

Queueing Model Based Network Server Performance Control

Lui Sha and Xue Liu, UIUC
lrs, xueliu@cs.uiuc.edu

Ying Lu and Tarek Abdelzaher, UVA
ying, zaher@cs.virginia.edu

Abstract

Controlling the timing performance of a network server is a challenging problem. This paper presents a Queueing Model Based Feedback Control approach to keep the timing performance of a network server close to the service level specification. We show that in an instrumented Apache server, combining feedback control with a queueing model leads to better tracking of QoS specifications than with feedback control alone or queueing model based feed forward control alone.

1 Introduction

Network based server systems, e.g., Web servers, have now become an integral part of our information services infrastructure. Controlling the timing performance of each individual connection to a network server is a challenging theoretical problem with important practical implications. This paper presents a queueing model based feedback control approach to keep the timing performance of a network server close to the service level specification.

A network server's response to allocated resources is highly non-linear. In addition, the workload is stochastic and its parameters could change abruptly over a wide range of values. The differential (or difference) equation model used by control theory does not model Web servers well, except in the limited case of heavy workload that allows for fluid approximations.

In spite of the tradition of using differential equation models for network servers, we believe that the key to success is not to force-fit a queueing system into a differential equation model. Rather, we should model network servers for what they are. Our approach is to develop a model predictive control, where the queueing model's online solution provides feed forward control, which augments the feedback loop. As we shall see later, the feed forward control keeps the system state near an equilibrium operating point, in spite of abrupt changes in the arrival process. This makes the design of a controller much easier. In return, the controller corrects errors due to the inaccuracies in the queueing model for the real networked server system.

To integrate queueing theory with control, we need to develop a general procedure that will allow us, based on experimental data alone; to get an accurate linear model for residue errors from any queueing model based feed forward control. However, some of the best practices for identifying models for physical systems are actually counter-productive. It is

intellectually satisfying to overcome such difficulties and see that two separately developed powerful theories can work synergistically in theory and in actual network servers as evidenced from our experimental evaluation on an Apache web server instrumented to implement our performance control extensions.

In Section 2, we present our key ideas on modeling and control design for a network server node. In Section 3, we provide experimental evidence of the usefulness of this approach. Section 4 reviews related work and Section 5 is the conclusion.

2 A Conceptual Framework

We would like to keep the timing delay experienced by users close to the agreement, e.g., an average delay of 1 second in a window of 1,000 events for a workload that is up to 100,000 requests per second. Delays consistently longer than the specification are unacceptable to the users. Delays consistently shorter indicate over-provisioning of resources that could have been used for other users and applications.

2.1 Feed Forward and Feedback Control

In this section, we describe the fundamental elements of the queueing model based control. Classical results from queueing theory are used for computing the service rate

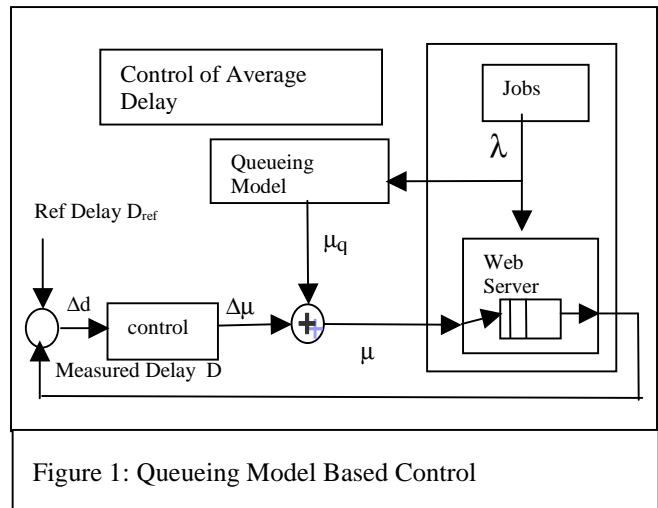


Figure 1: Queueing Model Based Control

necessary to achieve a specified average delay given the currently observed average request arrival rate. Server resources are then allocated to achieve the computed service rate. We call this feed forward loop the *Queueing*

Model Predictor. Finally, a feedback control loop compares the actual delay achieved to the desired average and adjusts the resource allocation accordingly in an incremental manner to ensure that the desired delay is maintained. The feedback and queueing components operate concurrently in a complementary manner as shown in Figure 1.

