task to that node, in case of tie decide arbitrarily.
    d. Re compute the load for the node which was just loaded with the task.
    e. Re-compute the threshold value for the node.
2. Repeat till all tasks are allocated.

3.1.4 Random Allocation using IO Threshold

In this algorithm, the server to which the task is allocated is selected in a random manner if the server to which it is initially allocated is found to be overloaded using the IO threshold discussed previously.

Algorithm 4 – RAT

Input:

- The set of incoming tasks having Poisson distributed inter-arrival time and their I/O explicit I/O requirements

Output:

- A valid schedule for the set of tasks.

The Steps of execution are follows:

1. For each task in the system perform the following steps:

a. Calculate the I/O load on each server as given in formula 2.7

b. Calculate the I/O threshold on each server as given in formula 2.8

c. Select a server at random from all the under-loaded servers.

d. Check if it is under-loaded even after allocating the task. If it's not, repeat step 1c. If it is overloaded select one more server at random that after being allocated is still under loaded.

2. Repeat till all tasks are allocated.

All these algorithms are suitable for IO intensive workloads. A more versatile algorithm can be proposed to deal with all kinds of workloads. This algorithm would take into account the three characteristics of each task namely the memory, CPU and IO. It is presented in the section below.

3.2 IO-CPU-Memory based Load balancing

Since the main target of the IORE-M policy is exclusively I/O-intensive workload, IORE-M is unable to maintain a high performance when the workload tends to be CPU- or memory-intensive which is a possibility for a different kind of web application. To overcome this limitation of IORE-M, a new approach, referred to as IOCMRE, attempts to achieve the effective usage of CPU and memory in addition to I/O resources in heterogeneous clusters. More specifically, when the explicit I/O load of a node is greater than zero, the I/O-based policy will be leveraged by IOCMRE as an efficient means to make load-balancing decisions. When the node exhibits no explicit I/O load, either the memory-based or the CPU-based policy will be utilized to balance the system load. In other words, if the node has implicit I/O load due to page faults, load-balancing decisions are made by the memory-based policy. On the other hand,

the CPU-based policy is used when the node is able to fulfill the accumulative memory requirements of all tasks running on it.

Algorithm 5 – IOCMRE

Input:

- The set of incoming tasks having Poisson distributed inter-arrival time and their I/O explicit I/O requirements
- CPU and Memory and IO requirements of each task
- CPU and Memory and disk performance capabilities of each node.

Output:

- A valid schedule for the set of tasks

Steps of Execution:

1. For each task do the following steps:
    a. Allocate the first task j at node i by means of the IORE-M policy discussed above.
    b. If IO(j) + $\sum_{j=1}^{n} IO(i,j)$ > 0 then use the same policy IORE-M to allocate jobs
    c. Else if page(i,j) + $\sum_{j=1}^{n} page(i,j)$ > 0 use a memory based policy to allocate jobs
    d. Else use the CPU based policy to allocate jobs.
    e. Repeat the above steps till all tasks are allocated.

This algorithm has not been implemented by us and just proposed here to illustrate a complete approach to allocate tasks under a wide range of workload conditions. In the next chapter, we discuss the simulation of the four algorithms mentioned above and compare them with respect to the performance metrics given in chapter 2.

Chapter 4 Simulation and Results

The task allocation problem for n servers has been proved to be a NP complete problem [14]. There are basically three approaches to solve a NP complete problem. It can be simulated to find results or it can be solved using an approximation algorithm or the exact algorithm that finds it actual solution can be used. We use the first approach to solve this problem. The details of the simulation framework and the related results have been described in the subsequent pages.

4.1 Simulation Environment:

In this section we shall be discussing the several important assumptions made before the simulation of these algorithms was undertaken. We also discuss an experiment that was conducted to fix the number of nodes in the architectural model assumed in the previous chapter.

4.2 Finding the right number of servers

Before simulation was undertaken, a series of steps were carried out to find the right number of servers that would be suitable for handling the set of tasks was found out by taking a sample set of 60 tasks that followed the specifications of the nature of tasks defined in the previous chapter. These tasks were scheduled on different numbers of servers starting from 1. Each time an experiment was performed using the random allocation policy mentioned above and the performance metric total make span of server allocation was measured. The results are illustrated by the graph given below. As we can observe, the increase in number of servers after

a certain point does not help in increasing the balancing of load or task allocation time by a major factor. This can be viewed in the light of Amdahl's law [10].

