CHAPTER 6

CLOSED LOOP SEQUENTIAL AND DIFFERENTIATED REAL-TIME WEB SERVERS

6.1 INTRODUCTION

Applications based on WWW are increasing day-by-day, in which the time-critical Web based applications (e.g. on-line trading and e-business server) are more popular in providing particular services. In this scenario, it becomes a challenge for researchers to design the Web servers providing time bounded services. In recent years, real-time applications are rapidly growing that execute in open and unpredictable environments such as on-line trading and e-business server where neither the resource requirements nor the arrival rates of service requests are known apriori. However, performance measures such as miss ratio, response time and CPU utilization are essential for these applications (Bonomi et al 1990). In recent years, several adaptive scheduling algorithms have been developed to achieve these performance measures with cost effective approaches in unpredictable environments.

Most of the real-time scheduling algorithms are open loop algorithms since scheduling decisions are based on the worst-case estimation of task parameters. They neither continuously observe the performance of the system nor dynamically adjust the system parameters to improve performance. Earliest Deadline First (EDF) scheduling is one of the open loop algorithms where the task set doesn’t have the complete knowledge about its timing constraint. In this proposed work, EDF is refined as a closed loop
scheduling and its performance criteria such as task miss ratio and response time are evaluated. The miss ratio is defined as the number of deadlines missed to the total number of completed and aborted task instances in a sampling window. The miss ratio is usually the most important performance metric in a real-time system.

Manipulated variable is defined as a system attribute that is dynamically changed by the scheduler. The new value of the manipulated variable is based on errors that can occur at run time. In this proposed architecture, the relative deadline of the requests acts as a manipulated variable. The reason for choosing the relative deadline as a manipulated variable is that most real-time scheduling policies may miss the deadline when the system is overloaded and the miss ratio increases as the system load increases. In this proposed work, the systems in unpredictable environments are taken where relative deadlines and CPU utilizations are unknown and dynamically changing over time.

A Closed loop real-time scheduling has been developed as a unified framework which provides Quality of Service (QoS) in unpredictable environments e.g. on-line trading and e-business servers on the Internet. The challenging problem in the closed loop scheduling is, changing the scheduling parameters dynamically at run-time.

The requests in the queue for longer time than threshold value are dropped in best effort model Web server. So the miss ratio and average response time for those requests are increased. These requests are effectively handled by the proposed sequential and differentiated real-time Web server architecture; in turn it provides better response times and task miss ratio guarantees for the requests in unpredictable environments.
6.2 BACK GROUND OF THE CLOSED LOOP WEB SERVER ARCHITECTURE

The proposed Web server architecture achieves the desired response times and task miss ratios for the requests, is shown in the Figure 6.1. The closed loop manages to process the requests again which are rejected due to the server's timeout threshold.

![Figure 6.1 Closed Loop architecture in Web Server](image)

The architecture is composed of a connection scheduler, a monitor, a controller and a fixed server process. The connection scheduler accepts every incoming TCP connection request, shuffles the requests depending on the relative deadline using closed EDF algorithm, and dispatches a request at the front of a scheduling queue to the server process. The server process reads requests, executes them, closes TCP connections and notifies to the connection scheduler that is free to process new request. Connection delay is defined as the time interval between the arrival of a TCP connection (establishment) request and the time the connection is accepted (dequeued) by
a server process. The monitor is invoked to measure the average connection delays and average deadline for each sampling period $S$. The controller calculates a new relative deadline value for the requests based on their position in the server queue and reallocates the server processes subsequently. The variables used for calculating the new relative deadline are Error $E(S)$, assigned deadline $U(S)$ and actual deadline $C(S)$. The Error for $n^{th}$ request is $E_n(S) = C_n(S) - U_n(S)$. If $E_n(S)$ is less than or equal to zero then that request is processed successfully by the Web server. If $E_n(S)$ is greater than zero then that request is rejected and sent back it to the server queue after recomputing the relative deadline using the equation below

$$U_n(S) = U_n - i(S) + g(E_n(S) - rE_n - i(S))$$

(6.1)

where 'g' is the parameter whose value depends on the position of the request in the server queue and 'r' is the percentage of recent error value of relative deadline $n$ is the index of the request. The error indicates the value of deadline missed for a particular request. An equation (6.1) has been applied to recompute the relative deadline for a request to guarantee the desired performance in the Web server. Following assumptions are taken into account for each task.

