Asian Journal of Information Technology 4 (10): 954-956, 2005

A Study on Enhancing QoS Through Dynamic Service Prioritization in Web Services

[1]M. Aramudhan and [2]V. Rhymend Uthaiaraj
Computing Center, Faculty of Ramanujan Anna University, Chennai-25, India

Abstract: This study focus on regulating the throughput of a request to the service class to achieve differentiated Quality of Service (QoS). Assume, all incoming requests are higher priority in differentiated service then its performance degrades. In order to avoid such things, Dynamic Prioritization Middleware (DPM) is proposed. Throughput is the mixture of requests. Throughput of requests is controlled using penalty function present in the scheduler. Managed QoS policies encode both static and dynamic scheduling.

Key words: Web services, service differentiation, dynamic prioritization, access control list

INTRODUCTION

Web services are functional components packaged as a single entity whose capabilities can be accessed at an Internet URI (Uniform Resource Identifier). Standards based web service use XML (Extended Markup Language) to interact with each other, which allow them to link up on demand using loose coupling. Web services use standard Internet protocols such as HTTP (Hyper Text Transfer Protocol), XML, SOAP (Simple Object Access Protocol) to provide connectivity and interoperability between companies. Web services are not tied to any one operating system or programming language. Web services are also called application services. Service differentiation is to treat client requests differently based on client needs and servers resource limitations[1]. Network alone is not sufficient to support end-to-end service differentiation.

Middleware is called plumbing because it connects two sides of the applications and passes the data between them. Middleware is a layer of software between client and server processes that deliver the extra functionality. This software provides services such as identification, authentication, authorization, directories and security. The proposed middleware receives the request from the clients and prioritize the requests based on application, device and user level. These requests are placed in the appropriate queues and processes them using strict priority scheduling. This policy brings about the danger of request starvation in which a low priority request waits indefinitely for its turn amid a continual flow of higher priority requests[2]. It is possible to provide prioritize without starvation using probabilistic scheduling policy. Throughput is the mixture of requests. Therefore, for every m requests of higher priority n requests of a lower priority should also be processed. Generally, differentiation of service is achieved with overhead.

This study focus on managing QoS through control of request throughput. Initially, requests are prioritized by user level such as member and non-member using static IPaddress. All incoming requests are members and it is placed on higher priority queue. Once, the incoming requests exceed threshold limit then its performance starts degrades. In order to avoid such condition a DPM is proposed.

EXISTING WORK

Static service differentiation is based on Application, Device and User level. In existing work, random number generation is applied for priorities the request[1]. Yoon and Kim proposed application level TCP connection management mechanisms of web servers to provide two different levels of web service high and default priority by setting different timeouts for inactive TCP connections[3]. The service prioritization is done by user level. All registered users are stored their IPaddress either in the server side database or DB server. The high level priority performance is improved by as much as 10-15%.

The Message riented iddleware voids rocessing of multiple requests at a same time. The bottleneck is all requests process via this middleware. Its performance is better than greater number of threads by the server at a time[2,4]. The strict priority scheduling is used for processing stored requests in the queues. This scheduling is suboptimal because of not only starvation and also its performance in the applications traffic. The probabilistic scheduling helps to avoid starvation in kernel level and penalty function regulates the throughput in application level. There is no industrial standard for scheduling algorithm. Each scheduling algorithm provides better performance in any one of the applications traffic.

Corresponding Author: M. Aramudhan, Computing Center Faculty of Ramanujan Computing Center, Anna University, Chennai-25, India

DYNAMIC PRIORITIZATION MIDDLEWARE (DPM)

The architecture of DPM is shown in Fig. 1. The middleware contains listener, classifier, access control list and counter. Listener watches the incoming requests to the server in the default port 80 and 90.

The requests of PC and Mobile devices are entered through 80 and 90, respectively. Initially, classifier differentiates service based on user level priority. Member addresses are stored in the database. Members are identified and placed in the higher priority queue. Counter is a flag which gives number of requests in the higher priority queue. The request in the higher priority queue exceeds the threshold limits composite priority level is enforced. The DPM algorithm is given below:

Step1: Observe the number of requests placed in higher priority queue using first level prioritization.

Step2: Observe the number of requests left for processing.

Step3: Every 5 seconds looks the number of requests in the higher priority queue using both end counters.

Step 4: The value of step3 exceeds the threshold limits (request where the performance degrades) invoke composite level prioritization (Two level prioritization say user and application level). Other wise continues with step 1 operation.

Step 5: go to step 3.

The penalty function adjusts throughput on the basis of the following algorithms:

Step1: Observe the number of requests of each type. Let Ni denote the number of incoming web services requests.

