# Queuing Theory Based Open Loop Control of Web Server

K.Hareesh Kumar and Somanath Majhi

***Abstract*— All the webservers today consider incoming connections on equal ground. As long as the server is lightly loaded there is not much a large difference in the response time but as the server becomes heavily loaded the response times may increase dramatically. Some E-commerce sites would like to give preferential treatment to their regular customers who are generating revenues and provide faster response time to them over other users who are merely browsing through their website. In this work the requests are prioritized on the basis of their URL in middleware layer. To select the requests in middleware layer queues, queuing theory is used.**

## I. INTRODUCTION

The traditional approach to achieving performance guarantees in computing systems has been to quantify hardware speed and software execution requirements, then apply an appropriate combination of pre-run-time analysis, admission control, and resource allocating algorithms to ensure that the system is not overloaded and the desired performance is achieved [1]. Support for different classes of QoS on web has been investigated recently. An important measure of quality is the responsiveness of the server. In the simplest case it is desired to differentiate between two classes of clients; *premium* and *basic*, such that premium clients receive better service at overload. In the literature a server architecture is proposed that maintains separate service queues for premium and basic clients allowing them to be handled separately in an appropriate order [2]. Maintaining QoS and overload protection using classical feedback control techniques are also investigated in literature [1]. But the problem with this method is selection of the requests must be based on the error value. This technique may not be suitable for dynamic environments. The utilization bound for periodic requests is given in [3] and for aperiodic requests was given in [4].

## II. PROBLEM DEFINITION

The problems in loaded servers which lead to lower response times are: a) The TCP listen queue does not distinguish between preferred requests and ordinary request to be serviced. b) Mostly used webservers do not distinguish the preferred requests over non-preferred ones for allocation of CPU time as well. c) Present, operating systems such as Linux, have only a single outgoing queue without priority. In this work, the above 2nd point has been taken up.

This work was not supported by any organization

K.Hareesh Kumar was a post-graduate student in the Department of Electronics and Communication Engineering, Indian Institute of Technology Guwahati

Somanath Majhi is with the Department of Electronics and Communication Engineering, Indian Institute of Technology Guwahati, Guwahati - 781039, India smajhi@iitg.ernet.in

## III. SERVER UTILIZATION

If a Uniform Resource Locator (URL) of size $x$ is requested, the request service time can be approximated by

$$T(x) = a + bx \tag{1}$$

where $a$ and $b$ are platform constants. Summing the service times of all requests in a particular observation period and dividing by the length of the period we can obtain server utilization $\rho$.

$$\rho = aR + bW \tag{2}$$

where $R$ is the observed request rate, and $W$ is the aggregate delivered bandwidth [5]. The values of platform constants $a$ and $b$ can be found using online estimators described in [6].

## IV. BOUND FOR APERIODIC REQUESTS

*Theorem*:A set of aperiodic tasks is schedulable using an optimal fixed-priority scheduling if $\forall t : \rho(t) \leq \frac{5}{8} + \frac{1}{8(M-1)}$, where $M$ is the maximum number of current tasks in the system, and $\rho(t)$ is the current utilization at time $t$.

From the above theorem, the least upperbound to the processor utilization for the aperiodic requests is nearly $\frac{5}{8}$ [4]. For better service of requests at overload, server must operate at server utilization nearly around $\frac{5}{8}$ . This theorem is valid only for single processor and not applicable for multiprocessor environment.

## V. ARCHITECTURE OF THE SERVER

Figure 1 depicts the architecture of a server that uses *middleware* approach described in [7], [8]. This server architecture comprises of an unmodified web server that is augmented with a QoS-aware middleware layer. The middleware layer enables the server to offer differential qualities of service to users, without requiring any changes to the web server code. In order to do so, the middleware layer intercepts a web server's TCP socket calls [9], interfaces directly with the server's TCP/IP stack, receives and process requests, and forwards them to the web server for service.

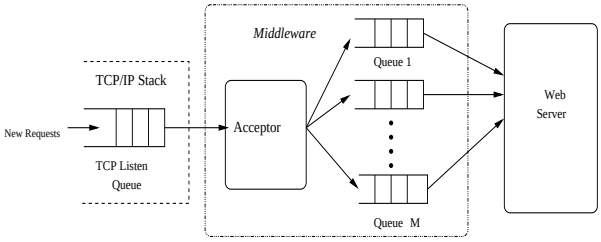


Fig. 1. Architecture of a Server

### A. Acceptor

The acceptor is the component of the middleware layer that interfaces with the server's TCP/IP stack. The acceptor listens on the server's TCP port (usually 80) and is responsible for receiving new connections. For each request that it receives on an incoming connection, the acceptor performs *classification* of requests based on the URL. If the URL is important for transactions, the request will be placed in high priority queue (Queue1). These priority's are not related to operating system. These are only for user importance classification. The priority of queues decreases from Queue 1 to Queue M.

