Modified Application Level Differentiated Web Service Management for TCP Connection

M. Aramudhan [1]and V. Rhymend Uthariaraj[2]
Anna University, Chennai-25, India
Email: aranagai@yahoo.co.in

Abstract: In application level TCP connection management, service differentiation module starts after the requests are entered into the server log. The incoming requests are differentiated into member and non-member based on the IP Address stored in the database on the server side. In the proposed model, the request starts its user level differentiation ahead of entering into the server log and assigned its holding time at the earliest. The number of requests processed at a time in every queue depends on the arrival category of the request. An innovative architecture and algorithm is proposed for the worst case scenario and the performance is evaluated. The experimental result shows that the performance of proposed model is better compared to the existing application level TCP management.

Keywords: Service differentiation, Best-Effort model, adjustment algorithm, timeliness.

1. Introduction

The changing nature of the service requirements among Internet applications term for an architecture that is flexible and able to distinguish the needs of the applications. The architecture of the Internet is challenged by continues growing traffic volume, the need to deliver revenue-generating services and improved scalability must be designed into the routing system. At present, web servers use a queuing policy based on Best-Effort model which employs First-in-First-out (FIFO) scheduling to process the incoming requests [1]. Best-Effort model is defined as stored and forward without guarantee service. In recent times, web servers suffer from the escalating resource demands by reason of the explosive growth of the web. Therefore, Web server is not able to process all the requests with the satisfaction of end-users. At the same time, all transactions are not equally important to the clients or servers besides several applications need to treat them differently. In such case, the multiple levels of service are desirable, where one level of users get benefit at the cost of other levels. The multiple levels of service efforts attempt both in server Operating System (OS) and in the network. Although it takes long run time to replace the OS of end systems or upgrading all routers in the network [2]. The same substantial benefit is achieved with server side application level mechanisms. The past research works suggested that the network and end-system OS are the best places to provide multiple levels of services [2]. But, there are some difficulties in deployment mechanisms. Service differentiation is defined as levels of service, which provides Quality of service (QoS) to one level at the cost of others in terms of response time, throughput etc.Timeliness says how fast an output is produced for an input.

The multiple levels of service in application level are simple, extensible and portable. In proposed application level TCP connection management mechanisms for web server provides two different levels of web service, member and non-member by setting different holding time for connections. In past work, the levels of service are differentiated following the request entered into the server log. In the proposed system, the service is differentiated ahead of entering into the server log in addition analysis the situation where all incoming requests are belonging to higher priority user level. The amount of requests processed in every queue changes dynamically based on the quantity of arrival request category using modified adjustment algorithm. This algorithm manages the variation in request throughput in each queue.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 presents the design and highlights of the proposed mechanism. Section 4 describes the results of implementation. Finally, Section 5 provides the conclusions.

2. Related work

Yoon-Jung Rhee [2] proposed two levels of web service in application level by setting different time outs for inactive TCP connections. The levels of service are differentiated following the requests entered into the server log. Persistent and Non-Persistent type HTTP connections are established for high and low priority levels respectively. The different holding time is allotted for each client request after differentiated its level. The idle user's connections are preserved till its end of holding time. The levels of priorities are assigned using random number generator.

Amit Sharma and Sengupta [3] proposed the levels of service achieved using application, device and user level priority. Application level priority classifies the requests on the basis of the application context. Device level priority is based on the device that is being used to access the service. User level priority is based on member or non-member of the site. Each request is accorded a composite priority value by combining the individual application, device and user levels through a certain priority calculating function. The summation of all individual priorities is one of the possible standard functions. The paper focused only on regulating the throughput of a particular type of request to the service class to achieve differential quality of service.

One of the application level and kernel approaches to web QoS is by Almeida et al. [4]. Their first application level mechanism limits the server pool sizes allocated to requests of different classes. The second mechanism they have implemented is a kernel level scheduler that allows preemption of low level requests and assigns process priorities based on the request class.

Nikolaos [5] presented the design of a QoS architecture which can be added to the Apache web server to allow the server to provide a differentiated QoS.This work discusses the services needed to provide a differentiated QoS to clients of a web site,

based on the client's identity and attributes.Nickolaos's work integrated an implementation of the services with the Apache web server and describes a service that can be used to create algorithms to suit a specific goal.