In the following, we use a running example to explore the research issues that arise in the aforementioned scheme. We begin by illustrating the ideas using a simple M/M/1 queueing model, which will be replaced by a G/G/1 model in Apache server control. Modeling a Web server by a single queue is a commonly used simplification, because the performance of a computer system is often dominated by a bottleneck stage. Let the request process (shown at the top right corner of Figure 1) have arrival rate λ , which may change abruptly. The queueing model estimates the arrival rate λ online. The equilibrium delay of a M/M/1 queue is simply $1/(\mu-\lambda)$. Suppose the agreement with the users specifies that, over a desired window of events, the reference delay is $D_{ref} = 2$. Thus, the corresponding service rate $\mu_q = \lambda + 0.5$, which will give exactly the long-term equilibrium delay of $D_{eq} = D_{ref} = 2$. However, there may still be sizable fluctuations around $D_{ref} = 2$ and we wish to reduce the fluctuations within the user observation window. Thus, the mean sample delay D over a window will be computed and compared with the reference delay. The difference, Δd , is the error to be corrected by the controller via service rate adjustment $\Delta\mu$ as shown in Figure 1.

Next, we show how the presence of a queueing predictor makes a *linear* feedback controller sufficient by compensating for system non-linearity. For feedback control to improve system performance, the controller must be properly tuned. Figure 2 depicts the average queueing delay d of a M/M/1 model as function of $(\mu-\lambda)$. In addition, the reference delay was chosen to be 2 units. Note that the point $D_{ref} = 2$ is the most difficult operating condition to control because it has the highest curvature, which makes it least amenable for linear modeling.

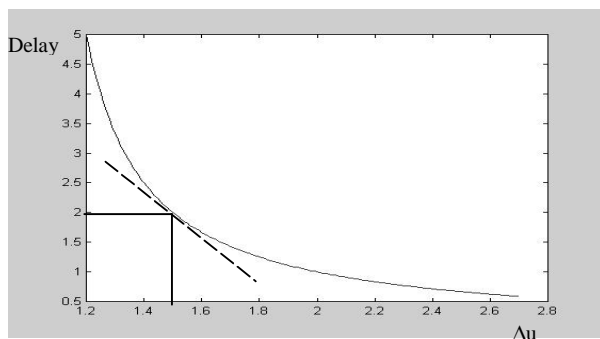


Figure 2: Linearization of $1/(\mu-\lambda)$ at delay = 2

The controller uses the error, Δd , the difference between the measured delay from the desired delay to compute a change in service rate, $\Delta\mu$, that would reduce the delay deviation. Hence, the controller must know the effect of $\Delta\mu$ on Δd , which is

known as the process gain $\Delta d/\Delta\mu$. For a given arrival rate λ , we have $\Delta d/\Delta\mu = \Delta d/\Delta(\mu-\lambda)$. Thus, at the point of linearization, for small deviations, the process gain is approximately the slope of the non-linear rate-delay relation of the queueing system, $d = 1/(\mu-\lambda)$.

As suggested by the slope of Figure 2, changing resource allocation by the same amount will change the average delay by different amounts depending on the current service rate. In control-theoretic terms, a linearized controller for a non-linear system works well, only when the system state is in the neighborhood of linearization. The enforcement of this condition is achieved via the Queueing Model Predictor.

We illustrate by an example how the queueing predictor maintains the system in the neighborhood of linearization. The objective is to adjust $\Delta\mu$ to make the delay close to 2, the reference delay.

Suppose that we linearize the system at $D_{ref} = 2$. Initially $\lambda = 1$. Let $D_{ref} = 2$. The feed forward control model provides $\mu_q = 1.5$. This generates an equilibrium delay $D_{eq} = 2$. The controller's task is to handle the sample mean fluctuations around 2. If the arrival rate suddenly jumps from 1 to 3, the queueing system moves far away from the point of linearization. Fortunately, as soon as the new arrival rate is detected, the feed forward service rate μ_q changes from 1.5 to 3.5 by solving the equation $2 = 1/(\mu-\lambda)$. As a result, the feed forward control restores the equilibrium condition to delay $D_{eq} = 2 = D_{ref}$. Thus, the feed forward control helps the feedback controller by keeping the system in the vicinity of linearization, no matter what the workload is. Indeed, the change of service time parameters can be handled in a similar way.

In the following, we shall refer to the difference between the measured sample delay and the reference delay after the feed forward control as the *residual error*. In queueing model based control, the purpose of the controller design is to suppress residual errors around the target delay. The linear feedback controller not only can help us reduce the fluctuations but also can correct inaccuracies in modeling and in the estimation of arrival rates. Having presented the basic ideas, we now raise the following questions.

- How can we accurately develop a linear model for the residual errors in a stochastic environment?
- How can we design a minimal order controller, and pick the correct rates of observation and control?

2.2 Control Model Development

We now develop a general procedure that will allow us, based on experimental data alone, to get an accurate linear model for the residual errors in any queueing model based feed forward control.

When the output of the system is the average delay, the response time formula, $1/(\mu-\lambda)$, depicts the average delay as a function of the difference between arrival rate and average service rate. From the perspective of control system modeling, this is exactly the transfer function between control action and response that we try to capture. Unfortunately, this function is valid only when the system can be modeled accurately by a M/M/1 model. We need a general procedure that works for any queueing model.