**Case (I)**

- Every task is periodic in nature and arrives in equal intervals at the system.
• All tasks are independent in nature and have no precedence constraints.
• All tasks are non-preemptive in nature

Case (II)

• Every task is aperiodic in nature and arrives randomly at the system.
• All tasks are independent in nature and have no precedence constraints.
• All tasks are non-preemptive in nature

6.2.1 Request Model

Each request is characterized by the attributes such as relative deadline, arrival time, estimated execution time, actual execution time, and actual CPU utilization. A key feature of this model is that it characterizes systems in unpredictable environments where the requests' CPU utilizations are varying with time and unknown to the scheduler. Such systems are amenable to the use of closed loop that dynamically update the scheduling parameters. So these systems adapt to the load variations at runtime. Case (I) assumption is used as the request model for sequential Web servers.

6.3 SIMULATION OF CLOSED LOOP REAL-TIME SCHEDULING ALGORITHMS

6.3.1 Sequential Real-Time Web Server
A Web server is modeled as a sequential server, that is, the Web server processes only one incoming request at a time. This concept is different from the multiprocessor or multithreaded Web server model. The rejected requests are sent back to the server queue with recomputed relative deadlines. The queue size is fixed. The estimated execution times of the requests may range from 1ms to 5ms. The maximum relative deadline of the request is 90ms. The timeout threshold value is 90ms. The tolerable value of the relative deadline for the requests should be less than or equal to the maximum timeout value for the Web servers. The maximum size of the queue is 500. A dummy request is inserted into the queue for identifying the repetitive requests. A cycle refers the number of times requests are rejected before the successful completion by the server. The algorithm given below is used for processing the incoming Web requests in sequential real-time Web server without closed loop.

Step 1: Read the number of requests for the sampling period 'S'.

Step 2: Each request is generated with arrival times, estimated execution times, and estimated relative deadlines using random number generator.

Step 3: The requests are arranged in the order based on estimated relative deadlines.

Step 4: When the actual execution times of the requests exceed the estimate relative deadlines, then the requests are rejected; otherwise, they are processed successfully and their execution times and CPU utilizations computed.

Step 5: Repeat step 4 until there are no requests in the queue.
The proposed algorithm is implemented as separate application level software. The requests are rejected by the server whose queuing time is more than 90 ms. Relative deadlines of the requests vary from 1 to 90 ms. Table 6.1 shows the performance of the sequential real-time open loop scheduling. Experimental results show that the open loop sequential real-time scheduling has 2% to 89% of task miss ratio for the requests vary from 25 to 500. The result shows in Table 6.1 that the less number of requests at the Web server satisfies all client requests i.e. the requests receive excellent service but as soon as request increases the server starts rejecting in open loop sequential real-time server. The scheduler generates access and error log files. The access log file stores the successful completion of the requests whereas the error log stores the rejected requests.

Table 6.1 Performance of the open loop sequential real-time scheduling

<table>
<thead>
<tr>
<th>S.No</th>
<th>Requests/Tasks</th>
<th>Number of requests processed</th>
<th>Total Response time (ms)</th>
<th>Task Miss Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>25</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>49</td>
<td>89</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>47</td>
<td>89</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>44</td>
<td>90</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>300</td>
<td>42</td>
<td>90</td>
<td>82</td>
</tr>
<tr>
<td>6</td>
<td>500</td>
<td>41</td>
<td>90</td>
<td>89</td>
</tr>
</tbody>
</table>

The real-time scheduling algorithm that uses a closed loop mechanism is given below.

Step 1: Read the number of requests for the sampling period 'S'.
Step 2: Each request is generated with arrival time, estimated execution time, and estimated relative deadline using random number generator.

Step 3: The requests are arranged in the order based on estimated relative deadlines.

Step 4: When the actual execution times of the requests exceed the estimated relative deadlines, then the requests are rejected and then sent back to the queue with recomputed relative deadlines using equation (6.1); otherwise, they are successfully processed and their actual execution times and CPU utilizations are computed.

Step 5: Repeat step 4 until there are no requests in the queue.

The proposed algorithm is implemented as separate application level software. The queue size is 500. The experimental results show that there is no change in the miss ratio as open loop approach for the given 500 requests in the sampling period 'S'. The performance of the client requests using closed loop sequential real-time scheduling is shown in Table 6.2. The experimental results show that the closed loop sequential real-time scheduling has a considerably less task miss ratio compared to the open loop real-time scheduling algorithm.