Sook-Hyun Ryu and Kim [6] proposed two approaches to implement differentiated quality of service. In the user level approach, the web server is modified to include a classification process, priority queues and a scheduler. It is difficult to achieve portability in this approach. In the kernel level approach, real time scheduler is used to prioritize user requests. In this section, the application and kernel levels related works are briefly discussed. In this paper, service differentiation is achieved in the application level and the performance is evaluated.

3. Proposed Model: Differentiated service in application level

In application level TCP connection management, web server provides two levels of service member and non-member, by setting different holding time for connections. The incoming request is differentiated using stored IP address in the database on the server side and later authenticated by the web server. This approach use static information in its decision process to differentiate the requests. The proposed idea starts user level differentiation ahead of entering into the server log and assigns proper holding time to every client immediately after establishing a TCP connection. The differentiated mechanism manages TCP connections by means of offering different levels. The service differentiation model is implemented inside the web server. This model manages class table based on membership DB and determines proper holding time of the client when a new client requests access to the web server. The architecture of modified application level TCP connection management mechanism is as shown in Figure-1.

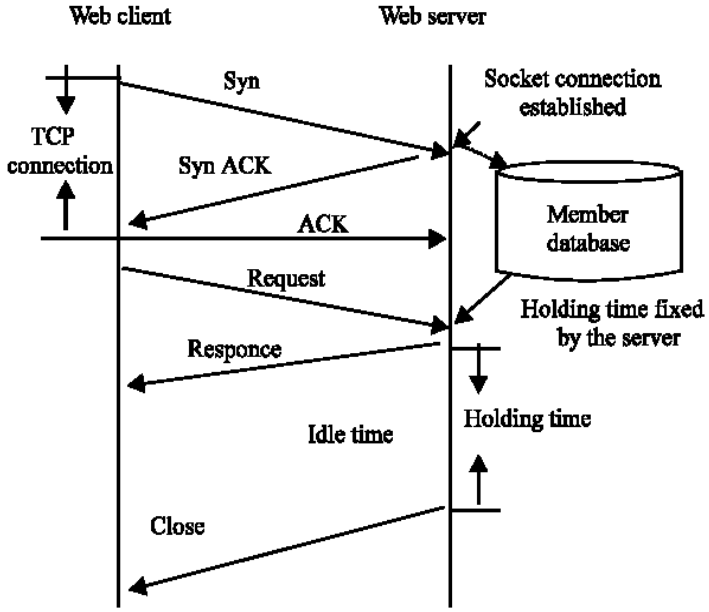

Figure-1: Modified application level TCP connection management mechanism

Client established connection with the server and requests the HTML file. A TCP connection is established with three handshaking signals such as SYN, SYNACK, and ACK. In general, all handshaking signals are bounded as a single system call in JAVA. Therefore, the following assumption is taken for implementation. Service differentiation module is instantly called for the requests ahead of entering into the server log and a delay of three times of connection time for requests after entered into the server log.

The rigid policy in the proposed system brings about the danger of request starvation, in which a non-member request waits indefinitely for its turn amid a continual flow of member requests. It is possible to provide prioritize requests without starvation and sub-optimality by using a randomized or probabilistic scheduling policy. A lower than normal arrival of request category should penalize the priority accorded to the request while the greater than expected arrivals should reinforce the priority positively. The variations in the category request throughput is managed, adjusted using a correcting factor based on the feedback. The given below adjustment algorithm adjusts the throughput in each queue, depends on the arrival amount of each category request.

Step 1: Observe the number of requests in each category. Let N_i denote the amount of incoming requests in each category..
Step 2: Calculate a ratio parameter for each queue. The formula used is

$$R_i = \frac{(n * N_i)}{\sum N_i}$$

Where n is the number of queues and R_i is the ratio parameter for i th queue and the number of requests in the i^{th} queue is N_i. For example, there is member (N_M) and non-member (N_{NM}) queues. The requests in each queues are N_M= 35, N_{NM} =20.the ratio parameter for member queue is $R_M = \frac{(2*35)}{55}$ =1.041(>1) and non-member queue is $R_{NM} = \frac{(2*20)}{55}$ =0.727 (< 1).

