ADAPTIVE LOAD BALANCING FOR CLUSTER ARCHITECTURE USING TRAFFIC MONITORING WITH CONTENT AWARENESS

A DISSERTATION

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

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

ARCHANA NIGAM

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

MAY, 2012

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "**ADAPTIVE LOAD BALANCING FOR CLUSTER ARCHITECUTRE USING TRAFFICE MONITORING WITH CONTENT AWARENESS**" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from July 2011 to April 2012, under the guidance of **Prof. Padam Kumar, Head of the Department,** Electronics and Computer Engineering, IIT Roorkee.

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

Date: 25/05/12
Place: Roorkee

(Archana Nigam)

CERTIFICATE

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

Date: 25/05/12
Place: Roorkee

(Prof. Padam Kumar)
Head of Department
Electronics and Computer Engineering
IIT Roorkee.

ACKNOWLEDGEMENTS

It has been a great privilege to be an M.tech student in the Electronics and Computer Engineering department at IIT, Roorkee and work closely with my advisor **Prof. Padam Kumar,** Head of the Department, Electronics and Computer Engineering, IIT Roorkee.

I express my deepest gratitude to him for his valuable guidance, support and motivation in my work. I have a deep sense of admiration for his inexhaustible enthusiasm and readiness to help me. The valuable discussion and suggestion with him have helped me a lot in supplementing my thoughts in the right direction for attaining the desired objective.

My special sincere heartfelt gratitude to all my friends, whose sincere prayer, best wishes, support and unflinching encouragement has been a constant source of strength to me during the entire work. On a personal note, I owe everything to the Almighty and my parents for always being my side and their support.

(Archana Nigam)

ABSTRACT

With the rapid growth of both information and users on the Internet, how to effectively improve the quality of network service becomes an urgent problem to be addressed. Load balancing is a solution to this problem in an effective way. Different adaptive load balancing methods have been developed to estimate servers load performance. However, they suffer from either increased processing load on the servers, or additional traffic on the network and the servers, or are impractical in real time scenario. Need of high scalability, high reliability and high availability are key issues in load balancing. Load balancing algorithm along with cluster architecture fulfills all the needs.

In this dissertation, we introduce the concept of content based queues, and have used RTT passive measurement technique to get an adaptive load balancing algorithm inside the cluster. Using the same queue for each type of request results into overload on server, so we introduce the concept of different queue for different type of request. The whole load balancing task is performed by a webproxy inside and outside the cluster, a proxy that has access to all servers so it also removes the drawback of dynamic algorithm in which most of the load balancing task is performed by server itself.

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction and Statement of the Problem

1.1 Introduction and Motivation

Distributed System:

Coulouris et al

A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages [1].

Tanenbaum, Van Steen

A distributed system is a collection of independent computers that appears to its users as a single coherent system [2].

Distributed System is needed to cope with the extremely higher demand of users in both processing power and data storage. For example:

Facebook by the end of 2010[3]

- Total users: 500 millions
- Total servers: 60, 000 servers (estimate, Oct 2009)
- 50 millions operations per second
- 1 million photos are viewed every second
- Each month more than 3 billion photos are uploaded

With this extremely demand, I do believe single system could not achieve it. That's one reason why distributed systems come in place. There are many reasons that make distributed systems viable such as high availability, scalability, resistant to failure, etc. When the demand for computing power increases the load balancing problem becomes important.

Load balancing is given the initial job arrival rates at each computer in the system find an allocation of jobs among the computers so that the response time of the entire system over all jobs is minimized. So the load balancing task in distributed computing is very important task.

There are three typical approaches to load balancing problem in distributed systems.

1) *Global approach*: In this case, there is only one decision maker that optimizes the response time of the entire system over all jobs and the operating point is called *social optimum* [4].

2) *Cooperative approach*: In this case, there are

several decision makers (e.g., jobs, computers) that cooperate in making the decisions such that each of them will operate at its optimum. Decision makers have complete freedom of preplan communication to make joint agreements about their operating points[4].

3) *Non cooperative approach*: In this case, each of infinitely many jobs optimizes its own response time independently of the others, and they all eventually reach equilibrium [4].

Load distributing improves performance by transferring tasks from heavily loaded computers, where service is poor, to lightly loaded computers, where the tasks can take advantage of computing capacity that would otherwise go unused. The usefulness of load distributing is not so obvious in systems in which all processors are equally powerful and have equally heavy workloads over the long term. However, Livny and Melman have shown that even in such a homogeneous distributed system, at least one computer is likely to be idle while other computers are heavily loaded because of statistical fluctuations in the arrival of tasks to computers and task-service-time requirements [5]. Therefore, even in a homogeneous distributed system, system performance can potentially be improved by appropriate transfers of workload from heavily loaded computers (senders) to idle or lightly loaded computers (receivers).

Here performance mean is *average response time* of tasks. The response time of a task is the time elapsed between its initiation and its completion .Minimizing the average response time is often the goal of load distributing. But taking only the average response time into consideration is not sufficient. Because in real time scenario it may happen that server connected to optical fiber have average response time less compare to server connected through twisted pair, so it will result into overload condition on a server connected through optical fiber. Therefore along with average response time load distribution should also be taken into consideration.

Server load balancing is highly significant for network research and has broad market prospects. Different load balancing methods have been developed to transfer load among servers. Some of them are impractical in real time scenario while others increase processing load on the server or on the network.

In this dissertation a new adaptive load balancing algorithm (I) inside the cluster has been introduced where the concept of rtt passive measurement technique for selecting the cluster and content awareness is used. By content awareness we mean having different queue for different request types rather than having the same queue for different requests, which improves the total execution time. Also introduced a new adaptive load balancing algorithm (II) outside the cluster. Even if the adaptive load balancing method doesn't work then rtt passive measurement will do all load balancing task which increases the reliability of the system. The Server Selection policy used in dissertation is in compliance with the policy described in [6] where application layer RTT between the router and the server is a key parameter to monitor load/performance of server. RTT calculation is done through a passive measurement policy to remove any burden on the network.

1.2 Statement of the Problem

A distributed system with a number of server connected with each other may suffer from uneven load distribution due to different population densities and interests of end-users. Various static and dynamic methods were proposed for evenly load distribution among server. Overloaded server causes degradation in performance of the whole system and under loaded server causes poor network utilization. Even if they work fair for simple architecture it may happen that there performance degrades for cluster architecture.

The problem addressed in this dissertation is to distribute load evenly among the server in a distributed system by developing a load balancing algorithm for cluster architecture with the following key properties:

1. *Dynamic*: Load balancing algorithm is invoked whenever there is an uneven distribution of load among servers, or whenever a server becomes overloaded. This can happen if a large number of requests is transferred to a particular server.

2. *Adaptive*: Load balancing algorithm is modifiable as the system states changes.

3. *Reliability*: Not depend on single method of load balancing. If algorithm fails then also there must be a mechanism to perform load balancing.

4. *Distributed*: A distributed load balancing algorithm is more scalable, and preserves the original system's property of no single-point-of-failure.

4. *Transparent*: Client does not experience interruptions in service while load balancing occurs. Its entire operation requires zero-human involvement.

6. *Content Awareness*: Having different queue for different request types rather than having the same queue for different requests, which improves the total execution time.

7. *Cluster:* It provides high availability, high reliability and high scalability.

To date, very limited amount of research has been done in load balancing that cover all the seven above mentioned parameters.

"Adaptive load balancing for cluster architecture using traffic monitoring with content awareness" cover all the above mentioned parameters. This can be achieved by:

- Implementation of rtt measurement technique for selecting the appropriate cluster to improve reliability of the system.
- Implementation of proposed adaptive load balancing algorithm (I) inside the cluster.
- Implementation of proposed adaptive load balancing algorithm (II) outside the cluster.

1.3 Organization of the Report

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

Chapter 2 gives the specification of different types of load balancing

Chapter 3 describes the proposed algorithms for load balancing in both inside and outside the cluster to improve the performance of the system.

Chapter 4 gives the implementation details and result of the proposed approach.

Chapter 5 concludes the dissertation work and gives suggestions for future work.

Chapter 2

Background and Literature Review

In the field of load balancing in distributed system significant research work has been done. This section presents a review on different existing load balancing algorithm. This section explains different approach for load balancing in distributed system. Also classify the load balancing algorithm and explain their advantage and disadvantage. The goal of load balancing is improving the performance by balancing the loads among computers. There are two main categories of load balancing policies: *static policies* and *dynamic policies* [7]. Static policies base their decision on statistical information about the system. They do not take into consideration the current state of the system. Dynamic policies base their decision on the current state of the system. They are more complex than static policies.

2.1 Static Load Balancing

Static policies can be distinguished between *distributed policies* and *centralized policies* or can classify them on the basis of deterministic and probabilistic. In a distributed policy the work involved in making decisions is distributed among many decision makers. In a centralized policy there is only one decision maker or the common decision of many cooperating decision makers is made in a centralized way. In deterministic policy there must be some static method basis on which server selection is made. In probabilistic policy servers will be selected on the basis of some probability based static approach. Figure 2.1 gives classification of static load balancing algorithm.

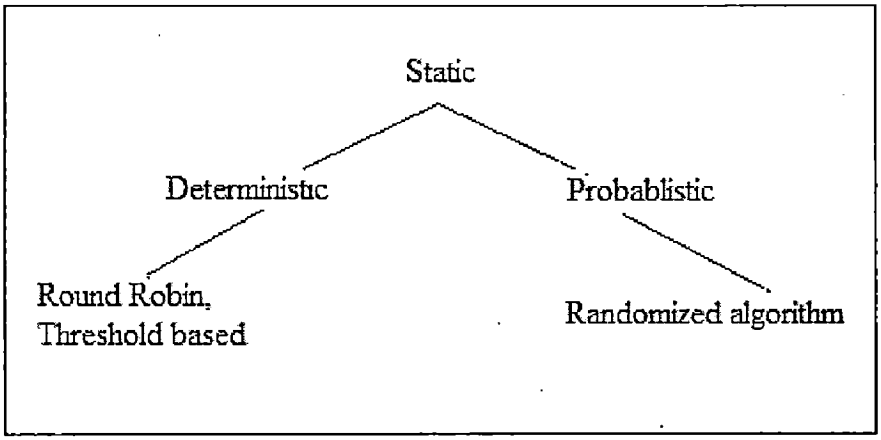

Figure 2.1 Classification of static load balancing algorithm [7]

In static load balancing, the performance of the processors is determined at the beginning of execution. Then depending upon their performance the work load is distributed in the start by the master processor [8]. The slave processors calculate their allocated work and submit their result to the master. A task is always executed on the processor to which it is assigned that is static load balancing methods are non preemptive. The goal of static load balancing method is to reduce the overall execution time of a concurrent program while minimizing the communication delays. A general disadvantage of all static schemes is that the final selection of a host for process allocation is made when the process is created and cannot be changed during process execution to make changes in the system load [7].

2.1.1 Round Robin Algorithm

This algorithm distributes jobs evenly to all slave processors. All jobs are assigned to slave processors based on Round Robin order, meaning that processor choosing is performed in series and will be back to the first processor if the last processor has been reached. Processors choosing are performed locally on each processor, independent of allocations of other processors [9]. Flow chart of this algorithm is shown in figure 2.2.

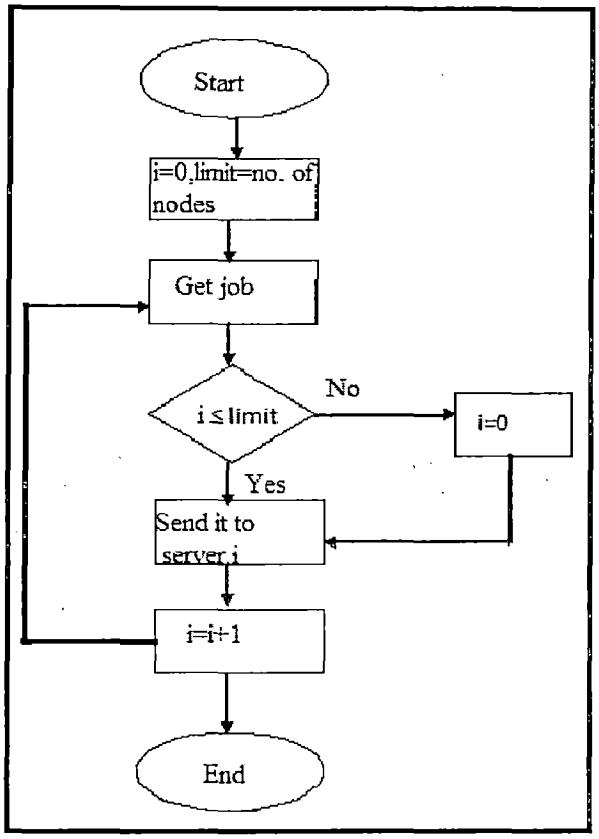

Figure 2.2 Flow chart of Round Robin Algorithm [10]

2.1.2 Randomized Algorithm

This algorithm uses random numbers to choose slave processors. The slave processors are chosen randomly following random numbers generated based on a statistic distribution [11]. The flowchart of Randomized algorithm is shown in Figure 2.3.

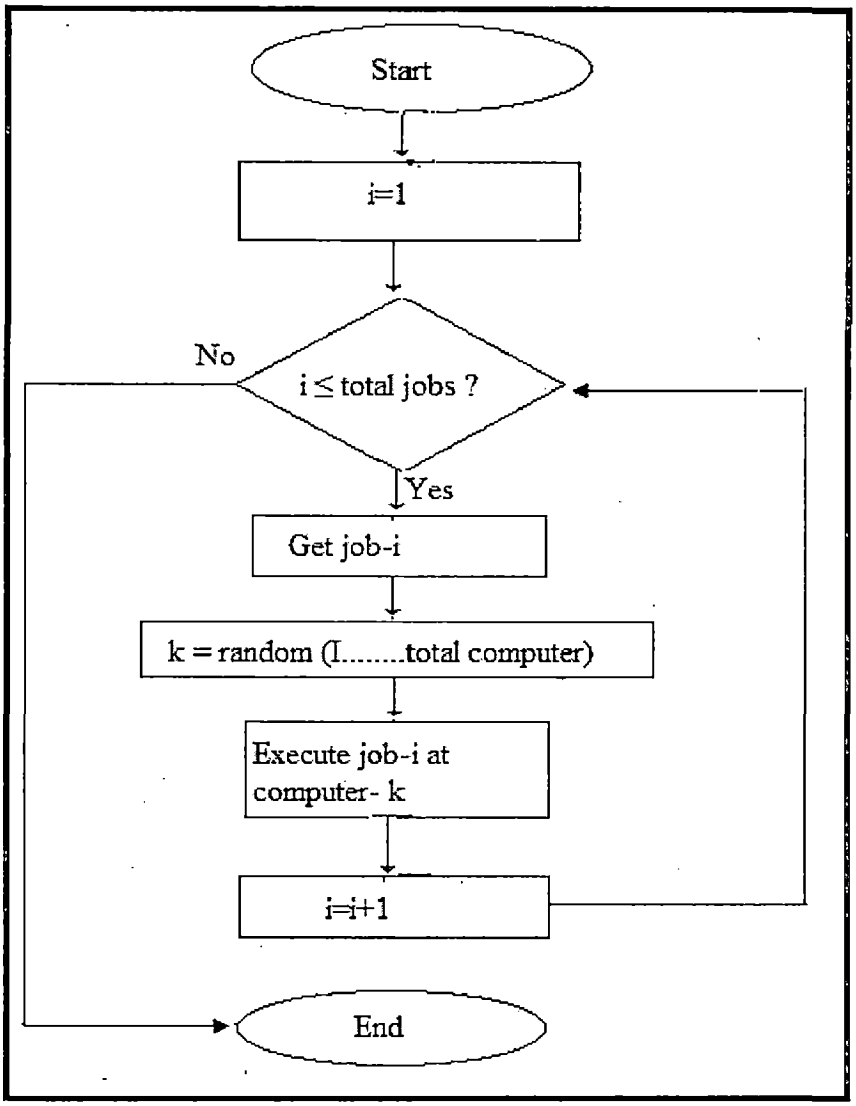

Figure 2.3 Flow chart of Randomized Algorithm [10]

2.1.3 Static Centralized Load Balancing Algorithm

In this algorithm [12], a central processor selects the host for new process. The minimally loaded processor depending on the overall load is selected when process is created. Load manager selects hosts for new processes so that the processor load confirms to same level as much as possible. From then on hand information on the system load state central load manager makes the load balancing judgment. This information is updated by remote processors, which send a

message each time the load on them changes. This information can depend on waiting of parent's process of completion of its children's process, end of parallel execution .The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the process created. High degree of inter-process communication could make the bottleneck state. This algorithm is expected to perform better than the parallel applications, especially when dynamic activities are created by different hosts [13].Flow chart of this algorithm is shown below.

