CHAPTER 3

CONTENT AWARE LOAD BALANCING FOR
HETEROGENEOUS WEB SERVERS

In this chapter,

3.1 Introduction
3.2 Content Aware Load Balancing
3.3 Framework
3.4 Proposed Algorithm
3.5 Experiment and Result Analysis
   3.5.1 Simulation
   3.5.2 Simulation Results
   3.5.3 Analysis of Results
3.6 Summary

3.1 INTRODUCTION

The state-of-art web applications communicate and coordinate with number of
geographically dispersed information sources providing information to huge
number of users. Homogeneous server clusters are not capable of satisfying the
growing demand of such applications that include real time audio and video,
automotive design, climate predictions, aerospace engineering, financial
forecasting, oil and gas discovery etc. Heterogeneity involves handling of low
level interoperability issues e.g. mismatch of hardware, operating platforms,
programming languages, database schema, topology etc. Scalable server cluster
allows addition or deletion of new server machines as the processing load
increases or decreases without disrupting the services. Moreover, it also provides
better reliability by gracefully transferring the load from server which is
unavailable due to failure or for preventive maintenance. Heterogeneity with
scalability makes the system more complex and existing dynamic load balancing (DLB) algorithms are not directly useful for distributed scheduling in such environments. The chapter proposes a DLB algorithm for scalable heterogeneous server cluster using content awareness. The proposed algorithm, considers server’s processing capability, queue length and utilization ratio as load indices. As the cluster supports multiple services, we have used content awareness forwarding algorithm at the primary level and waited round robin algorithm at the secondary level.

The user’s dependence on the web applications like utility bills, net banking, e-Learning, information based services etc. is increasing at an exponential rate. The service providers try to fulfill the expectations of the users by providing huge amount of information. Owing to this, the contents of web are increasing at the accelerating rate in past few decades. However, the number of requests from the users are not uniformly distributed in 24 hour time span. Some websites receive huge number of requests for some special category of services in a short time interval during the peak hours. The bursty load on the server degrades response time or results into denial of service which eventually affects the user satisfaction for the particular website and the users have to switch to some other alternative for web based services. The situation worsens if no alternate solution is available to these organizations like in case of online railway reservation, internet banking etc.

To improve the availability and reliability, Web Server Cluster (WSC) is widely used by the business organizations and scientific institutions. A WSC is a collection of servers connected via high speed LAN. It works like a single virtual web server with a single host name and the user’s requests are transparently redirected to one of the node in the WSC. As the proposed WSC is scalable, the cluster needs to be expanded by addition of nodes as per need when the load on WSC increases during peak hours and allows contraction by removing some nodes when loads on the cluster decreases. This expansion and contraction
introduces heterogeneity as the new servers may have different configurations viz. memory, processing capability and operating system.

To evenly distribute the load among the servers on the WSC, dynamic load balancing (DLB) techniques are used. DLB optimizes request distribution among servers based on various parameters like server capacity, current load level and historical performance. It also improves mean response time and overall throughput of a WSC. To further improve the performance of the WSC, we have used content aware load balancing i.e. load balancing on the basis of category of the request received. For different categories of requests, we have used different scheduling techniques e.g. round robin (RR), weighted round robin (WRR) and shortest queue (SQ).

Most of the existing DLB algorithms in the literature serve homogeneous WSC and can’t be directly used in the heterogeneous environment. Therefore, we propose a DLB algorithm which supports heterogeneity, scalability and content awareness. Our objective is to improve the throughput and response time of the WSC. The algorithm proposed in this chapter finds applicability in following scenarios:

- High performance computing clusters
- Grid scheduling algorithms
- Improved dynamic content handling techniques
- Load balancing strategies for middleware and backend systems

The rest of the chapter is organized as follows. Section 3.2 describes content-aware load balancing with the available algorithms in the field. The proposed architecture and frame work is presented in Section 3.4. Section 3.5 covers the proposed algorithm and discusses the simulation results.
3.2 CONTENT AWARE LOAD BALANCING

To maintain the availability and stability of the system, idea of multiple servers has been proposed. A WSC comprises of multiple servers and can be accessed by using a single host name. It is clear that in a WSC, all the requests arrive at single point known as dispatcher or scheduler that redirects requests to the appropriate server. Therefore, the overall performance of the system depends on how the requests are distributed among the servers to minimize the load and improve the response time of the users of the WSC. Load balancing techniques are used to distribute these requests for efficient functioning of WSC.