In our running example, as shown in Figure 2, the line tangent at the curve at $D_{ref}=2$ has a slope of -4 . This is because $dD/d\mu = -1/(\mu-\lambda)^2 = -(D_{ref})^2$. However, we want to rediscover it accurately from data of the residual errors alone. The standard approach in model identification is to 1) use a white noise control signal that is sufficiently large but not causing saturation; and 2) to rapidly sample the output. The collected data is then fed to a least squares based modeling tools such as the ARX model in Matlab. The reason for the white noise signal is that it will expose the full spectrum of system responses. The reason for large inputs is to make the responses larger and thus easier to measure. The reason for rapid sampling is not to miss any signal in the response due to the Nyquist rate consideration.

Unfortunately, the ideas of large excitation at the input and fast sampling at the output are exactly the wrong things to do in the development of a control model for the residual errors in a queueing system. The correct approach is to use a small white noise signal, a very large observation window and average all the observation samples in the window. This is counter-intuitive. In common stochastic control problems such as Linear Quadratic Gaussian control problems and Kalman filters, the plant is fundamentally governed by differential equations, albeit random processes disturb both the observations and actuations. However, our plant here is a random process. And we want to control, in probability, the short-term behaviors of a random process by adjusting its probability distribution parameters.

To illustrate the need for a long observation window, suppose that we want to correct a biased coin with probability of head equals to 0.4. We begin by soldering a small weight to the side of head of the biased coin and then do some experiments. If the resulting probability of head is greater than 0.5, we will reduce weight. Otherwise, we increase it. Over a period of time, we may correct the biased coin. The critical question here is how frequently should we adjust the weight? Obviously, we cannot succeed if we change the weight, flip the coin once, and then change the weight again. Once we adjust the weight, we must flip the coin a large number of times until a statistical equilibrium is reached. This allows us to determine the effect of the added weight upon the probability distribution with a high degree of confidence. From a control perspective, the problem here is that we are controlling a *probability*, and not an instantly measurable quantity such as temperature or pressure. Since probability itself is not directly measurable, the sensor

can only estimate it by the mean of large samples. In other words, the transfer function between the change of weight and the change of probability of head manifests itself only when the sample variance becomes negligible. Unlike a

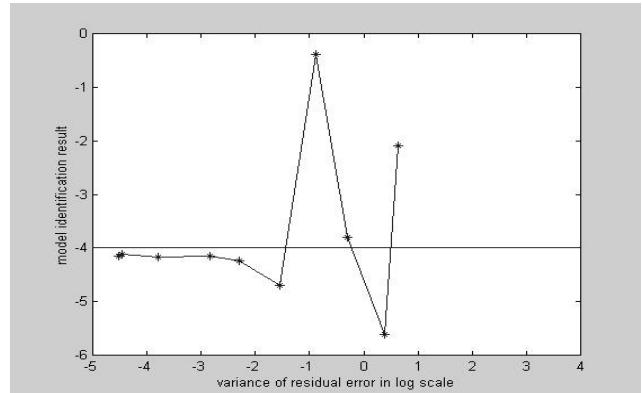


Figure 3: Effect of Sample Variance on Model Accuracy

Bernoulli process where each coin flip is independent, the delays between nearby outputs from a queueing system are correlated. The sample variance drops and the mean of the window of events converge to the process mean as the window size increases and contains more and more epochs. This is the condition that allows the relationship between the control and its impact on delay to reveal itself.

But there is one more important detail: we must keep the size of the excitation signal small. A close examination of Figure 2 tells us that the queueing delay curve is *asymmetric*. The effect of reducing $\Delta\mu$ has a greater impact than increasing it by the same amount, especially when the size of $\Delta\mu$ is large. This asymmetry causes problems in least squares based estimation, because it will pick a line with equal sums of squared errors on both sides of the tangent. The need for small excitation also argues for a large sample size so as not to drown the responses to small excitations in sample variances.

We have implemented a simulator as shown in Figure 1. In the simulation, we randomly change the sign of $\Delta\mu$ of a certain size and collect the sample means for model identification. Figure 3 depicts the slope produced by Matlab's ARX model under a small excitation of $\Delta\mu = \pm 0.01$. The horizontal axis is the sample variance in log scale. As we can see, when the variances become very small (i.e., very large sample size), Matlab's estimation of the slope converges towards the theoretical value of -4 . Figure 4 shows the effect of using a larger excitation of $\Delta\mu = \pm 0.1$. Due to the large excitation, the sample variance does not converge to *zero* as window size increases. In addition, the slope estimated by Matlab ARX model does not converges to -4 .

One further fundamental distinction must be made in the choice of a sampling interval. In traditional control, the output of the process under control is a function of time.