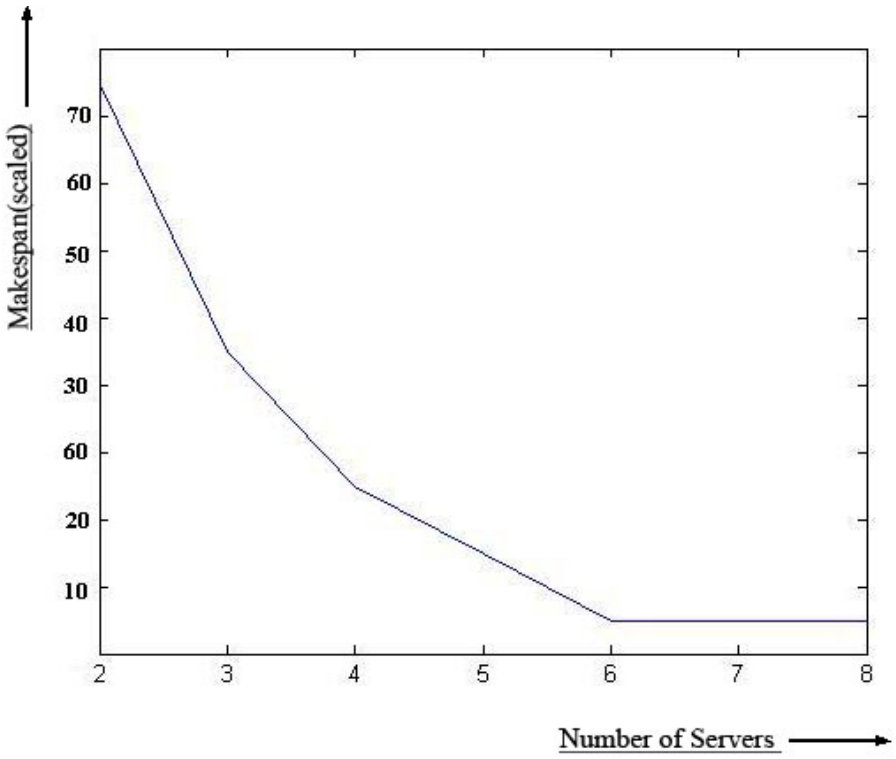

Figure 4.1 Determining no. of servers

These are the assumptions while the simulation of the algorithms was undertaken –

1. When we begin simulation for a set of nodes, we assume that we start from a zero state. It means that all the servers are now allocated zero tasks.
2. The values of all data items required for simulation have been either taken in a random fashion to maintain uniformity or from standard sources.
3. A fully connected network is simulated without considering the communication costs for any algorithm because the model proposed consists of servers arranged in a

high speed LAN system and the costs remain negligible in comparison to costs due to retrieval.

4. A simple disk model [1] is used for considering IO load on each of the tasks in the proposed system framework.
5. The service time for each IO access is the summation of seek time, rotational latency and transfer time.
6. The file sizes for each of the tasks have been assumed to be of size between 100 to 10000 bytes generated by the Weibull distribution with a mean of 2024 bytes.
7. The system has been considered to be heterogeneous only with respect to its disk performance and its heterogeneity H_d is calculated according to the formula given in the previous chapter. The values of H_d for different systems has been mentioned in table 4.1.
8. The redirection overhead due to virtual IP in the load balancer is not included while simulating because it remains indifferent to the algorithm used and hence not a suitable parameter to be included in the framework of comparison.
9. A unit amount of the value that represents the task strength is serviced in a unit amount of time. The real time elapsed has no relation with the time needed for task execution.

4.3 Results:

We have simulated the four algorithms on four different system models differing only in their disk performance parameters. The CPU and Memory sizes are kept same for all of these four systems. The disk parameters are represented in the table 4.1 below.

