Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers *

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

Chenyang Lu
Department of Computer Science and Engineering
Washington University in St. Louis
lu@cse.wustl.edu

Lui Sha, Xue Liu
Department of Computer Science
University of Illinois
lrs, xueliu@cs.uiuc.edu

Abstract

The use of feedback control theory for performance guarantees in QoS-aware systems has gained much attention in recent years. In this paper, we investigate merging, within a single framework, the predictive power of queueing theory with the reactive power of feedback control to produce software systems with a superior ability to achieve QoS specifications in highly unpredictable environments. The approach is applied to the problem of achieving relative delay guarantees in high-performance servers. Experimental evaluation of this approach on an Apache web server shows that the combined schemes perform significantly better in terms of keeping the relative delay on target compared to feedback control or queueing prediction alone.

1 Introduction

Feedback control theory has recently gained much popularity as an analytic foundation for providing soft QoS guarantees in computing systems such as Internet servers [3, 18] and Internet routers [7, 12]. An advantage of feedback control is that it causes system performance to exhibit a self-correcting, self-stabilizing behavior. Interpreting QoS specifications as control loop set points, the system converges towards an equilibrium around an operating point defined by the specification, thereby producing the desired QoS. Due to the robustness of common feedback controllers, system convergence is observed even in the presence of modeling inaccuracies, inherent system nonlinearities, and variations of system parameters over time. These desirable properties have brought about much interest in control theory as a means to achieve software QoS guarantees.

A disadvantage of feedback control is that it essentially is a *reactive* approach. The feedback loop operates by responding to measured deviations from the desired performance, i.e., exerting corrective action that attempts to reduce the deviation to zero. In many cases, however, it may be possible to *predict* that system performance is about to degrade before the degradation actually occurs. For example, in web server delay control, where server response time is the controlled parameter, an increase in current input load is likely to cause an increase in the average response time in the near future. Unfortunately, a feedback controller, which measures the current delay, will remain oblivious to the impending overload until it occurs. This delay in response is particularly significant since the server response time is a moving average that is slow in response to changes. Augmenting feedback control with a predictive framework would have averted this problem, thereby preventing the overload from occurring.

Queueing theory provides the predictive framework needed such that expected delays can be inferred directly from input load. In a previous paper [19], the authors described the first attempt to combine queueing prediction and feedback control in software services in the context of providing absolute delay guarantees in web servers. In this paper, we extend the approach to the problem of meeting relative delay guarantees in web servers. Relative guarantees are particularly useful for service differentiation on overloaded systems. They have gained much popularity since the formulation of the proportional differentiated services

* The work reported in this paper was supported in part by NSF grants CCR-0093144, CCR-0098269, CCR-0208769, and MURI grant N00014-01-1-0576

framework [8, 9]. Unlike other differentiated service models, the proportional differentiated service provides both *predictable* and *controllable* relative differentiation. It is *predictable*, as the differentiation is consistent (i.e., higher classes are better, or at least no worse) regardless of the variations of the class loads. It is *controllable*, meaning that the network operators are able to adjust the quality spacing between classes based on their selected criteria.

We compare empirically three different variants of the general formulation of the relative delay guarantee problem as a combination of queueing-theoretic prediction and feedback control. These variants follow the common architecture shown in Figure 1. The queueing predictor is placed on the feedforward path to make estimates of per-class resource allocation that satisfies the delay requirements. Several feedback control loops correct this allocation for their respective classes in response to measured per-class performance deviations that result due to inaccuracy of the predictor. Variants of this architecture, explored in this paper, differ in the relation between the number of feedback loops and the number of client classes, the definition of per-loop set points, the definition of performance error, and the way the feedback controller output is mathematically combined with that of the queueing predictor. We describe how these variants are implemented inside an Apache web server, analyze their advantages and disadvantages, and report on our experiences with tuning their parameters in practice such that best performance is achieved.

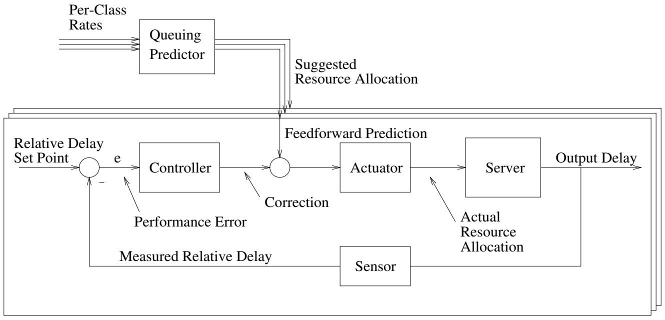


Figure 1. The augmented control loop

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents the design of the control loop and a description of its sensors and actuators. Section 4 describes our experimental framework and evaluation results. The paper concludes with Section 5.

2 Related Work

To the authors' knowledge, this and our previous [19] papers present the first attempt to combine queueing-theoretic prediction with control-theoretic correction in a single framework for performance control in web services. While queueing theory and feedback control are both mature disciplines, the opportunity for their combined use has not presented itself in prior work. The situation changes when control is applied to software performance. Server queues, modeled by difference equations as integrators of request flows, satisfy the assumptions of both the control-theoretic and the queueing-theoretic frameworks.

