Improved Prediction for Web Server Delay Control

Henriksson, Dan; Lu, Ying; Abdelzaher, Tarek

Published in:
Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04)

DOI:
10.1109/EMRTS.2004.1311001

2004

Improved Prediction for Web Server Delay Control [*]

Dan Henriksson
Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
dan@control.lth.se

Ying Lu, Tarek Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
ying, zaher@cs.virginia.edu

Abstract

Control methods are being used increasingly for uncertainty management and QoS in modern web server systems. Previous approaches have suggested combined feed-forward and feedback control strategies, using queuing theory for feed-forward delay prediction. While queuing theory allows one to predict delay as a function of arrival and service rates, the prediction applies only to long-term averages, and is therefore insensitive to sudden load changes. Unfortunately, Internet load is very bursty, leaving room for predictor improvement. The main contribution of this paper is an extension of the combined feed-forward/feedback framework in which the queuing model is replaced with a predictor that instead uses instantaneous measurements to predict future delays. The proposed strategy is evaluated in simulation and by experiments on an Apache web server. It is shown that the new approach performs better than the combined queuing model based feed-forward and feedback control presented in earlier papers.

1. Introduction

The use of feedback control techniques has recently gained much attention in the real-time community, especially as a means to influence the performance of complex software applications, such as web server and router systems. The attractive features of feedback control lie mainly in its inherent robustness to uncertainties, such as modeling inaccuracies, system nonlinearities, and time-variation of system parameters. These types of uncertainties are very common in unpredictable poorly modeled environments such as the Internet.

The traditional way to model server systems is by fluid approximations related to the steady-state response of the server queues. The differential (or difference) equations arising from these flow models are then used to design feedback controllers using traditional linear or nonlinear control methods [21, 4].

The drawback of using feedback-based control alone is that it is a reactive approach, where corrections are made only after disturbances have had a chance to influence the system. Traditional control design often uses a combination of feedback and feed-forward, where the feed-forward controller will react to disturbances before they affect the current system under control. In [18, 22] a solution is presented where a queuing-theoretic prediction was used to augment the feedback loop. However, since predictions from queuing theory apply only to long term averages, they do not handle transient behavior very well. In this paper, we derive a different feed-forward delay predictor that directly relates instantaneous measurements of arrival times and queue length to the average delay over a finite prediction horizon, hence allowing finer-grained prediction and achieving tighter control.

The predictor is then used to dynamically change the service rate in order to fulfill the delay specifications. One example is to use dynamic voltage scaling to manipulate the processing speed of the CPU hence saving energy when the server is under-loaded. The energy cost has been identified as a substantial component of the total cost of running large server farms [8, 23]. Our proposed scheme also applies to other actuator mechanisms, such as resource allocation in a shared environment. The proposed delay prediction and control scheme is evaluated both in simulation and by experiments on a real Apache web server. The simulation study is performed using the TrueTime [10] simulator. It is shown that the new predictor outperforms the previous schemes where queuing theory was used to provide the feed-forward control action.

The rest of this paper is outlined as follows. Section 2 describes the design of the feed-forward predictor and the feedback controller, respectively. A simulation study evaluating the proposed scheme is given in Section 3 and an experimental study is given in Section 4. Section 5 describes related work. The paper concludes with Section 6.

[*]The work reported in this paper was supported in part by the National Science Foundation under grants EHS-0208769, ANI-0105873 and CCR-0093144 and MURI grant N00014-01-1-0576. We are grateful to the anonymous reviewers for their valuable comments.

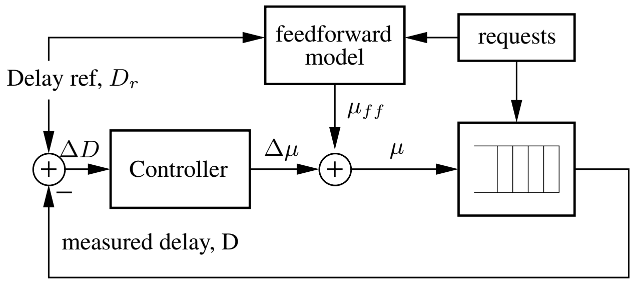

Figure 1. Server control loop.