Major goal of most of the existing load balancing algorithms is to maximize cluster throughput and resource utilization and improve average response time by minimizing request execution time and communication overheads. The load balancing algorithms can be categorized as static or dynamic. Static algorithms use the parameters whose value is known in advance viz. number of servers, the order in which requests are distributed, processing capability of servers etc. Thus, the systems using static algorithms can’t ensure efficient load balancing. Random, round robin (RR) and weighted round robin (WRR) algorithms are static load balancing algorithms. On the other hand, DLB attempts to balance the workload dynamically by responding to the server’s current state. Dynamic algorithms collect the current state of the server cluster and distribute the requests based on the run time information [Mehta, 2010]. Therefore, dynamic algorithms are more complex as compared to static algorithms. Least-connection, shortest queue and load-based algorithms are commonly used dynamic load balancing algorithms [Chen, 2004].

Round-robin algorithm distributes the load equally among the servers regardless of the current number of connections or the response time. A weighted round-robin algorithm takes care of processing capabilities of each server. Cluster administrators manually assign a performance weight to each server and a scheduling sequence is automatically generated according to the server’s weight.
Requests are then distributed to the different servers using round-robin algorithm. Least-connection, shortest queue and load-based algorithms send requests to the servers in cluster, based on the number of connections, number of jobs in the queue and process load respectively.

Aweya et al. have described an overall control scheme for distributed web server systems. They have proposed two algorithms namely Admission Control (AC) algorithm-1 and AC algorithm-2 and then compared these algorithms with RR. They have shown that the performance improvement of RR, AC algorithm one and AC algorithm two are in increasing order. Both the algorithms generate satisfactory results with homogeneous servers [Aweya, 2002]. However, the algorithms are not capable of converging to the desired operational level in a heterogeneous environment. Colajanni and Cardellini have proposed adaptive time-to-live (TTL) algorithm that takes care of uneven distribution of requests in geographically distributed heterogeneous web servers [Colajanni, 1998]. The algorithm assign different TTL value for each address by considering the selected server capacity and gives better results for geographically distributed and heterogeneous servers. Bryhni and Kure have compared load balancing techniques for scalable web servers. The comparison was made on a trace driven simulation based on redirection of request in the network and redirection in the mapping between a CNAME and IP address [Bryhni, 2000]. The study shows that better performance is achieved with redirection in the network. Lin et al. have presented a content-based load balancing algorithm that allocates requests to the server with lowest load according to the load state of the server [Lin, 2009]. Yang and Cheng have proposed a load balancing algorithm supporting QoS in web server cluster. The algorithm solves the problem of locality of static load and the equal distribution of dynamic load and has better performance compared to other traditional algorithms [Yang, 2007]. Chen and Chen considered the content type for balancing work load on scalable WSC. They have observed that for different content requests, the load for each web server is different. They have proposed a load balance algorithm for supporting better response in scalable WSC. They have shown that their approach is suitable for the
high variation of processing time [Chen, 2004]. Lu and Lau have proposed an adaptive load balancing algorithm for heterogeneous distributed system. The algorithm transfers a suitable processing workload from senders to receivers. They have also proposed a load state measurement scheme for heterogeneous systems [Lu, 1996]. However, to the best of our knowledge in literature review, we did not notice a content-aware algorithm for scalable heterogeneous web server cluster.

3.3 Framework

As discussed in Chapter 2, load balancing strategies can be classified as Client-based approach, DNS-based approach, Server-based approach and Dispatcher-based approach [Cardellini, 1999]. DNS based load balancing is useful in multiple replicated servers with distinct host names and the method provides limited load balancing. On the other hand, in dispatcher based distribution method, the client requests are distributed by a central node known as scheduler [Werstein, 2006]. The proposed WSC architecture uses dispatcher based approach to forward the incoming client requests to least loaded node in the WSC. For the implementation of content-aware load balancing, web requests are classified as follows [Chiang, 2009]:

- Static Requests: The category includes HTML pages and embedded objects. These requests require high speed access to Web objects and are high cache bound and low CPU bound requests. As accessing cache memory is faster than disk, static requests are also known as cache bound.
- Transaction Requests: These are the requests for dynamic pages built at runtime by accessing static objects, databases etc. Database operations are highly dependent on secondary memory accesses. Therefore, these requests are also known as I/O bound requests.
- Secure Transaction Requests: This type of requests include online payments, net banking, e-commerce etc. and depend on databases, dynamic pages and high class of secure computing. Security measures require extensive
processing. Therefore, this category is known as I/O bound requests and CPU bound requests.