The application of control theory to software performance control has met much success in recent years. The main motivation has been to develop robust techniques for QoS adaptation that have convergent, self-correcting properties. Several recent papers [1, 2, 4] 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 [14, 21, 22], 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 [18]. In [20], guarantees were made on power dissipation by applying control-theoretical techniques to microprocessor thermal management. In our previous work [15, 16], we applied feedback control theory to proxy cache QoS, as well as the problem of automating controller tuning. At the network layer, control theory was applied to packet flow control in Internet routers [7, 12]. Due to the usefulness of the control-theoretic approach and its versatile applications, middleware frameworks emerged for control-based QoS assurances [24]. The authors of [10, 13, 24] provided tools to help apply control-theoretic design techniques to a larger class of systems. We demonstrate that adding queueing-theoretic prediction capabilities to control loops can enhance feedback loop performance in a non-trivial way. We believe this technique will be of great value whenever time-related QoS metrics are being controlled.

3 Design

We consider the problem of providing relative delay guarantees in high-performance servers.[1] We define a high-performance server as any server which handles a very large number of requests per second, such that client response time is dominated by the server's queueing delay (e.g., in the server's external socket queue, or in the CPU ready queue). We assume that server traffic is classified into a finite fixed number (N) of classes. At each sampling time, k, the average delay, $W_i(k)$, is computed for each traffic $class_i$. The relative delay guarantee ensures the quality spacing between classes by imposing constraints of the following form on successive pairs of classes

$$\frac{W_{i+1}(k)}{W_i(k)} = \frac{c_{i+1}}{c_i}, i = 1, \cdots, N-1 \quad (1)$$

[1] Absolute delay guarantees can be provided similarly.

where c_i is a constant weighting factor of $class_i$, representing the user's relative delay specifications. To evaluate the system performance on providing the relative delay guarantee, we measure the deviation $E_i(k)$ of the actual delay ratio from the desired ratio, i.e.

$$E_i(k) = \frac{c_{i+1}}{c_i} - \frac{W_{i+1}(k)}{W_i(k)} \tag{2}$$

We further assume that all classes share a single server resource. Relative delay guarantees are achieved by allocating that resource appropriately among the competing classes. While in general, servers have multiple resources (such as CPU cycles, network bandwidth, and disk bandwidth), it is often the case that only one resource is the bottleneck. This observation was experimentally verified for the server used in this study. Thus, the single-resource approximation holds well in practice.

An initial solution to the above relative delay guarantee problem has been presented by the authors in [14]. It has been shown that feedback control is capable of keeping the different delays in proportion to their relative weight factors. However, abrupt changes in the traffic pattern result in large deviations before corrective action is taken. This fundamental limitation of a pure feedback-control solution motivates us to design a queueing predictor that is sensitive to traffic changes and re-formulate the relative delay guarantee problem as a combination of prediction and feedback. Three different formulations of the combined problem are presented and evaluated in this paper illustrating significantly improved performance. The queueing predictor, the common component in the three formulations, is described in Section 3.1. Its integration techniques with feedback control are presented in Section 3.2. Analysis of how the queueing predictor improves the controller performance in the integrated architectures is given in Section 3.3.

3.1 The Queueing Predictor

Consider an arbitrary high-performance server serving N client classes. Assume the request streams of each class are Poisson distributed, with an arrival rate of λ_i. In our server model, each traffic class is served by a separate M/M/1 queue with an average service rate of μ_i. Although Poisson distribution is not an accurate model for Internet traffic, we use M/M/1 queue because first, its solutions are simple and easy to manipulate and second, the model errors can be corrected by feedback control. From queueing theory, we know that for an M/M/1 queue with arrival rate λ_i and service rate μ_i, the long term average queueing time (i.e. connection delay) for the clients is

$$W_i = \frac{\lambda_i}{\mu_i(\mu_i - \lambda_i)}, \; where \; \lambda_i < \mu_i \tag{3}$$

Hence, we can estimate the queueing delay ratio of two classes at k^{th} sampling time as

$$\frac{W_i(k)}{W_m(k)} = \frac{\lambda_i(k)\mu_m(k)(\mu_m(k) - \lambda_m(k))}{\lambda_m(k)\mu_i(k)(\mu_i(k) - \lambda_i(k))}. \tag{4}$$

Since we assume that in our server the queueing delay dominates the clients' waiting time, to provide the relative delay guarantee (Equation 1), we control service rates $\mu_i(k)$ and $\mu_m(k)$ in a way that makes the estimated queueing delay ratio approach the desired ratio, that is

$$\frac{\lambda_i(k)\mu_m(k)(\mu_m(k) - \lambda_m(k))}{\lambda_m(k)\mu_i(k)(\mu_i(k) - \lambda_i(k))} = \frac{c_i}{c_m}. \tag{5}$$

We also assume that the service rate μ_i of each queue is isolated and proportional to the amount of its allocated resources. By dynamically adjusting the resource allocation, we have the control on service rates. There are several techniques for allocating system resources among competing client classes. In servers with a process-per-class or a thread-per-class architecture, it is possible to control per-class resource allocation using mechanisms such as capacity reserves [17], CPU reservations [11] or resource containers [5]. These mechanisms allow a certain percentage of CPU time to be allocated to (the processes or threads) serving requests of a particular class. The average service rate μ_i for that class is proportional to the resources reserved.