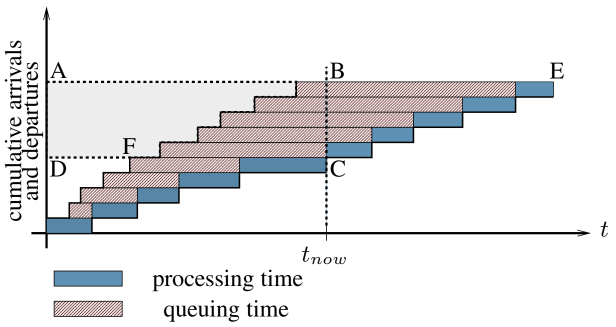

Figure 2. Server queuing and processing delay.

2. Design

In this section, we will describe the assumptions and the developed model used for delay prediction. The derived scheme uses instantaneous measurements of queuing delays and estimated processing times to predict and influence the average delay of future requests. In the derivation of the scheme, we will assume a server that supports some actuation mechanism that can influence the processing rate (referred to as *server speed* in the sequel) of the incoming web requests. Let us denote by C, the average number of processor cycles required to process a typical request. The actual execution time of the request is then given by $\frac{C}{\mu}$, where μ is the current server speed. Server speed can be changed by techniques such as dynamic voltage scaling (DVS), or scheduling algorithms such as the constant bandwidth server, where only a fraction of CPU capacity is allocated to the server.

The general structure of the delay control system under consideration is shown in Figure 1. A feed-forward control action, μ_{ff}, is computed from the reference delay and information regarding the past arrival pattern. This feed-forward signal is then adjusted by a feedback term, $\Delta\mu$. The feedback controller is a gain-scheduled PI-controller that uses feedback from actual delay measurements.

The objective of the control is to keep the service delay (averaged over a short finite interval) as close as possible to a pre-specified reference value. This reference value is typically determined by the QoS specification, where delays consistently longer than the specification are unacceptable to the user. On the other hand, delays consistently shorter than the specification, is an indication of unnecessary consumption of resources. The derivation of the feed-forward predictor and the design of the feedback controller will be described below.

2.1 The Feed-forward Predictor

In order to provide an efficient feed-forward control, it is important to have an accurate model of the delays induced by queuing in the server. The total delay for a request consists of a queuing delay (e.g., in the server's external socket queue, and in the CPU ready queue) and a processing delay (the request being processed by a server thread, and blocking waiting for I/O). This is illustrated in Figure 2.

Consider the situation at the time, t_{now}, of the most recent departed request. At this time there are a number of requests queued up waiting to be processed. Let us consider N of these requests and compute their average delay as a function of the processing speed, μ. In Figure 2 we have $N = 5$.

Assuming that these requests have an average nominal processing time, $\hat{C}$, their individual processing time in next sample will be, $\hat{C}/\mu$. Let $\hat{D}$ denote the average delay experienced by the N requests. We will now provide a geometric derivation of the equation relating the average delay, $\hat{D}$, and the server speed, μ.

The total delay experienced by the N requests, $N\hat{D}$, can be computed geometrically from Figure 2 as the area $BECF$. This is given as the area of the rectangle $ABCD$, plus the area of the triangle BEC, minus the light-gray shaded area ($ADFB$). Noting that the area $ADFB$ is the sum of the arrival times of the requests, we arrive at the equation

$$N\hat{D} = N \cdot t_{now} + \frac{N \cdot (N\hat{C}/\mu)}{2} - \sum_{k=i}^{i+N-1} A_k \qquad (1)$$

Dividing by N, we get:

$$\hat{D} = t_{now} - \hat{A} + \frac{N\hat{C}}{2\mu} \qquad (2)$$

Now introduce $\hat{A}_i = \frac{1}{N}\sum_{k=i}^{i+N-1} A_k$ as the average arrival time, D_i as the average delay, C_i as the average computation time, and μ_i as the server speed for requests being dequeued in the ith sample. We also see that $t_{now} - \hat{A}_i = Q_i$ is the average queuing time for the requests being dequeued in the ith sample. Solving for μ_i then gives:

$$\mu_i = \frac{NC_i}{2(D_i - Q_i)} \qquad (3)$$

which is a predictor equation telling us how to choose the server speed in order to obtain an average delay, D_i, of the next N requests. The feed-forward controller is invoked after each departure and computes the new processing speed according to Equation (3). D_i is chosen as the current delay set-point, D_r. The average queuing time, Q_i, is measured exactly. The average nominal processing time, C_i, is estimated from past measurements. N will be chosen as the current queue length at each sampling instant.