- Multimedia requests: These requests consist of real-time audio and video requests and are heavy I/O bound processes.

In this section, the architecture of proposed WSC is discussed. As shown in Figure 3.1, WSC contains $n$ types of servers where server types have been categorized on the basis of their content types. Further, each server type comprises of variable number of replicated servers. Initially, there are four categories of servers namely static, transaction, secure transaction and multimedia.

![Fig. 3.1 Web Server Cluster](image-url)
Let $S$ be the set of category servers and $S_i$ be set of replicated servers of $i^{th}$ category then the sets can be represented as:

$$S = \{S_1, S_2, ..., S_n\}$$

$$S_i = \{S_{i1}, S_{i2}, ..., S_{ij}\}$$

where, $n$ is the number of server categories and $j$ may vary between $1$ to $m$ for each of the category $i$.

Replication represents that every server in the category has the capability to respond the user request of that category. As the servers are heterogeneous in terms of hardware and operating system, their response time will be different. There is a scope of increasing or decreasing the number of categories or number of servers in each category on the basis of demand. As shown in Figure 3.1, in the proposed WSC architecture, a scheduler is connected to router and on the other side it is connected to a LAN switch. All the servers are also connected with LAN switch. The scheduler is single point entry for client requests of WSC as only scheduler’s IP address is public. After receiving the request, scheduler determines request category and least loaded sever $S_y$ in the $i^{th}$ category and then forwards the request to server $S_y$. It also updates the load and response table on the basis of information received from the servers. The load table is used for even distribution of workload to the servers; whereas response table helps in calculating the average response time and the throughput of the WSC for evaluating the overall performance of the WSC.

At the server end, servers directly send the response of the requested query to the client through the router via LAN switch using one-way mechanism. By by-passing the scheduler for response, proposed method has tried to reduce the workload of the scheduler. The method has also defined three load levels for the servers in WSC namely low load, moderate load and heavy load. Servers calculate their load on the basis of queue length ($q$), memory utilization ($m$) and
processor utilization (p) parameters. The weight value $w_{ik}$ is associated with these parameters by WSC administrator according to content category. Therefore, the weighted load $l_{ij}$ at server $S_y$ can be computed as:

$$l_{ij} = \left( \frac{w_{iq} \cdot q_{ij} + w_{im} \cdot m_{ij} + w_{ip} \cdot p_{ij}}{w_{iq} + w_{im} + w_{ip}} \right)$$

(1)

where, $q_{ij}$, $m_{ij}$ and $p_{ij}$ are values of queue length, memory utilization and processor utilization parameters respectively of server $S_y$. The values of these parameters change frequently with time. Therefore, it is reasonable to average the load value of the servers for some assumed interval. Thus, the average load $L_{ij}$ of the server $S_y$ over the period of $t$ time interval may be calculated as:

$$L_{ij} = \frac{\sum l_{ij}(x)}{t}$$

(2)

where, $l_{ij}(x)$ is the weight of server $S_y$ at time $x$ for $x = 1, 2...t$ and the actual load (AL) of server $S_y$ is calculated by dividing $L_{ij}$ by its heterogeneity factor i.e.

$$AL_{ij} = \frac{L_{ij}}{h_{ij}} \text{ for all } h_{ij} \in H = \{h_{ij}\}$$

where, $i$ represents number of categories and $j$ represents number of servers in each category. $h_{ij}$ is proportional to the capability of the servers i.e. the value of $h_{ij}$ is one for least capable server which may vary up to ten and proportionate increase based on the servers capability. The value of heterogeneity factor, $h_{ij}$ is assigned by the WSC administrator after running a complex problem on all the nodes of cluster. The load level can be determined for the server $S_y$ as follows:

- $= \text{low}$ for $AL_{ij} \leq 30\%$
- $= \text{high}$ for $AL_{ij} \geq 70\%$
- $= \text{Moderate}$ otherwise.

