Improving User-Perceived Performance at a World Wide Web Server

Amy Csizmar Dalal
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

Scott Jordan
University of California at Irvine
544D Engineering Tower
Irvine, CA 92697

***Abstract*— We examine a method to improve the service of incoming requests at a World Wide Web server. The motivating factor is the existence of an impatient user pool: a user aborts a pending web request if a response is not received within a random timeout period. We use a queueing theory approach to derive an optimal service ordering for this server, assuming Poisson arrivals and exponential service times. We find that the optimal policy is greedy, in that at any time the server processes the request with the highest perceived payoff. We verify these results both analytically and via simulation.**

I. Introduction

Most of today's web servers handle incoming web requests in a round robin manner. This method of scheduling requests is partly a consequence of operating system design and partly due to web server software design. Traditionally, it has been held that processor sharing policies such as round robin yield the best "user-perceived" performance, in terms of fairness and response time, when service time requirements are variable. Processor sharing policies typically allow shorter jobs to complete before longer jobs, and the average time a job spends in a system that employs processor sharing is linearly proportional to its required service time.

However, this theory falls short when applied to the construction of web server software. Web users tend to be very impatient and will abort a pending request if a response is not received in a matter of seconds. Users that time out in such a manner cause the server to waste resources on a request that never completes. At a heavily-loaded server with many requests arriving per second, this may ultimately prove disastrous, and lead to a situation of "server deadlock", where the server works at capacity but does not complete any requests.

The objective of the web server in the system studied here is to maximize user perceived performance, which is a decreasing function of the amount of time the user spends waiting for a file to download from a web server. We assume that the network connecting the client and server is a static quantity and look solely at what the server does in processing requests. Our metric of interest is a quantity we call "revenue", defined as the probability that the user has not aborted a request before it is filled by the server. We describe revenue as a decaying exponential function of the time the request spends in the system before completing service. Our goal is to find the service policy that maximizes the average revenue earned per second by the server.

Several previous research efforts have utilized similar queueing methods to analyze quality of service issues in web servers. Both [2] and [4], for instance, demonstrate improvements in response time at a web server by using a non-traditional service ordering (shortest-connection-first and shortest-remaining-processing-time, respectively) in place of processor sharing. [5] uses a queueing model to determine how response time at a web server is affected by several parameters, including network bandwidth, processor speed, and adding additional hosts to a distributed server system. However, [5] only considers first-in, first-out service ordering, and does not study how alternate policies may affect response time. [1] proposed an improvement over FIFO service at web servers by using a combination of admission control and a set of priority queues, thus providing better QoS at the web server and increasing the throughput of a web server as seen by the clients. However, none of these studies considers the "impatient user" problem.

The rest of the paper is organized as follows. In Section II, we briefly outline the analytical model developed to represent the web server of interest and describe the derivation of the optimal service ordering policy for two similar web server systems. Section III contains simulation results obtained from a model constructed using BONeS software and a discussion of the results. In Section IV, we present a modification to the original model involving the service time distribution, and discuss its implications. Finally, in Section V we present some brief conclusions.

II. Web server model

The model presented here is an abstraction of the actions that occur at the session level layer due to the HTTP/1.1 protocol. We ignore all actions associated with the network layer, including specifics about individual TCP connections associated with requests. We assume the server processes requests blindly and is not aware if or when a request is aborted before it is processed to completion.

We model the web server as a single-server queue with a single stream of Poisson arrivals. The service times of incoming requests are drawn from an exponential distribution with pa-

This paper represents work completed while both authors were at Northwestern University.

rameter μ, and service times are independent and identically distributed. In this model, the service time of a pending request is directly proportional to the number of bytes in the requested file. Further, we assume that swap times are negligible.

For each request that enters the web server system, the server earns an amount of *revenue* equal to $e^{t_{\text{response}}}$, where t_{response} is the total time a request spends in the system under a given service policy. We additionally define the *potential revenue* of an arbitrary request i at a time t prior to the departure of that request from the system as $g_i(t) = e^{-(t-x_i)}$ where x_i is the arrival time of request i.