Step 3: Calculate the adjustment value for each queue.

If (R_i>1) $P_i = 1 + \left(\frac{R_i * R_i}{K_P}\right)$

else if (R_i = = 1) P_i = 1.0

else if (R_i < 0) P_i =Lower- Bound

Step 4: The number of requests processed at a time is the weight for each queue is denoted by q_W. The weight in the queue after adjustment algorithm is $q_W * P_i$. If R_i>1 then the request processed at the queue is higher bound say 9.1 means 10. If R_i<1 then the requests processed at a time is lower bound say 9.1 means 9. If R_i=1 then the result is an integer.

Initially, the weight for member and non-member queue is 10 and 3 respectively. Here K_R and K_P are the negative and positive normalization constants and serve to keep the adjustment values within desired bounds. For examples, K_R =100 and K_P =40.These constants control the range of the curve and need to be modified depending on the nature of the requirement. The weight is changed for member and non-member queue after adjustment algorithm is 11 and 2. Web traffic is unpredictable in nature. Therefore, the worst case scenario is taken as a constraint in the web service. All incoming requests are similar category then the performance degrades. The architecture proposed for handling all incoming requests are identical category is as shown in Figure-2.

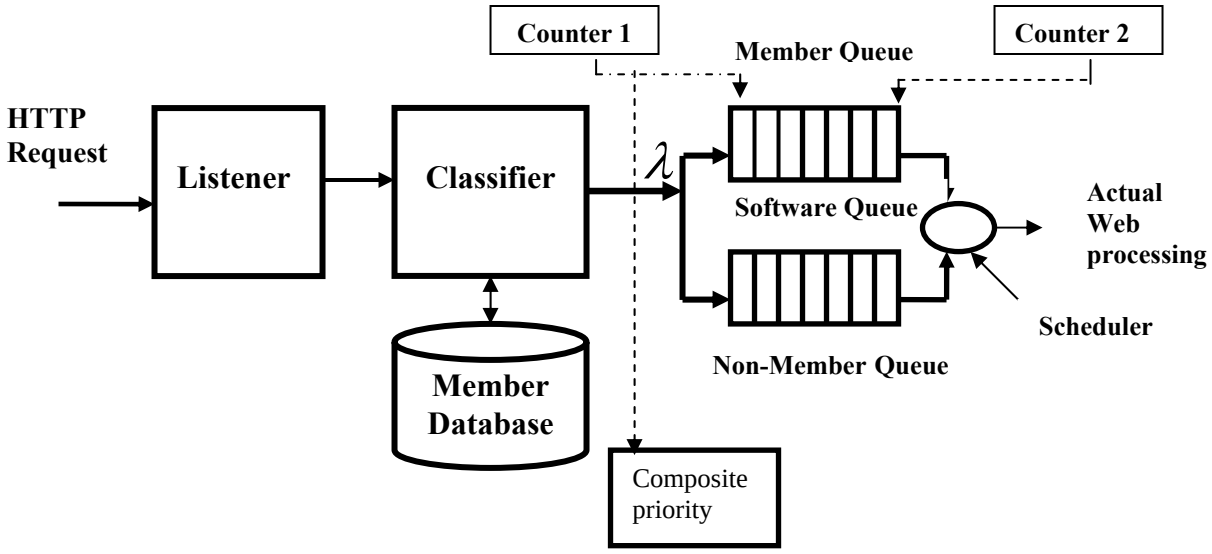

Figure-2: Architecture for handling similar category user level service

The disadvantage of the proposed architecture is simple counter measure would pick a request from the non-member queue after servicing n packets in the member queue. This would some times cause a priority inversion. But this could occur anyway if a non-member request arrives only a short moment before a request of member. The web server may run different services each with different characteristics like time requirements. The architecture receives two requests, one to an elastic service and the second to a real time service, the first coming from a highly privileged user and the second from a less privileged one. Such variation of contexts is inevitable, even desirable in service oriented architecture and the resulting variation in the demand needs has to be appropriately taken into account. The algorithm for handling unexpected identical user level service is as follows.