Let c be the total capacity of the server. Let a resource reserve of size $b_i \leq c$ be allocated to $class_i$. The corresponding service rate μ_i will be $b_i\mu$, where μ is the service rate per resource unit. Then Equation (3) becomes

$$W_i = \frac{\lambda_i}{b_i\mu(b_i\mu - \lambda_i)} \tag{6}$$

To meet the relative delay guarantee specified by Equation (1), the following equations should be satisfied

$$\left.\begin{array}{rcl} \frac{\lambda_2(k)b_1(k)(b_1(k)\mu - \lambda_1(k))}{\lambda_1(k)b_2(k)(b_2(k)\mu - \lambda_2(k))} & = & \frac{c_2}{c_1} \\ \frac{\lambda_3(k)b_2(k)(b_2(k)\mu - \lambda_2(k))}{\lambda_2(k)b_3(k)(b_3(k)\mu - \lambda_3(k))} & = & \frac{c_3}{c_2} \\ & & \cdots \\ \frac{\lambda_N(k)b_{N-1}(k)(b_{N-1}(k)\mu - \lambda_{N-1}(k))}{\lambda_{N-1}(k)b_N(k)(b_N(k)\mu - \lambda_N(k))} & = & \frac{c_N}{c_{N-1}} \end{array}\right\} \tag{7}$$

and

$$b_1(k) + b_2(k) + ... + b_N(k) = c \tag{8}$$

There are N equations in the N unknowns, b_i, $i = 1, 2, \cdots, N$. By solving these equations, we can get the desired values for b_i, which describe the resource allocation suggested by the predictor. These values constitute the queueing predictor output (Figure 1), which can also be translated into a pair-wise ratio, $\frac{b_i}{b_{i+1}}$. In the combined queueing plus feedback framework, the output of the queueing predictor is not directly applied. Instead it is mathematically combined with the output of the feedback controller whose purpose is to adjust for errors in resource allocation, as described next.

3.2 The Feedback Controller

We will present three ways relative differentiated service guarantees can be translated into a feedback control framework, which is then integrated with the queueing predictor described above. All three approaches share the architecture shown in Figure 1. The approaches differ in how many control loops are used, how control set points are computed, and how the error is formulated to drive the correction process. Below, we describe each of the three approaches and discuss their relative merits and drawbacks. Comparisons of their performance on an Apache web server are shown in the evaluation section.

3.2.1 Approach 1: Per Class Control with Fixed Set Point

In this variant, a single control loop exists per traffic class. To satisfy the relative delay guarantee expressed by Equation (1), at every sampling time k, it is enough that the relative delay of each $class_i$, defined as $\frac{W_i(k)}{\sum_{j=1}^N W_j(k)}$, be equal to the relative delay ratio computed from user specifications, i.e., $\frac{c_i}{\sum_{j=1}^N c_j}$. Hence, the performance error $e_i(k)$ of $class_i$ at sampling time k is given by the expression

$$e_i(k) = \frac{c_i}{\sum_{j=1}^N c_j} - \frac{W_i(k)}{\sum_{j=1}^N W_j(k)} \tag{9}$$

where $\frac{c_i}{\sum_{j=1}^N c_j}$ is the fixed set point of $class_i$ and $\frac{W_i(k)}{\sum_{j=1}^N W_j(k)}$ is the value returned by the relative performance sensor. It is easy to see that if the performance error is reduced to zero for each class, the relative guarantee of Equation (1) holds true for the N-class system. The feedback controller in each loop is a linear function $f(e_i(k))$. It translates the performance error $e_i(k)$ into a correction Δb_i to the resource allocation b_i computed by the queueing predictor for $class_i$. The controller function satisfies $f(0) = 0$ (i.e., no correction if there is no error). Thus, if the predictor is accurate, the performance error and the correction would be equal to zero. Since the predictor is only approximate, the actual (i.e., adjusted) resource allocation b_i^{adj} enforced by the actuator for $class_i$ is computed from

$$b_i^{adj} = b_i + \Delta b_i \tag{10}$$

It is important that the adjusted resource allocation obeys the constraint of server capacity expressed by Equation (8), i.e. $\sum_{i=1}^N b_i^{adj} = c$. We call it, the *constant sum* constraint. This property holds because

$$\sum_{i=1}^N e_i(k) = \frac{\sum_{i=1}^N c_i}{\sum_{j=1}^N c_j} - \frac{\sum_{i=1}^N W_i(k)}{\sum_{j=1}^N W_j(k)} = 1 - 1 = 0 \tag{11}$$

Hence

$$\sum_{i=1}^N \Delta b_i = \sum_{i=1}^N f(e_i(k)) = f(\sum_{i=1}^N e_i(k)) = f(0) = 0 \tag{12}$$

Consequently

$$\sum_{i=1}^N b_i^{adj} = \sum_{i=1}^N b_i + 0 = c \tag{13}$$

In other words, the condition on total server capacity, expressed by Equation (8), is satisfied for the adjusted resource allocation.