2.2 Design of Feedback Controller

The main short-coming of the feed-forward controller is the fact that the actual processing times of the future requests are not known. As mentioned above, these are estimated based on past measurements. To deal with this uncertainty and to improve the performance of the previously described scheme, a feedback controller is added. The feedback controller uses feedback from current delay measurements, as shown in Figure 1. The control action is computed from the difference, ΔD, between the measurement and the set-point, D_r. The controller is a proportional, integral (PI) controller implemented according to the formula:

$$\begin{aligned}
&\Delta\mu(k) = \Delta\mu(k-1) + g \cdot (\Delta D(k) - r \cdot \Delta D(k-1)) \\
&\Delta\mu(0) = 0
\end{aligned} \tag{4}$$

where g and r are controller design parameters known as the controller gain and zero, respectively.

The relation between the average delay, D_i, and the server rate, μ_i, is nonlinear which means that a change, $\Delta\mu$, in the processing speed, will have different effects depending on the current delay set-point around which we are controlling. The behavior of the system also depends on the arrival rate of the incoming requests. For this reason the PI-control implementation exploits gain-scheduling [27] where different controller parameters are used based on the current set-point and estimated arrival rate.

Other, more sophisticated, non-linear control techniques, or even stochastic control could be proposed but are hardly necessary. The main reason for this is the effectiveness of the proposed feed-forward predictor as shown in the simulation and experimental sections below. Since the average delay is brought close to the delay set-point by the feed-forward action it is sufficient to use linear PI-control action to increase the performance around the set-point. This mainly means eliminating small stationary errors.

System models used for controller design were estimated using simple step response experiments for different set-points and arrival rates. Small changes of the server speed were applied and it was observed how the average delay was influenced. These experiments gave the stationary gain and the approximate speed of the system. Based on these first-order approximations of the system dynamics, a number of controller parameters, g and r, were pre-calculated for different set-points and arrival rates. These values were stored in a lookup table and appropriate parameters were then chosen dynamically on-line as the set-point or arrival rate changed.

Observe that all actuators have saturation limits (such as when all available resources have been allocated to the server, or when the DVS scheme operates at the maximum voltage level.) Hence, some extra care must be taken in the control design. Typically the most direct implication of this is that the control signal will often saturate at its limits. This makes it crucial to implement anti-reset windup of the I-part to prevent integrator windup in the controller.

3. Simulation Study

The proposed feed-forward predictor has been evaluated in a simulation study, which is described below. In the study the performance of the feed-forward predictor is compared with previous approaches based on queuing theory. It is also shown how the performance is influenced by the addition of the gain-scheduled PI-controller.

3.1. Simulation Environment

The simulations were performed using the MATLAB/Simulink-based toolbox TrueTime [15, 10]. The simulator allows co-simulation of several computer nodes each running pre-defined tasks scheduled according to a chosen policy. The code that the tasks execute is written as MATLAB m-files.

A client/server scenario was simulated with a client application sending web requests to a server simulating the basic properties of the HTTP/1.1 protocol. Using this simulation model it is possible to experiment with different processing times and priorities of the requests, different scheduling policies of the threads in the server node, and different control strategies to change the service rate of incoming requests.

3.1.1 The Client

The client node generates synthetic web requests that are sent to the server. The inter-arrival times, (i.e., the intervals between the sending of subsequent requests on a connection), follows a bounded Pareto distribution. The Pareto distribution has been reported to fit measurements of real web traffic very well [14]. When sending the first request on a closed connection (a connection is closed if no new requests are sent within the HTTP/1.1 TIMEOUT), the client awaits an acknowledgment from the server before sending further requests. Thereafter, requests are pipelined on the connection (i.e., multiple requests can be issued without waiting for each response). Each request has an associated simulated processing time and a blocking time that it will consume on the server side. These are also Pareto distributed.

3.1.2 The Server

The server node contains a number of simulated server threads, and a high-priority thread that handles incoming

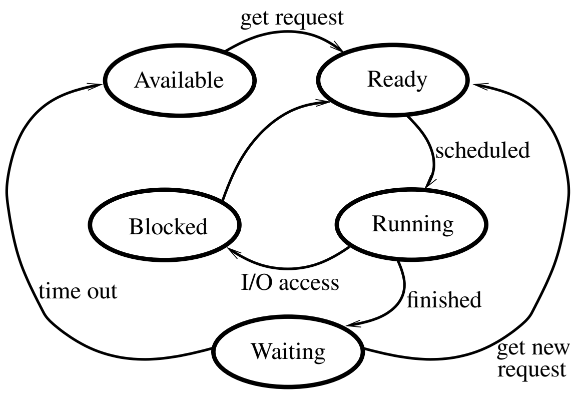

Figure 3. Service request state diagram.