For updating the scheduler's load table, state change driven policy has been used. According to this policy, servers send the information to the scheduler whenever load changes from one level to another. On receiving the load change information, scheduler updates its load table.
3.4 PROPOSED ALGORITHM

The process starts with the establishment of cluster and initialization of the weight value to queue length \( (q) \), memory utilization \( (m) \) and processor utilization \( (p) \) parameters, load table and response table of all content categories. As soon as scheduler receives the requests, it identifies its category based on the information stored in HTTP request and refers to the load table to determine the least loaded server in this category for forwarding the request to most appropriate server. The client request is responded by the processing server directly without involving the scheduler. In order to test the proposed algorithm, we have compared it with weighted round robin and the Lin’s algorithm [Lin, 2009]. The flowchart is shown in Figure 3.2 and the algorithm is described as follows:

Table 3.1 Informal Description of Algorithm

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 request, scheduler identifies the request category
   c) Scheduler refers the load table to identify least loaded server
   d) Request is redirected to the least loaded server by rewriting the server’s address
   e) Response table is updated
   f) Load table is updated if the change in load level of the server occurred
   g) If all the servers in a category are critically loaded, addition of servers is requested
   h) Go back to step 2 (a)
Table 3.2 Formal Description of Algorithm

class Node
{
    double queueLength, memoryUti, processorUti, hetroFactor;
    int wq, wm, wp;  // The weight value associated with these parameters according to content category

    double AL;  // AL: Actual load of node

    Node (int wq, int wm, int wp)
    {
        queueLength = 0;
        memoryUti = 0;
        processorUti = 0;
        this.wq = wq;
        this.wm = wm;
        this.wp = wp;
    }

    double calculateLoad()
    {
        double load = 0;
        for (n unit of time)
            load += (wq * queueLength + wm * memoryUti + wp * processorUti)
                / (wq + wm + wp);
        return load / (n * hetroFactor);
    }

    void updateLoadTable()
    {
        AL = calculateLoad();
        if(AL != TL)  // TL: Table load of the node
            TL = AL;
    }

    void updateResponseTable()
    {
        calculate the response time for the current job and update the response table;
    }

    void hetroFactor()
    {
        run a complex program for given parameters and accordingly set the value of heterFactor;
    }

    void processJob(int q, int m, int p)
    {
        Set the value of queueLength, memoryUti and processorUti;
        updateResponseTable();
    }
}

class LoadBalancer
{
    Node[][] n;
    static double [][] loadTable;
    static double [][] responseTable;
    void receiveForwardRequests();
    int checkHealth();
    void scaleNode();
}
Fig. 3.2 Flowchart Depicting the Working of Proposed WSC
3.5 EXPERIMENT AND RESULT OF ANALYSIS

This section describes simulation results and result analysis.

3.5.1 Simulation

A Java program generates requests where arrival follows Poisson distribution and service follows exponential distribution. WSC simulation uses these requests which are received by scheduler. Scheduler distributes these requests using WRR, Lin’s algorithm and proposed algorithm one at a time and calculated mean response time (in seconds) using WRR, Lin’s algorithm and proposed algorithm. We have implemented the algorithms using Java multithreading that allows program to calculate load of the nodes, update the load table and response table along with forwarding the requests to the nodes in continuous manner. User threads generate requests and handed over to the scheduler thread. The generated requests are of different categories with varying resource requirement in each category. On the basis of scheduling policy, scheduler thread stores the request in the circular queue of server thread. Scheduler thread maintains response time of every dispatched request. The same set of input values is simulated for weighted round robin, Lin’s algorithm and proposed algorithm.