While the above formulation of the relative delay guarantee problem has the advantages of having a fixed set point and an adjusted resource allocation which satisfies the *constant sum* constraint, on the disadvantage side, it has three undesirable features. First, there is tight coupling among the control loops. This is because the plant output, $\frac{W_i(k)}{\sum_{j=1}^N W_j(k)}$, involves division of the class's delay by the sum of all classes' delays, which are outputs of other control loops. Modeling a system in such a way causes a single input single output controller not suitable for the performance control, unless the denominator, in this case, the sum of all classes' delays remains relatively constant. Second, the control framework gives rise to an asymmetric formulation of the original relative differentiation problem. It is true when the performance error (Equation 9) $e_i(k) = 0$ (for all i), the QoS specification is achieved, as $E_i(k) = 0$ in Equation (2). However, $e_i(k)$ isn't always strictly proportional to $E_i(k)$. Suppose there are two classes in the system, $c_1 = 1$, $c_2 = 3$; at kth sampling time, we have $W_1(k) = 1$, $W_2(k) = 1$; while at jth sampling time, we have $W_1(j) = 2$, $W_2(j) = 10$. Although in both kth and jth sampling time, $|E_1(k)| = |E_1(j)| = 2$, the performance errors fomulated by this approach do not reflect this, as $|e_1(k)| \neq |e_1(j)|$ with $e_1(k) = -\frac{1}{4}$ and $e_1(j) = \frac{1}{12}$. Third, the *constant sum* constraint isn't achieved free of charge. It requires that for all control loops, the designed control functions are the same, e.g. f. This restricts controller design. The relationship between resource allocation and delay may vary from one class to another (because, for example, the class traffic patterns are different) making it advantageous to tune the different per-class controllers differently. This, however, is impossible without violating the *constant sum* constraint.

3.2.2 Approach 2: Per Class Control with Variable Set Point

We can formulate the relative delay problem differently by noticing that when the relative guarantee (Equation 1) is satisfied, the ratios $\frac{W_1(k)}{c_1}$, $\frac{W_2(k)}{c_2}$, ..., $\frac{W_N(k)}{c_N}$ are equal. Moreover, they are also equal to their average. This average is therefore a good candidate for a common control set point. The error of each loop i is the deviation of the i^{th} ratio from the common set point. When all deviations are zero, the relative guarantee is satisfied.

In this architecture, we use a single control loop per class. The sensor in each loop i multiplies the output delay $W_i(k)$ of the class by the constant $\frac{1}{c_i}$. The set point is the same for all classes and is equal to the average of all sensor measurements, i.e., $avg(k) = \frac{1}{N}\sum_{i=1}^{N}\frac{W_i(k)}{c_i}$. The performance error of $class_i$ is

$$e_i(k) = avg(k) - \frac{1}{c_i}W_i(k) \tag{14}$$

Observe that the control loops in this formulation are linear, since the sensor measurement $\frac{W_i(k)}{c_i}$ is the delay of the class scaled by a constant. The controller is a linear function that computes $\Delta b_i = f(e_i(k))$, such that actuator input is $b_i^{adj} = b_i + \Delta b_i$, where b_i is the output of the queueing predictor. The *constant sum* constraint is satisfied because

$$\sum_{i=1}^{N} e_i(k) = \sum_{i=1}^{N}\frac{W_i(k)}{c_i} - \sum_{i=1}^{N}\frac{W_i(k)}{c_i} = 0 \tag{15}$$

which leads to

$$\sum_{i=1}^{N}\Delta b_i = \sum_{i=1}^{N} f(e_i(k)) = f(\sum_{i=1}^{N} e_i(k)) = f(0) = 0 \tag{16}$$

and consequently, $\sum_{i=1}^{N} b_i^{adj} = c$. Although the approach removes a nonlinearity from the system, it is an indirect method in that each control loop controls the *absolute* delay of its own class. The relative delay guarantee is achieved indirectly as a consequence of changing the set point which depends on all control loop outputs from the previous iteration. Similar to the first approach, this is an asymmetric formulation of the relative differentiation problem (Equation 1)and it requires the same control function design for all the loops to meet the *constant sum* constraint.

3.2.3 Approach 3: Class Ratio Control

Another disadvantage of both of the previous approaches is that each individual feedback loop controls the resource allocation of a *single* class. However, the relative delay guarantee is not well correlated with resource allocation to any individual class, but with the *ratio* of such allocations to different classes. Hence, controller derivation becomes a problem, as the actuator's enforced resource allocation, b_i^{adj}, is not well correlated with the measured performance error (Equation 2), $E_i(k)$, in any single loop.

To avoid this problem, we change the architecture such that one feedback control loop is used for every *pair* of adjacent classes. At every sampling time k, the loop measures the deviation in the delay ratio $\frac{W_{i+1}(k)}{W_i(k)}$ of the two classes from its ideal target $\frac{c_{i+1}}{c_i}$. The performance error is therefore defined as $e_i(k) = \frac{c_{i+1}}{c_i} - \frac{W_{i+1}(k)}{W_i(k)}$. If the queueing predictor is ideal, the ratio $\frac{b_i}{b_{i+1}}$ of resources allocated to the two classes will be such that no error is observed. In general, the queueing predictor is only approximate, causing a finite error to develop. The loop controller is a linear function, $f(e_i(k))$, which determines the *allocation ratio* adjustment $\Delta(\frac{b_i(k)}{b_{i+1}(k)})$ needed to correct the resource allocation ratio.

To combine the queueing predictor and the controller, we simply add their outputs. The input to the actuator which enforces resource allocation is therefore

$$\frac{b_i^{adj}(k)}{b_{i+1}^{abj}(k)} = \frac{b_i(k)}{b_{i+1}(k)} + \Delta(\frac{b_i(k)}{b_{i+1}(k)}) \tag{17}$$

The actuator solves these equations, one for each of the $N-1$ control loops in the system, together with the total capacity condition

$$b_1^{adj}(k) + b_2^{adj}(k) + ... + b_N^{adj}(k) = c \tag{18}$$

for the individual values of resource allocation $b_i^{adj}(k)$. Since this is a system of N equations in N unknowns, a solution is always possible. The solution simultaneously satisfies the *constant sum* constraint and the adjusted ratio $\frac{b_i^{adj}(k)}{b_{i+1}^{adj}(k)}$ of each control loop. As we show in the evaluation section, this formulation yields the best performance.