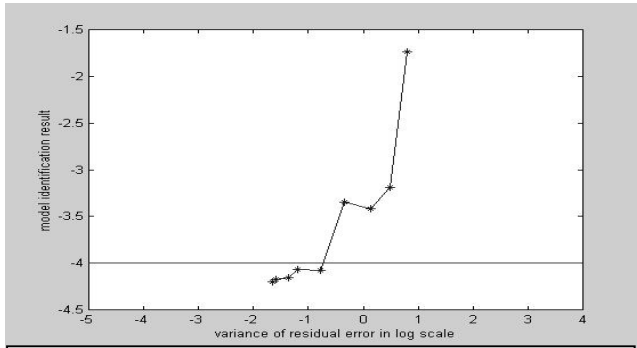


Figure 4: Effect of Excitation on Model Accuracy

Thus, the unit of sampling is measured in units of physical time since they are the independent variables, upon which the process evolves. However, we now manipulate the probability distribution of a random process, where what counts is the mean and variance of delays. The independent variable is now the number of events that affect delay mean and variance, i.e., the number of served requests. Thus, it is preferable to perform our task in the domain of events rather than in the domain of time. This is also consistent with our event based model development process. Our sampling window is therefore measured as the number of elapsed events.

We have so far developed a simple 1st order ARX model that links the effect of increases (reduction) in service rate $\Delta\mu$ to reduction (increase) in output delays Δd : $\Delta d(n) = -4\Delta\mu(n-1)$, where n is the n^{th} control sampling instance. The control gain k is simply $-1/\text{slope}$. That is, $k = 0.25$. This is because for next sampling instance of $n+1$, we want to correct the delay to $-\Delta d(n) = \Delta d(n+1) = -4\Delta\mu(n)$, so $\Delta\mu(n) = 0.25\Delta d(n)$.

We now come to the final issue of designing the control rate. The observation window and the control window are often the same in many control applications. For example, when control period is l second, the signal used for the control is often based on what has been observed in this l sec window. However, longer observation windows can be used. Consider the case where Kalman filter is used to condition the signal for a controller. Whatever the rate of control one chooses, the recursive nature of Kalman filter implies that at any time t , the signal used by the control is based on all the samples that have been observed from the very beginning, albeit the effect of samples in the remote past is small.

For any chosen window of observation, we should update the control as soon as possible so as to minimize the build-up of errors. Suppose that the observation window contains 1000 events. We can produce the 1st control update using the first 1000 events, the 2nd update using the average of events 2 to 1001 etc. To summarize, our strategy is to use a long observation window but a short control update window. To reduce the computation overhead, we may choose to update once every N events instead of every event. Figure 5 shows the

resulting control performance using an update window of 500 events, 200 events, ..., until 10 events. The horizontal axis depicts both the size of update windows and the corresponding size of the control effort, i.e., the average size of corresponding $\Delta\mu$. As we can see, a shorter update window has two important effects. First, it reduces the sample mean fluctuations. Second, it reduces the control efforts $\Delta\mu$. For examples, at $N = 500$, the average control effort is about 0.058 . When $N = 10$, the average control effort is about 0.04 . Smaller control effort makes the sharing of connections in the server easier.

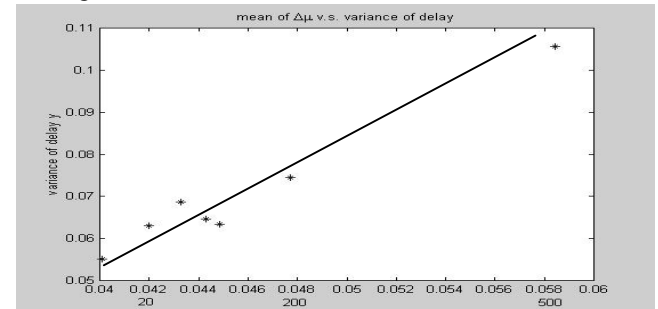


Figure 5: Effect of Control Window Size on Sample Variance

2.3 Performance Evaluation

Having introduced the basic concepts, we replace the M/M/1 model with a general G/G/k model, whose waiting time formula follows the Allen-Cunneen Approximation:

$$D_q \cong \left(\frac{c_a^2 + c_s^2}{2} \right) \frac{\rho \sqrt{2(k+1)-1}}{\mu \cdot k (1 - \rho)} \quad c = \frac{s}{m} \quad (1)$$

where traffic intensity is $\rho = \lambda/\mu$, and where s and m are the standard deviation and mean of the sample data respectively. Subscripts a and s stand for arrival process and service process respectively. The steady state response time is:

$$D_{eq} = D_q + 1/\mu \quad (2)$$

In the case of a server cluster with $k=1$ node, we have

$$D_q \cong \left(\frac{c_a^2 + c_s^2}{2} \right) \frac{\rho}{\mu \cdot (1 - \rho)} \quad (3)$$

In our experiments, we replace the M/M/1 response time model with G/G/1 response time model in Figure 1. Both the mean and variance of the arrivals are now estimated online. For simplicity, we assume that the service time distribution is exponential but this can be easily replaced by online measurement of mean and variance as well. In the experiments, the reference delay is $D_{ref} = 2$. In the first set of experiments, the inter-arrival time has Pareto distribution, which was reported to fit measurements of actual Web traffic well [9]. In the beginning of the run, there are 0.65 arrivals per unit time. This rate jumps to 0.85 arrivals per unit time in the middle of the simulation. Figure 6(a) uses queueing model feed forward control only.