3.5.2 Simulation Results

For simulation, a set of 400, 800 ....4000 requests are generated in each category and mean response time is calculated using response table maintained by scheduler. As shown in Table 3.3 to Table 3.6 and Figure 3.3 to Figure 3.6, we compared these algorithms for static, transaction, secure transaction and multimedia requests and as shown in Table 3.5 and Figure 3.7, these algorithms are compared for WSC.
Table 3.3 Mean Response Time for Static Requests

<table>
<thead>
<tr>
<th>No of Requests</th>
<th>WRR</th>
<th>Lin's algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>0.3</td>
<td>0.33</td>
<td>0.35</td>
</tr>
<tr>
<td>800</td>
<td>0.29</td>
<td>0.32</td>
<td>0.35</td>
</tr>
<tr>
<td>1200</td>
<td>0.28</td>
<td>0.31</td>
<td>0.34</td>
</tr>
<tr>
<td>1600</td>
<td>0.28</td>
<td>0.31</td>
<td>0.34</td>
</tr>
<tr>
<td>2000</td>
<td>0.27</td>
<td>0.31</td>
<td>0.32</td>
</tr>
<tr>
<td>2400</td>
<td>0.28</td>
<td>0.31</td>
<td>0.33</td>
</tr>
<tr>
<td>2800</td>
<td>0.27</td>
<td>0.31</td>
<td>0.33</td>
</tr>
<tr>
<td>3200</td>
<td>0.26</td>
<td>0.31</td>
<td>0.32</td>
</tr>
<tr>
<td>3600</td>
<td>0.26</td>
<td>0.31</td>
<td>0.32</td>
</tr>
<tr>
<td>4000</td>
<td>0.26</td>
<td>0.31</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Fig. 3.3 Comparison of Mean Response Time for Static Requests
Table 3.4 Mean Response Time for Transaction Requests

<table>
<thead>
<tr>
<th>No of Requests</th>
<th>WRR</th>
<th>Lin's Algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>0.55</td>
<td>0.5</td>
<td>0.55</td>
</tr>
<tr>
<td>800</td>
<td>0.59</td>
<td>0.52</td>
<td>0.55</td>
</tr>
<tr>
<td>1200</td>
<td>0.65</td>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td>1600</td>
<td>0.62</td>
<td>0.5</td>
<td>0.48</td>
</tr>
<tr>
<td>2000</td>
<td>0.62</td>
<td>0.47</td>
<td>0.43</td>
</tr>
<tr>
<td>2400</td>
<td>0.65</td>
<td>0.46</td>
<td>0.42</td>
</tr>
<tr>
<td>2800</td>
<td>0.62</td>
<td>0.44</td>
<td>0.39</td>
</tr>
<tr>
<td>3200</td>
<td>0.62</td>
<td>0.44</td>
<td>0.38</td>
</tr>
<tr>
<td>3600</td>
<td>0.61</td>
<td>0.43</td>
<td>0.38</td>
</tr>
<tr>
<td>4000</td>
<td>0.59</td>
<td>0.41</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Fig. 3.4 Comparison of Mean Response Time for Transaction Requests
Table 3.5 Response Time for Secure Transaction Requests

<table>
<thead>
<tr>
<th>No of Requests</th>
<th>WRR</th>
<th>Lin’s Algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>0.78</td>
<td>0.7</td>
<td>0.68</td>
</tr>
<tr>
<td>800</td>
<td>0.75</td>
<td>0.67</td>
<td>0.65</td>
</tr>
<tr>
<td>1200</td>
<td>0.72</td>
<td>0.64</td>
<td>0.62</td>
</tr>
<tr>
<td>1600</td>
<td>0.73</td>
<td>0.65</td>
<td>0.62</td>
</tr>
<tr>
<td>2000</td>
<td>0.71</td>
<td>0.62</td>
<td>0.57</td>
</tr>
<tr>
<td>2400</td>
<td>0.73</td>
<td>0.62</td>
<td>0.57</td>
</tr>
<tr>
<td>2800</td>
<td>0.72</td>
<td>0.61</td>
<td>0.55</td>
</tr>
<tr>
<td>3200</td>
<td>0.72</td>
<td>0.59</td>
<td>0.55</td>
</tr>
<tr>
<td>3600</td>
<td>0.72</td>
<td>0.59</td>
<td>0.54</td>
</tr>
<tr>
<td>4000</td>
<td>0.72</td>
<td>0.59</td>
<td>0.53</td>
</tr>
</tbody>
</table>

Fig. 3.5 Comparison of Mean Response Time for Secure Transaction Requests
Table 3.6 Mean Response Time for Multimedia Requests