Note that the actuator in this case creates a potential coupling between individual loops in that the actual resource shares of individual classes, $b_i^{adj}(k)$, depend on the allocation of other control loops, as determined by the solution of Equations (17) and (18). This, however, is not a concern since in the controlled system it is the ratio $\frac{b_i^{adj}(k)}{b_{i+1}^{adj}(k)}$ that is most tightly correlated to the delay ratio $\frac{W_{i+1}(k)}{W_i(k)}$. The actuator does not change the former ratio. Thus, the coupling does not affect the desired guarantee. Unlike the previous two approaches, this formulation does not have the asymmetry problem and the controllers for each loop could be different linear functions.

3.3 Combining Prediction and Feedback Control

In this section, we analyze some implications of the integration of the queueing predictor from Section 3.1 with the feedback control loops from Section 3.2. As we know, the average delay of queueing systems (such as web servers) is essentially nonlinear in the resource allocation which determines average service rate. We call the curve that relates the resource allocation (or service rate) of a class to the delay of that particular class the *input-output* curve. The slope of this curve is very important since it describes how changes in resource allocation affect the average delay of the class. This slope is used in tuning the feedback controller. For linear systems, linear control theory can determine controller parameters given the constant slope of the input-output curve. From queuing theory, however, we know that the input-output curve for an M/M/1 system is given by $Delay = \frac{1}{(\mu - \lambda)}$. Hence, its slope ($\frac{d\ Delay}{d\ \mu}$) is not constant, which means the controller must be tuned based on the "worst case" slope. It is useful to think of system delay as defining an *operating point* on the curve. As current delay changes, so do the operating point and the slope. The worst case slope is the maximum slope of the input-output curve over all operating points that might be visited by the system during normal operation. A controller tuned for the worst case will ensure eventual system convergence to specifications (known as system *stability*), as long as the system operates at a point where the slope of the input-output curve does not exceed the stipulated worst case. This convergence, however, will generally be too slow when the slope is significantly lower than the worst-case. In this case, the controller design is said to be pessimistic.

In our previous paper [19], we demonstrated that a queueing predictor helps alleviate this problem. The predictor determines a resource allocation that attempts to keep system delay around a target set point. Hence, the system operates in a small region of the input-output curve and can be linearized around that operating point. The slope of the input-output curve in that small region does not change significantly, leading to less pessimistic controller design.

An interesting observation we point out in this paper is that unlike the case with absolute delay control, in relative delay control the above argument does not hold. Relative delay control must guarantee that the average delays seen by different classes satisfy a specified proportion. For example, when the control loops are set up as described in Section 3.2.3, the "output" in the input-output curve is the fraction $\frac{W_{i+1}}{W_i}$. The input is the fraction $\frac{b_i}{b_{i+1}}$. In section 3.1, we derived the input-output formula (Equation 4) for the corresponding queueing delay ratio using queueing theory

$$\frac{W_i}{W_m} = \frac{\lambda_i b_m (b_m \mu - \lambda_m)}{\lambda_m b_i (b_i \mu - \lambda_i)} \tag{19}$$

Figure 2, plots this formula, demonstrating how the delay ratio $\frac{W_{i+1}}{W_i}$ (the metric that the system intends to control) is related to the resource allocation ratio $\frac{b_i}{b_{i+1}}$ (the control signal generated in the scheme) for three different request arrival rates. Observe that, unlike with absolute delay control, in this case for the *same* output delay ratio, the input-output curve does not have a unique slope. Instead, its slope depends on the request arrival rate. Thus, keeping the relative delay around the set point (say, the rectangular area in Figure 2) does not yield a uniquely linearizable system. The controller has to be designed for the worst-case slope which depends on the expected range of request arrival rates. The same observation is found true of the other two relative delay control schemes discussed in this paper.

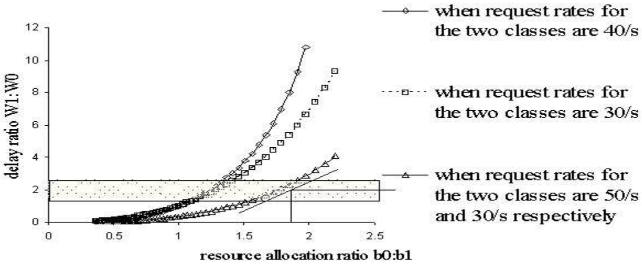


Figure 2. Non-linearity of the system

Despite our inability to uniquely linearize the system around the operating point even in the presence of the queueing predictor, this predictor is still very useful in that is does attempt to keep the delay ratio of all classes around the set point. The more accurate the queueing model used in the prediction, the closer the system is to the set point, and the lower is the burden on the feedback controller (i.e., the less relevant is the effect of pessimism in controlled design). In Section 4.3 we prove this predictive mechanism significantly improves the control system performance when large workload changes occur.

4 Experimentation

In the previous section, the combined queueing plus feedback frameworks are presented. We now describe how we implemented and evaluated them in an Apache web server to provide relative delay guarantees.