System	Seek Time (ms)	Rotational Latency (ms)	Bandwidth (MB/s)	Heterogeneity
A	5.3 – 8.98	3 – 5.07	3.72 – 20	0.1428
B	4.9 – 9.78	2.5 – 6.91	2.42 - 26	0.1945
C	3.99 – 10.68	2.9 – 8.87	2.12 - 32	0.2321
D	6.83	3.86	10.7	0

Table 4.1: Disk parameters for various systems of varying heterogeneity

From the table above, it is evident that system D is homogenous and the heterogeneity level increases from system A through system C. The disk heterogeneity is calculated using equation 2.3 from the second chapter.

The standard deviation values of server utilizations of different algorithms have been compared in figure 4.1 under increasing levels of disk heterogeneity that is in terms of System D, System A, System B and System C respectively.

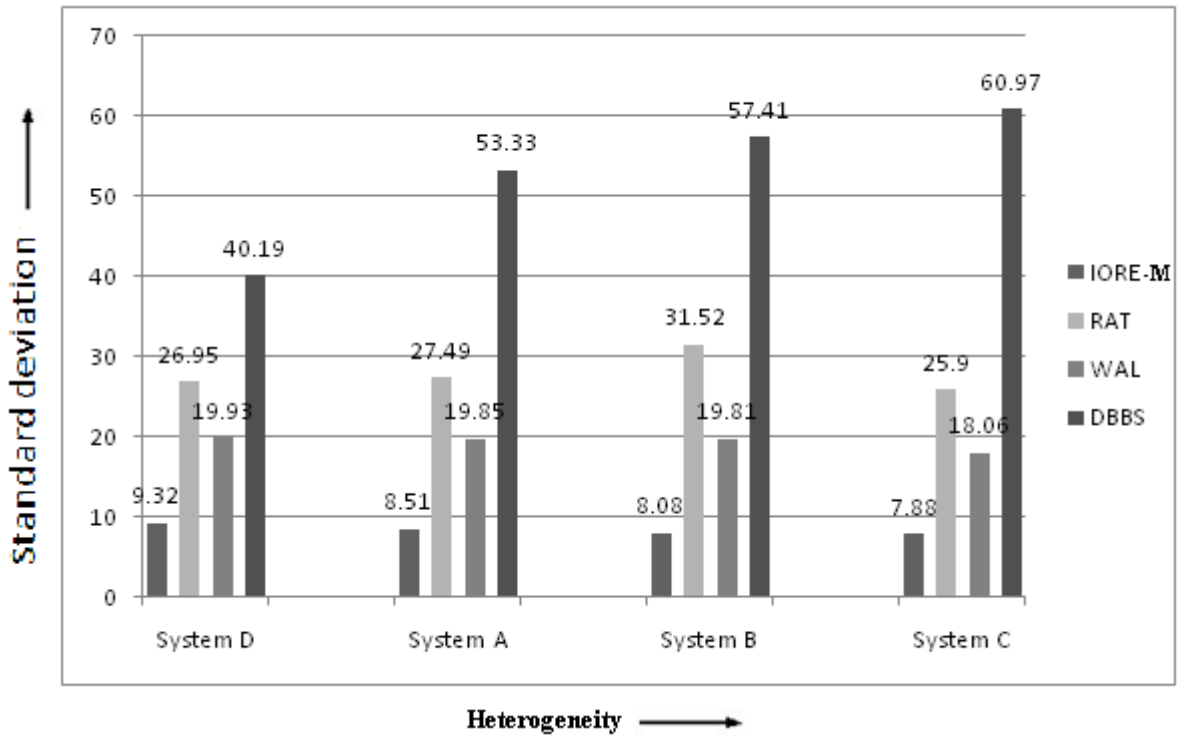

Figure 4.2: Comparison of standard deviation of server utilizations of different algorithms on different systems

From the above graph, it can be seen that IORE-M algorithm performs best across all systems while its own performance is improved as the level of heterogeneity increases. The first part of this can be explained by considering the fact that this algorithm takes into account IO loads at each node due to allocated tasks and most of the tasks are IO intensive. And since the algorithm computes the IO load of each node using a factor that involves the ratio of disk performances, a more spread disk performance of all the nodes helps in choosing the threshold effectively. (Higher values of standard deviation metric imply lower performance and vice versa). DBBS algorithm shows the worst performance. The RAT and WAL algorithms show mediocre performance in all the systems. RAT's performance shows no interpretable trend as the level of heterogeneity increases because the algorithm does not take into account the disk performances or the I/O load on the nodes.