We define the "optimal" policy as the one that maximizes the average revenue the server earns per unit time. To this end, we show that the following is true:

Theorem 1 *The optimal policy for the web server defined above is greedy. That is, it chooses to serve the job with the highest potential revenue $g_i(t)$ at any time t.*

The proof of this theorem is outlined below; see [3] for the full proof. Briefly, we show this is true by showing that each of the the following items hold:

1. *An optimal policy is non-idling*; that is, the server will choose to work on a request rather than remaining idle as long as there is at least one request awaiting service. The proof is by contradiction. Supposing that the optimal policy is idling, we construct an alternate policy that swaps an interval from the idle period with a later interval from a non-idle period; this results in an increased expected revenue over the original policy. The result follows.
2. *An optimal policy switches between requests in service only upon an arrival to or departure from the system.* The proof uses the fact that the revenue function is identical for all requests, and thus decays at the same rate for each request over an interval of time.
3. *An optimal policy works on one request at a time.* The proof considers three policies over a sample path containing two requests. Two of the policies operate as follows: work on one request to completion, then work on the other request to completion. The third policy works on both requests simultaneously (using generalized processor sharing) until one departs, and then completes the remaining request. We show that the expected revenue generated by the third policy is a weighted sum of the two work-on-one-request-at-a-time policies over this sample path, and thus is nonoptimal. This result easily expands to a sample path with an arbitrary number of requests.
4. *An optimal policy is Markov*; it is independent of both past service history and future arrivals. The proof is based on the Exponential distribution for the service times and for the interarrival times of requests.

The proof proceeds by using these four lemmas to establish the optimal policy. In this last step, we differentiate between two types of web server systems: one in which there is no differentiation in service level among incoming requests, and one in which incoming requests require different levels of service. We term the first system the "non-QoS system" and the second the "QoS system".

A. Non-QoS system

In this system, each incoming request is weighted equally. The revenue of each request depends solely on its response time. At any time t, we define the *state vector* of the system under an arbitrary service policy p as

$$G_{t_p} = (g_{1_p}(t)\ g_{2_p}(t)\ \ldots\ g_{M_p}(t))$$

where M denotes the number of jobs that have arrived at the system since the beginning of the current busy cycle[1] and have not yet departed under policy p, and $g_{i_p}(t)$ is the revenue function for request i under policy p. Since jobs that have already departed do not affect the service selection, we can assume without loss of generality that these jobs depart the state vector as well upon departure from the system. Thus, the server makes its selection based only on the state at time t, as Step 4 of the proof indicates.

Because the distribution of remaining service time is identical for every job in the system at time t, the expected revenue gained from serving any job i beyond time t with remaining service time t'_i is $E[g(t'_i)] = E[g_{i_p}(t)e^{-t'_i}] = g_{i_p}(t)\,\mu/(1+\mu)$, where the constant 1 in the denominator is the decay rate of the revenue function and μ is the parameter of the service time distribution. The job with the highest expected payoff satisfies $\max_i\ g_{i_p}(t)\mu/(1+\mu) = \max_i\ g_{i_p}(t)\ , i \in 1, 2, \ldots, M$. The optimal policy is thus to serve the job with the highest potential revenue $g_i(t)$, and the theorem follows.

In the non-QoS system, the optimal policy processes incoming requests in a preemptive last-in, first-out (LIFO-PR) manner. That is, the web server processes the newest job in the queue until either the request completes service, at which time it begins processing the next newest request in the queue; or until another request arrives, at which time the server preempts the job in service to process the new job.

B. QoS system

In this system, each incoming request is weighted differently. We denote the initial revenue of an arbitrary request i as C_i. The potential revenue equation for request i for this system is thus $g_i(t) = C_i e^{-(t-x_i)}$.

The proof of the optimal policy for this system is similar to that of the optimal policy for the system with no QoS differentiation. The entries in the state vector rely on both the response time and the initial reward of each request in the system at time t. Theorem 1 still holds, but with the new $g_i(t)$ to account for the QoS differentiation. (Again, please refer to [3] for the proof.)

[1] A busy cycle refers to any time period where there is at least one request pending at the web server.

In the QoS system, therefore, the optimal policy processes requests in the following manner: At any time, the web server will choose to serve the request that is the most profitable combination of time-in-system and initial payoff. We refer to this policy as a *parameter-based*, or PB, policy.

III. Simulation results

To further verify our results, we model the system described above using BONeS software. The model implements the optimal service ordering policy for both the QoS and non-QoS systems, along with two other service orderings commonly found in web server software architecture: first-in, first-out (FIFO) and processor-sharing (PS).[2]

We compare the total normalized revenue earned by the server per simulation in each service ordering for a range of service rates and arrival rates. Because the simulation time is much longer than the time required for the queue to empty, we look at the total revenue collected per simulation rather than the revenue per busy cycle. The total revenue measure serves as a good approximation, when normalized, to the average total revenue seen per busy cycle.

The requests are generated according to a Poisson distribution with parameter λ; this quantity varies among the simulations to represent a variety of systems, from light-traffic to heavy-traffic. The service rate of the server, μ, is calculated from the arrival rate and the offered load.[3] The offered load is the variable in each simulation.