<table>
<thead>
<tr>
<th>No of Requests</th>
<th>WRR</th>
<th>Lin's Algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>1.09</td>
<td>1.08</td>
<td>1.06</td>
</tr>
<tr>
<td>800</td>
<td>1</td>
<td>0.88</td>
<td>0.86</td>
</tr>
<tr>
<td>1200</td>
<td>0.96</td>
<td>0.8</td>
<td>0.78</td>
</tr>
<tr>
<td>1600</td>
<td>0.98</td>
<td>0.8</td>
<td>0.73</td>
</tr>
<tr>
<td>2000</td>
<td>0.96</td>
<td>0.78</td>
<td>0.7</td>
</tr>
<tr>
<td>2400</td>
<td>0.98</td>
<td>0.77</td>
<td>0.69</td>
</tr>
<tr>
<td>2800</td>
<td>0.96</td>
<td>0.76</td>
<td>0.65</td>
</tr>
<tr>
<td>3200</td>
<td>0.97</td>
<td>0.75</td>
<td>0.63</td>
</tr>
<tr>
<td>3600</td>
<td>0.96</td>
<td>0.74</td>
<td>0.63</td>
</tr>
<tr>
<td>4000</td>
<td>0.96</td>
<td>0.73</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Fig. 3.6 Comparison of Mean Response Time for Multimedia Requests
Table 3.7 Mean Response Time for WSC

<table>
<thead>
<tr>
<th>No of Requests</th>
<th>WRR</th>
<th>Lin’s algorithm</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1600</td>
<td>0.68</td>
<td>0.65</td>
<td>0.66</td>
</tr>
<tr>
<td>3200</td>
<td>0.66</td>
<td>0.6</td>
<td>0.61</td>
</tr>
<tr>
<td>4800</td>
<td>0.65</td>
<td>0.58</td>
<td>0.57</td>
</tr>
<tr>
<td>6400</td>
<td>0.65</td>
<td>0.57</td>
<td>0.54</td>
</tr>
<tr>
<td>8000</td>
<td>0.64</td>
<td>0.55</td>
<td>0.51</td>
</tr>
<tr>
<td>9600</td>
<td>0.66</td>
<td>0.54</td>
<td>0.5</td>
</tr>
<tr>
<td>11200</td>
<td>0.65</td>
<td>0.53</td>
<td>0.48</td>
</tr>
<tr>
<td>12800</td>
<td>0.64</td>
<td>0.52</td>
<td>0.47</td>
</tr>
<tr>
<td>14400</td>
<td>0.64</td>
<td>0.52</td>
<td>0.47</td>
</tr>
<tr>
<td>16000</td>
<td>0.63</td>
<td>0.51</td>
<td>0.46</td>
</tr>
</tbody>
</table>

Fig. 3.7 Comparison of Mean Response Time for WSC
3.5.3 Analysis of Results

From Table 3.3 and Figure 3.3, it is clear that for static requests, WRR gives better performance than other two algorithms. As shown in Table 3.3 to Table 3.6 and Figure 3.3 to Figure 3.6, it is obvious that up to 400 requests, all these three algorithms are giving approximately similar results. However, we observe a significant performance improvement after 800 requests. The proposed algorithm gives better result than Lin’s algorithm when number of requests is more and the contents are more heavier i.e. the load on WSC increases. We observe that proposed algorithm provides -9%, 6%, 13% and 16% improvement over WRR and -4%, 4%, 9% and 13% improvement over Lin’s algorithm in mean response time for static requests, transaction requests, secure transaction requests and multimedia requests respectively. Therefore, if we use WRR scheduling for static requests and proposed algorithm for rest of the request types, the overall improvement in the response time of the WSC is 12% and 8% over WRR and Lin’s algorithm respectively. If number of requests increase exponentially, better performance improvement can be achieved.

3.6 Summary

In this chapter, we have proposed content aware load balancing algorithm for scalable heterogeneous WSC. We have compared the proposed algorithm with the weighted round robin and Lin’s algorithm. The results show that proposed algorithm gives better result as the load on the WSC increases. For lightly loaded server clusters, WRR algorithm can be used which is simpler as compare to other two algorithms and hence reduces the complexity of the scheduler. However, for huge workload, the proposed algorithm can be used effectively.