The horizontal axis is the number of user observation windows with a size of 1000 events. The vertical axis is the sample mean of the delay in the user observation window. As we can see, feed forward control performs quite well by itself. This demonstrates the effectiveness of the queueing model. Figure 6(b) adds the additional feedback control to reduce the fluctuations around 2.

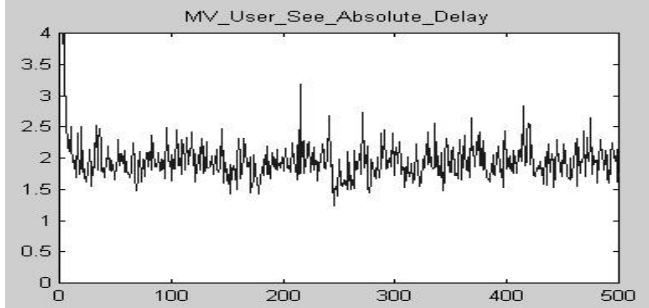


Figure 6(a): Feed Forward Control Only

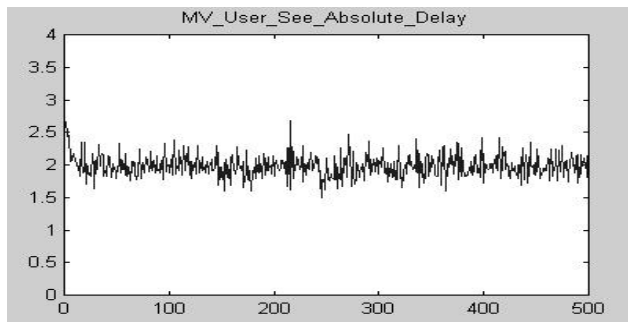


Figure 6(b): Feed forward + Feedback

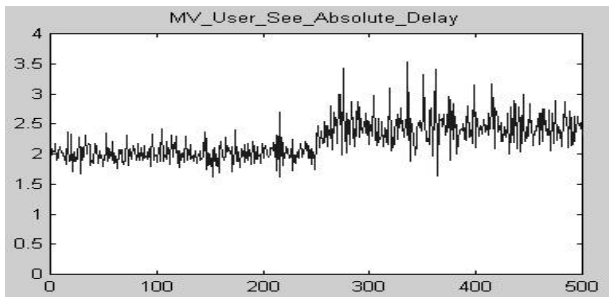


Figure 6(c): Control with fixed offset

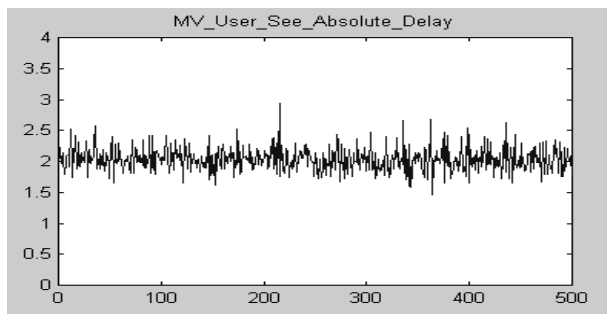


Figure 6(d)): From Poisson to Pareto

A precondition of a successful linearization of a non-linear system is that the point of linearization should be in an equilibrium state. The equilibrium service rate is the one that keeps the system in the equilibrium state. For a reference delay of 2, we know that the equilibrium service rate is $\mu = (\lambda + 0.5)$. To reduce the random fluctuations around 2, the controller adjusts the service rate by a small amount according to the linearized model.

In queueing model based control, the equilibrium service rate is automatically provided by the feed forward control from the queueing model. A statically constructed equilibrium service rate is sensitive to the change in the arrival process. As we can see in Figure 6(c), a large steady state error and a larger variance materialize after the midpoint on the x-axis.

Since a wrong equilibrium service rate results in a steady state error, we could use integral control to correct the equilibrium service rate. When a workload change causes steady state errors, the integral controller sums up the errors, and produces a negative feedback control that adjusts the service rate. However, until the set point is crossed, the sum of errors keeps increasing and so does the integral control. Thus, integral control has a tendency to produce overshoot. A small integral gain is preferred in practice. It allows for a system to slowly correct steady state errors without excessive oscillation.

Unlike a load increase in a mechanical system that produces consistent steady state errors that can be directly canceled by integral control, a queueing system has sizable random swings around the process mean in steady state. From error history alone, it would take a long integration time to filter out random fluctuations and detect the steady state error caused by workload changes. This makes it difficult in practice to rely on the integral controller alone to provide the correct offset. On the other hand, a small integration term remains a useful tool to correct bias introduced by inaccuracies in the queueing model, such as the G/G/1 model used in our experiments.