For the non-QoS system, the initial revenue multiplier is set to 1.0 for each incoming request; for the QoS system, the initial revenue multiplier is drawn from a uniform distribution with endpoints $[0, 1)$.

Fig. 1 shows the normalized revenue plots for average arrival rates of 10 and 100 requests per second for the non-QoS system. Additionally, we plot the revenue function evaluated at the average response time for an M/M/1 system, i.e., $e^{-w},\ w = \rho/(\lambda(1-\rho))$, for comparison.

The plots decay slowly at low loads before rapidly falling off at higher loads. As the arrival rate increases, the load at which this rapid decay starts increases. Higher arrival rates translate into higher service rates at fixed load, which means that jobs are serviced in a shorter amount of time in the simulations with high arrival rates than in those with low arrival rates.

The benefits of using LIFO-PR are particularly pronounced at high server loads, where there is a significant difference in revenue between this service order and the traditional service orders. This improvement is partly due to the high variances of the LIFO-PR response times as compared to the response time variances under FIFO and PS. Requests with long service time requirements have a high probability of being preempted by requests with shorter service times. A shorter job has a better chance of completing quickly, thus registering a short response time with the server. Since the revenue function is exponential, it decays rapidly at first but then slows to a gradual decay within a short time period. The shorter response times "seen" in LIFO-PR weigh more heavily on the total revenue in the simulation.

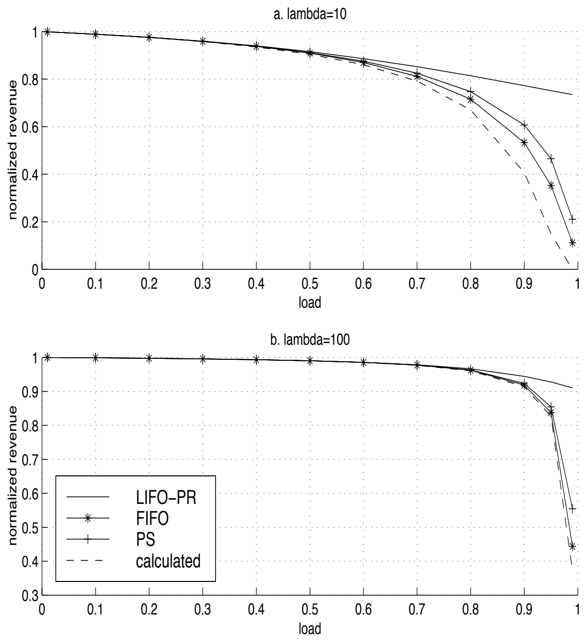

Fig. 1: The normalized revenue as a function of the offered load for the non-QoS system. (a) $\lambda = 10$ requests/sec, (b) $\lambda = 100$ requests/sec

Fig. 2 shows the normalized revenue as a function of load for the same two arrival rates for the QoS system. The maximum values for each plot are around 0.5, the mean of the initial reward distribution. We plot $0.5e^{-w}$, the revenue calculated at the expected M/M/1 response time and scaled by the mean initial reward, for comparison.

The traditional service order plots and the calculated plots are the same as the plots from the original system scaled by 0.5. We see here, though, that the new optimal service order, labeled "PB" in the figure, outperforms the previous optimal policy, LIFO-PR, albeit slightly. The difference between these two policies is most pronounced at high loads.

IV. An extension to the original model

So far, we have described a system in which all service times are drawn from the same distribution. In this section, we explore the implications of relaxing this requirement. We consider a system in which service times are drawn from a set of M independent exponential random variables, with parame-

[2]For the second system, we additionally consider LIFO-PR as one of the "other" policies to which to compare the optimal policy.

[3]Recall that the service rate is directly proportional to the file size; the file size is the actual variable used here, and we use a multiplier to convert to service rate in the simulation run. We use a mean file size of 2 kilobytes.

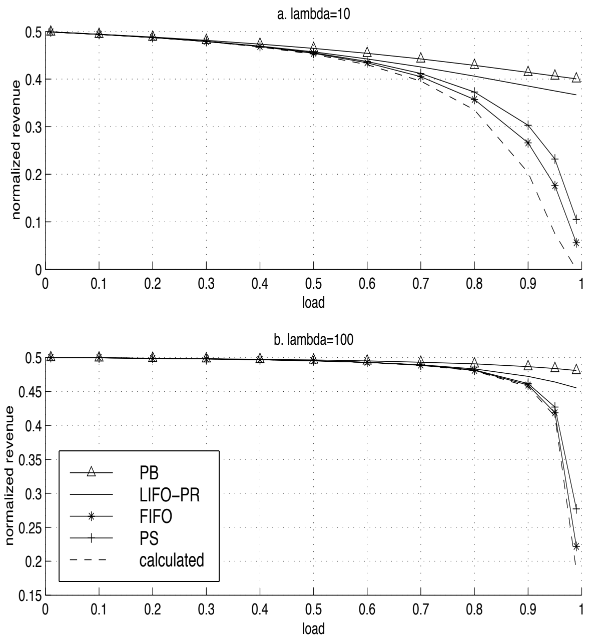

Fig. 2: The normalized revenue plotted as a function of offered load for the QoS system. (a) $\lambda = 10$ requests/sec, (b) $\lambda = 100$ requests/sec