4.1 Implementation

We considered server processes as the resource to be controlled and implemented a process allocation mechanism to achieve relative connection delay differentiation. We used an instrumented version of the Apache 1.3.9 [23] server first described in [14]. In this version a new library implements a Connection Manager (including a Monitor, a Queueing Predictor, a Controller, and a Connection Scheduler) to adjust the allocation of server processes among classes of connections. The total capacity c of the server is the number of server processes created when the system boots up.

$b_i(k)$ represents the number of server processes reserved for $class_i$ ($\forall i \in 1, \cdots, N$) at k^{th} sampling instant, which sum up to be c. In order to decide the process quota for each class, we implemented the Queueing Predictor and the Controller. According to the resulting process quota, the Connection Scheduler serves as an actuator to allocate certain number of server processes to connections from each class. Monitor is responsible for system measurements.

- *Queueing Predictor*

  System profiling was carried out beforehand to get an approximate value for μ, the service rate per process unit. Using the estimated μ and average request arrival rates λ_i calculated by the Monitor, the Queueing Predictor will produce the new desired values for b_i by solving N equations (7) and (8).

- *Controller*

  For every control loop, an Integral controller is implemented, whose digital form is

  $$U_i(k) = U_i(k-1) + g * e_i(k),\ U_i(0) = 0 \qquad (20)$$

  The interpretation of $U_i(k)$ and $e_i(k)$ depends on the control loop formulation

Approach	$U_i(k)$	$e_i(k)$
1	$\Delta b_i(k)$	$\frac{c_i}{\sum_{j=1}^{N} c_j} - \frac{W_i(k)}{\sum_{j=1}^{N} W_j(k)}$
2	$\Delta b_i(k)$	$\frac{1}{N} \sum_{i=1}^{N} \frac{W_i(k)}{c_i} - \frac{1}{c_i} W_i(k)$
3	$\Delta \frac{b_i(k)}{b_{i+1}(k)}$	$\frac{c_{i+1}}{c_i} - \frac{W_{i+1}(k)}{W_i(k)}$

  and their measurements are carried out by the Monitor; g is the design parameter called the controller gain, which we tune based on the worst-case slope of the input-output curve.

- *Connection Scheduler*

  The Connection Scheduler serves as an actuator to control the relative delays of different classes. It adds up the outputs of the Queueing Predictor and the Controller and computes the individual values of process allocation $b_i^{adj}(k)$ for each class of clients. The Connection Scheduler listens to the well-known port and accepts every incoming TCP connection request. For each $class_i$, the Connection Scheduler maintains a (FIFO) connection queue Q_i and a process counter R_i for enforcing the number of processes to allocate.

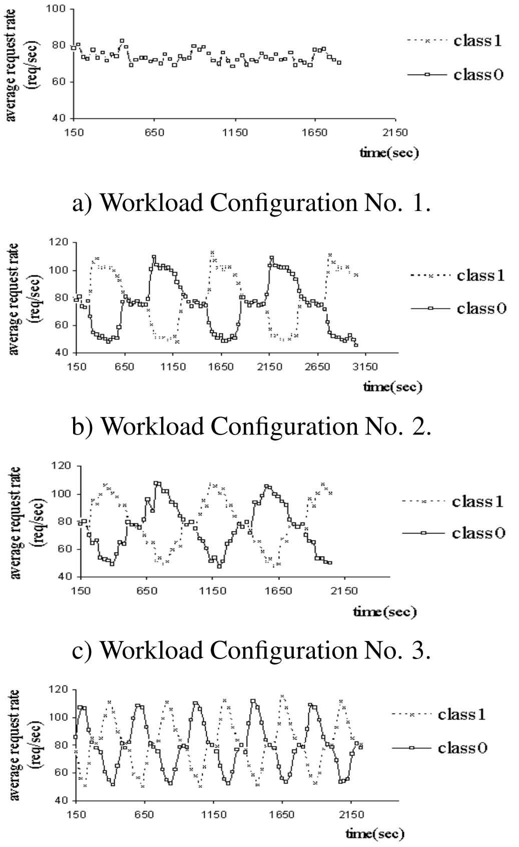
4.2 Experimental Setup

All experiments were conducted on a testbed of PC's connected with 100Mbps Ethernet. Seven 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)[6] 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. Client machines were evenly divided into two or three classes. The system goal was to guarantee that the connection delay of different classes to satisfy relation $\frac{W_q[1]}{W_q[0]} = \frac{3}{1}$ for two classes case or $\frac{W_q[1]}{W_q[0]} = \frac{W_q[2]}{W_q[1]} = \frac{2}{1}$ for three classes case. To test the performance of the system on providing relative delay guarantees, the designed workload configurations differed in the relative workload volume between classes. The justification for such experiment configurations is that the relative change of the class workloads produces the biggest disturbance to the system performance, the relative delay. Four workload configurations were designed for the two classes case. For the first one, the average request arrival rates of the two classes didn't change dramatically and were around 75 requests per second (Figure 3-a). For the remaining configurations, numbered 2, 3, and 4, the user population of each class changes in every 300, 130 and 70 seconds respectively, with increasing frequency. The corresponding request arrival rates are shown in Figure 3-b,c,d. The workload configurations for the three classes case were similarly designed.