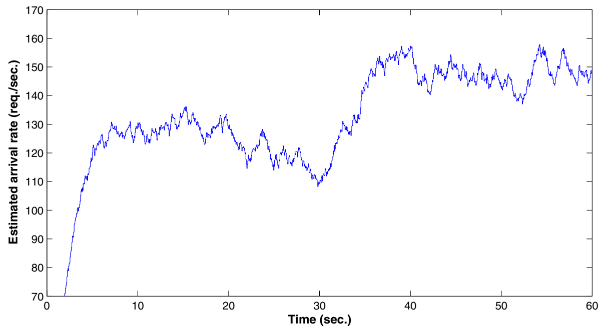

Figure 4. Estimated arrival rate.

requests. All incoming requests are time-stamped, and then either forwarded to the socket queue of the server thread serving the connection, or put in a global request queue if there is no idle server thread available. When server threads are finished serving their current connection they check the global input queue for new requests.

Each server thread operates according to the state diagram shown in Figure 3. The time associated with the Running and Blocking states are Pareto distributed and specified in each request. When the server thread is finished, it waits for a certain time (HTTP/1.1 TIMEOUT) for new requests, before closing the connection.

The server node also contains an h-periodic low-priority thread that estimates the arrival rate by measuring the number of arrivals between each invocation. The arrival rate computed in each sample, $\lambda_a(k) = \frac{Arr(kh)-Arr(kh-h)}{h}$, is smoothed using a recursive first-order filter to obtain the estimate, $\hat{\lambda}_a$:

$$\hat{\lambda}_a(k) = \alpha \cdot \hat{\lambda}_a(k-1) + (1-\alpha) \cdot \lambda_a(k) \tag{5}$$

The forgetting factor, α, was chosen according to the trade-off between transient and steady-state behavior of the estimate. If the value is too high, the estimate reacts too slowly to changes in the arrival pattern, whereas a too low value gives large fluctuations in the estimate. In the simulations below, α was chosen to 0.85.

3.2. Predictor and Controller

The combined feed-forward predictor and feedback controller described in Sections 2.1 and 2.2 were implemented in an aperiodic task that was triggered after each departure. The estimated nominal processing time, $\hat{C}$, of the requests was estimated using the recursive first-order estimator:

$$\hat{C}_k = \beta \cdot \hat{C}_{k-1} + (1-\beta) \cdot c_k \tag{6}$$

where c_k is the last measured processing time. The forgetting factor λ was chosen to 0.85.

N in Equation (3) was chosen as the number of currently queued up requests at each sampling instant. In the case that no requests were queued up, the server speed was put to its minimum value. In the other special case, where Equation (3) produces a negative value (corresponding to the situation where the average queuing time of the requests already exceeds the delay set-point), the server speed was set to its maximum value.

In this section, we consider a DVS actuator with voltage levels in the interval $[1, 5]$. The value $\mu = 5$ then corresponds to the maximum speed, and $\mu = 1$ to the minimum server speed.

3.3. Simulation Results

A 60-second simulation scenario was run, during which a reference delay of 0.6 seconds was enforced. The average arrival rate in the beginning of the run was around 125 requests per second. This rate then jumped to 150 requests per second at time 30. Figure 4 shows the arrival rate computed by the estimator task during a simulation. The average processing times of the requests were chosen such that the input load in the beginning of the run corresponded to 190% of server capacity when running at the minimum speed. The average input load was then increased to 225% at time 30. The results of the simulations are shown in Figure 5.

The obtained average delay when using the feed-forward predictor (Equation (3)) only are shown in the bottom left (plot c.) of Figure 5. The average delay is measured using an observation window of 100 departed requests. The results are quite satisfactory already without the addition of feedback control. However, a small steady-state error is visible in the graphs at the higher arrival rate.

To remove this error, the PI-controller described in Section 2.2 was introduced. The corresponding simulation results are shown in the bottom right (plot b.) of Figure 5. Now the steady-state error is removed.