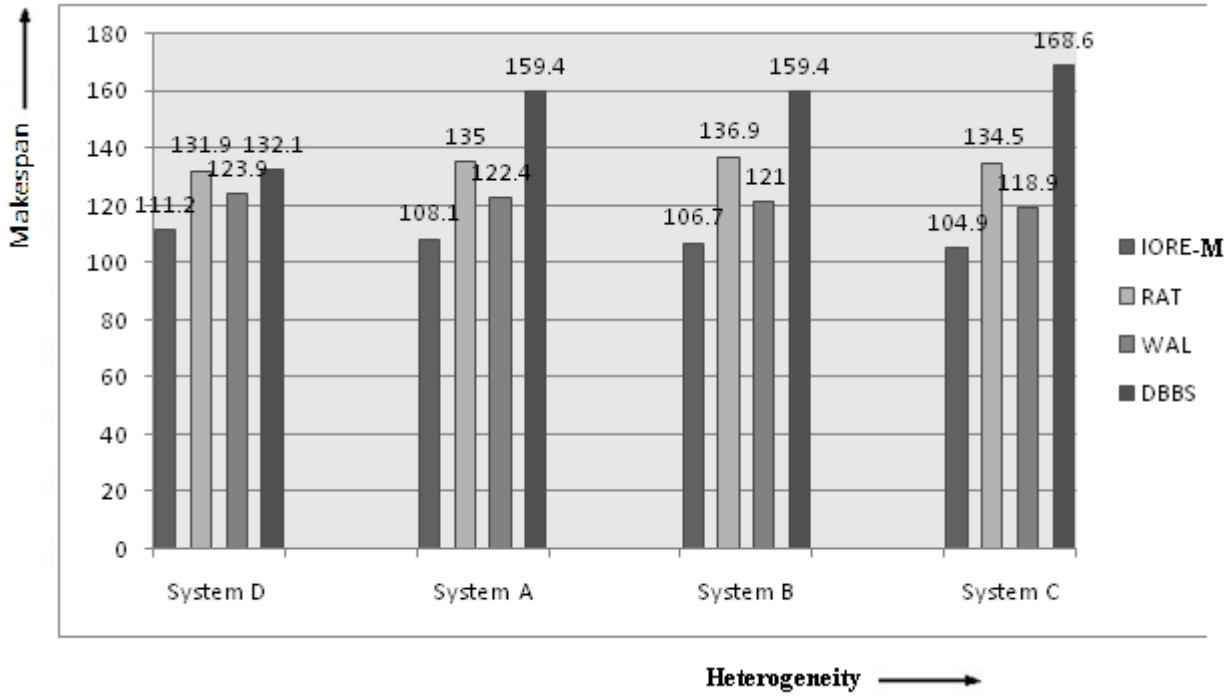

Figure 4.3 Comparison of scheduling makespan of different algorithms on different systems

From Figure 4.2 above, we see that the makespan metric virtually follows the standard deviation metric. This asserts the consistency of our algorithms. DBBS algorithm does not consider the IO load on any node while allocating tasks, rather it considers only a threshold defined solely on basis on total system load which is static and depends on disk performance. As a result, the servers with the better performing disks get most of the tasks and this leads to low performance. The WAL algorithm is better than both RAT and DBBS for both the metrics.

We also simulated the IORE-M algorithm over a less IO intensive and more CPU intensive workload. It is called as IORE-M degraded in this case. In this experiment, it was seen that its performance degraded. Clearly, this algorithm is designed only for a task with an IO intensive

nature and not very suitable for tasks which are of a wider range of workload. This observation has been made on basis of Figure 4.3.