Table 6.2 Performance of the closed loop sequential real-time scheduling

<table>
<thead>
<tr>
<th>S.No</th>
<th>Requests/ Tasks</th>
<th>Number of requests processed</th>
<th>Total Response time (ms)</th>
<th>Task Miss Ratio (%)</th>
<th>Maximum Cycle (number)</th>
</tr>
</thead>
</table>

The table provides the performance metrics for the closed loop sequential real-time scheduling algorithm, including the number of requests processed, total response time, task miss ratio, and maximum cycle number.
The number of requests processed in the first cycle decreases as the input to the sequential real-time scheduling Web server is increased. The sequential closed loop real-time scheduling offers a better performance over the open loop approach under heavy and unpredictable workloads. As the number of requests increases and the assigned queue size is the exact number of requests then there is no much performance difference between closed and open loop algorithms. To overcome this bottleneck, the queue size is redefined as three times higher than that of the actual size required for the given requests. Performance evaluation of the closed loop sequential real-time scheduling architecture demonstrates that it provides better miss ratio and average response time despite of instantaneous changes in the web workloads. The experimental studies show that the average response time for the requests is less in close loop compared to the open loop approach.

### 6.3.2 Sequential Differentiated Real-Time Web Server

The best effort differentiation model (Nong ye et al 2005) on Web server provides better service to the clients depending upon their priority, but they do not provide any guarantees for timely completion of requests. The differentiation approach on Web servers usually offer better service to higher priority requests compared to lower priority requests. The performance difference depends heavily on load conditions of the Web server. The proportional differentiated service models and the absolute guarantee models
provide stronger guarantees in service differentiation compared with the best effort differentiation model. In the *absolute guarantee model* (Chen Yang Lu 2002), a fixed service delay is to be enforced for each type of requests. A disadvantage of this model is that it is difficult to determine appropriate deadlines for the Web services. The absolute delay guarantee requires that all request types receive satisfactory delays if the server is not overloaded; otherwise, desired delays are violated in the predefined priority order, i.e., low priority requests always suffer guarantee violation earlier than high priority requests. In the *proportional differentiated service model* (Chen Yang Lu 2002) a fixed ratio between the delays seen by the different service classes can be enforced. Depending on the nature of the overload condition, either the proportional differentiated service or the absolute guarantee may become more desirable.

In this proposed model, the concept of *absolute guarantee model* is incorporated in the sequential differentiated real-time scheduling algorithm because it is more suitable for the time critical Web based applications.

The Web server architecture for sequential differentiated real-time scheduling has been developed which provides better task miss ratio and response time guarantees to different service classes even in overloaded conditions. The client requests are differentiated into member and non-member classes of requests. Either of the types of these requests receives excellent service but as soon as the request rate increases, the server starts rejecting both classes of requests in open loop differentiated algorithm.

At the same time when closed loop differentiated scheduling is used, the member class gets less request rejection compared to the non-member class. When the load increases, the average response time for member requests are very less while the response times for non-member tasks
are very high. The architecture for a sequential differentiated real-time Web server is as shown in Figure 6.2.

![Figure 6.2 Differentiated service supported closed loop approach Web server](image)

The relative deadlines for member and non-member requests are 90 ms and 50 ms respectively. The member requests are of a higher priority compared to the non-member requests.

Therefore, the relative deadlines of the member requests are given lower values than those of the non-member requests. The relative deadlines of 95% of the members are higher than those of the non-member requests. The Web server satisfies both classes of client requests at lower request rates where as the server starts rejecting both classes of requests as soon as the request rate increases. Table 6.3 shows the performance of the open-loop sequential differentiated real-time scheduling Web server.
### Table 6.3 Performance of the open-loop sequential differentiated real-time scheduling

<table>
<thead>
<tr>
<th>Requests</th>
<th>Request processed</th>
<th>Total Response Time(ms)</th>
<th>Miss Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Member</td>
<td>Non-Member</td>
<td>Member</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>40</td>
<td>38</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>60</td>
<td>47</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>48</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>41</td>
</tr>
</tbody>
</table>

In closed loop sequential differentiated Web server, the requests that exceed their relative deadlines are rejected and sent back to their respective queues with customized relative deadlines. The relative deadlines for member and non-member requests are 90 ms and 50 ms respectively. The maximum size of the queue is 500. The estimated execution times of the requests fall from 1ms to 5ms. A dummy request is inserted in both the queues for identifying the repetitive requests. The performance of the member and non-member requests are evaluated using closed loop sequential differentiated scheduling and it's compared to the open loop differentiated sequential real-time scheduling.