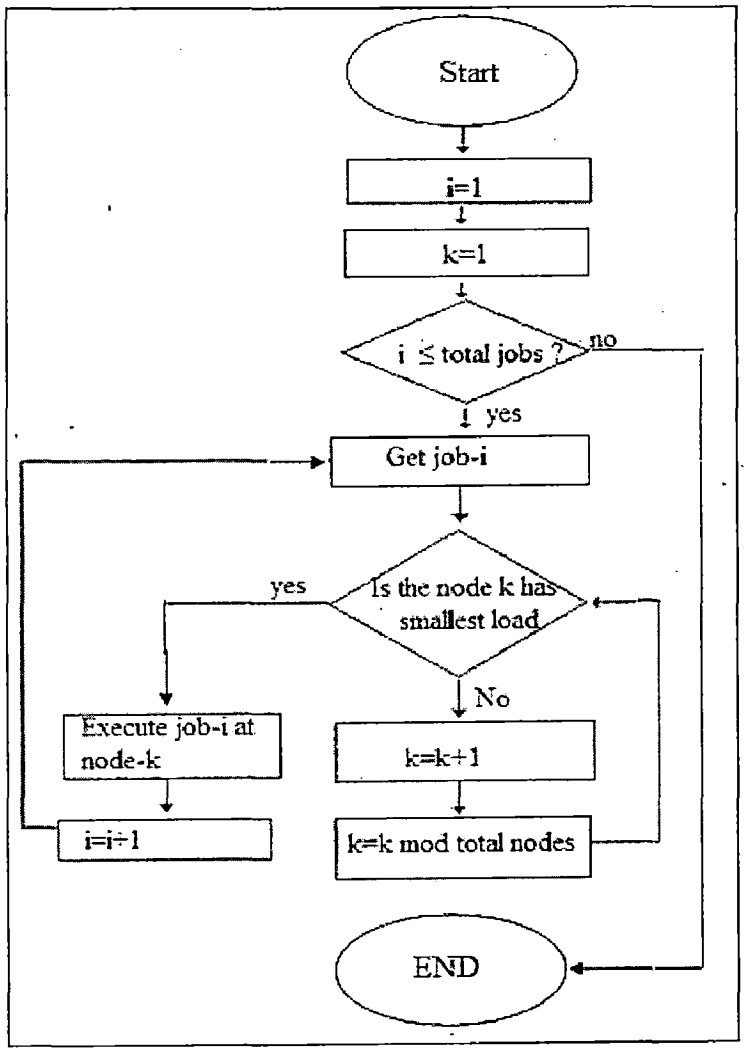

Figure 2.4 Static Centralized Load Balancing Algorithm [10].

2.1.4 Threshold Based Load Balancing

Processor choosing in this Algorithm is performed based on two threshold values, t_upper and t_lower, that represent upper and lower threshold respectively. Both of these threshold values are used to characterize states of a slave processor that described below:

Processor State	**Threshold**
Under loaded	Load $\leq$ t_under
Medium	t_under $\leq$load $\leq$t_upper
Overloaded	Load $\geq$ t_upper

Initially, all the processors are considered to be under loaded. When the load state of a processor exceeds a load level limit, then it sends messages regarding the new load state to all remote processors, regularly updating them as to the actual load state of the entire system. If the local state is not overloaded then the process is allocated locally. Otherwise, a remote under loaded processor is selected, and if no such host exists, the process is also allocated locally [14]. Figure 2.5 depicts the flowchart of Threshold algorithm [10].

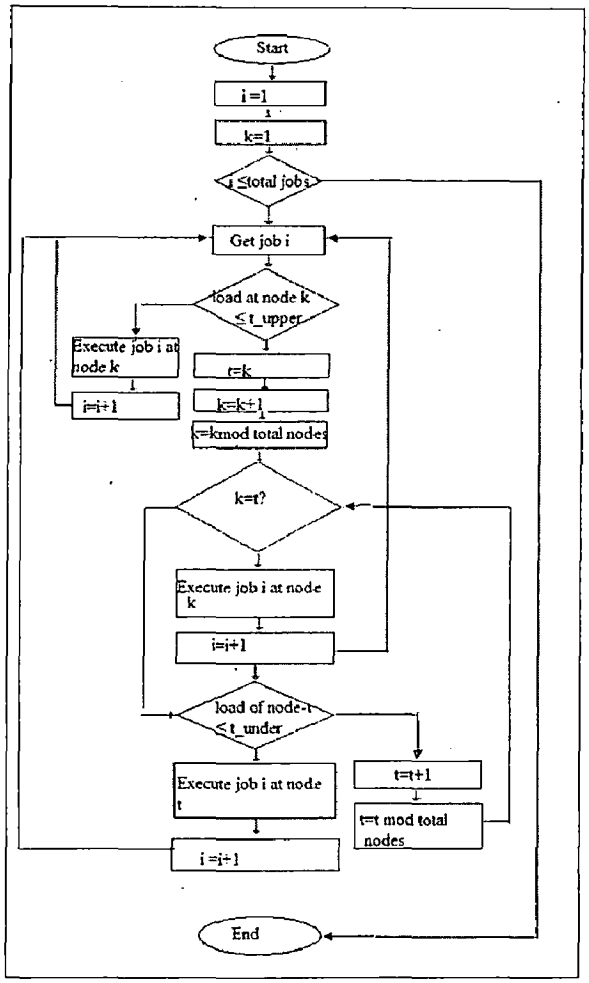

Figure 2.5 Threshold Based Static Load Balancing Algorithm.

2.2 Dynamic Load Balancing

Dynamic policies base their decision on the current state of the system. Despite the higher runtime complexity dynamic policies can lead to better performance than static policies. There are two main classes of dynamic load balancing policies: centralized and distributed [15].

2.2.1 Central Queue Algorithm

Central Queue Algorithm [16] works on the principle of dynamic distribution. It stores new activities and unfulfilled requests as a cyclic FIFO queue on the main host. Each new activity arriving at the queue manager is inserted into the queue. Then, whenever a request for an activity

is received by the queue manager, it removes the first activity from the queue and sends it to the requester. If there are no ready activities in the queue, the request is buffered, until a new activity is available [10]. If a new activity arrives at the queue manager while there are unanswered requests in the queue, the first such request is removed from the queue and the new activity is assigned to it. When a processor load falls under the threshold, the local load manager sends a request for a new activity to the central load manager. The central load manager answers the request immediately if a ready activity is found in the *process-request queue*, or queues the request until a new activity arrives. Figure 2.6 shows the flow chart for the same.

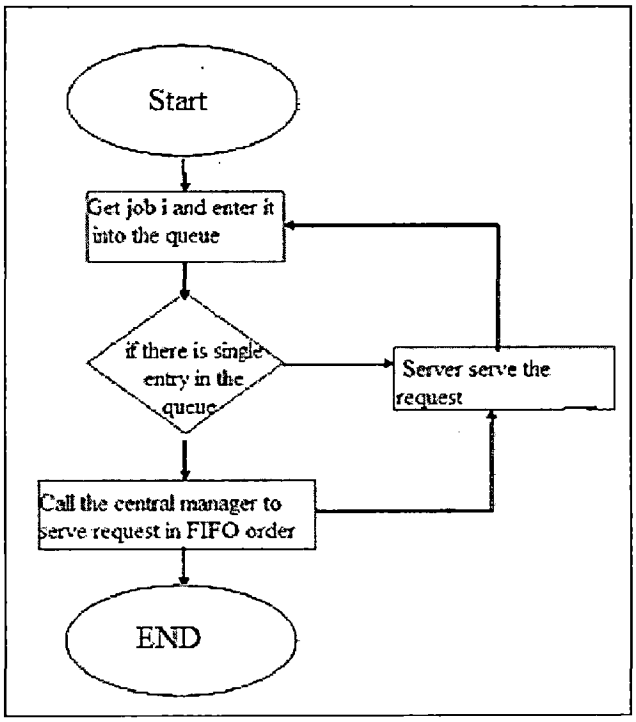

Figure 2.6 Central Queue Algorithms [16].

2.2.2 Local Queue Algorithm

Main feature of this algorithm [17] is dynamic migration support. The basic idea of the local queue algorithm is static allocation of all new processes with process migration initiated by a host when its load falls under threshold limit, is a user-defined parameter of the algorithm. The parameter defines the minimal number of ready processes the load manager attempts to provide on each processor. Initially, new processes created on the *main* host are allocated on all under

loaded hosts. The number of parallel activities created by the first parallel construct on the main host is usually sufficient for allocation on all remote hosts. From then on, all the processes created on the main host and all other hosts are allocated locally. When the host gets under loaded, the local load manager attempts to get several processes from remote hosts. It randomly sends requests with the number of local ready processes to remote load managers. When a load manager receives such a request, it compares the local number of ready processes with the received number. If the former is greater than the latter, then some of the running processes are transferred to the requester and an affirmative confirmation with the number of processes transferred is returned. Figure 2.7 shows the flow chart of the algorithm.

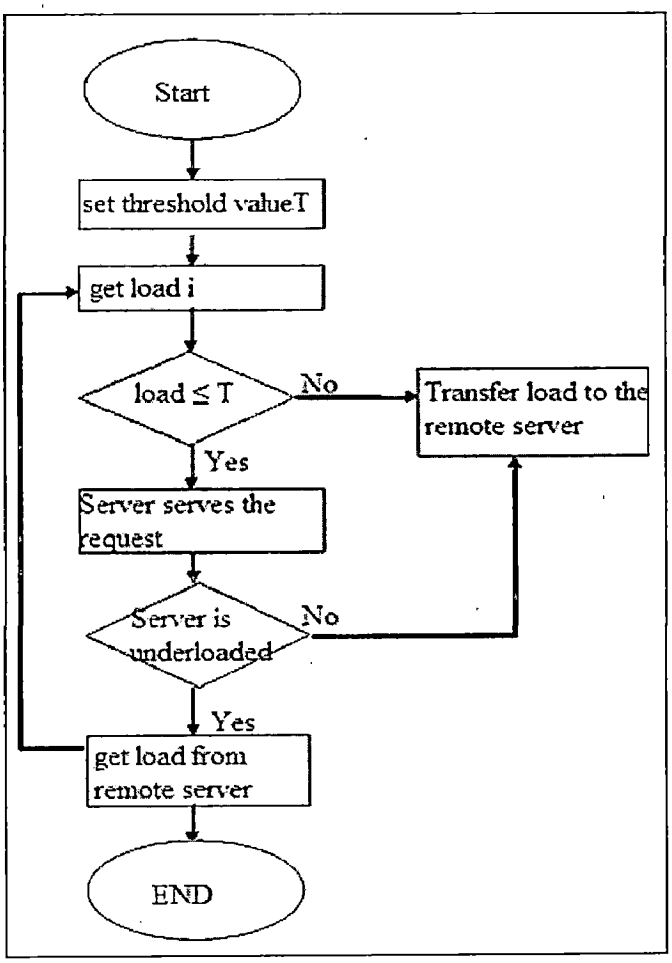

Figure 2.7 Local Queue Algorithms [17]

2.2.3 Symmetrically-Initiated Algorithm

1. Sender-Initiated Algorithm

The sender-initiated algorithm, as the name implies, is activated by a sender that wishes to off-load some of its computation. This algorithm facilitates job migration from a heavily loaded node to a lightly loaded node. There are three basic decisions that need to be made before a transfer of a job can take place:

- Transfer policy: When does a node become the sender?
- Selection policy: How does a sender choose a job for transfer?
- Location policy: What node should be the target receiver?

If the queue size is the only indicator of the workload, a sender can use a transfer policy that initiates the algorithm when detecting that its queue length *(SQ)* has exceeded a certain threshold *(ST)* upon the arrival of a new job[18]. The location policy requires knowledge of load distribution to locate a suitable receiver. The sender can send a *multicast* message to all other nodes asking for a reply about their queue sizes. Upon receiving this information, the sender can select the node with the smallest queue length *(RQ)* as the target receiver, provided that the queue length of the sender *(SQ)* is greater than the queue length of the target receiver *(RQ)* (i.e. $SQ>RQ$). Figure 2.8 depicts the flowchart of the sender-initiated algorithm.

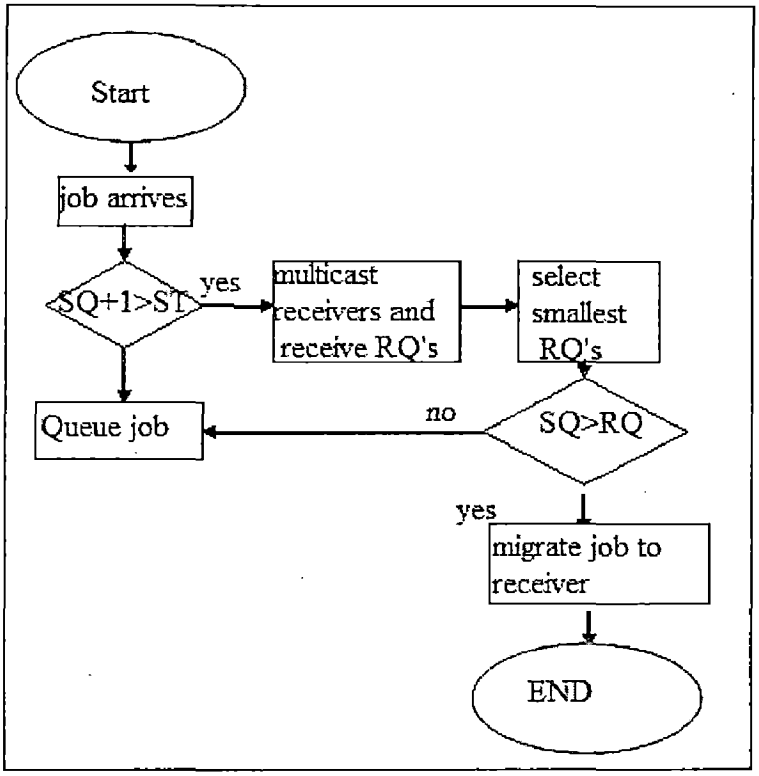

Figure 2.8 Flowchart of the Sender-Initiated Algorithm [18].

2. Receiver-Initiated Algorithm

Sender-initiated algorithm is a *push* model, where jobs are pushed from one node to other nodes. A receiver can *pull* a job from other nodes to its queue if it is underutilized [18]. The receiver-initiated algorithm can use a similar transfer policy of the sender-initiated algorithm, which activates the pull operation when its queue length falls below a certain threshold (*RT*), upon the departure of a job. Figure 2.9 depicts the flowchart of the receiver-initiated algorithm.

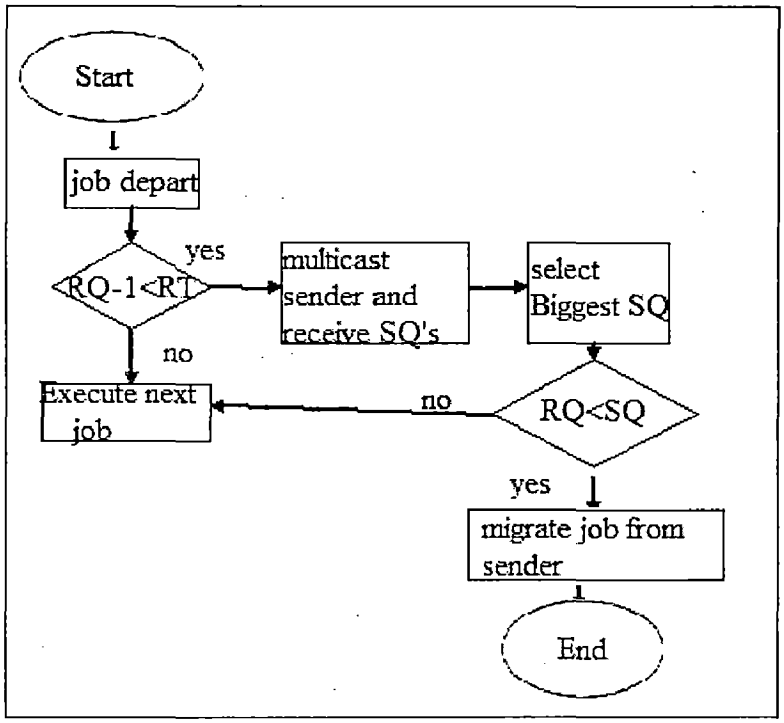

Figure 2.9 Receiver Initiated Algorithm [18]

3. Symmetrically-Initiated Algorithm

Since the sender-initiated and receiver-initiated algorithms work well at different system loads, it seems logical to combine them [18]. A node can activate the sender initiated algorithms when its queue size exceeds one threshold *ST,* and can activate the receiver-initiated algorithm when its queue size falls below another threshold *RT.* As such, each node may dynamically play the role of either a sender or a receiver.