Step 1: Compute the number of incoming HTTP requests in member (higher priority) queue following user level differentiation. Confirm it exceeds the maximum degrade threshold value, if it so, call composite priority for advancing the request to the appropriate queue (consider both user and application level priority) in order to reduce the performance degradation.

Step 2: Locate the number of requests in the member queue using subtraction between the counters and assigned it in the counter1 often.

Step 3: The counter1 value is less then Max.degrade threshold value then, call single priority (User level) else composite priority (User +Application level).
Step 4: Repeat the steps 1-3 for advancing the request.

4. Implementation and Result Analysis

A prototype modified application level TCP connection is implemented on an IBM machine using JDK1.2. This experiment is conducted in small scale client/server environment. Server is capable to connect maximum of 50 clients at a time using thread and process based architecture [7].The server consists of two processes and each having single multithreaded, each thread handles one request at a time. The crash of one process does not affect the others, so that server continues to operate and process other requests even when one of its processes is killed or restarted. The disadvantage of this architecture is single malfunctioning thread can bring the process down because all threads in the particular process share the same address space. The memory requirement is small and new threads do not need additional address space. The process and thread architecture is as shown in Figure-3. The size of the member database is varies 20 k, 50 k and 100 k respectively. The incoming requests are prioritized into member and non-member and advanced to the appropriate queues. Each member threads hold 50 s where as non-member holds 30 s. The inactivity member and non-member threads are abruptly terminated after 20 s and 15 s respectively. All member threads are processed before non-member threads. Even one non-member thread in the queue may get delayed until all threads in the member queue get processed. Both types of user level requests are established with persistent connection. Each thread created an object on a server, on behalf of the client, has information about its state stored in the server's memory. A decision to assign an object to a server based on the current conditions stated in the model. The server creates a fixed number of processes at startup time. The number of threads in each process varies with the number of client requests.

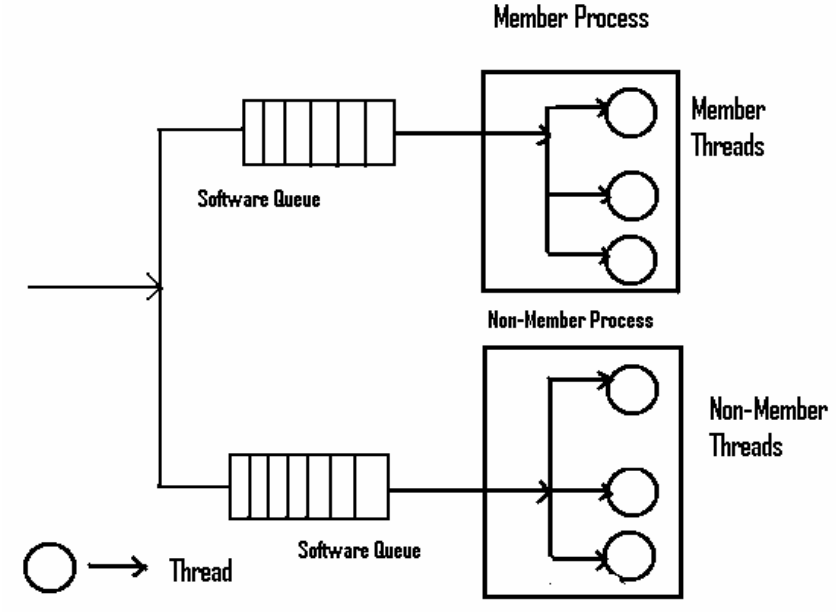

Figure-3: Process and Thread based software web architecture

The service differentiation takes place after and ahead of requests entering into the sever log and advanced to the appropriate queue for processing is as shown in the Table-1 and 2.