The suggested approach was compared to a feed-forward scheme based on an M/M/1 queuing model. Assuming an average arrival rate of λ_a requests per second, and a service rate of λ_s requests per second, the average long-term delay, $\hat{D}$, is given by:

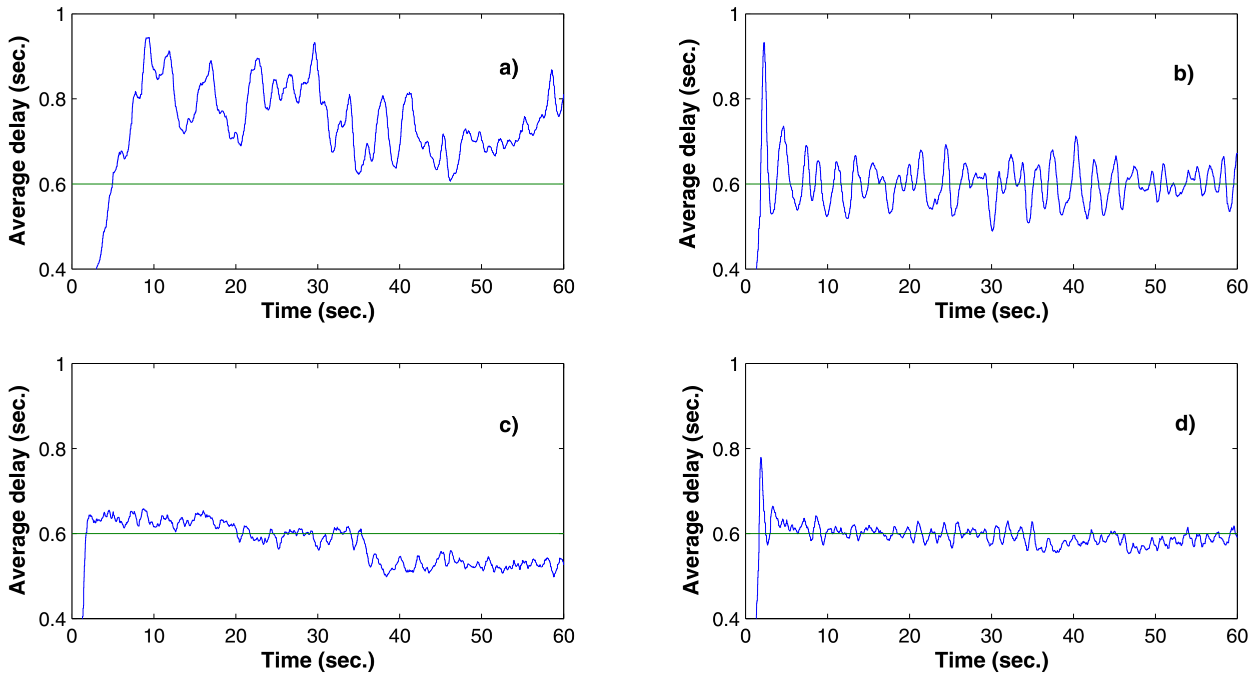

Figure 5. Simulation results for the four compared control strategies. (a) M/M/1 queuing model based feed-forward. (b) M/M/1 queuing model based feed-forward + PI-control. (c) Proposed feed-forward predictor. (d) Proposed feed-forward predictor + PI-control.

$$\hat{D} = \frac{\lambda_a}{\lambda_s(\lambda_s - \lambda_a)} \tag{7}$$

where the service rate in our case is given by $\lambda_s = \mu/\hat{C}$, where $\hat{C}$ is the estimated nominal processing time of the requests and μ again is the processing speed. Given a desired average delay, D_{des}, and the estimated arrival rate, $\hat{\lambda}_a$, Equation (7) can be used to compute the corresponding service rate giving the equation:

$$\lambda_s = \frac{\lambda_a}{2} + \sqrt{\frac{\lambda_a^2}{4} + \frac{\lambda_a}{D_{des}}} \tag{8}$$

Simulation results of this queuing-based feed-forward action are shown in the top left (plot a.) of Figure 5. The main reason that this approach performs worse than our suggested scheme, is that the M/M/1 model assumes Poisson distributed arrivals, which is not an accurate model of Internet traffic. We also get transient errors due to the fact that the queuing prediction is true only for long term averages. Finally, the steady-state variance is considerably higher than for the proposed predictor.

The addition of the PI-controller removes the steady-state error and reduces the steady-state variance. This is shown in the top right (plot b.) of Figure 5. However, the performance is still not comparable with the performance

	Loss $(\cdot 10^{-2})$
M/M/1 model	20.9
M/M/1 + PI-ctrl	4.08
delay prediction model	3.92
delay prediction model + PI-ctrl	2.79

Table 1. Summary of the performance loss for the four different control strategies.

obtained using the proposed predictor in combination with PI-control.