Finally, to demonstrate the adaptivity of our approach, we begin with a Poisson arrival process with arrival rate 0.65 and then switch to a process whose inter-arrival time has a Pareto distribution with arrival rate 0.85. As we can see from Figure 6(d), queueing model based control is insensitive to the changes of distribution and rate of the arrival process.

3 Experimental Investigation Using an Instrumented Apache Web Server

We now present the application of the Queueing Model Based Control in an instrumented Apache web Server. The control goal is to keep the actual sample delays close to the specified reference delay.

3.1 Implementation

We have conducted experiments of queueing model based control in an instrumented Apache Web Server [11] to evaluate the efficacy of the aforementioned scheme in controlling the timing behavior of real-life applications. When an Apache server boots up, a pool of processes is created to serve incoming TCP connection requests. When a HTTP connection request arrives, it is put into a service queue to wait for an available process. In HTTP 1.1, the current web protocol, persistent connections are implemented. Hence, a server process that accepts a connection must wait (i.e., block) for a specified amount of time for further requests from the same client. This blocking time dominates the time a process spends on serving the connection. Consequently, the average connection service rate is determined primarily by the number of server processes, and not by CPU allocation. The more processes are allocated to a client class queue, the higher the service rate. Thus, by adjusting the allocated processes, we can *approximately* control the service rate to the incoming requests and hence achieve the desired reference delay guarantees.

We modified the source code of Apache and added a new library that implements a Connection Manager that includes a Monitor, a Queueing Model Predictor, a Controller, and a Connection Scheduler. Let c denote the maximum number of server processes that the system could have (i.e. server capacity). Let $b(k)$ represent the number of server processes reserved for the given client class at the k^{th} sampling instant, where $b(k) \leq c$. In order to decide the process quota for the client class to be controlled, we implemented two modules, the Queueing Model Predictor and the Feedback Controller. According to the resulting process quotas, the Connection Scheduler serves as an actuator to allocate a certain number of server processes to connections from this client class.

The Connection Manager process runs a high-priority loop that listens to the web server's TCP socket, accepts incoming connection requests, and queues them up in the application layer. This design allows us to control the request queue without kernel modifications. The Connection Scheduler module dispatches a connection by sending its descriptor to a free server process through a corresponding UNIX domain socket. The Connection Manager time-stamps the acceptance and dispatching of each connection. The difference between the acceptance and the dispatching time is recorded as the connection delay that models the service delay we are concerned with in this study. Although the service delay also includes the queueing time in the TCP listen queue in the kernel, in our prototype this kernel-level queueing delay is negligible by design. This is because the Connection Manager always greedily accepts all incoming connection requests in a tight loop.

3.1.1 Monitor

At each sampling instant k , the Monitor is invoked to compute the average request arrival rates $\lambda(k)$, mean request inter-arrival

time m , and its standard deviation s , during the last sampling period. They are used by the Queueing Model Predictor to calculate new process quotas $b(k)$. The Monitor also computes the average connection delays $W(k)$ of the client class during the last sampling period, which will be used by the Feedback Controller to calculate adjustment $\Delta b(k)$ for process allocations.

3.1.2 Queueing Model Predictor

System profiling was carried out beforehand to get an approximate value for μ the service rate per process unit. The Connection Manager time-stamps the dispatching and closing of each connection. The difference between the dispatching and the closing time is recorded as the service time the server process spends on the connection. Using the average service time of the server process per connection, we calculated an estimate of the service rate μ of one server process.

As discussed in Section 2.3, we assume a G/G/1 queueing model for the Apache web server. For simplicity, we assume that the service time distribution is exponential. Our experimental results presented below can be further improved if a better estimate of the distribution of service times is available. In practice, however, accurate online measurement of this distribution may be both difficult and unnecessary. At each control period, the Queueing Model Predictor will use the estimated service rate μ , the online measured average request arrival rates $\lambda(k)$, the average request inter-arrival time $m(k)$ and its standard deviation $s(k)$ to produce the new desired values for $b(k+1)$ by solving the equation (3).

3.1.3 Controller

Both a P and a PI controller were implemented to control adjustment of server process quota. Both controllers are variations of the general PID controller known for its robustness and predominant use in industry. The derivative control action was not used, because it is sensitive to measurement noise. The advantage of a P controller is simplicity. The advantage of a PI controller is better elimination of residual errors. The Monitor computes and passes the average delays to the Feedback Controller, which computes the error $e(k)$ between the measured delay $D(k)$ and the desired delay D_{ref} . The output $U(k)$ of the P controller is simply proportional to the error. A digital form of the PI control function is:

$$U(k) = U(k-1) + g(e(k) - r \times e(k-1)), U(0) = 0 \quad (4)$$

where $U(k) = \Delta b(k)$, the adjustment of the server process quota; g and r are design parameters called the controller gain and the controller zero, respectively (see [1] for the parameters design using Root Locus method).