Database Size (k)	Clients connected with server	Average latency time (ms)
20	5	301.23
	10	366.25
	15	388.33
50	5	311.23
	10	366.25
	15	404.96
100	5	386.00
	10	392.92
	15	396.64

Table -1: Average latency time taken for service differentiation after TCP Establishment

Database Size(k)	Clients connected with server	Average latency time (ms)
20	5	294.00
	10	333.45
	15	356.65
50	5	306.30
	10	335.45
	15	368.93
100	5	372.00
	10	378.00
	15	390.40

Table-2: Average latency time taken for service differentiation ahead of TCP Establishment

The average response time of Best-Effort model (FIFO) and service differentiation is as shown in the Table-3.

Clients	Database size(k)	Average response time (ms)	
		Differentiation before TCP connection	Differentiation after TCP connection
5	50	318.60	340.60
10	50	377.20	399.20
15	50	404.40	421.20
5	100	374.46	415.60
10	100	392.20	434.20
15	100	409.90	441.06

Table-3: Average response time for processing requests using Best-Effort Model

Clients	Database size(k)	Average response time (ms)	
		Differentiation before TCP connection	Differentiation after TCP connection
5	50	318.20	329.40
10	50	369.90	383.70
15	50	379.20	404.80
5	100	388.20	392.40
10	100	390.18	407.20
15	100	398.34	416.60

Table -4: Average response time for processing request using service differentiation

Even the substantial benefit is achieved, the performance varies depends on the OS used [7]. In past works, polling time is set as a weight for processing the requests at a time in each queue. Depending upon the number of requests of each category that the system receives the polling values are adjusted using adjustment algorithm .For example, polling time periods such as 200 ms, 400 ms and 600 ms are initially assigned for each queue. These values may be changed marginally by adjustment algorithm to 198 ms, 402 ms and 589 ms respectively. In the proposed system, the amount of requests processing at a time is set as a weight in each queue. Table-5 shows the performance using with adjustment algorithm. The system is implemented in Apache Axis 1.0 framework to host the web service. The service is run using Tomcat 4.0 and Apache Axis 1.0.Client side class is made to generate requests at the rate of 50 requests per second through multithreading. In past works, the levels of priorities are assigned using random number generator from 1 to 3. In the proposed work, the levels of priorities are assigned using IP address of the client or proxy.

Number of requests	:	50
Incoming throughput per category	:	18 requests per second
Total Service Time	:	1760 ms (29.37 Sec)
Average service Time per request	:	0.488 seconds

Request Type	Number of Incoming Request	Initial weight (No. request)	After Adjustment Algorithm	Throughput observed
Member	35	10	11	12
Non-Member	15	4	3	3

Table-5: performance using adjustment Algorithm.

Throughput is measured by finding the requests being taken for processing by the server per second. The deviation observed between theoretical and observed value is not much, indicating a good degree of regulation. The proposed system is able to manage the random incoming request types and give throughputs close to the desire levels.

Mathematical Model

Two queues, member and non-member, the average arrival rate of both queues is λ_1 and λ_2 and its service time is μ_1 and μ_2. The average mean arrival rate of queue is λ. Each request in the queue is processed in FIFO order. The number of request processed at a time is defined as weight is w_1 and w_2 respectively. Poisson probability provides a way to calculate the probability of users simultaneously accessing the system. The worst-case response of the system would occur if all users access the system at exactly the same time. The theory behind Poisson probability is that requests enter the system with an exponential probability that other users are entering the system at the same time. The equation for the Poisson probability is

$$f(x) = p(x,u) = \frac{e^{-\mu} \mu^{x}}{x!}$$

μ is total service time of member and non-member requests. X is the number of requests accessing the server at a time. Applying priority queues, most of the requests fall outside the typical Poisson probability. The average response time of each queue is

$$E[R_M] = \frac{2}{2\mu - \lambda}$$

The maximum number of request processed at a time in the member (E [R_M]) and non-member (E [R_{MN}]) queue is

$$E[R_M] = \frac{1}{W_1 - \lambda_1} \qquad \text{and}$$

$$E[R_{NM}] = \frac{1}{W_2 - \lambda_2}$$

W_1, W_2 are subject to change based on the arrival of requests in each queue. There are 2 priority classes, with request in member class having highest priority and those in non-member priority having lower priority. Assume that request of class i arrive as an independent Poisson process with rate λ_i. and that the mean and mean-square service times are $\overline{X}$ and X^2, respectively. Assume that there is no queue limit (m=∞**).** Under this assumptions, the mean waiting time for request of class i, W_i is given by the formula is

$$W_{i=} \frac{\sum_{i=1}^{i=n} \lambda_i \overline{X2}}{2(1 - \rho_1 - \ldots \rho_{i-1})1 - (1 - \rho_1 - .. \rho_i)} \qquad [\rho = \frac{\lambda}{\mu}]$$