To quantify the simulation results, the *performance loss*

$$J = \int_0^{T_{sim}} (D_r - \hat{D})^2 \tag{9}$$

was recorded in each simulation case. The results are summarized in Table 1.

It is seen that the prediction scheme described in this paper significantly improves the performance loss metric. It can be specifically noted that the feed-forward predictor alone obtains comparable performance to the M/M/1 model in combination with PI-control.

Next, we implemented the proposed scheme in a real Apache server and showed that a performance improvement is seen on the real platform as well.

4. Experimental Evaluation

This section presents the implementation and evaluation of the proposed algorithm in an Apache web server. To regulate the processing speed of a web server, a resource control mechanism must be implemented. In our prorotype, we chose the number of spawned server processes as the variable to be controlled. Due to the UNIX time-sharing policy, the number of processes allocated to serve a particular class affects the fraction of the CPU allocated to that class. The proposed feedforward predictor was integrated with an instrumented version of Apache 1.3.9[17] where we implemented feedback control of absolute delay. The control system incudes a monitor, predictor, controller, and connection scheduler. Let the total capacity, c, of the server be defined as the number of server processes created when the system boots up. Let $b_i(k)$ represent the number of server processes reserved for $class_i$ ($\forall i \in 1, \cdots, N$) at the kth sampling instant, where $b_i(k) \leq c$. The predictor together with the controller decide the process quota $b_i(k)$ for each class. The connection scheduler enforces the quota allocation. The monitor measures achieved delay.

All experiments were conducted on a testbed of PC's connected with 100Mbps Ethernet. Four machines with a 450MHz AMD K6 processor and 256MB RAM were used. One of them was assigned to run the web server with HTTP 1.1, and the rest to run clients that stress the server with a synthetic workload.

We used an enhanced version of Surge (Scalable URL Reference Generator) [7] to generate synthetic web workloads in our experiments. It generated URL requests with a rate independent of the server load while keeping the self-similarity characteristics in the resulting traffic. Experiments with a single guaranteed-delay class of clients were conducted. The goal of the system was to provide the delay guarantee for that class with as few resources as possible. The motivation for such minimization was to leave as much resources as possible for background best effort traffic. The input load patterns generated by the clients are shown in Figure 6. The desired delay for the class was set to $D_{ref} = 4sec$ in all experiments.

For the apache web server, we configured the maximum number of server processes to be $c = 128$. To reduce the control overhead, instead of executing the control mechanism on every dispatched connection, we set the sampling period to 500 dispatched connections. System profiling was carried out beforehand to get an approximate value for the service rate. Using the estimated single process service rate, the measured number of requests in the waiting queue that will be dispatched at the next sampling time, and the average of their arrival times, the predictor would produce the desired process quota by solving Equation (3). We further added a low-pass filter to make the predictor output smoother:

	α	β
G/G/1 model	0.0	-
G/G/1 + PI-ctrl	0.0	0.5
delay prediction model	0.5	-
delay prediction model + PI-ctrl	0.5	0.5

Table 2. The low-pass filters configuration for the four different control strategies.

$$\hat{\mu}_k = \alpha \cdot \hat{\mu}_{k-1} + (1 - \alpha) \cdot \mu_k \qquad (10)$$

where μ_k is the service rate calculated from Equation (3) at the kth sampling interval. When the feedback controller is activated, the monitor measures and calculates the average delay as follows and reports it to the controller:

$$AvgD_k = \beta \cdot AvgD_{k-1} + (1 - \beta) \cdot D_k \qquad (11)$$

where D_k is the measured average delay at the kth sampling interval. For comparison, we carried out experiments for the following four scenarios. First, only the proposed predictor is activated for the Apache web server. Second, the proposed predictor is integrated with a PI controller. Third, only a G/G/1 queuing predictor is activated. Last, the G/G/1 queuing predictor is integrated with a PI controller. The low-pass filters configuration for the four strategies are shown in Table 2.

We choose $\alpha = 0.5$ for the proposed predictor and $\alpha = 0$ for the G/G/1 queuing predictor. This is because the proposed predictor is based on instantaneous measurments which introduces some noise into the predictor outputs. The configured low-pass filter is used to reduce such noise. While the G/G/1 queuing predictor is based on queuing theory, which essentially is applied to long-term averages, therefore, the low-pass filter is deactivated ($\alpha = 0$) in this case.

In Figure 7, the experimental results for the proposed predictor are presented. It demonstrates that the new proposed predictor incurs only a small steady-state error. When integrated with the linear PI controller, the combined system converges to the reference.