The proposed algorithm is implemented as separate application level software. The performance of the closed loop differentiated sequential real-time scheduling is shown in Table 6.4. In the differentiated closed loop approach, the member requests get considerable benefits whereas the non-member requests get improved performance in the form of a better task miss
ratio and average response time. The closed loop differentiated real-time scheduling has considerably reduced task miss ratios for members (2% to 5%) and non-members (15% to 45%) classes. All the member and non-member requests are rejected if they have been in the queue for longer than 90 ms and 50 ms. Performance evaluation of closed loop sequential differentiated real-time scheduling architecture demonstrates that it provides better miss ratio and response time despite of instantaneous changes in the Web workloads.

Table 6.4  Performance of the closed-loop sequential differentiated real-time scheduling

<table>
<thead>
<tr>
<th>Request</th>
<th>Requests</th>
<th>Requests processed</th>
<th>Total Response Time (ms)</th>
<th>Task Miss Ratio (%)</th>
<th>Max. Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Member</td>
<td>Non-Member</td>
<td>Member</td>
<td>Non-member</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>149</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>49</td>
<td>41</td>
<td>186</td>
<td>2%</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>50</td>
<td>45</td>
<td>202</td>
<td>50%</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>44</td>
<td>32</td>
<td>208</td>
<td>73%</td>
</tr>
</tbody>
</table>

6.4  CONCLUSION

A sequential real-time scheduling Web server has been developed which gives better task miss ratio and response time guarantees to Web requests in overload conditions. These algorithms are implemented in C++ and tested in VXworks platform. The performance of the open and closed loop approaches are evaluated using simulation studies. Results show that the closed loop real-time scheduling offers a better performance over the open loop scheduling under all practical conditions. Closed loop sequential
real-time architecture demonstrates that it provides least task miss ratio and response time guarantees for Web servers when the number of users is large and unpredictable. Probable future research is to extend the single node solution to distributed systems such as server farms. The current applications are implemented as simulators or as separate application level software. It would be interesting to implement the proposed architecture as part of an OS kernel or middleware that provides a general set of service APIs which in turn provides performance guarantees to applications.
CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Current Web service model treats all the transactions uniformly according to the best effort paradigm. Since the web server contains inadequate resources, it is impossible for a Web server to process the incoming requests from the clients with high quality of service and when the requests increase rapidly. Differentiated service improves the performance of higher priority requests at the expense of the lower priority requests. The proposed TCP connection management mechanisms at application level for Web servers provide two different levels of Web services by setting different holding times. Differentiated service module starts ahead of the requests entering into the server log, and assigns proper holding times to every client immediately after establishing a TCP connection. The rigid policy in the system carries with it the danger of request starvation, in which a non-member request waits indefinitely for its turn along with a continual flow of member requests.

A weight is predefined in this proposed system for each queue to avoid the request starvation, and the initial weight depends on the priority of the queues. The number of requests processed at a time is the weight for each queue. When the number of requests in the queue is lower than normal, then the predefined weight to that queue is reduced. Similarly, when the number of requests in the queue is higher than normal, then the predefined weight to that queue is increased. The customized tuning algorithm makes gradual
incremental corrections in priority values as the request arrival rates deviates from the expected pattern.

The service differentiation module called before or after the requests entering into the server log is compared with the best effort paradigm. It helps in improving the average response time of the requests in turn the performance will be increased. The performance improvement range is affected by the size of the database and the number of requests in the queues. A novel architecture is proposed and implemented for managing the incoming requests of identical categories. This proposed architecture takes some requests from the member queue and puts in non-member queue using composite priority to retain the QoS. The limitation of this proposed architecture is that a simple counter measure would pick a request from the non-member queue after servicing 'n' packets in the member queue. This would sometimes cause a priority inversion. This mechanism can effectively provide QoS guaranteed services even in the absence of operating system and network support. The simulation results proved that the service differentiation at the application level produces substantial benefits for Web service applications when compared to the best effort paradigm.