2.2.4 Dynamic Load Balancing Algorithm for Scalable Heterogeneous Web Server Cluster with Content Awareness

The dependence on Internet for the web applications like utility bills, net banking, e-Learning, information based services etc. is increasing at an exponential rate. To improve the availability and reliability, the web sites need more than one server and a Web Server Cluster (WSC) is used. To evenly distribute the load among the servers on the WSC, dynamic load balancing (DLB) techniques are used. DLB algorithms serve homogeneous WSC and can't be directly used in the

heterogeneous environment. Therefore, author [19] proposes a DLB algorithm which supports heterogeneity, scalability and content awareness. The process starts with the establishment of cluster. Algorithm starts with initialization of parameters and load tables in different categories. As soon as scheduler receives the requests, it identifies its category and least loaded server in this category and the request is forwarded to the least loaded server. Response to the request is provided by the server without involving the scheduler. Figure 2.10 depicts the flowchart of the algorithm for the same.

Algorithm [19]

1) Parameters, load tables and response table of WSC are initialized.

2) Step (a) to step (h) will be repeated infinitely.

a) WSC scheduler waits for the client requests.

b) After arrival of requests, scheduler identifies the requests category.

c) Scheduler refers the load table to identify least loaded server.

d) Requests are redirected to the least loaded serer by rewriting the servers address.

e) Response table is updated.

f) Load table is updated if the change in load level of the servers occurred.

g) If all the servers in a category are critically loaded, addition of servers is requested.

h) Go back to step 2 (a).

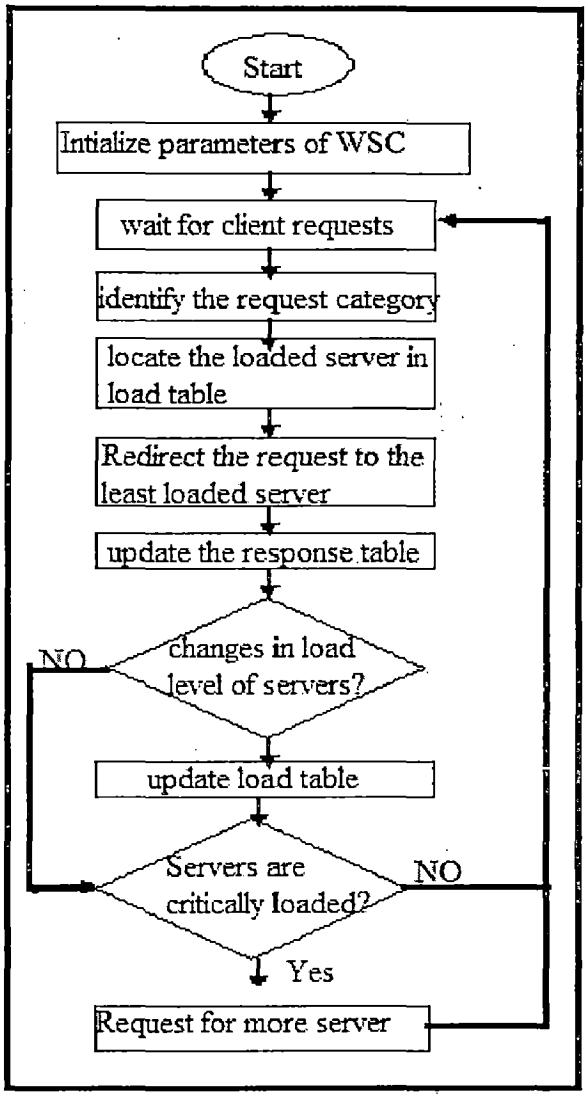

Figure 2.10 Flow Chart Of Dynamic Load Balancing Algorithm for Heterogeneous Web Server Cluster [19].

2.2.5 Centralized Dynamic Load Balancing Algorithm

Centralized dynamic load balancing [20] works on the principle of dynamic distribution. As shown in diagram, initially processes are stored in queue or process can be allotted as they arrive. Figure 2.11 depicts the architecture of centralized load balancing algorithm.

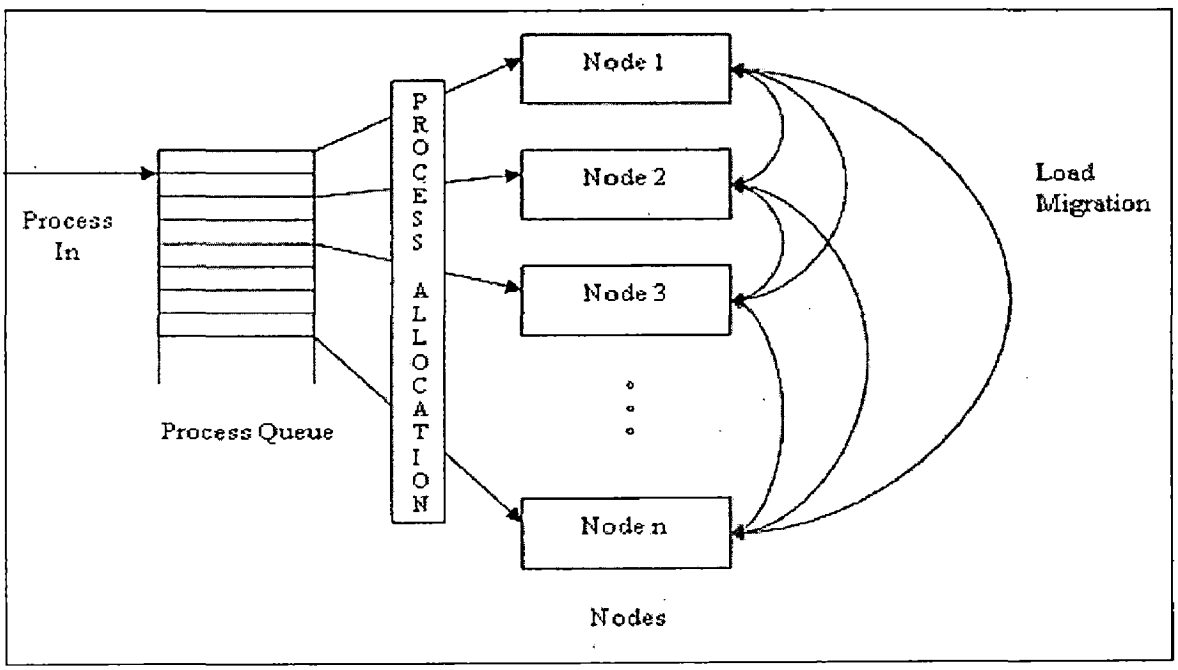

Figure 2.11 Architecture of Centralized Load Balancing Algorithm [20].

If these are placed in queue, processes are allotted one by one to primary nodes. Processes are migrated from heavily loaded node to light weighted node. Process migration is greatly affected by the network bandwidth and work load. In order to reduce the traffic, nodes are grouped into clusters. First a light weighted node is checked in the same cluster, if such primary node is available, load transfer takes place between these two nodes and load is balanced, otherwise if such light weight node is not available, one centralized node is available to accommodate the overload of a primary node. This centralized node is not assigned any process initially; it is given only the overload of primary nodes [20]. Centralized node has some better structure as compared to other nodes in the cluster. Traffic between centralized node and primary nodes kept minimum to avoid network delay. Figure 2.12 flow chart centralized load balancing algorithm.

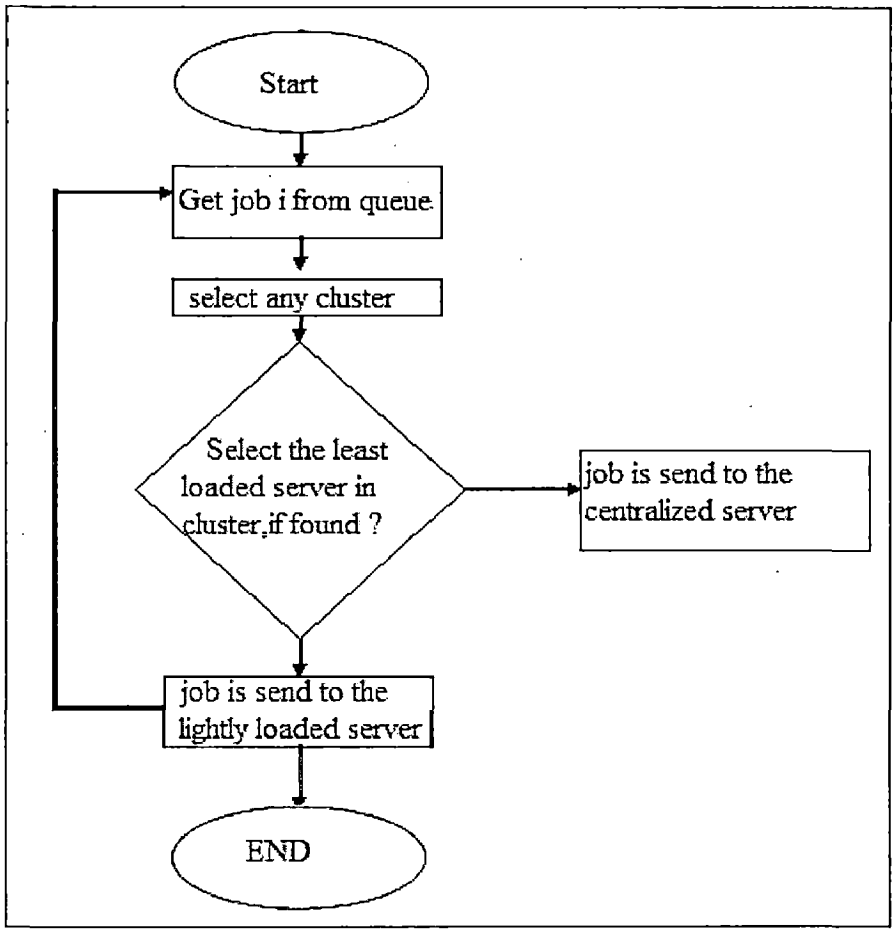

Figure 2.12 Flow chart of Centralized load balancing algorithm [20].

2.2.6 Modified Centralized Approach for Dynamic Load Balancing

In Centralized approach there is single node, so process the load at high speed by using switching but still a limitation is there. An approach is there to remove the limitation is to split the centralized node into small nodes called supporting nodes (SNs). Figure 2.13 architecture of modified centralized load balancing algorithm.

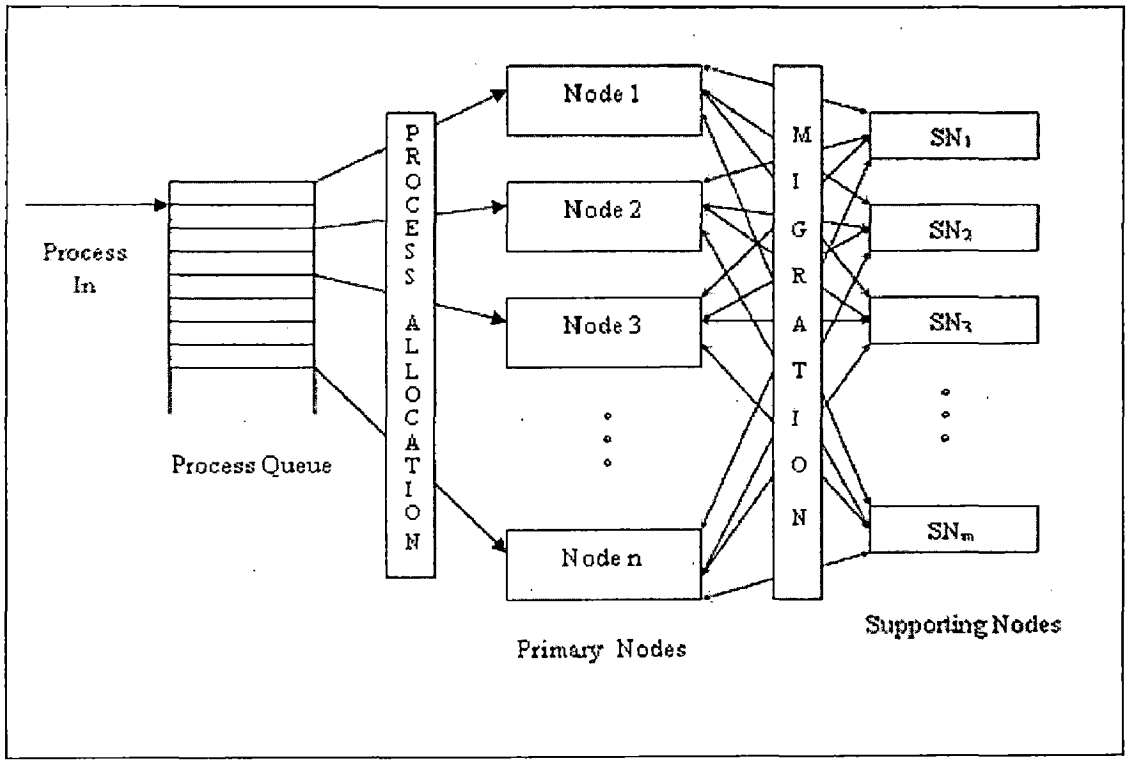

Figure 2.13 Architecture of Modified Centralized Load Balancing Algorithm [20].

But still here supporting node are not allotted load initially. Many times supporting nodes is idle or they are not properly loaded as only overload is assigned to supporting nodes. This is wastage of power of supporting nodes [20]. We can also use the free time of SN by making them busy for this free time. So a further approach is developed here in which supporting nodes are given some load initially and SNs maintain a priority list of process or order in which the process at the SN will execute. Suppose a process Pi is currently executed by SNi and a Primary node Ni is overloaded so that it finds a supporting node SNi suitable for transferring its overload, so Ni will interrupt the SNi, then SNi will assign Priority to the coming process and call the interrupt service routine to handle the interrupt. Interrupt Service Routine actually compares the priority of each coming process with the currently executing process and perform the switching between the currently executing process and process coming from the primary nodes, Otherwise, each supporting node is maintaining a priority queue in which process to be executed are sorted according to the priority, in which coming process are stored in this queue with a priority. In figure 2.13 Each node whether primary node or secondary node (assuming initially process is

there) is assigned some Task .Process and transferred from PN to SN if overload occur at any PN. Figure 2.14 flow chart of modified centralized algorithm.

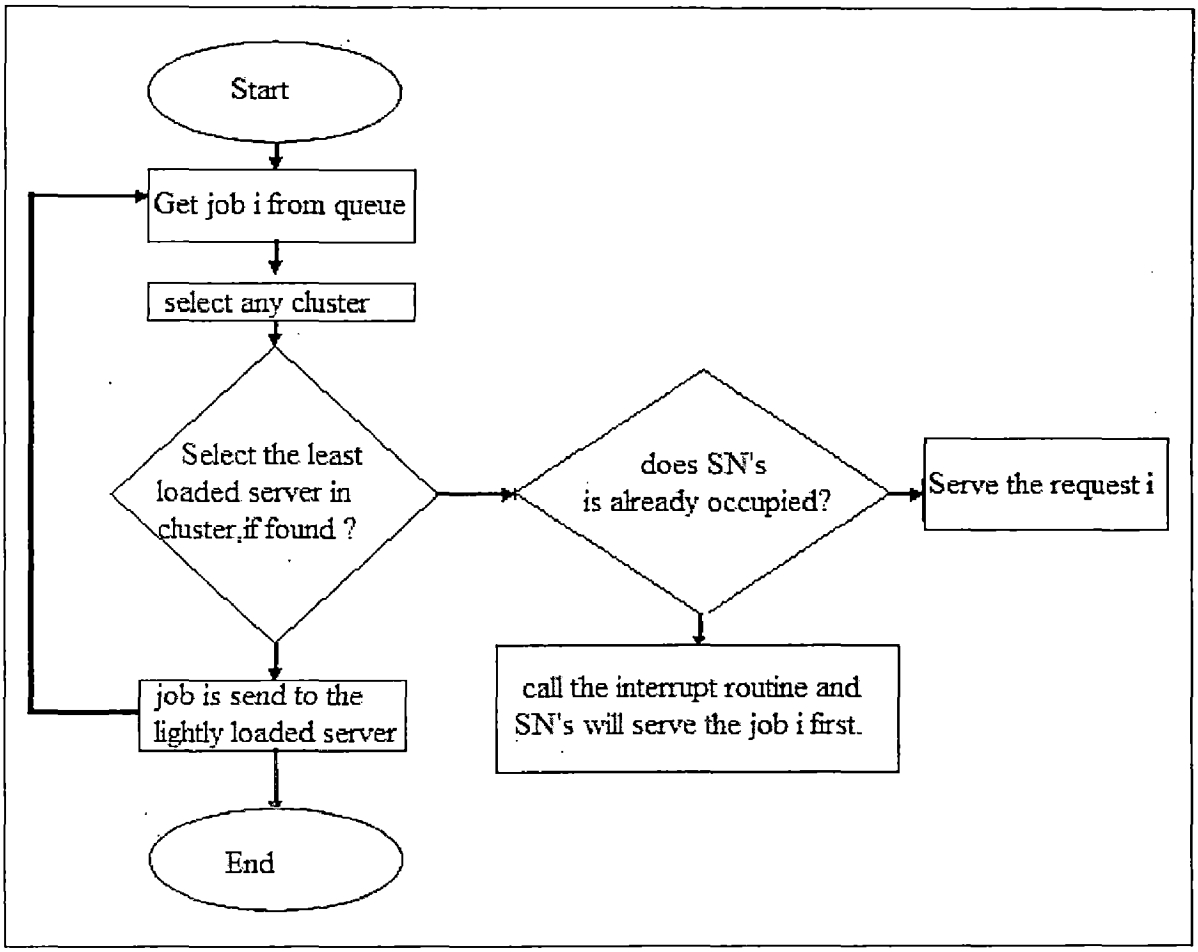

Figure 2.14 Flow chart of Modified Centralized Algorithm [20].