ters $\mu_m, m \in 1, 2, \ldots, M$. This corresponds to the case where a web server contains several different content types (HTML, images, CGI scripts, and so on) that can be best described by different distributions.

A. *Optimal policy*

To derive the optimal service policy for this system, we assign each incoming request i a file size drawn from an exponential distribution with parameter μ_i, and assume a non-QoS system. (A similar argument could be used to derive similar results for a QoS system as well, although we will not explicitly show this here.) Define the class of non-processor sharing policies to exclude policies which do not work on one request at a time and those which switch between requests in service at times other than arrival or departure epochs.

We present a characterization of the optimal policy, similar to Theorem 1, but among a more limited class of policies.

Theorem 2 *The optimal policy for the web server defined above, among the class of non-processor sharing policies, is greedy. That is, it chooses to serve the job with the highest potential revenue per unit time $g_i(t)\,\mu_i$ at any time t.*

The proof of the theorem can be found in [3]. It proceeds by showing that the optimal policy is both non-idling and Markov, as before. The state vector is now defined as

$$G_{t_p} = \{(g_{1_p}(t), \mu_1)\ (g_{2_p}(t), \mu_2)\ \ldots\ (g_{M_p}(t), \mu_M)\}$$

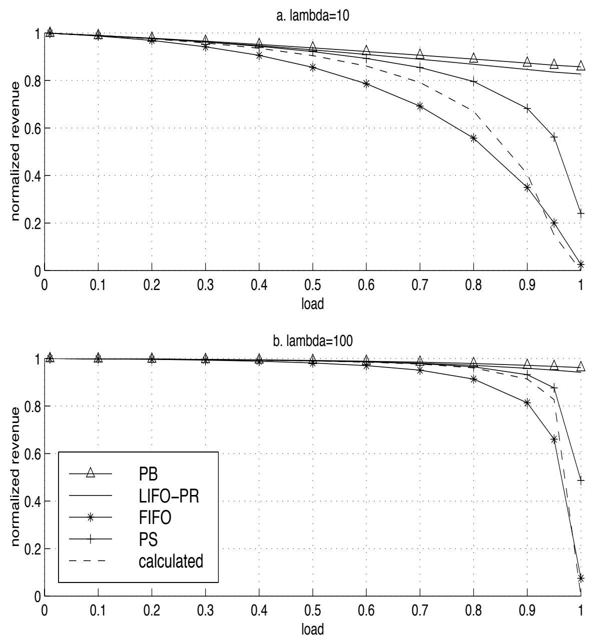

Fig. 3: The normalized revenue plotted as a function of offered load for the variable service times extension. (a) $\lambda = 10$ requests/sec, (b) $\lambda = 100$ requests/sec

Since the requests are serviced at different rates, we must now consider potential revenue *p*er unit time, which is equal to the expected revenue divided by the expected completion time, or $g_i(t)\,\mu_i$.

Fig. 3 shows the normalized revenue earned by the server under the optimal policy and compared to the "traditional" service orderings for two values of arrival rate. We limit the subset of file sizes to three, with means of 2 kilobytes (representing HTML files), 14 kilobytes (representing image files), and 100 kilobytes (representing "other" files, such as CGI and Java programs). Note that the plot also includes the revenue evaluated at the average response time for an M/M/1 system as a point of comparison.

For this set of simulations, the differences between the revenue earned by the optimal service order ("PB") and LIFO-PR are even smaller than in the QoS system simulations. Both policies hold up well even under very high server loads. Note that the FIFO policy degrades much more rapidly in terms of generated revenue as the load increases than in either of the two previous systems, which can be seen by comparing the FIFO line to the M/M/1 line in the plot.

B. *Processor sharing policies*

The optimal policy was shown to be greedy among non-processor sharing policies. In this subsection, we demonstrate that processor sharing policies may generate a higher average revenue than non-processor sharing policies when service times are drawn from a set of M independent exponential random variables,

Consider a sample path that consists of only 2 jobs. Suppose job 1 arrives at time zero, and job 2 arrives at some time $a > 0$, where a is less than the service time required by job 1. The busy cycle ends at some time $b, b > a$.