The LDMA framework creates a foundation for developing decentralized load balancing schemes on distributed Web servers. The LDMA is based on all-to-all communication without gathering workload information and request transfer policies among the clustered Web servers in LAN. Mobile agents are used for implementing the LDMA. The LDMA reduces the workload of a dispatcher. However, the LDMA suffers from the obvious drawback of single-point of failure. The mobile agent is used to communicate among the servers for decision-making and to delete the requests in the replica queues. The LDMA framework uses the concept of “ranked Web-servers”, i.e., initially, each replica is assigned a rank, which shows the
priority for processing a request. The rank assigned to the replica is based on the time at which it enters the cluster. A replica, which enters first, has been assigned highest rank. The performance of the LDMA scheme is compared with the MALD, an existing mobile agent based load balancing. Simulation results show that the load deviation of LDMA compared to the MALD is less.

The LTPT is a classical centralized decision making algorithm and is rarely used in practice because processing times should be known in advance. In the LTPT load balancing, new client requests are forwarded to the Web server that has least processing time needed for executing the previously assigned requests. The LTPT acts as a benchmark for load balancing in a LAN environment even though it is rarely used in practice. Simulation results show that the load deviation of the LTPT compared to the Mod_backhand load balancing scheme is less.

The WLDMA is a new dynamic load balancing strategy, which redistributes the client requests from heavy loaded Web servers to less load servers. It could be very useful in WAN environments. In the WLDMA, each of the servers processes the client requests independently and interacts with other servers periodically to share the workload. A client can have access to the Web server that is located geographically closer to minimize the WAN delay. In the existing WAN load balancing schemes, job reallocation is done only when the workload on a server exceeds locally fixed threshold value. In the WLDMA, job reallocation is based on the load of the other servers in the WAN. Simulation results show that the load deviation of the WLDMA compared to the scheme without load balancing scheme is less.

Sequential and differentiated real-time scheduling with open and closed loops are proposed, discussed and implemented in VXworks. A Web server is modeled as a sequential Web server, which means that the requests
are processed one at a time. The requests are shuffled based on the relative deadlines using open and closed loops EDF algorithms. The server queue size is fixed in simulation. Each request is generated with arrival times, estimated execution times, and estimated relative deadlines using a pseudo number generator. When the actual execution times of the requests exceed the estimate relative deadlines then the requests are discarded; otherwise, they are successfully processed and their execution times and CPU utilization computed. The rejected requests are sent to the server queue with recomputed relative deadlines. A dummy request is inserted in the queue after each cycle in order to compute the repetition of the requests. The task miss ratio of the sequential real-time server with closed loop has reduced compared to the open loop EDF real-time scheduling Web server.

Sequential differentiated real-time scheduling with open and closed loops are proposed and implemented in VXworks. The client requests are differentiated into member and non-member requests, both of which receive excellent services initially, but as soon as the request rates increase, the server starts rejecting them in open loop server. At the same time with the use of closed loop real-time scheduling, the member requests get less request rejections compared to the non-member requests. When the number of requests increases, the average response time for the member requests are very less while that of the non-member tasks is very high. Experimental results show that the closed loop sequential differentiated real-time scheduling has reduced the task miss ratio for member and non-member requests compared to the open loop differentiated sequential real-time time scheduling algorithm.
7.2 SUGGESTIONS FOR FUTURE WORK

The proposed application level architecture receives two requests from member and non-member sites. Member requests belong to an elastic traffic and non-member requests belong to real-time traffic. Such variation of contexts is inevitable, is even desirable in Service Oriented Architecture (SOA) and the resulting variation in the demand needs to be appropriately taken into account in future work. As future work, the performances of different applications may be tested over a complex topology and new management configurations included for comparison.

The proposed load balancing algorithms would be implemented in real LAN and WAN environments in the future. These algorithms may also be refined to act as fault tolerant load balancers. The LDMA and the LTPT algorithms, their issues and their performance in WANs are well thought-out as a future work. The LDMA, the LTPT and the WLDMA load balancing policies are under fixed delay assumptions, so, the policies may not perform as expected when the delays are random. This may be considered in the future. Moreover, the delays in actuality are not constant, that is, they depend on factors such as network availability, execution of the software, etc. The WLDMA framework proves to be effective in distributing the workload among Web servers in WAN environments. These simulations may be conducted on real time environment to analyze the performances of the algorithms.

The future course of research is to extend the single node solution to distributed systems such as server farms. The proposed scheduling algorithms are implemented as simulators and as separate application level software. It would be interesting to implement the proposed architecture as part of an OS kernel or a middleware that provides a general set of service APIs that provide performance guarantees to applications.