2.2.7 Centralized dynamic cluster based load balancing algorithm

In Centralized dynamic cluster based load balancing algorithm there are two load balancer. In cluster architecture one balancer is outside the cluster for performing load balancing task outside the cluster and other balancer is inside the cluster for performing load balancing inside the cluster. In this method balancer inside the cluster keeps data of all the servers. It transfers the request to the server which is minimal loaded. Balancer outside the cluster keeps data of all the clusters and it is responsible for transferring the load from one cluster to another cluster to make

whole system balanced. In centralized dynamic cluster based load balancing algorithm outside balancer and inside balancer is bottleneck. Figure 2.15 shows the architecture centralized dynamic cluster based load balancing algorithm.

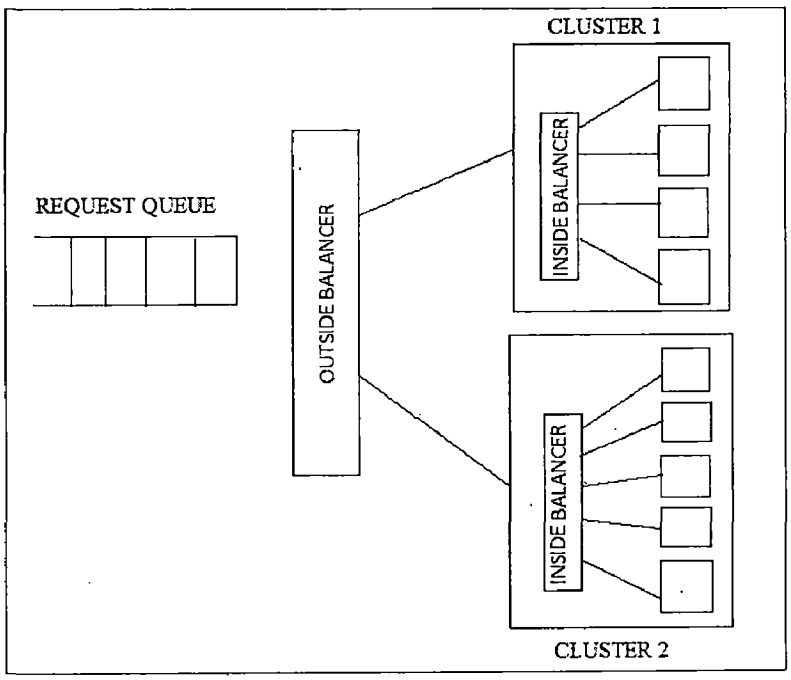

Figure 2.15 Centralized dynamic cluster based load balancing algorithm

2.2.8 A Content-Based Load-Balancing System

Content-based routing is another technology that can be used to enhance a network's features. In the above example, the load-balancing router simply distributed network traffic evenly across a list of servers. The router selects the best server then forwards the request to the chosen web server and acts as a middleman thereafter, passing packets from the client to the server and from the server to the client in a forwarding mode. So to select the appropriate web server application layer RTT is used. The application layer RTT includes the network delay and the server processing delay [6] as shown in figure 2.16

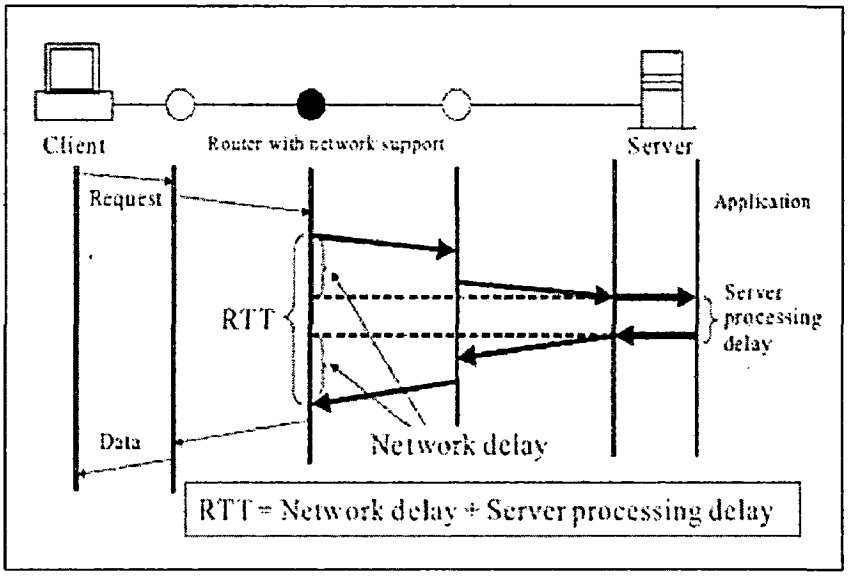

Figure 2.16 RTT Passive Measurement [6]

In some situations, server latency may be dominant due to the load on the particular server. In this case, user response time may be improved by selecting a server that is lightly loaded. In another situation, network delay may be dominant due to congestion. In this case, network delay should be used for server selection. To realize good selection of servers, we believe that both server processing delay and network delay should be taken into account. To do so, we use the application layer RTT between the router and the server as information for server selection [6]. When a router in a network selects a server which has small application layer RTT, total response time can be improved. Now the question arises if router selects a server based on its application layer RTT, client request may have a tendency to concentrate at a particular server. To avoid this situation, we apply probabilistic server selection policy [6] at the router. Selection probability of the server whose RTT is large should be small and the server whose RTT is small should be selected with large probability. The following simple method for calculation of the server selection probability. A router i calculates $P_{ij,}$, a probability of selecting server j, as

$$P_{ij} = \frac{\frac{1}{RTT_j}}{\sum_{m=1}^{n}\frac{1}{RTT_m}} \qquad [6]$$

Where n is total number of servers serving the same service and RTT_m is the RTT between router i and server m.

Chapter 3

Proposed Algorithm for load balancing

To improve the existing technique, it has been suggested to include a new adaptive load balancing algorithm (I) where the concept of rtt passive measurement technique and content awareness has been used. Content awareness means having different queue for different request types rather than having the same queue for different requests, which improves the total execution time. The new adaptive load balancing algorithm (I) is implemented for the servers inside the cluster, this algorithm improves the total execution time as compare to static and dynamic algorithm. The advantage of having cluster architecture is it provides high availability, high reliability and high scalability. So to improve the execution time of the cluster architecture along with new adaptive load balancing algorithm (I) that would be implemented inside the cluster, a new adaptive load balancing algorithm (II) would also be implemented outside the cluster to improve total execution time. Proposed architecture and algorithm are as follows.

3.1 New Adaptive Load Balancing Algorithm (I)

3.1.1 Architecture

As described earlier, Load balancing algorithms can be classified as either static or dynamic. Unfortunately these methods generates additional processing load on the server, deteriorating its performance. The function of a web server is to service HTTP requests made by client. Typically the server receives a request asking for a specific resource, and it returns the resource as a response. A client might reference in its request a file, then that file is returned or, for example, a directory, then the content of that directory (codified in some suitable form) is returned. A client might also request a program, and it is the web server task to launch that program (CGI script) [7] and to return the output of that program to the client. Various other types of resources might be referenced in client's request. Different request have different sizes thus required server time is different for them.

Client request is transferred to the web server via router, and the router distinguishes the request on the basis of content. As shown in Figure 3.1, each back end server keeps queues for handling each type of request. This database is maintained by the router for each server and router transfers the request to a particular queue of a server after rtt passive measurement. . There is a common module for all the servers which we call as web proxy. This is the main improvement over other adaptive load balancing algorithms. It performs the load balancing task by running the technique and thus improves efficiency.

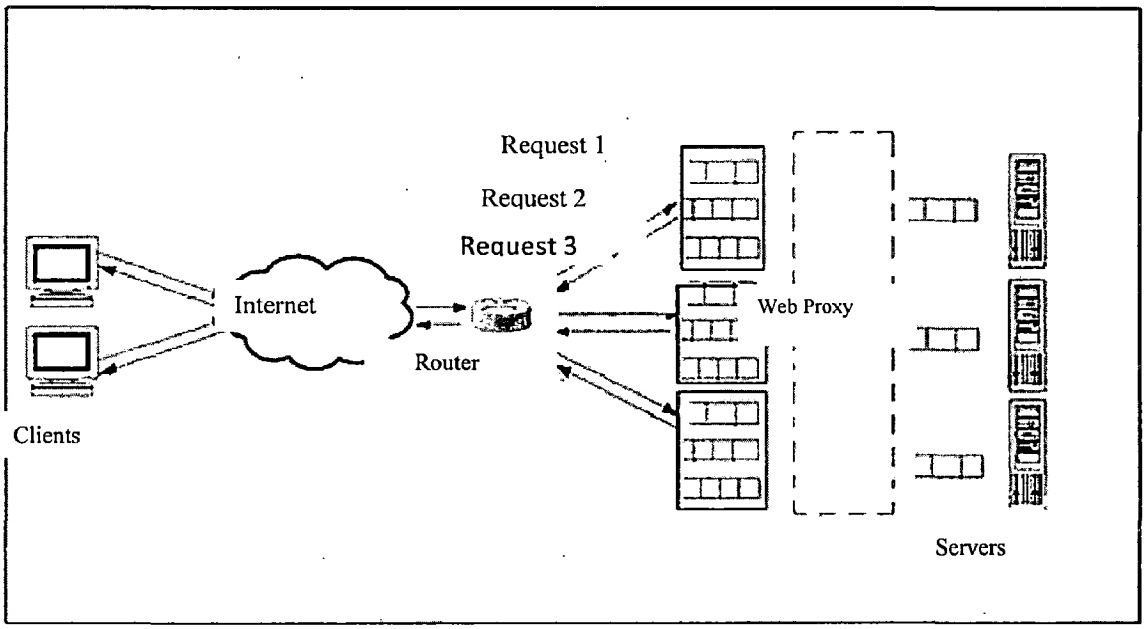

Figure 3.1 Architecture of Proposed Adaptive Load Balancing Algorithm (I)

RTT passive measurement technique that is used here is similar to the technique described above for content routing based dynamic load balancing method.

Before describing the algorithm, first let us understand what is exactly mean by content awareness and its need. Suppose there are three servers in the system and each can handle three different type of request i.e. video, audio, and image. The current load of each of the servers is as follows.

Server 1 : 3 video requests.

Server 2 : 3 audio requests.

Server 3: 3 image requests.

Consider just the length of the queues then all have same length but if the request is forwarded to server 1 then it will result in higher response time. So the technique that has been introduced here is to have different queue for different kind of request and whenever a server gets overloaded due to the requests on a particular queue then transfer the request from this server to the same queue of other server thus balancing the load.

3.1.2 Algorithm (I)

This is the adaptive load balancing algorithm run by web proxy, common for all the servers. So it has access to the entire server's queue and can update them as needed.

Let there be n Servers and each server has m queues, where each queues is for a different types of request. Each type of request is of different size e.g. Videos are of size say x units whereas audio are of size say y units whereas images are of z units (much smaller than x and y) and so on. Define the initial size of the queue which will change according to the current state of the system.

Define $load_i$ where $1 \leq i \leq n$, load on i_{th} server.

Max_j where $1 \leq j \leq m$, maximum load in j_{th} queue.

Min_j where $1 \leq j \leq m$, minimum load in j_{th} queue.

Initialize limit value by taking the average of the length of all the m queues on n server and divide it into the half. If value of $load_i$ is greater than limit value then transfer job from queue of that server to same type of queue of other server. It may also possible that taken the full average value and compared load with it, but it will be the situation where server is already overloaded and then measures have to be taken to transfer its load and make it under loaded or lightly loaded. In our case such a situation is not allowed to occur because here precautionary measures are taken i.e., divide the average into half and transfer the load well before any overloaded condition.

Algorithm for webproxy

Steps

1. For each i and j initialize $load_i$, Max_j, Min_j to 0.
2. Repeat step 3 to 6, till server is on
3. If ($load_i$<limit)

Converts the m queues into the single queue on the basis of time and server serves the request in FCFS form.

4. If ($load_i \geq$limit) Repeat Step 5 until ($load_i$<limit)
5. For each queue (from 1 to m) calculate max_j and min_j and transfer the request from queue j of server x having maximum load to queue j of server y having minimum load.
6. Go to step 3.
7. Exit

Flowchart of proposed algorithm is shown in figure 3.2.

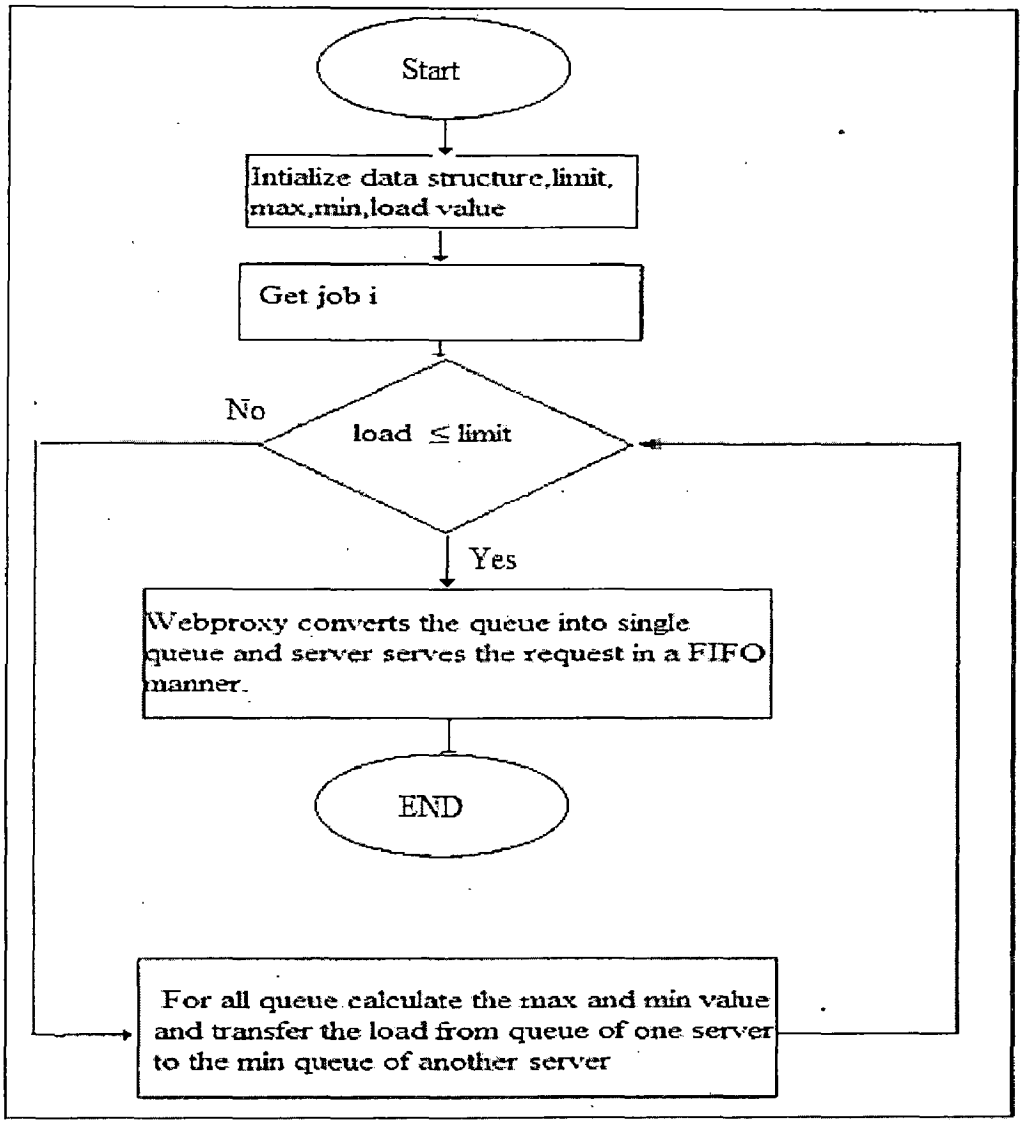

Figure 3.2 Flow Chart of Proposed Adaptive Load Balancing Algorithm (I).