4.3 Experimental Results

4.3.1 Two Classes Case

In the first set of experiments, we compared the performance of the three different implementations of the combined framework. We carried out the experiments with the four different workload configurations described. We used the average absolute difference between the measured and desired delay ratio $|E(k)|$ (Equation 2) as a measure of performance. The smaller the average difference, the better the system performance. We calculated the average $|E(k)|$ using the data collected in the middle of the experiments, when the system had been warmed up by the incoming workloads. The results are shown in Figure 4. From this figure, it can be seen that the scheme in Section 3.2.3 gives the best performance of all configurations. This could have been expected because approach 1 (Section 3.2.1) results in a non-linear control loop, and approach 2 (Section 3.2.2) has a variable set point which makes linear control more difficult if the system exhibits any internal non-linearities.



a) Workload Configuration No. 1.

b) Workload Configuration No. 2.

c) Workload Configuration No. 3.

d) Workload Configuration No. 4.

Figure 3. Request rates of the two classes

Hence, we believe that the superior performance of the scheme in Section 3.2.3 is an artifact of using linear controllers in our experimental study. If nonlinear controllers are used, the relative performance of the three schemes remains an open issue. This issue, however, is outside the scope of this paper. The rest of the evaluation section is concerned only with the the best-performing scheme.

In order to evaluate the combined system performance, we compare its performance to that of the controller alone and the queueing predictor alone for the mentioned workload configurations. Experimental results show that the queueing predictor is useful in catching the traffic volume changes and improves the system performance when combined with the controller. As an illustration, Figure 5 shows the results for workload configurations numbered 1, 2, and 3. Note from Figure 5-a that the performance of the combined system does not beat that of the controller alone when the traffic volume is stable. It indicates that feedback control of web server delay performs well when the workload does not change significantly. If that is the case, there is no need for a queueing predictor.

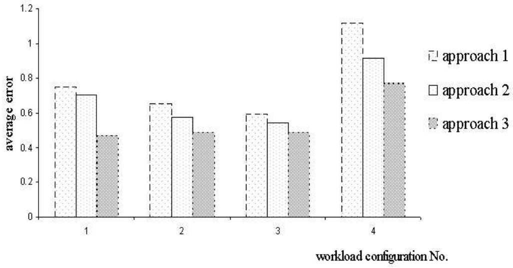


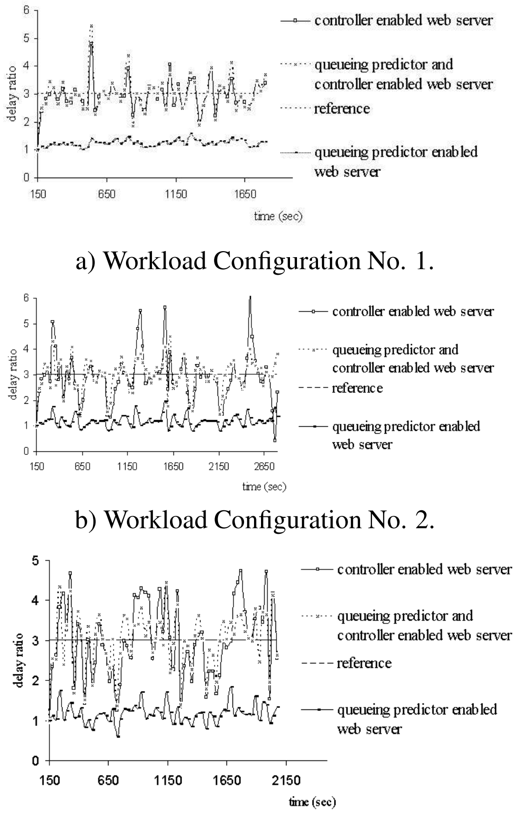
Figure 4. A comparison of different architectures

However, dramatic workload changes are very common in web servers. Performance results in Figure 5-b,c indicate that the controller cannot handle more rapid workload changes well. Without the help of the queueing predictor, big performance deviations are observed from the desired delay ratio when the request rates change. In contrast, with the queueing predictor integrated, the combined system responds to traffic changes more rapidly, thus reducing the performance deviation.

Figure 5 also shows that the predictor alone does not let the system converge to the target delay, which is caused by inaccurate assumptions made by the predictor about the probability distributions of the workload. The feedback controller in the combined architecture demonstrates its usefulness in correcting this deficiency.

The average measured deviations of the compared systems from the desired ratio are summarized in Figure 6, which indicates the controller alone degrades much faster than the combined system when traffic changes increase.

To show how system performance is affected by the choice of sampling interval, we carried out a series of experiments using the workload configuration No. 3. We calculated the average $|E(k)|$ using the data collected during 1360 seconds in the middle of the experiments. As shown in Figure 7, the choice of sampling interval leads to significantly different average deviation from the target. The difference is caused by the fact that we are controlling a probability, instead of an instantly measurable quantity such as temperature or pressure. Since probability itself is not directly measurable, the sensor can only estimate it by averaging over large periods. Therefore, if we choose very short sampling intervals, the system performs poorly. As we increase the sampling interval, system performance improves. However, increasing of the sampling interval beyond some point will make the system less responsive causing performance to degrade again. Hence, there is one optimal choice of sampling interval where the system performs best. The analytic derivation of this optimal is an interesting research problem. Observe that Figure 7 shows that the combined system always performs better than the controller only one,



a) Workload Configuration No. 1.

b) Workload Configuration No. 2.

c) Workload Configuration No. 3.