The priority queuing reduces delays for member queue at the expense of increased delays for non-member queue. However, priority queue is result in a situation in which one or more of the non-member class of users receive no service at all. As

$$\sum_{1}^{i} \rho_i \rightarrow 1$$

From below, the mean waiting time for priority –i requests grows to infinity, i.e., the backlog of priority –i customers waiting for service grows indefinitely with time. The above results apply only for a system with no queue limit. For a system with a large but finite queue limit m, mean waiting times are always finite for all classes of requests. However, the quality of service for priority –i requests still degrades severely as

$$\sum_{1}^{i} \rho_i \rightarrow 1$$

The mean waiting times tend to become very large and blocking probabilities approach unity, reducing the queue limit reduces mean waiting time at the cost of increased blocking. Total delay (T_D) in the proposed model is

T_D = mean waiting time for each queue + response time of each queue+ time for prioritization

5. Conclusions

Service differentiation takes place ahead of entering into the server log yields better timeliness performance in the application level TCP connection management. This performance obviously depends on the OS. The amount of requests processing at a time is set as a weight in each queue. Depending upon the number of requests of each category that the system receives, the quantity of request is adjusted using adjustment algorithm. In practice, measurement of requests and service differentiation may be overhead but, it shows a little improved performance compared to existing models. In the future, a real time feedback scheduling will be incorporate and its performance is evaluated. This experiment will also be conducted for Intranet and Internet environment. The current infrastructure is being extended to Microsoft .NET platform. A better mechanism of handling of asynchronous web service will also be added to the infrastructure. Scalability issue will also be addressed in future.

References

1) Nong Ye, Esam S. Gel, Xueping Li, Toni Farley, Ying-Cheng Lai (2003), "Web Server QoS models : applying Scheduling rules from production planning" in journal of Computer and operations Research, pp.1147-1164

2) Yoon-Jung Rhee, Eun-Sil Hyun and Tai-Yun Kim (2002), "Connection management for QoS service on the Web" in journal of Network and Computer Applications Vol.25, pp. 57-68.

3) Amit Sharma, Hemant Adarkar and Shubhashis Sengupta (2004), "Managing QoS through Prioritization in Web Services", in Proc., of the Fourth International Conference on Web Information Systems Engineering Workshops.

4) J. Almedia, M. Dabu, A. Manikntty and P. Cao (1998), "Providing Differentiated Levels of Service in Web Content Hosting", in Proc. of Internet Server performance, Madison, Wisconsin.

5) N. Vasilious and H. Lutfiyya (2000), "Providing a Differentiated Quality of Service in a World Web Server" in Proc. of the Performance and Architecture of Web Servers Workshops, Santa Clara, California USA, June 2000, pp.14-20.

6) Sook-Hyun Jae-Young Kim and James Won-Ki Hong (2001), "Towards Supporting Differentiated Quality of Web Service" in Proc. of Performance and Architecture of Web Servers Workshops, Santa Clara, California USA, June 2001.

7) Daniel A. Menasce (2003), "Web Server Software Architectures" in journal IEEE Internet Computing Nov/Dec. pp.78-81.

Bibliography

V.Rhymend Uthaiaraj received his PhD degree in computer science & Engg. from Anna university .Now, he is working as a professor in Anna University. His areas of interest includes Network security, Optimization of algorithms, Object oriented modeling, Web design and mobile computing. He has published 40 papers in International conferences and journals.

M.Aramudhan received his Master of Engineering degree in computer science & Engg. from Regional Engg.College, Tirchy. Now, he is research scholar in Anna University. His area of interest includes load balancing techniques, performance and analysis of networks and web technology. He has published 4 international conferences and 2 journals.