3.2 New Adaptive Load Balancing Algorithm (II)

3.2.1 Architecture

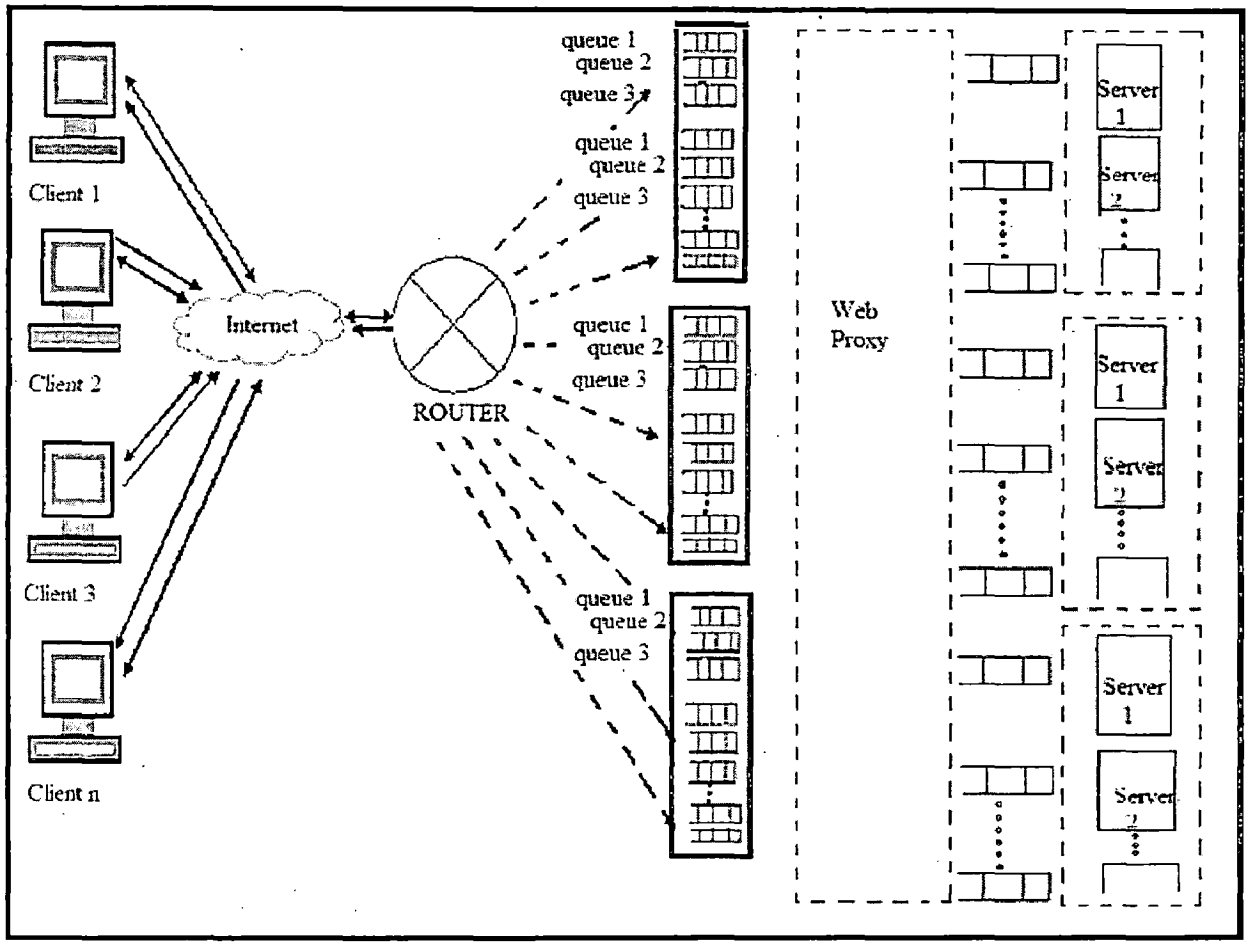

Figure 3.3 Architecture of Proposed Adaptive Algorithm (II).

The concept of cluster is used to provide higher availability,reliability and scalability than can be obtained by using a single system. Benefit of having cluster architecture is, it provides high availability by making application software and data available on several servers linked together in a cluster configuration. If one server stops functioning, a process called failover automatically shifts the workload of the failed server to another server in the cluster. The failover process is designed to ensure continuous availability of critical applications and data. Clusters can be used to solve three typical problems in a data center environment: Need for High Availability, High Reliability and High Scalability.

So the concept of cluster has been introduced with our previously proposed architecture. In the architecture (II), again the concept of content awareness along with RTT passive measurement is used. Here also Client request is transferred to the web server via router, and the router distinguishes the request on the basis of content. As shown in Figure 3.3, there are clusters of server; each back end server inside the cluster keeps queues for handling each type of request. This database is maintained by the router for each server and router transfers the request to a particular queue of a server after rtt passive measurement. There is a common module for all the servers which we call as web proxy. This is the main improvement over other adaptive load balancing algorithms. It performs the load balancing task by running the algorithm (explained in the next section) thus freeing the servers from performing load balancing technique and thus improves efficiency.

3.2.2 Algorithm (II)

The above mentioned proposed adaptive load balancing algorithm (I) run by webproxy, common for all server inside the cluster, is used to perform load balancing inside the cluster and the new proposed load balancing algorithm (II) common for all the cluster is used to perform load balancing outside the cluster. So webproxy has access to the entire server's queue and can update them as needed.

Let there be n Servers and each server has m queues, where each queues is for a different types of request. Each type of request is of different size e.g. Videos are of size say x units whereas audio are of size say y units whereas images are of z units (much smaller than x and y) and so on. Define the initial size of the queue which will change according to the current state of the system.

Define $load_i$ where $1 \leq i \leq n$, load on i_{th} server inside the cluster.

$Load_c$ where $1 \leq c \leq k$, load on c_{th} cluster.

Max_j where $1 \leq j \leq m$, maximum load in j_{th} queue.

Min_j where $1 \leq j \leq m$, minimum load in j_{th} queue.

Initialize $limit_s$ value by taking the average of the length of all the m queues on n server and divide it into the half. If value of $load_i$ is greater than limit value then transfer job from queue of

that server to same type of queue of other server. It may possible that taken the full average value and compared load with it, but it will be the situation where server is already overloaded and then measures have to be taken to transfer its load and make it under loaded or lightly loaded. In our case such a situation is not allowed to occur because we take precautionary measures i.e., divide the average into half and transfer the load well before any overloaded condition. The same precautionary would be taken for cluster architecture also. Let ∂ the value by taking the average of the length of all the m queues on n server of k cluster. Let cluster C_r, $0 \leq r \leq k$, has maximum number of server say α and total number of server in a cluster C_r is β_r.

Initialize $limit_c$ value by following formula

$Limit_c = (\partial/\alpha) * \beta_r$ where $0 \leq r \leq k$.

Algorithm for webproxy

Steps

1. For each i and j initialize $load_i$, Max_j, Min_j to 0.
2. Repeat step 3 to 6, till server is on
3. If ($load_i < limit_s$)

Converts the m queues into the single queue on the basis of time and server serves the request in FCFS form. Go to step 6.

4. Else ($load_i \geq limit_s$) Repeat Step 5 to 6 until ($load_i < limit_s$)
5. For each queue (from 1 to m) calculate max_j and min_j and transfer the request from queue j of server x having maximum load to queue j of server y having minimum load.
6. if ($load_c > limit_c$) Repeat Step 7 to 8 until ($load_c < limit_c$)
7. For each queue (from 1 to m of cluster c1 and c2) calculate max_j and min_j and transfer the request from queue j of server x of cluster c1 having maximum load to queue j of server y of cluster c2 having minimum load.
8. Go to step 3.
9. Exit

Flow chart of proposed adaptive algorithm (II) is shown below.

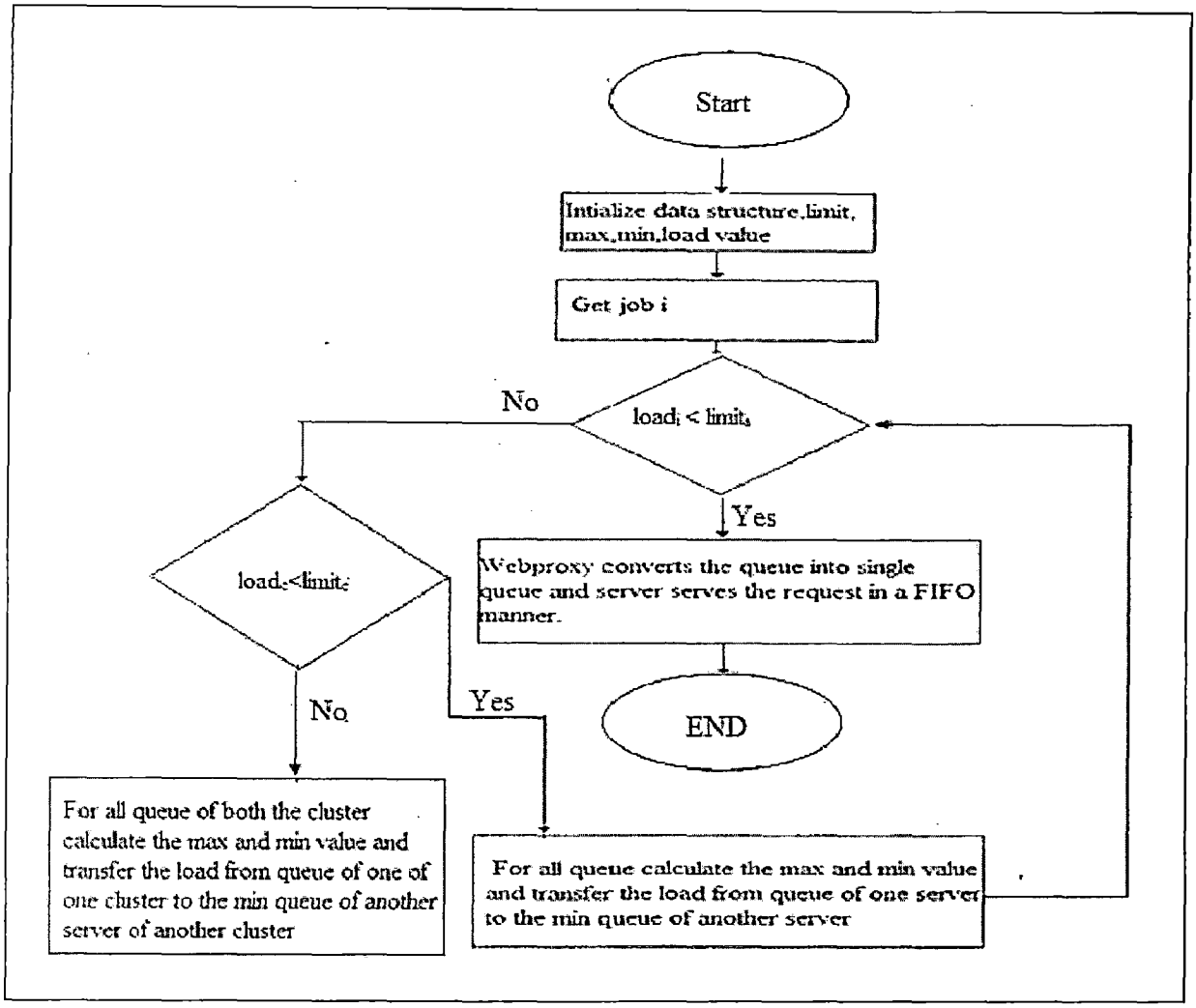

Figure 3.4 Flow chart of Proposed Adaptive Algorithm (II)

Chapter 4.

Implementation Detail and Experimental Result

4.1 Implementation detail and experimental result of algorithm (I)

A. *Simulation*

We have implemented the adaptive algorithm (I) in netbeans 7.1 using multithreading. Each thread created corresponds to a particular server. We designed a client module that generates requests from the web and handed over to the router. The generated requests are of different types. On the basis of type of request, router puts them in an appropriate queue. Router transfers the request to the webproxy where the entire relevant load balancing task is performed. We have also implemented the static algorithm based on round robin technique [9], randomized technique [11], threshold technique [14] ,dynamic algorithm based on load balancing by content based routing technique [6] and modified centralized approach for dynamic load balancing algorithm[20] for making necessary comparison.

B. Simulation Results

For simulation, we have used the request from web and these requests are of different sizes and types. Algorithm 1 is round robin static algorithm, Algorithm 2 is randomized static algorithm, Algorithm 3 is threshold static algorithm, Algorithm 4 is content based routing algorithm , Algorithm 5 is modified centralized approach for dynamic load balancing algorithm and Algorithm 6 is proposed adaptive load balancing algorithm (I). We compare these algorithms on the basis of load distribution among servers and total response time.

C. Analysis of Results

From Figure 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 it may be concluded that the load distribution is fairer in Algorithm 5 i.e. in adaptive algorithm (I).

In case of round robin static algorithm, request are transferred from server 1 to m, where m is the total server and when server m is reached process is repeated but it may happen that large size request are concentrated over a particular server so that server might get overloaded. As shown in figure 4.1, Server 3 get overloaded. This algorithm works fine for same type of request.

In case of randomized static algorithm, server is selected randomly, as shown in figure 4.2 most of the time server 3 is selected by random method, it will result into the overloaded server.

In case of Threshold based static algorithm, as shown in figure 4.3 it work fine under normal condition till threshold value is not reached, but it may happen that threshold value is reached for all the server at the same time so it will result into the overloaded server.

But all the static method work fine for same type of request. For different type of request, load distribution is not balanced.

In case of Content based routing dynamic algorithm, as shown in figure 4.4 server is selected on the basis of their response time. But in real time scenario both network delay and processing delay is important. But the main emphasis of this algorithm is on network delay. So it will result in to the overload on the server having minimum network delay.

In modified centralized approach for dynamic load balancing algorithm, as shown in figure 4.5 supporting nodes are given some load initially, when load on server exceed, load is transferred to supporting nodes and supporting nodes process the request on the basis of its priority. In this method although the load at server remains balanced but load on supporting node increases with increase in number of requests.

Proposed Adaptive Algorithm (I) as shown in figure 4.6, works fine under the both scenario for same as well as for different type of request and also make decision on the basis of both network and processing delay. As compared to other algorithm for same number and type of request load distribution is fairer.

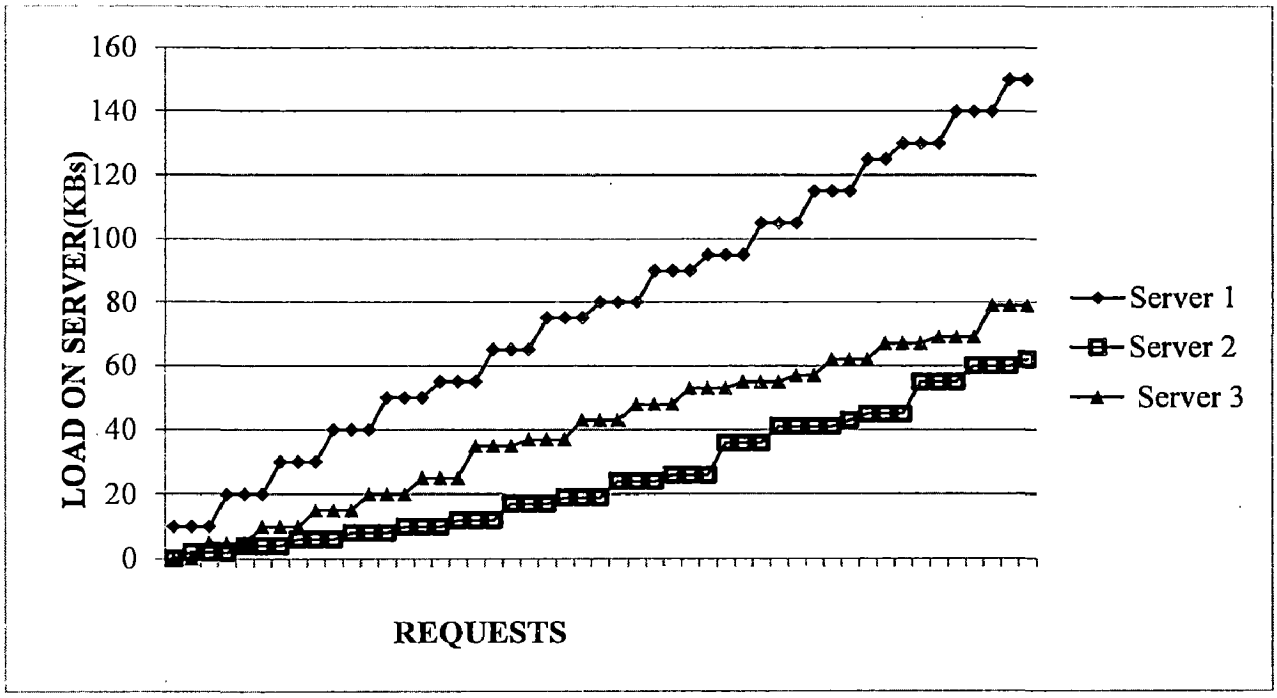

Figure 4.1 Graph Showing Simulation Result For Round Robin Static Algorithm.