Figure 5. Performance results with different workload configurations

which further demonstrates that the queueing predictor is helpful in improving the system performance.

4.3.2 Three Classes Case

We also carried out experiments with three classes of clients. The experimental setup was similar to the two classes case and was described in Section 4.2. As an illustration, one of the experiment configurations and resulted average deviations is shown in Figure 8, which indicates that the combined system performs much better when the workload volumes change dramatically for the three classes.

5 Conclusions

In this paper, we experimentally investigated the benefits of combining a queueing-theoretic predictor with a feedback control architecture in the context of achieving relative delay guarantees in high-performance servers. Our experimental prototype featured an Apache web server running the HTTP 1.1 protocol, which are presently the most prevalent web server and protocol on the Internet. Empirical evidence shows that significant performance improvement can be achieved in the combined scheme when workload changes (for instance, traffic spikes) exist and for all sampling interval choices. Multiple variants were investigated for the combined architecture. It was shown that the best is to use a single controller per class-pair as described in Section 3.2.3. We attribute this observation to 1) its symmetric formulation of the relative delay differentiation problem; and 2) the high correlation between the ratio of resource allocation and the delay ratio, compared to the correlation between an individual resource share and the relative delay ratio.

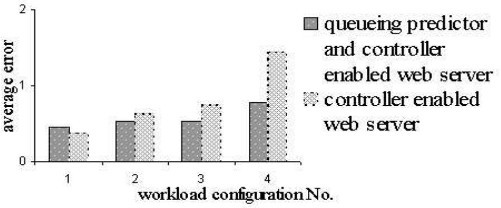


Figure 6. Average $|E(k)|$ with different workload configurations

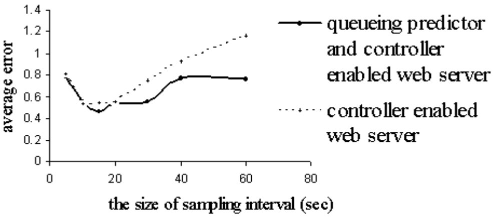
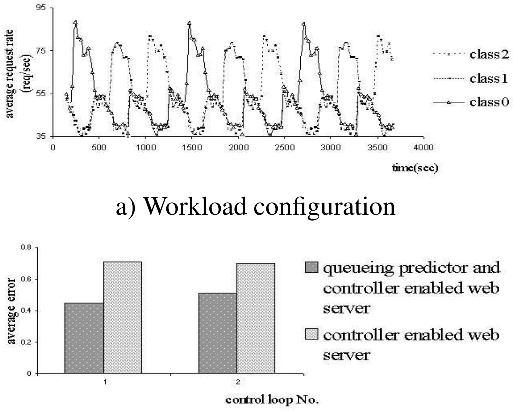


Figure 7. Average $|E(k)|$ with different sampling rates



a) Workload configuration

b) Average $|E_i(k)|$ of the two control loops

Figure 8. Three classes experiment

The paper did not address several important issues. For example, the derivation of the optimal sampling time in a combined queueing and control formulation is of significant interest. Other interesting directions include the extension of this scheme to a multi-resource architecture modeled by a queueing network. Finally, other predictive approaches could be explored. Of particular interest may be to explore the trend prediction techniques used to predict financial markets. This is currently an avenue for the authors' future work.

Acknowledgements

Special thanks goes to Jack Stankovic, Gang Tao, Ronghua Zhang and Nicolas Christin who contributed to an earlier stage of this work.

References

[1] T. F. 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 and C. Lu. Modeling and performance control of internet servers. In *39th IEEE Conference on Decision and Control*, Sydney, Australia, December 2000.

[4] 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*, January 2002.

[5] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Third USENIX Symposium on Operating Systems Design and Implementation*, pages 45–58, New Orleans, Louisiana, February 1999.

[6] 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.

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

[8] C. Dovrolis and P. Ramanathan. Proportional differentiated services, part ii: Loss rate differentiation and packet dropping. In *International Workshop on Quality of Service*, Pittsburgh, PA, June 2000.

[9] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *SIGCOMM*, pages 109–120, 1999.

[10] A. Goel, D. Steere, C. Pu, and J. Walpole. SWiFT: A feedback control and dynamic reconfiguration toolkit. Technical Report CSE-98-009, Oregon Graduate Institute, Portland, OR, June 1998.

[11] M. Jones, D. Rosu, and M.-C. Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

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

[13] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal on Selected Areas in Communications*, September 1999.

[14] 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.

[15] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework and its application to differentiated caching services. In *International Conference on Quality of Service*, Miami Beach, FL, May 2002.

[16] Y. Lu, A. Sexana, and T. Abdelzaher. Differentiated caching services; a control-thoeretical approach. In *Proceedings of the 2001 International Conference on Distributed Computing Systems*, pages 615–622, 2001.

[17] C. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[18] 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.

[19] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queueing model based network server performance control. In *IEEE Real-Time Systems Symposium*, Austin, TX, December 2002.

[20] K. Skadron, T. Abdelzaher, and M. Stan. Control-theoretic techniques and thermal rc modeling for accurate and localized dynamic thermal mangement. In *International Symposium on High Performance Computer Architecture*, Cambridge, MA, February 2002.

[21] J. A. Stankovic, T. He, 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.

[22] 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.

[23] The Apache Software Foundation. http://www.apache.org.

[24] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 2002 International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.