Step 2: Calculate a ratio parameter for each request type. The formula used is

$$Ri = (n * Ni) / \Sigma Ni$$

Where n is the number of different request types and Ri is the ratio parameter for the ith request type, whose count is Ni. For example, if the requests are N_a=10,N_b=12 and Nc=14 in number, the ratio for requests type A becomes Ra= 3 *10/36=.833(less than 1)

Step 3: Calculating a penalty function for each request type. Given a Ratio value of Ri, the formula is:

$Ti=1-(1/(K_R*R_i*R_i))$;
If (R_i>1)
$Pi=1+ ((R_i*R_i)/K_P)$;
else if (R_i = =1) Pi=1.0
else if (R_i <0) Pi=Lower bound;

K_p and K_R are the negative and positive normalization constants and keep the penalty values with in desired bounds. The constant control range of penalty curve K_p and K_R are 100 and 40 respectively. Lower bound is a lower ceiling for the penalty.

Step 4: The penalty parameter is multiplied with the Time for which the request queue is polled.

$$Ti=Ti*Pi$$

For example, there are 3 types of request priority. The numbers of requests in each priority are 40, 31 and 29 respectively. The penalty value attached in each priority is 1.03, 0.988 and 0.986 using above algorithm. The idea behind is maintaining the ratio of the request throughputs with respect to the incoming request traffic.

SIMPLE EXPERIMENT

This experiment was conducted on test bed with an IBM system x86 Family 6 models 8 stepping 10 AT/AT Compatible and 128 RAM running Servlet technology, Tomcat 4.1 as the web server under Windows 2000. All incoming requests were entered in to web server via port 8080. In static scheduling, the requests were prioritized based on application, user and device level. The incoming requests were prioritized using user level. The site member IPaddresses were stored in the database. The different version of web pages was viewed by the member and non-member of the site. The main advantages of servlet are portability, flexibility and integration with the server. In this experiment two different version of web pages were

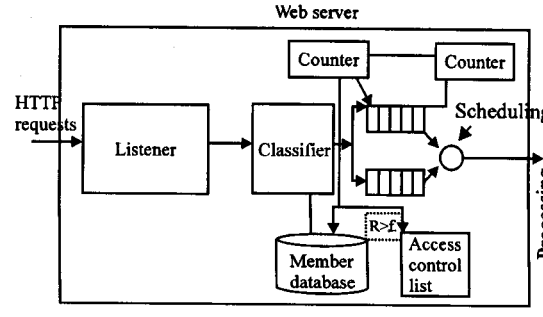

Fig. 1: Dynamic prioritization middleware

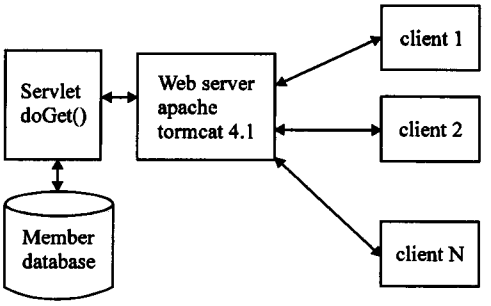

Fig. 2: Experimental architecture

created. First one is generated with restricted information where as next one was complete information including images and video clip. The classification identified the member and non-member of site using centralized database. The complete version page was accessed by the member. The correct web page was returned to the client's category. Figure 2 shows the architecture of simple experiment.

The strict priority scheduling yields poor performance because of not only starvation and also its application performance[5]. This issue is over come using probabilistic scheduling and for every m requests of higher priority n requests of lower priority should be processed. The throughput is regulated using penalty function. The dynamic and static algorithm is placed in the scheduling and it is performance will be evaluate in future.

The number of accesses made by the client to the web page was identified by session tracking. The classification was taken place only when session value is null. Another issue is that in many instance there will be multiple users behind the same proxy. In this situation the server sees all the users as having the same IP address.

In next level, the incoming requests are prioritized and placed in different queues. The higher priority queue is process before lower priority. Assume, all incoming requests are higher priority then the performance degrades. In order to maintain its performance DPM is proposed. DPM acts as a middleware, therefore, all incoming request enters through it.

CONCLUSIONS

In practice, service differentiation is achieved with little overhead. Web server could not able to processes all requests with good QoS having limited resources. In certain cases, differential resource allocation schemes for different requests based on request type have been developed. The resource allocations have been made on basis of request content or the percentages of request arrivals. Higher priority requests are processed with managing QoS through DPM.

REFERENCES

1. Yoon-Jung, R., H. Eun-Sil and K. Tai-Yun, 2002. Connection management for QoS service on the web. J. Network and Computer Applications, 25: 57-68.
2. Sharma, A., H. Adarkar and S. Sengupta, 2004. Managing QoS through prioritization in web services. Proceedings of the 4th international Conference on Web Information Systems Engineering Workshops (WISEW) IEEE.
4. Aramudhan, M. and V.R. Uthaiaraj, 2003. Web server: A study on model and its performance. Proceedings of National Conference on Recent Trend on Communication Longwal, Punjab, pp: 329-335.
3. Yoon-Jung, R. and K. Tai-yun, 2002. Connection management for QoS service on the web. J. Network and Computer Applications, pp: 57-68.
5. Aramudhan, M. and V.R. Uthaiaraj, 2004. Admission based web server model and its timeliness. Proceedings of Internation Symposium M2USIC '04, pp: 13-17.