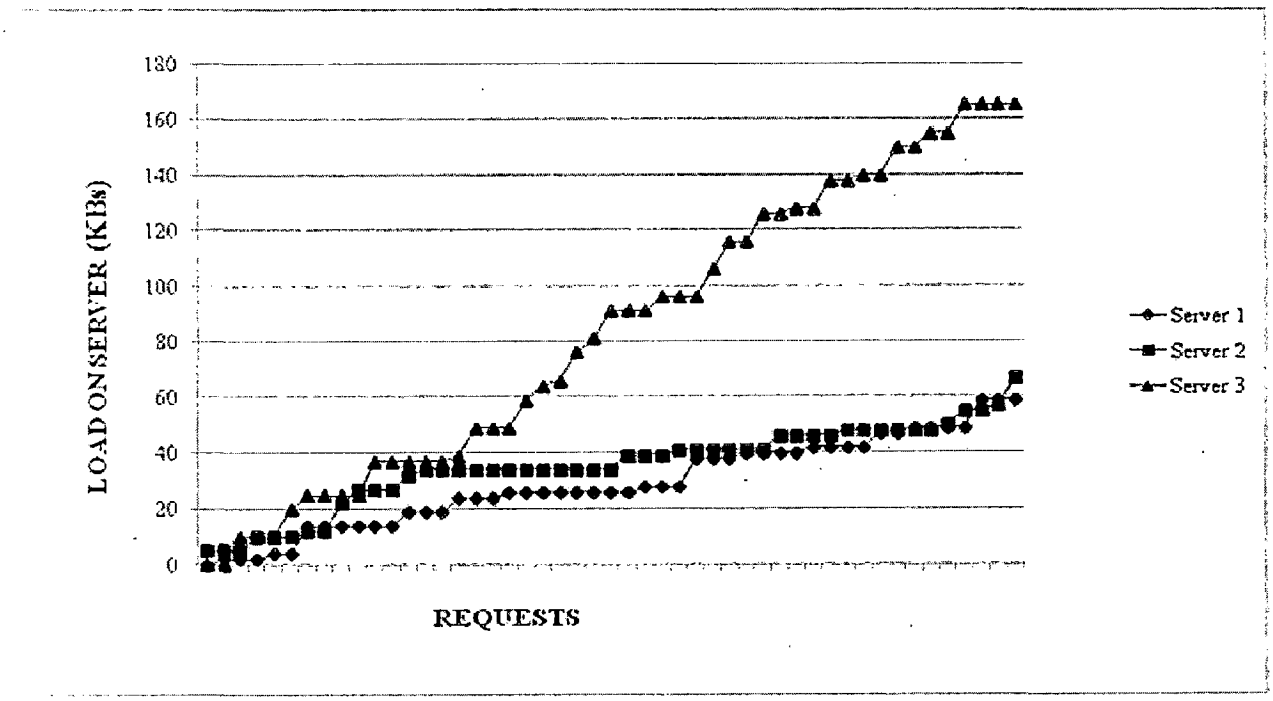

Figure 4.2 Graph Showing Simulation Result For Randomized Static Algorithm.

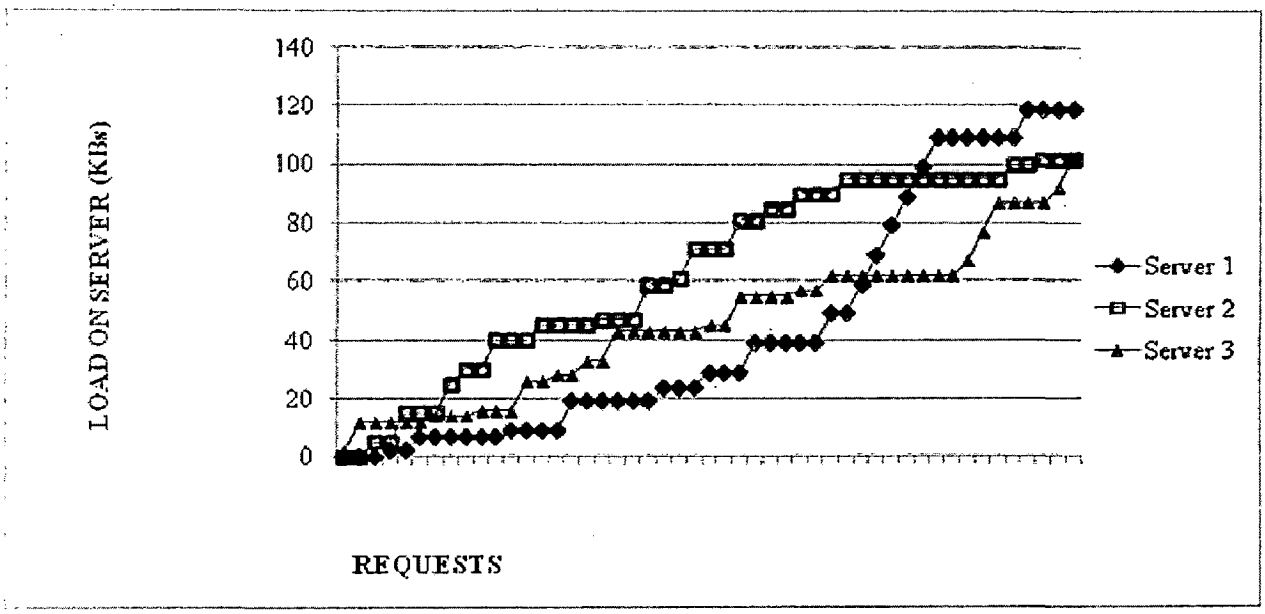

Figure 4.3 Graph Showing Simulation Result for Threshold Based Static Algorithm.

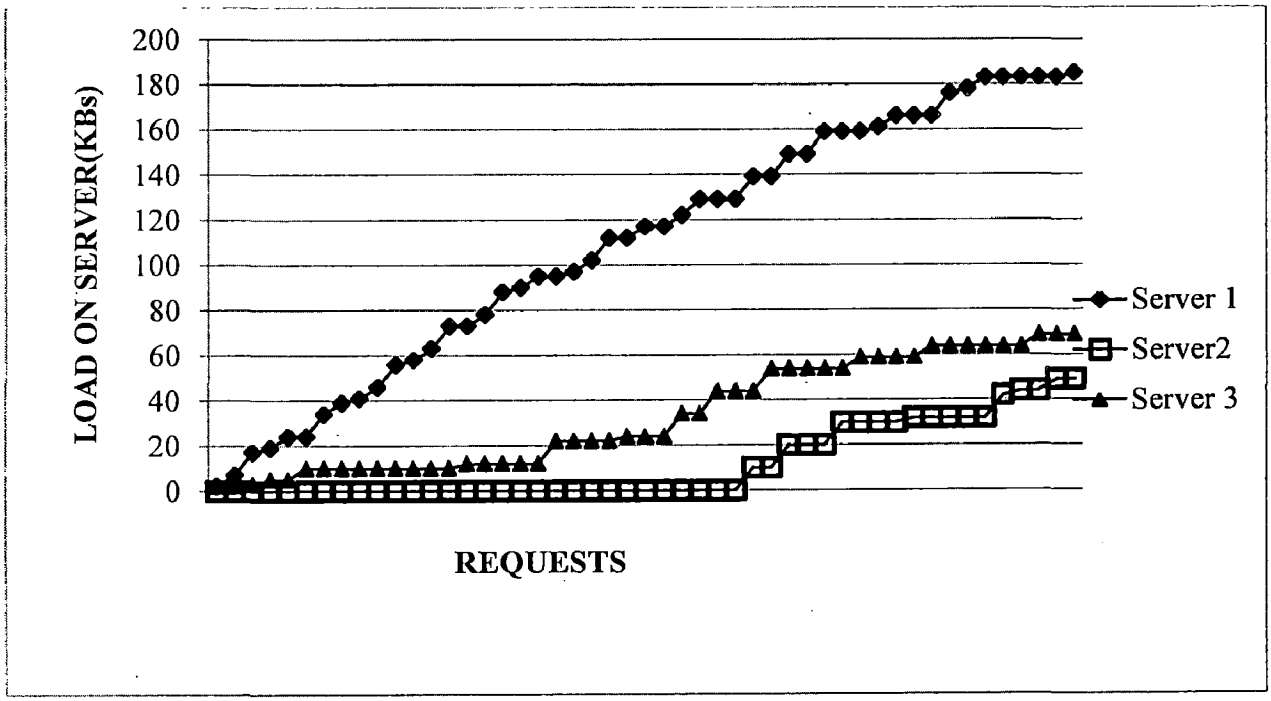

Figure 4.4 Graph Showing Simulation Result for Content Routing Based Dynamic Algorithm.

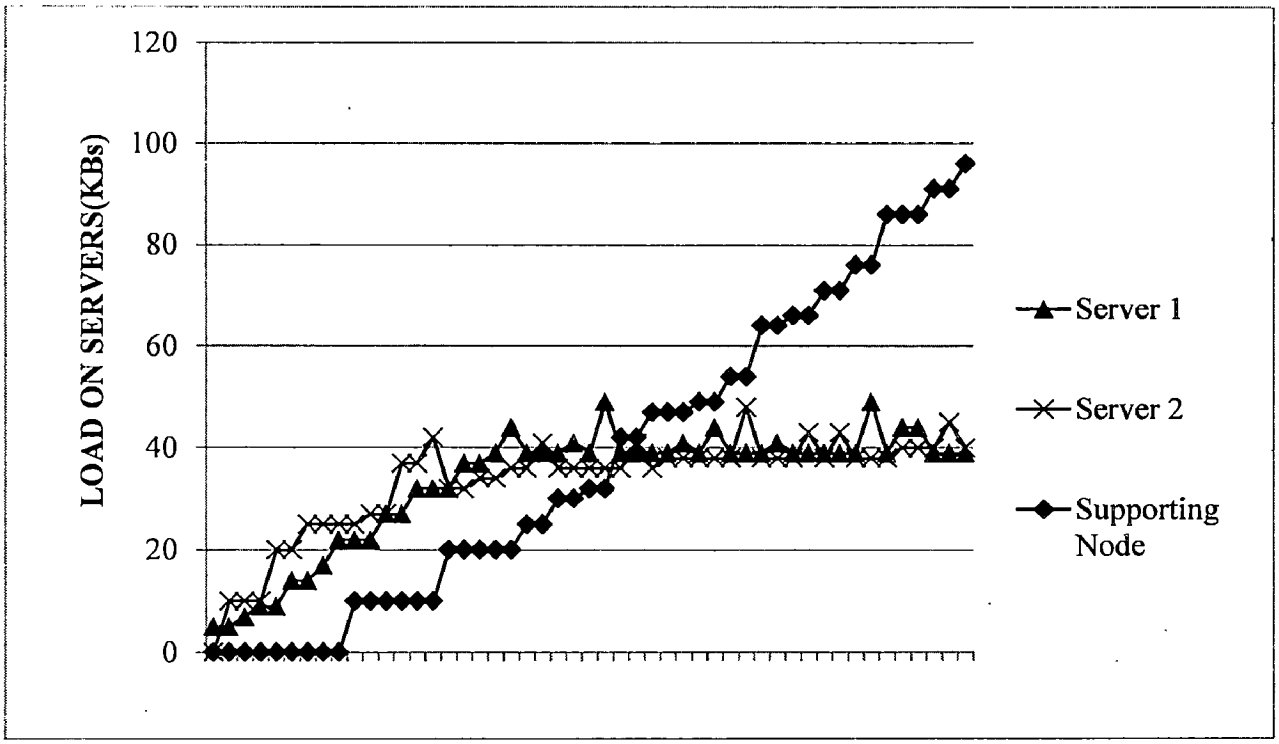

Figure 4.5 Graph Showing Simulation Result for Modified Centralized Dynamic Load Balancing Algorithm.

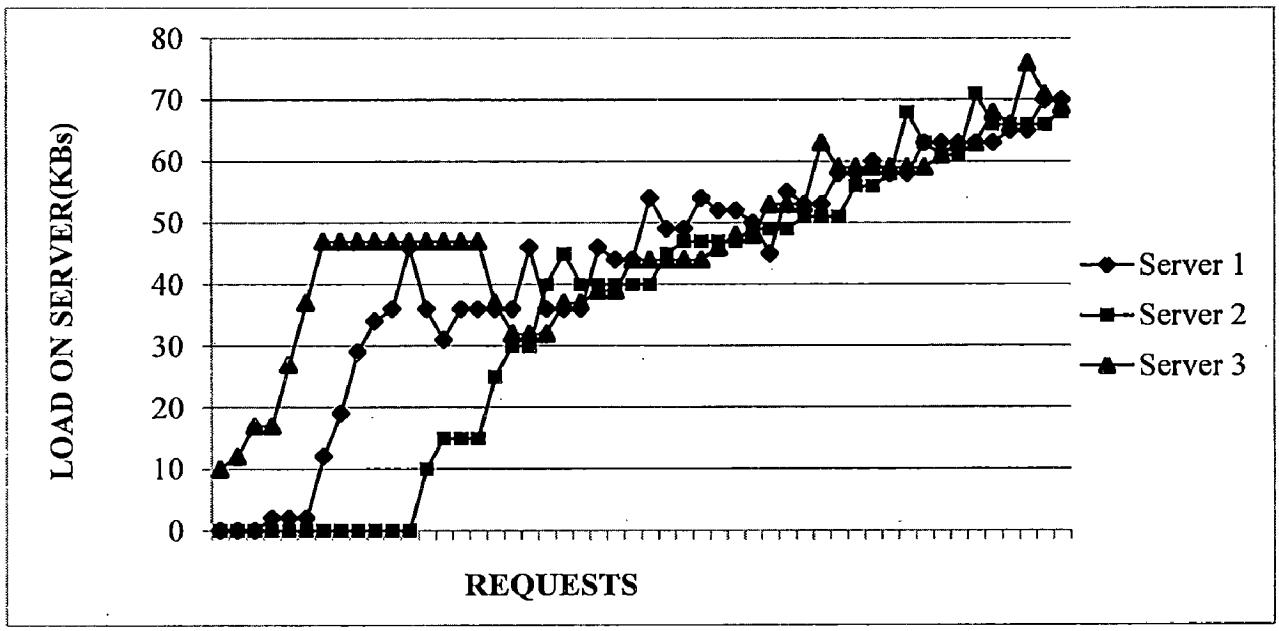

Figure 4.6 Graph Showing Simulation Result for Proposed adaptive load balancing algorithm (I)

Initially response time of Round robin algorithm is less or same as proposed adaptive algorithm (I), as shown in figure 4.7. But as the load increases response time of round robin algorithm increases exponentially.

Random static algorithm works fine till load on the servers is balanced, as shown in figure 4.8 as the load increases response time of the algorithm increases.

Threshold based static algorithm works fine till threshold value is not reached, as shown in figure 4.9 but if all the servers reached their threshold value then it will result into the overloaded server and hence response time increases.

Total response time of modified centralized approach for dynamic load balancing algorithm is less compared to proposed adaptive algorithm (I) for less number of requests, when the load on supporting nodes increases total response time of modified centralized approach for dynamic load balancing algorithm increases, as shown in figure 4.10.

For the less number of requests response time of content based routing dynamic algorithm is less compared to proposed adaptive algorithm (I),as shown in figure 4.11 but for the overloaded condition response time of proposed algorithm is less compared to content based routing dynamic algorithm.