### B. Selection Procedure

The queues will be processed for every $T$ seconds. First we will see how many requests are there in each individual queue. Let $\lambda_1, \lambda_2, ....., \lambda_M$ are arrival rates in corresponding queues. The server utilization corresponding to each queue is given by

$$\rho_i = \frac{\lambda_i}{\mu} \qquad (i = 1, ..., M) \tag{3}$$

where $\mu$ is departure rate of the server (this departure rate mainly depends on system resources) and $\rho_i$ and $\lambda_i$ are server utilization and arrival rate, respectively for the corresponding queues. The total server utilization is given by the following relation

$$\rho_{total} = \sum_{i=1}^{M} \rho_i \tag{4}$$

From the equation 3, the estimation equation is

$$\rho_{new(i)} = \frac{\lambda_{new(i)} \cdot \rho_{previous(i)}}{\lambda_{previous(i)}} \qquad (i = 1, ...., M) \tag{5}$$

The least upper bound for aperiodic requests from Theorem 1 is $\frac{5}{8}$. So server needs to be maintained at server utilization below this bound always. At overload, server is maintained at server utilization around $0.6$. The algorithm for selection of requests is given below. In this algorithm rejection of requests are indicated by sending a short message "SERVER IS BUSY, TRY AFTER SOME TIME" to the client.

Step 1: Initialize server utilization $\rho = 0$.

Step 2: Estimate the server utilization of each queue based on previous period data. If the sum of server utilization of processing queue and previous processed queues is less than 0.60, process all requests in that queue otherwise calculate how many requests to be served to achieve server utilization to be nearer to 0.60 and reject all the remaining requests in that queue and the requests in remaining unprocessed queues. This process is repeated for all queues.

Step 3:Calculate server utilization serviced requests using $\rho = aR + bW$.

Step 4:Go to step 1 after $T$ seconds.

## VI. Simulation Results

The multithreaded client, server programs are done in Java [10]. For implementation purposes jdk1.3.1 version was used. For simulation purposes some pages have been put in server. The clients are run from the other systems. In Fig 2 the solid line represents expected load at those time instants due to clients and the discrete line represents actual server utilization at those instants. The dotted line is 60% server utilization line.

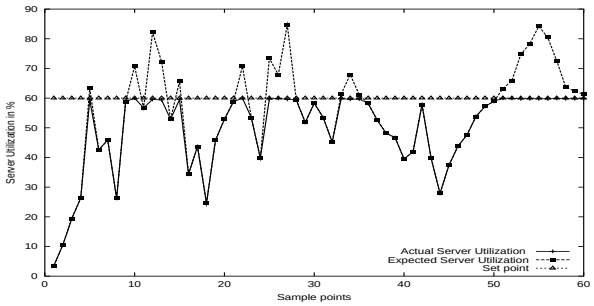


Fig. 2. Simulation result

## VII. Conclusions and Future Works

In this work the requests are prioritized on the basis of their URL in middleware layer. To select the requests in middleware layer queues, queuing theory is used. All the measurements in this work were done in Java, by writing client and server programs in Java. Our proposed algorithm works for both static and dynamic content cases. In this work the assumption is that the dynamic content (like CGI scripts) execution periods are same over a period of time. Until now all QoS enabled servers are open loop in nature. For designing the server in closed loop system we need adaptive controllers, which shall change its parameters based on data of previous periods. In future intelligent controller like fuzzy logic controllers shall play a major role in QoS enabled server architectures.

## References


[1] T.F.Abdelzaher and C.Lu, Modeling and performance control in Internet servers. *IEEE CDC*, Sydney, 2000.

[2] N.Bhatti and R.Friedrich, Web server support for tiered services. *IEEE Network*, 13(5), 1999.

[3] C.L.Liu and J.W.Layland, Scheduling algorithms for multiprogramming in a hard real-time environment. *J.of ACM*, 20(1), pp. 46-61, 1973.

[4] T.F.Abdelzaher, A schedulable utilization bound for aperiodic tasks. University of Virginia, CS-2000-21, August, 2000.

[5] T.F.Abdelzaher and N.Bhatti, Web server QoS management by adaptive content delivery. *Int. Workshop on Quality of Service*, London, UK, 1999.

[6] T.F.Abdelzaher, An automated profiling subsystem for QoS-aware services. *Real-Time Technology and Applications Symposium*, Washington, D.C, June, 2000.

[7] T.Jin and J.D.Salehi, Capacity guarantees for web servers. Hewlett-Packard Laboratories, HPL-98-155, June, 1998.

[8] P.Bhoj, S.Ramanathan and S.Singhal, Web2K: Bringing QoS to Web Servers. ISAL,HP Laboratories, Palo AltoHPL-2000-61, May, 2000.

[9] W.R. Stevens, UNIX Network Programming. Prentice Hall of India Private Limited, 2000.

[10] D.Flanagam, Java in nutshell. O'Reilly & Asoociates, 1996.