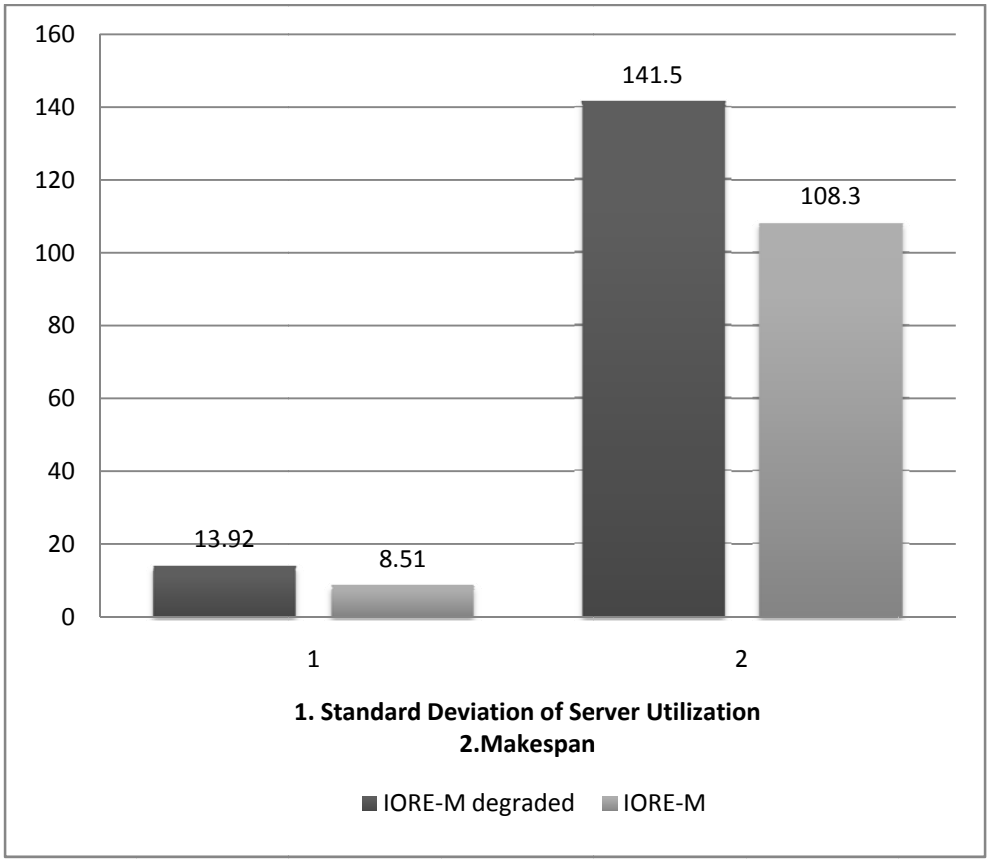

Figure 4.4 – Comparison of IORE-M and IORE-M degraded on system A

In the next chapter, we conclude the study based on the results obtained in this chapter.

Chapter 5 Conclusion

The replication of web site data and information across a cluster of heterogeneous web servers is essential for web applications or web sites that cater to a large number of users due to the rapid growth of internet traffic. Following this, load balancing plays a crucial role in adhering to the required QOS parameters by allocating the user tasks to each of the nodes in an efficient manner.

For a web application that uses file sharing and involves heavy data access from web servers thus dedicating more proportion of the total load to IO intensive tasks, IO aware load balancing policies are needed. The IORE-M algorithm was found to the best among the four algorithms considered for all the systems which show diverse levels of heterogeneity However, for a system which also services tasks that have a varied distribution of CPU, IO and Memory, the WAL algorithm is likely to perform better. The IOCMRE algorithm is also an apt choice for an environment of this kind but it has higher runtime overhead.

The proposed model thus solves the problem mentioned at the start of the thesis but within a framework that is limited by several assumptions. An actual implementation however, will enforce the results in support of the work done by simulation in this study.

Chapter 6 Future Enhancements

The approach proposed through out the study suffers from certain limitations which have already been discussed in the first chapter. The work done by us can be suitably extended in several other directions as suggested.

1. Considering a distributed model of load allocation instead of the centralized master load balancer model is a good choice for taking this work father due to a number of reasons. Resources can be shared and the reliability and fault tolerance is greatly increased. Also it is possible that some of the algorithms may be intrinsically more suitable to a distributed kind of node structure.
2. Developing an intelligent or adaptive load balancing algorithm which uses heuristics to determine which algorithm based either on more weight to CPU, IO or memory load or any viable combination of these factors to use in a given situation for a task allocation problem depending on the current system state.
3. Considering a task model in which the requests or tasks which have the potential to be parallel executed but are limited due to the sharing of resources with other tasks or some other form of control dependency.
4. The study of multimedia based web applications and the special requirements of such task allocation schemes in systems like video on demand [17] which are basically retrieval based systems presents a challenge.