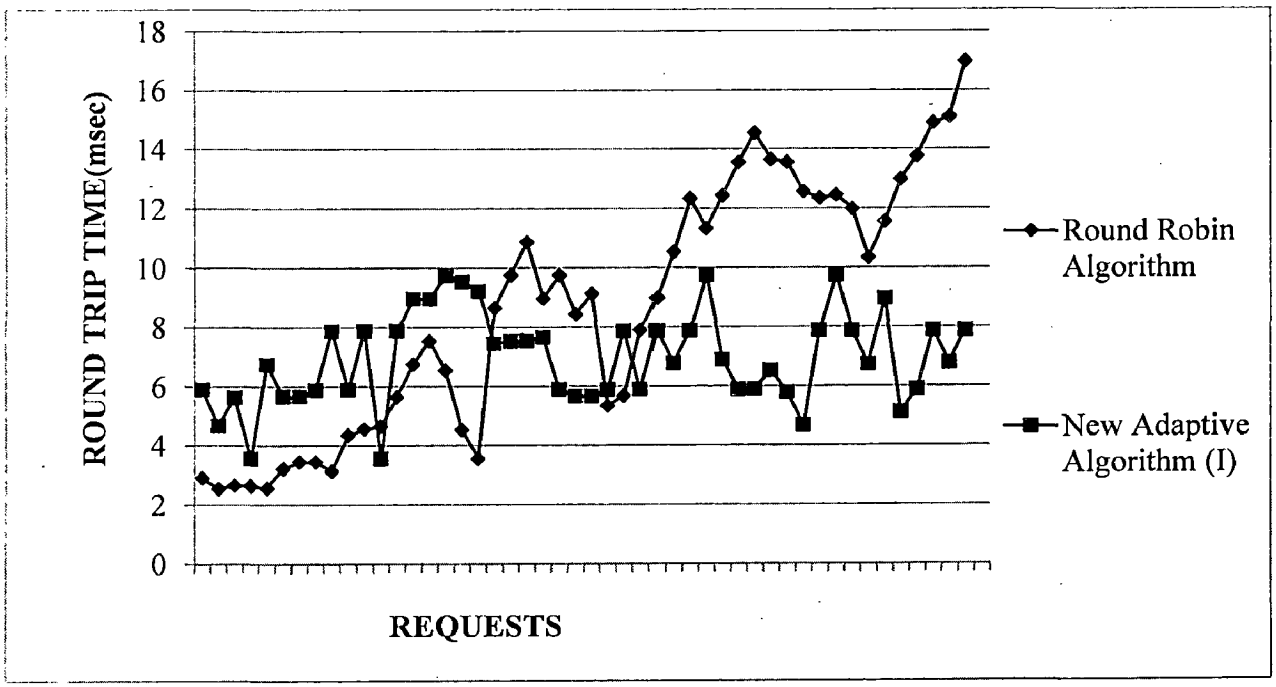

Figure 4.7 Total Response Time Comparison of Round Robin Algorithm And New Adaptive Algorithm(I)

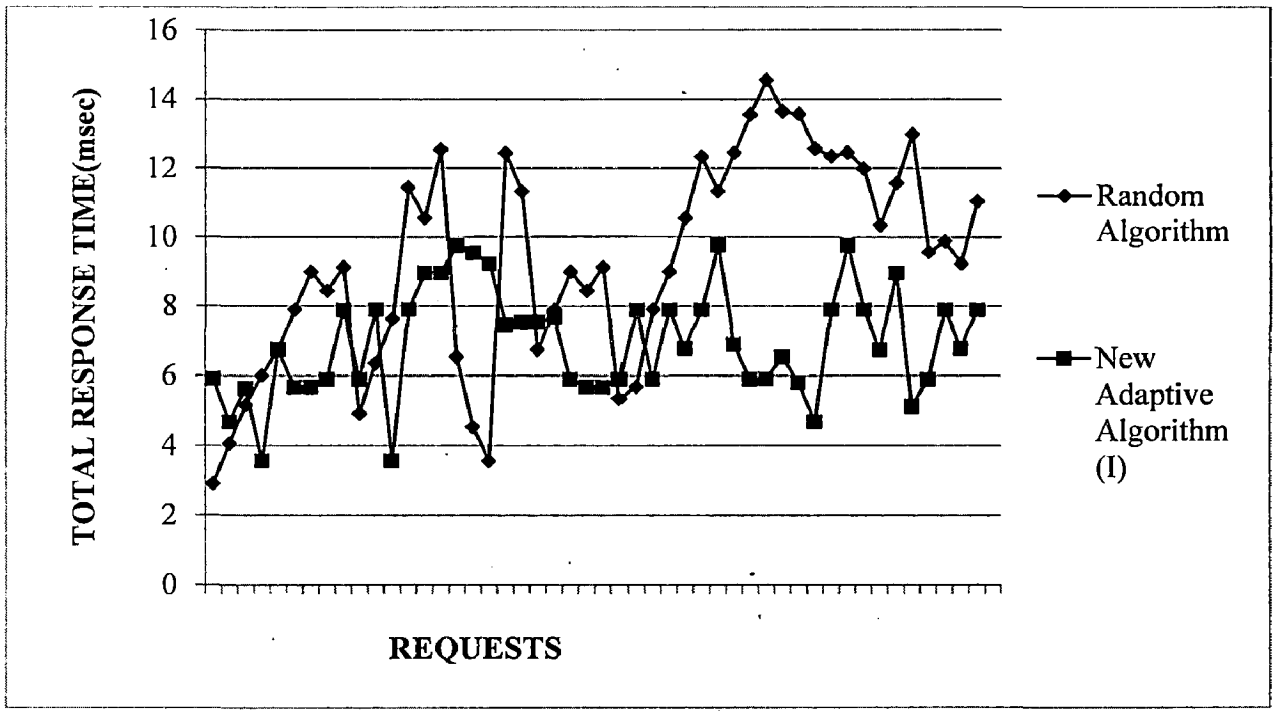

Figure 4.8 Total Response Time Comparison of Random Algorithm And New Adaptive Algorithm(I)

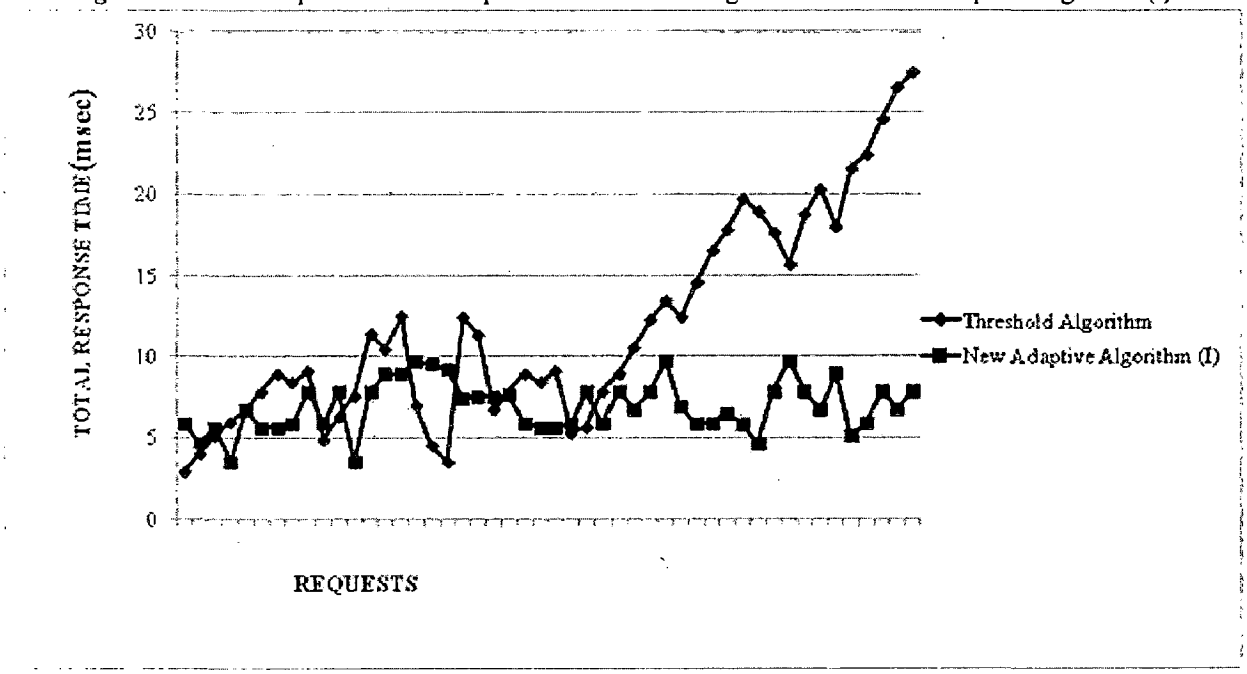

Figure 4.9 Total Response Time Comparison of Threshold Based Algorithm And New Adaptive Algorithm(I)

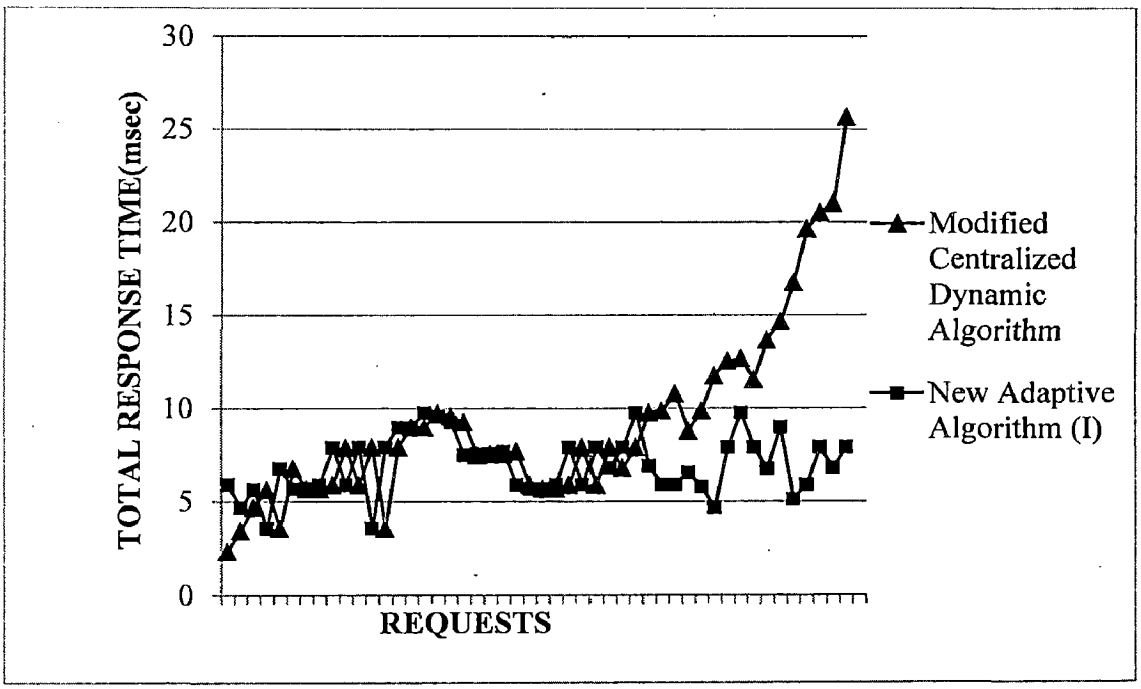

Figure 4.10 Total Response Time Comparison of Modified Centralized Dynamic Algorithm And New Adaptive Algorithm(I).

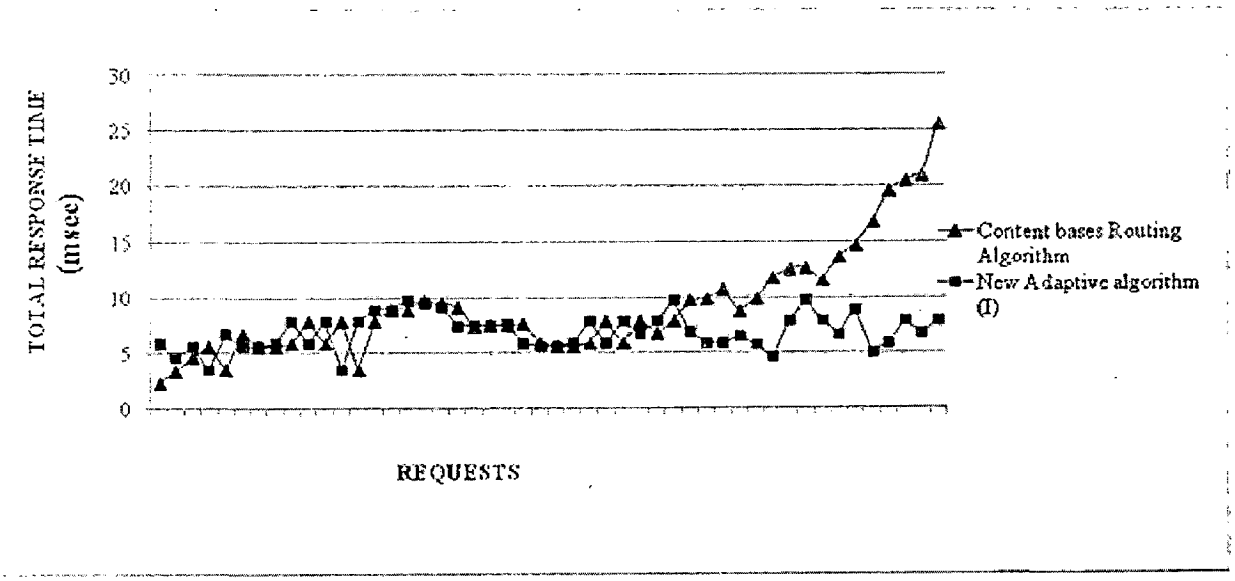

Figure 4.11 Total Response Time Comparison of Content Based Routing And New Adaptive Algorithm(I)

4.2 Implementation detail and experimental result of algorithm (II)

A. *Simulation*

We have implemented the adaptive algorithm (II) in netbeans 7.1 using multithreading. Each thread created corresponds to a particular server. We designed a client module that generates requests from the web and handed over to the router. The generated requests are of different types. On the basis of type of request, router puts them in an appropriate queue. Router transfers the request to the webproxy where the entire relevant load balancing task is performed. We have also implemented the Dynamic load balancing algorithm for web cluster [19] and centralized dynamic cluster based load balancing algorithm [21].

B. Simulation Results

For simulation, we have used the request from web and these requests are of different sizes and types. We compare these algorithms on the basis of load distribution among servers and total response time.

C. Analysis of Results

From Figure 4.12, 4.13, 4.14 it may be concluded that the load distribution is fairer in proposed Algorithm (II).

In dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness, this algorithm identifies the category of request and request is forwarded to the least loaded heterogeneous server, as shown in figure 4.12. But the performance of algorithm degrades when number of request of same type increases as it will get concentrated over a particular heterogeneous cluster of server while other server remain under loaded.

In case of centralized dynamic cluster based load balancing algorithm. Balancer inside the cluster perform load balancing for servers inside the cluster and balancer outside the clusters perform load balancing for clusters, as shown in figure 4.13.The main disadvantage of the centralized dynamic cluster based load balancing algorithm is bottleneck of balancer.

In proposed adaptive load balancing algorithm (II), it neither uses concept of heterogeneous server nor the concept of centralized adaptive algorithm. As shown in figure 4.14 , it is concluded that performance of proposed adaptive load balancing algorithm is fairer as compare to other two algorithm.

For the total response time, initially for both dynamic load balancing for scalable heterogeneous web cluster algorithm and centralized dynamic cluster based load balancing algorithm total

response time is less as compare to proposed algorithm (II).As shown in figure 4.15, when the number of requests increases there is slight increase in response time of proposed algorithm(II) but tremendous increase in response time of both the algorithm.