3.1.4 Connection Scheduler

The Connection Scheduler serves as an actuator to control the connection delays of the client class. It adds up the outputs from the Queueing Model Predictor and the Feedback Controller and computes the individual values of process allocation $b(k)$ for the client class. The Connection Scheduler listens to the well-known port and accepts every incoming TCP connection request. The Connection Scheduler maintains a (FIFO) connection queue Q and a process counter R for the number of processes allocated to a connection.

3.2 Experimentation

In this section, we present experimental results for our instrumented Queueing Model Based Control Apache web server, in which the Queueing Model Predictor and the feedback controller are integrated.

3.2.1 Experimental Setup

All experiments were conducted on a testbed of PC's connected with a 100Mbps Ethernet. One machine with a 450MHz AMD K6 processor and 256MB RAM was used to run the web server with HTTP 1.1 enabled, and two machines with 450MHz AMD K6 processor and 256MB or 64MB RAM were used to run clients that stress the server with a synthetic workload. The experimental setup is as follows.

Client

We modified SURGE (Scalable URL Reference Generator) [9] to generate synthetic web workloads in our experiments. The current version of SURGE assumes that users do not issue another request until they get the reply for the previous one. This assumption does not model well the behavior of a server with many independent clients whose request arrival times are not related to the times responses were sent to other requests.

Hence, we modified SURGE to generate URL requests with a rate independent of the server load while still keeping the self-similarity characteristics in the resulting traffic. We assumed that the time clients wait after they send the request for a web object follows Pareto distribution, which is not dependent on the server load or network delay.

Server

We implemented the Queueing Model Predictor and integrated it with our controller instrumented Apache web server [3]. The maximum number of server processes is configured to $c = 128$, which models a small size Web server. Each sampling period is 600 events long in all the experiments. Sampling period is defined in terms of the number of events and not absolute time because the controlled process is a probability. The dynamics of such a process depend on the number of trials. Thus, to obtain a consistent system model from one sampling period to another, we fix the number of events (trials) per sample. The connection TIMEOUT of HTTP1.1 is set to 15 seconds.

3.2.2 Experimental Results

In order to evaluate the performance of the Queueing Model Based Control approach, we compared its performance with that of the feedback controller only and Queueing Model Predictor only respectively. The goal of the system is to provide the service delay guarantee for a client class with as few resources as possible while meeting the performance requirement. In practice, it is common for the web service provider to promise the premier class a certain performance guarantee while trying to provide as good a performance as possible to the other classes. Therefore, it is desirable that only the necessary amount of resources is allocated for the premier class. Two experiments were conducted, where the service demands were changed dramatically. In both experiments, the desired connection delay for the premier class is set to $D_{ref} = 2$.

The input load patterns in the two experiments are shown in Figure 7 and Figure 9 and the average connection delays measured in every sampling period (i.e. every 600 events) are shown in Figure 8 and Figure 10 respectively.

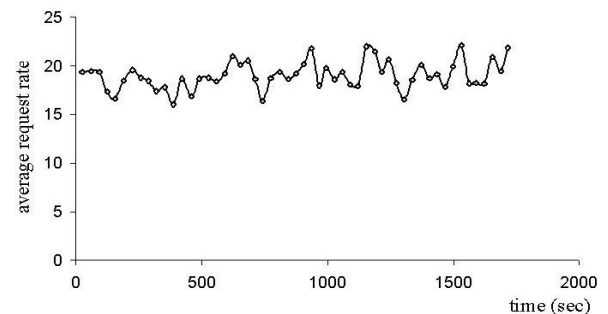


Fig 7: Input Load in Case 1

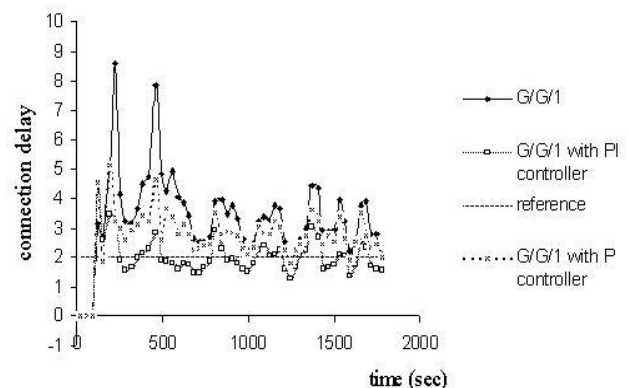


Fig 8: Exploring Control Benefits

In the first experiment, we explore the effect of adding feedback control to a resource allocation scheme that uses a G/G/1 predictor alone for resource allocation. Figure 8

shows how tightly the target delay is tracked by the system, comparing the performance of the G/G/1 predictor alone to that of the predictor augmented with a P and then a PI controller. In the second experiment, we explore the benefits of adding the predictor to a resource allocation scheme that relies on feedback control alone. Figure 10 compares how well the delay target is achieved in the feedback control scheme with and without the G/G/1 predictor.