Using the aggregate of the squared error between the desired and actual connection delay over the duration of the experiment, we compare the performance of different schemes. The smaller the aggregate error, the better the convergence. Table 3 summaries the results, which further show that the proposed predictor beats the G/G/1 queuing predictor and achieves better performance.

We conclude that the new predictor described in this paper, when used in conjunction with a feedback controller, offers superior performance not only in simulation but also in a practical system. The overhead of the scheme is minimal as it is incured only once every several hundred

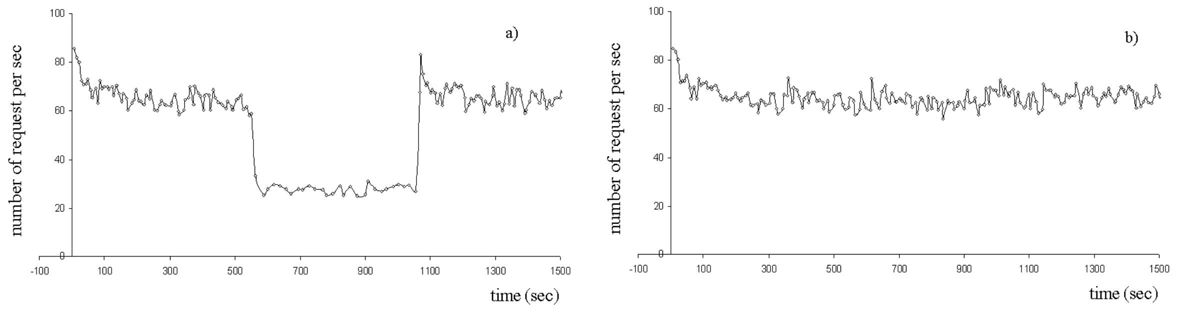

Figure 6. Arrival rate during the experiments. a) Input load1 b) Input load2

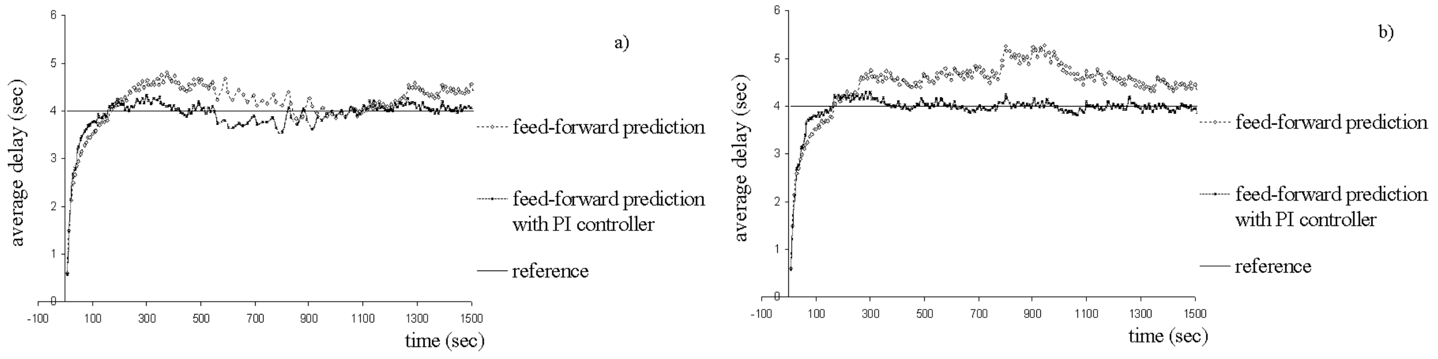

Figure 7. Average delays corresponding to input loads shown in Figure 6. The average is obtained over an observation window of 5000 requests.

	Input Load1	Input Load2
G/G/1 model	60.40	241.05
G/G/1 + PI-ctrl	4.76	3.59
delay prediction model	22.10	71.00
delay prediction model + PI-ctrl	3.20	1.59

Table 3. The aggregate error for the four different control strategies.

packet invocations. This scheme therefore provides the best absolute delay control in contemporary high-performance servers known to date.

5. Related Work

The complexity and increased growth of the Internet has made software applications, such as web servers and routers more and more dependent on feedback control for performance control and uncertainty management [4]. Several recent papers [1, 2, 3] presented a control theoretical approach to web server resource management based on web content adaptation. QoS guarantees on request rate and delivered bandwidth were achieved. In [17, 24, 25], control theory was used for CPU scheduling to achieve QoS guarantees on service delay. A similar approach was used for e-mail server queue management [19]. In [21], admission control is developed based on nonlinear control theory. At the network layer, control theory was applied to packet flow control in Internet routers [13, 16]. The case was recently made for energy saving in large complex systems, such as high-performance contemporary web servers [8]. An architecture for power-aware QoS management in web servers was presented in [23].