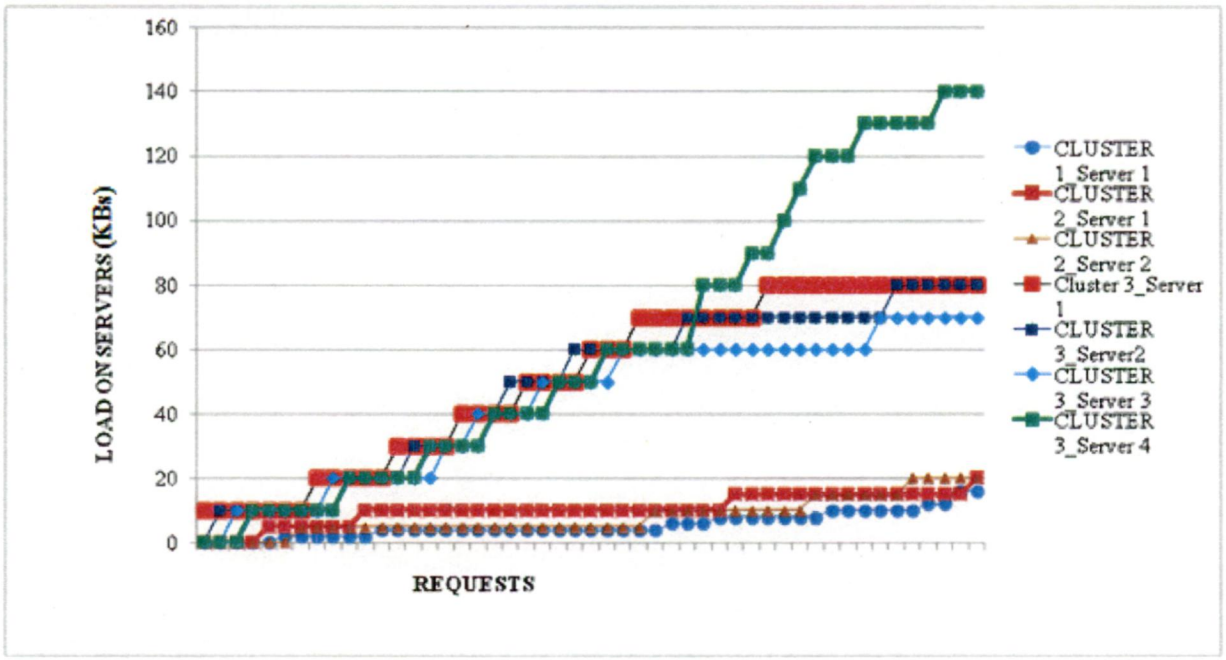

Figure 4.12 Graph Showing Simulation Result for Dynamic load balancing algorithm for heterogeneous web server cluster.

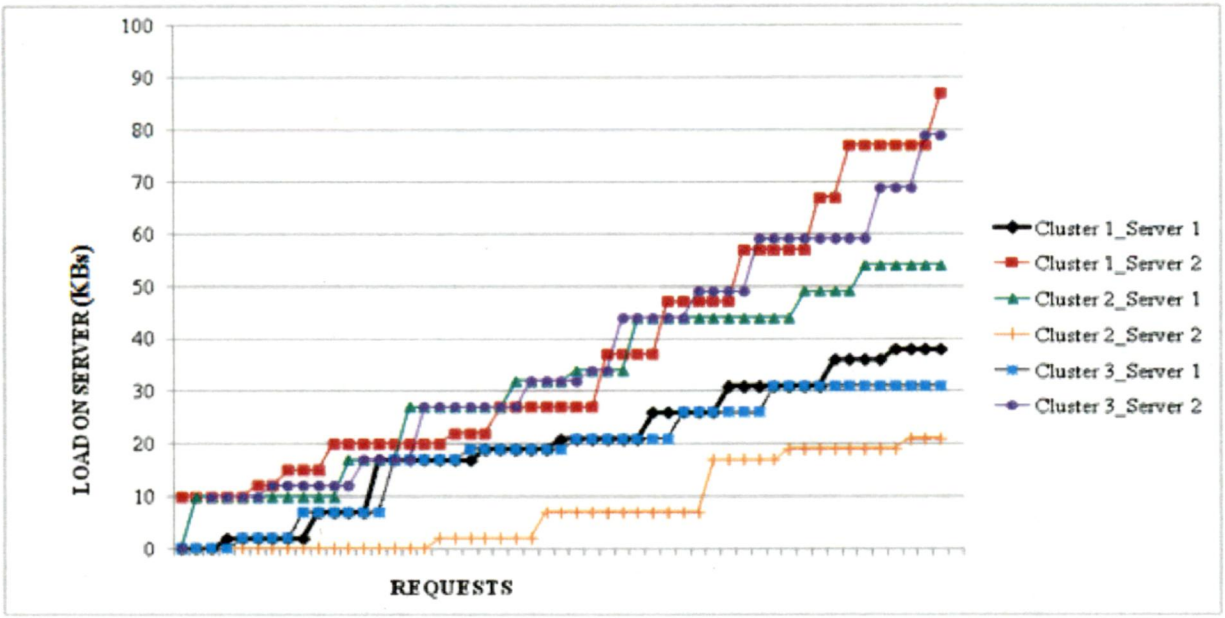

Figure 4.13 Graph Showing Simulation Result for Centralized dynamic cluster based load balancing algorithm

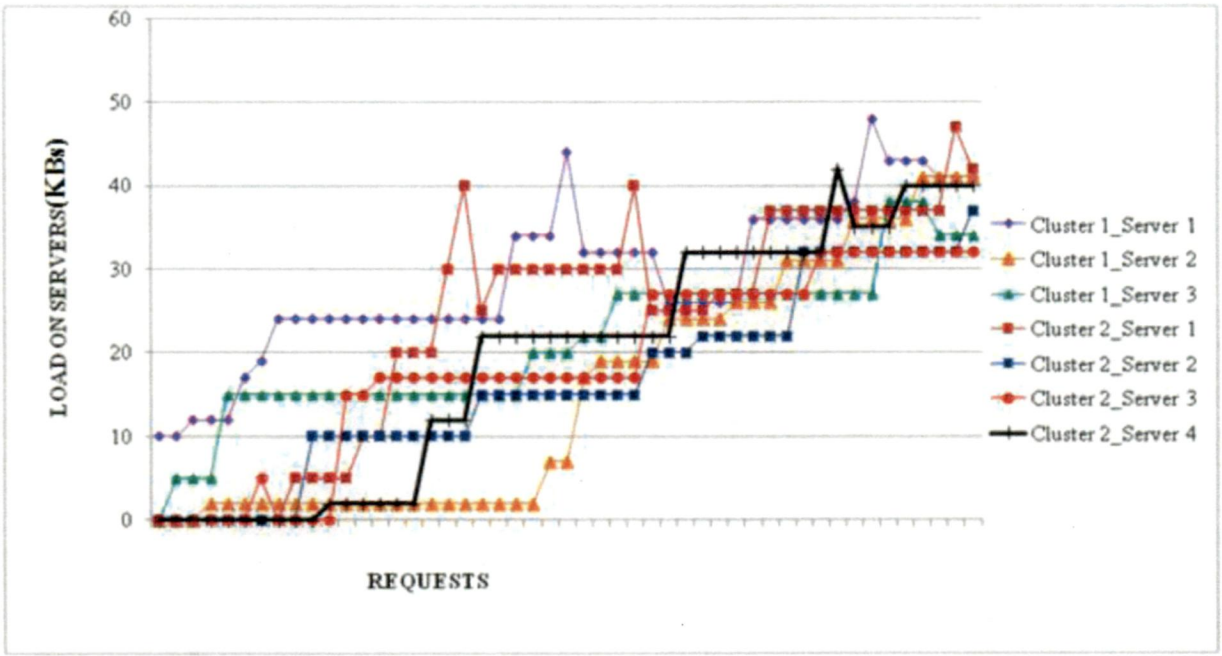

Figure 4.14 Graph Showing Simulation Result for Proposed Adaptive Algorithm(II)

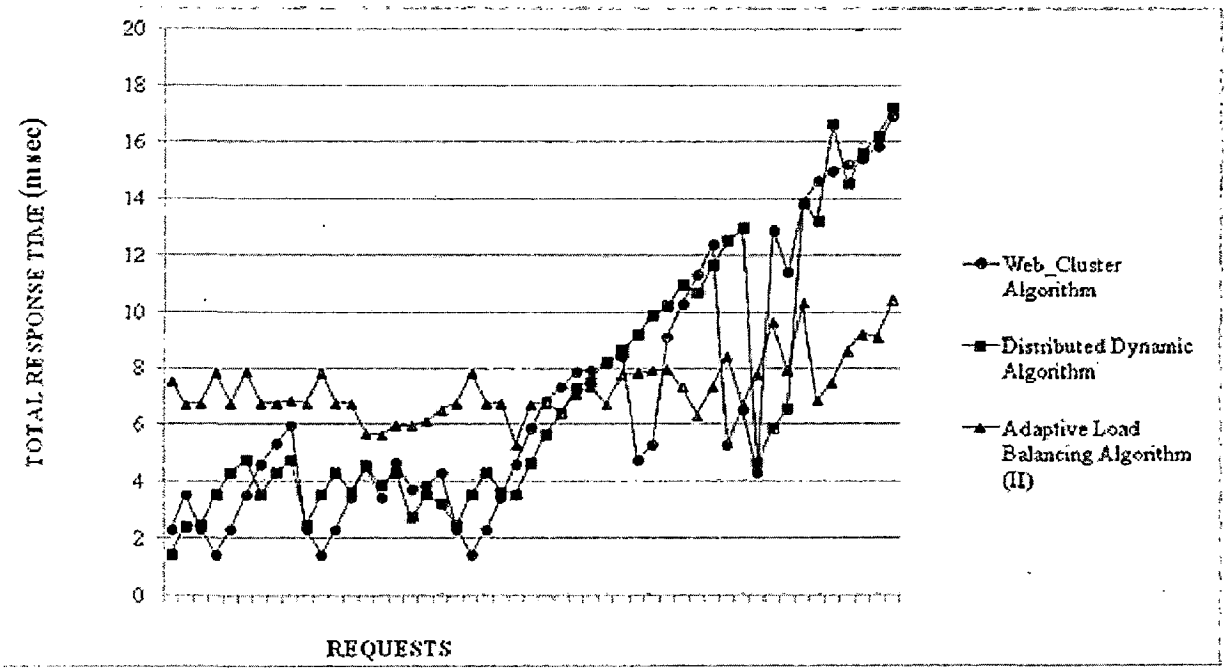

Figure 4.15 Total Response Time of Above Three Algorithm

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have implemented adaptive load balancing with some enhancements and concluded its valuable characteristics, which are as follows:

- Used the concept of Rtt passive measurement technique. It select the server on the basis of minimal Rtt. Thus improving the reliability of the whole system.
- We introduce content awareness. By content awareness we mean having different queue for different request types rather than having the same queue for different requests, which improves the total execution time
- We introduced the new adaptive load balancing algorithm (I) inside the cluster. Its performance is fairer as compared to other proposed algorithm.
- We introduced the new adaptive load balancing algorithm (II) outside the cluster and combine this algorithm with adaptive load balancing algorithm (I) inside the cluster along with rtt passive measurement technique for improving the overall performance of the system.

5.2 Future work

In future we are trying to deploy these algorithms over real time system where response time has a crucial point for different data source (Video, audio, Image, Text). In the future, distributed application frameworks will support mobile code, multimedia data streams, user and device mobility, and spontaneous networking so among these recent futures computing, fare load distribution is a vital demand and we have to modify and enhance different aspect of our proposed algorithm to sustain these demands.

References

[1] G. *Coulouris, J.Dollimore*, T. Kindberg, and G. Blair. "*Distributed systems: concepts and design*", Addison-Wesley Longman, 2005.

[2] A. Tanenbaum and M.V. Steen. "*Distributed Systems: Principles and Paradigms*", Prentice Hall, Pearson Education, USA, 2002.

[3] R. Miller. "Facebook Server Count : 60,000 or more", Internet: *http://www.datacenterknowledge.com/archives/2010/06/28/facebook-server-count-60000-or-more/*, June 28, 2010.

[4] D.Grosu,and A. T. Chronopoulos. "Algorithmic mechanism design for load balancing in distributed systems." *IEEE Transactions on Systems, Man, and Cybernetics- Part B: Cybernetics,* vol. 34, no. 1, pp. 77-84, 2004.

[5] N.G.Shivaratri, P. Krueger, M. Singhal. "Load distributing for locally distributed system.", *Computer*, vol. 25, no.12, pp. 33-44, 1992.

[6] H.Miura, M.Yamamoto. "Content Routing with Network Support Using Passive Measurement in Content Distribution Networks.", *IEICE Transation on Communications,* Special Issue on Content Delivery Networks E86-B(6), pp. 1805-1811, 2003.

[7] D. Grosu and A. T. Chronopoulos. 'A game-theoretic model and algorithm for load balancing in distributed systems.", *In Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium*, pp. 146-153, Ft Lauderdale, Florida, USA, 2002.

[8] K. Benmohammed-Mahieddine, P. M. Dew, and M. Kara. "A periodic symmetrically-initiated load balancing algorithm for distributed systems.", *In Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pp. 616–623, Norwood, June 1994.

[9] Zhong Xu, Rong Huang, "Performance Study of Load Balancing Algorithms in Distributed Web Server Systems", *CS213 Parallel and Distributed Processing Project.*

[10] H.Rahmawan and Y. S. Gondokaryono. "The simulation of static load balancing algorithms," IEEE., 2009 *International Conference on Electrical Engineering and Informatics* (IEEE 2009), pp. 640-645, Selangor, Malaysia, August 2009.

[11] R.Motwani and Raghavan. "Randomized Algorithms.", *ACM Computing Surveys*, vol.28, no. 1, pp. 33-37, 1996.

[12] P. L. McEntire, J. G. O'Reilly, and R. E. Larson, "*Distributed Computing:Concepts and Implementations*", New York : IEEE Press, 1984.

[13] S.Sharma, S.Singh and M.Sharma. "Performance Analysis of Load Balancing Algorithms," *World Academy of Science, Engineering, and Technology*, vol. 38, pp. 269-272, 2008.

[14] A. Saxena and D. Sharma," Analysis of Threshold Based Centralized Load Balancing Policy for Heterogeneous Machines ", *International Journal of Advanced Information Technology* (IJAIT), vol. 1, no. 5, pp.39-53, 2011.

[15] H. G. Rothitor. "Taxonomy of dynamic task scheduling schemes in distributed computing systems". *IEE Proc.-Computers and Digital Techniques*, vol. 141, no. 1, pp. 1–10, 1994.

[16] W.Leinberger, G.Karypis, V.Kumar, R. Vishwas "Load Balancing Across Near Homogeneous Multi-Resource Servers",In 9th Heterogeneous Computing Workshop (HCW*2000*), pp. 2-3, Cancun, Mexico, February 16, 2000.

[17] P.Saxena,G.Saxena, S. Kumar, A. Kumar, R. Belwal. "Functioning Analysis of Load Balancing Algorithms in a Distributed Computing Environment" International Journal of Computer Science and Telecommunications (IJCST 2012), vol. 3, no. 1, 2012.

[18] T.Taibi, A.Abid and E.F.E.Azahan. "A Comparison of Dynamic Load Balancing Algorithms." *Jordan Journal of Applied Science Natural Sciences*, vol. 9, no. 2, pp. 125-133, 2007.

[19] A.Tiwari and P.Kanungo. "Dynamic load balancing algorithm for scalable heterogeneous web server cluster with content awareness.", *Trendz in Information Sciences & Computing* (TISC), pp. 143 – 148, Chennai, 2010.

[20] P.Jain,and D. Gupta. "An Algorithm for Dynamic Load Balancing in Distributed Systems with Multiple Supporting Nodes by Exploiting the Interrupt Service.", *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, pp. 232-236, 2009.

[21] S. K. Goyal, R. Patel, and M. Singh. "Adaptive and Dynamic Load Balancing Methodologies For Distributed Environment: A Review," *International Journal of Engineering Science*, vol. 3,no. 3, pp. 1835-1840, 2011.

Publications

[1] Archana Nigam, Dr. Padam Kumar, Anuj Tiwari, Ankita Singhal, " Adaptive Load Balancing for Server Using Traffic Monitoring With Content Awareness ", *In International Conference on Recent Advances in Engineering and Technology (ICRAET 2012),* pp. 109-112, 29-30 April 2012 , Hyderabad, India.

[2] Archana Nigam, Dr. Padam Kumar, Anuj Tiwari, Ankita Singhal, " Adaptive Load Balancing for Server Using Traffic Monitoring With Content Awareness" In the Special Issue of *International Journal of Systems, Algorithms and Applications (IJSAA 2012),* vol. 2, issue. ICRAET12, pp. 13-16, May 2012.

[3] Archana Nigam, Tejprakash Singh, Anuj Tiwari, Ankita Singhal, "Adaptive Load Balancing for Cluster Architecture Using Traffic Monitoring With Content Awareness" In *International Journal of Advanced Research In Computer Engineering and Technology* Volume 1, Issue 1, pp. 18-23 March 2012.

[4] Archana Nigam, Dr. Padam Kumar, Ankita Singhal, Anuj Tiwari, "Load Balancing for Clusters Using Traffic Monitoring With Content Awareness", In *Third International Conference on Computing, Communication and Networking Technologies* (ICCCNT) , 2012 India. (communicated)