We consider three service policies, labeled NPS1, NPS2, and PS. The three policies behave as follows in the interval $[a, b]$:

- **NPS1**: The server processes job 1 to completion, queueing job 2. Upon job 1's departure, the server processes job 2 to completion.
- **NPS2**: The server processes job 2 to completion, queueing job 1. Upon job 2's departure, the server resumes processing job 1 to completion.
- **PS**: The server processes both job 1 and job 2 simultaneously, by means of some sort of processor sharing, until one of the jobs completes service and departs, at which time the server dedicates all of its resources to processing the remaining job to completion.

Unlike the equal service rate case (item 3 of Theorem 1), the expected revenue for policy PS is not a weighted sum of the revenues generated by policies NPS1 and NPS2.

In fact, we can find such sample paths for which the revenue gained by PS exceeds both the revenue gained by NPS1 and by NPS2. The revenues for each policy are shown in Fig. 4 for a set of such sample paths. The expected revenue is plotted as a function of the potential revenue of job 1 evaluated at the time when job 2 enters the system; the initial revenue for job 2 is set to be 1. The difference in expected revenues is small, but clearly the expected revenue from the PS policy is greater than the revenue from either NPS1 or NPS2 over the range of values shown.

It is interesting to note that we have so far been unable to duplicate these simulation results for sample paths that contain more than two requests, when arrivals form a Poisson process. We speculate that the periods where a PS-like policy performs better than an NPSx-like policy occur less frequently than period over which the reverse holds; thus, over a long time epoch, the effects of PS-like policies on the total simulation revenue are averaged out.

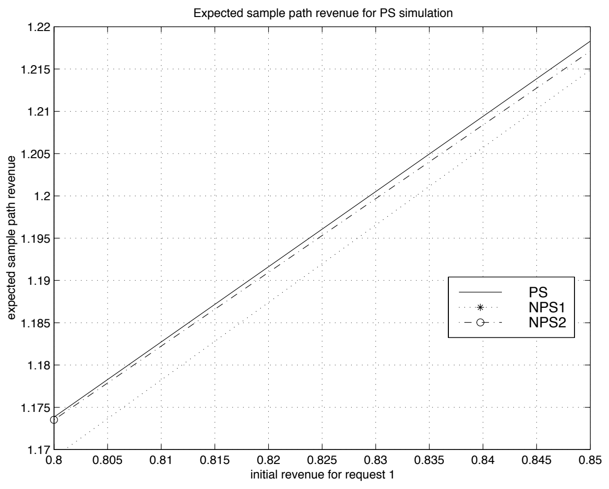

Fig. 4: Expected revenue generated by each of the policies under consideration in the counterexample. The plot is zoomed in to show one range in which the PS policy generates a higher expected revenue than the two non-PS policies.

V. Conclusions

We have shown that when network delays are ignored, an impatient user population is best served using a nontraditional, "greedy" service ordering policy. Server performance is maximized when the concept of fairness is ignored. The possible loss of revenue from requests that "give up" is compensated by the greater payoff from the requests that are served to completion.

The simulation model further verifies the analytical results, and illustrates the performance improvements that can result if such policies are implemented at a heavily-loaded web server. In the simulations considered, the revenue earned by the web server increases by as much as 6.6 times when the optimal policy is used instead of FIFO, and by as much as 3.5 times over the revenue generated by a processor-sharing policy in the non-QoS system. Similar improvements are seen in the QoS system and in the variable service time distribution system, although in the latter the improvement factor of the optimal policy over the FIFO policy is about twice as much as the same factor in the QoS and non-QoS systems. However, caution must be taken in considering variable service time distribution systems, as processor sharing policies may generate higher revenues.

References

[1] Nina Bhatti and Rich Friedrich. "Web server support for tiered services". *IEEE Network*, 13(5):64–71, September/October 1999.

[2] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. "Connection scheduling in web servers". In *USENIX Symposium on Internet Technologies and Systems*, pages 243–254, Boulder, Colorado, October 1999.

[3] Amy Csizmar Dalal. *Characterization of User and Server Behavior in Web-Based Networks*. PhD thesis, Northwestern University, December 1999.

[4] Mor Harchol-Balter, Mark E. Crovella, and Sung Sim Park. "The case for SRPT scheduling in web servers". Technical Report MIT-LCS-TR-767, MIT Laboratory for Computer Science, October 1998.

[5] Louis P. Slothouber. "A model of web server performance". StarNine Technologies, Inc.