5. The break up of individual tasks into finer elements and their concurrent execution or data retrieval by methods like striping can improve the performance of almost all algorithms in this area.
6. The entire data or information to be stored for the web application need not always be replicated on all the servers. This leads to unnecessary costs and infrastructure maintenance. Considering selective replication depending upon the average system loading and QOS can be an area where work can be extended on the framework proposed in this study.

Chapter 7 Bibliography

[1] Xiao Qin, Hong Jiang,Yifeng Zhu, David R. Swanson: Dynamic Load Balancing for IO intensive tasks on Heterogeneous clusters. Proceedings of 2003 International Conference on High Performance Computing.

[2] Zhongju Zhang, Weiguo Fan: Web Server Load Balancing – A queuing analysis. European Journal of Operation Research 186 (2008) 681-693.

[3] Lisa Wells, Soren Christensen, Lars M. Kristensen, and Kjeld H. Mortensen: Simulation based Performance Analysis of Servers. In: Proceedings of 9th International Workshop on Petri Nets and Performance Models, PNPM'01 Aachen, Sept. 11-14 , 2001, Reinhard German and Boudewijn Haverkort (eds.), IEEE, pages 59-68. 2001.

[4] Nartpong Ampronaramveth, Surasak Sanguanpong: Optimization of Cluster Web Server Scheduling from Site Access Statistics. Proceedings of the Sixth Annual National Symposium on Computational Science and Engineering, Nakhon Si Thamarat, Thailand, April 2002

[5] Zhong Xu, Rong Huang, Laxmi N. Bhuyan: Load balancing of DNS based distributed web server systems with Page caching. Proceedings of the 10th international conference on parallel and distributed systems 2004.

[6] Xiao, L.., Zhang, X., Qu, Y: Effective load sharing on heterogeneous networks of workstations. Proceedings of International Symposium on Parallel and Distributed Processing. (2000).

[7] Xiao Qin Hong Jiang, Yifeng Zhu, David R.Swanson: A dynamic load balancing scheme for IO intensive applications in distributed systems. Proceedings of International Conference on Parallel Processing Workshops 2003.

[8] M. Harchol-Balter and A. Downey: "Exploiting Process Lifetime Distributions for Load Balacing," Published in ACM Transactions on Computer Systems, vol. 3, no. 31, 1997.

[9] X. Zhang, Y. Qu, and L. Xiao: "Improving Distributed Workload Performance by Sharing both CPU and Memory Resources," Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000), Apr. 2000.

[10] L. Lee, P. Scheauermann, and R. Vingralek: "File Assignment in Parallel I/O Systems with Minimal Variance of Service time," IEEE Trans. on Computers, Vol. 49, No.2, pp.127-140, 2000.

[11] Y. Zhang, A. Yang, A. Sivasubramaniam, and J. Moreira: "Gang Scheduling Extensions for I/O Intensive Workloads," Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, 2003.

[12] Cho, Y., Winslett, M., S. Kuo, J.L., Chen, Y: Parallel I/O for scientific applications on heterogeneous clusters: A resource-utilization approach. In: Proceedings of Supercomputing. (1999)

[13] Chu-Sing Yang and Mon-Yen Luo: Building an adaptive Fault tolerant highly manageable web server on clusters of non dedicated workstations. IEEE Journal 2000.

[14] Mathijs de Weerdt, Yingqian Zhang, Tomas Klos: Distributed task allocation in social networks. Proceedings of autonomous agents and multiagent systems, ACM Press 2007.

[15] Rasaratnam Logendran and Nudtapon Nudtasomboon: Minimizing the makespan of a group scheduling problem- a new heuristic. Internation Journal of Production Economics Vol.22 Issue 3, Dec 1991.

[16] Thomas H. Cormen, Charles E. Leiserson, Donald R Rivest, Clifford Stein: Introduction to Algorithms 2nd Edition, MIT Press 2001.

[17] Yin-Fu Huang, Chih – Chiang Fang : Load Balancing of Clusters of VOD Servers. Information Sciences 164[2004].