Other research efforts focused on QoS control in web servers include [3, 5, 6, 9, 11, 12, 20, 26]. By CPU scheduling and accept queue scheduling respectively, Almeida et al. [5] and Abdelzaher et al. [3] successfully provide differentiated levels of service to web server clients. Demonstrating the need to manage different resources in the system depending on the workload characteristics, Pradhan et al. [20] develop an adaptation technique for controlling multiple resources dynamically. Like [20], Banga et al. [6] and Voigt et al. [26] provide web server QoS support at the OS kernel level. In [9, 12, 11], session-based QoS mechanisms are proposed, which utilize session-based relationship among HTTP requests. Our work is an extension of previous approaches in the area of web server control using both feedback and prediction [18, 22].

6. Conclusions

This paper presented a novel feed-forward predictor for web server delay control. The algorithm exploits an actuator that can change server speed, such as dynamic voltage scaling, or resource allocation mechanisms that assign a configurable fraction of machine capacity to individual tasks. The predictor uses instantaneous measurements of queuing delays and arrival times to provide a non-linear feed-forward control action. The predictor was combined with an event-based feedback controller. The proposed algorithms were evaluated both in a simulation study and by experiments on a real web server. It was shown that the predictor performed satisfactory even without the inclusion of the feedback controller, although feedback improved performance especially on the real Apache platform. It was also shown that the proposed scheme performed much better than previously reported combined prediction and feedback schemes based on queuing theory.

References

[1] T. Abdelzaher. An automated profiling subsystem for qos-aware services. In *IEEE Real-Time Technology and Applications Symposium*, Washington, D.C., June 2000.

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

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[4] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3), June 2003.

[5] J. Almeida, M. Dabu, A. Manikntty, and P. Cao. Providing differentiated levels of service in web content hosting. In *First Workshop on Internet Server Performance*, Madison, Wisconsin, June 1998.

[6] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Operating Systems Design and Implementation*, pages 45–58, 1999.

[7] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, pages 151–160, Madison, WI, 1998.

[8] P. Bohrer, E. Elnozahy, M. Kistler, C. Lefurgy, C. McDowell, and R. Mony. *The Case for Power Management in Web Servers*. Power Aware Computing, Kluwer Academic Publications, 2002.

[9] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *IEEE Infocom*, NEW YORK, NY, June 2002.

[10] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, June 2003.

[11] H. Chen and P. Mohapatra. Session-based overload control in qos-aware web servers. In *IEEE Infocom*, NEW YORK, NY, June 2002.

[12] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving the performance of an overloaded web server, 1998.

[13] N. Christin, J. Liebeherr, and T. F. Abdelzaher. A quantitative assured forwarding service. In *IEEE Infocom*, NEW YORK, NY, June 2002.

[14] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. *ACM/IEEE Transaction on Networking*, 1(1), 2003.

[15] D. Henriksson, A. Cervin, and K.-E. Årzn. TrueTime: Simulation of control loops under shared computer resources. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[16] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of the conference on Communications architecture & protocols*, pages 3–15, 1993.

[17] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *IEEE Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.

[18] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, May 2003.

[19] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. In *IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA, May 2001.

[20] P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy. An observation-based approach towards self-managing web servers. In *International Workshop on Quality of Service*, Miami, FL, May 2002.

[21] A. Robertsson, B. Wittenmark, and M. Kihl. Analysis and design of admission control in web-server systems. In *Proceedings of ACC'03*, 2003.

[22] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queuing model based network server performance control. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, Austin, TX, December 2002.

[23] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron. Power-aware QoS management in web servers. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[24] J. A. Stankovic, T. H. T. F. Abdelzaher, M. Marley, G. Tao, S. H. Son, and C. Lu. Feedback control scheduling in distributed systems. In *IEEE Real-Time Systems Symposium*, London, UK, December 2001.

[25] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Operating Systems Design and Implementation*, pages 145–158, 1999.

[26] T. Voigt, R. Tewari, D. Freimuth, and A. Mehra. Kernel mechanisms for service differentiation in overloaded web servers, 2001.

[27] K. J. strm and B. Wittenmark. *Adaptive Control*. Addison-Wesley, 2nd edition, 1995.