The experiments show that the combined scheme outperformed both the predictor and the feedback control scheme used in isolation. Experimental results reveal the following observations:

- Using the aggregate of the squared errors between the desired and actual connection delay over the duration of the experiment, we can compare the performance of different schemes. The smaller the aggregate error, the better the convergence. The following table summarizes the results.

| Exp 1 (Figure 8) | G/G/1 predictor alone | G/G/1 predictor with a P controller | G/G/1 predictor with a PI controller |
|-------------------|-----------------------|--------------------------------------|--------------------------------------|
| Aggregate error | 214.08 | 74.80 | 30.32 |
| Exp 2 (Figure 10) | PI controller alone | PI controller with a G/G/1 predictor | |
| aggregate error | 56.84 | 23.84 | |

Table 1 The Aggregate Error

- Using queuing model feed forward control alone; a large steady state error develops (Figure 8). This is attributed to the fact that the web server is not exactly a single queue system. Thus, the model derived in this paper is only an approximation of the true server. The fluctuation is also quite large.
- When P (proportional) control is added, both the steady state error and fluctuations decrease (Figure 8). Proportional controllers are not effective for correcting steady state errors caused by load disturbance, because they need the existence of instantaneous errors to generate corrections.
- When integral action is added (feed forward plus PI control), the steady state error is eliminated (Figure 8).
- The closed loop performance of the PI controller is much better in the presence of the queuing predictor (Figure 10). This is because the predictor is able to supply an approximate output that achieves a delay value close to the set point. The controller therefore has to handle the residual error only.

Our experimental evaluation demonstrates the advantages of integrating a queuing theoretic predictor with a feedback controller to achieve QoS guarantees in Internet servers. The

two components have complementary strengths, jointly offering more robust tracking of performance set points in the presence of widely unpredictable load.

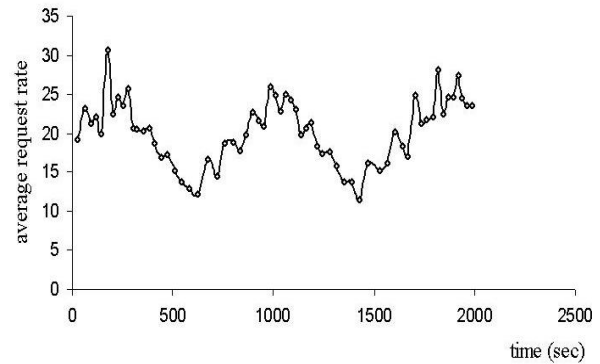


Fig 9: Input Load in Case 2

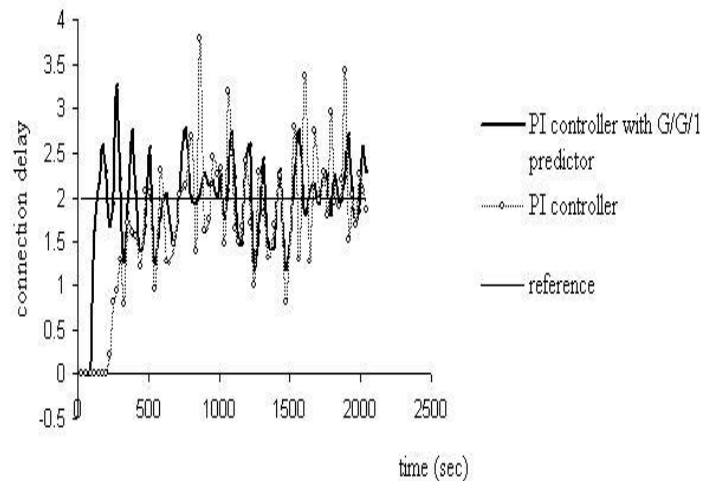


Fig 10: Exploring G/G/1 Benefits

While queuing theory and feedback control have been repeatedly used in isolation in many contexts to provide performance guarantees, the authors are not aware of any prior work that demonstrates and advocates their combined use within a single framework. The authors believe that such a combined framework merits deeper investigation to produce better theory for performance assurances in systems that can be modeled in both the queuing and the feedback control domains.

4 Related Work

Feedback control theory has been developed for more than 50 years. Many powerful tools and results have been obtained and deployed. Modern applications, such as multimedia streaming, real-time computing, transaction

processing, embedded systems, and e-commerce require some form of performance control [1, 28]. Hence, the use of control theory and tools to achieve performance guarantees in computing systems has recently emerged as an interesting and active research topic. TCP congestion control [5] is perhaps one of the most successful accomplishments in this area.

In [18, 27], congestion control in communication networks is treated as a feedback control problem. They model the available bandwidth as an autoregressive process driven by a white noise process and the congestion control problem is formulated as an LQG stochastic control problem that can be solved using a discrete-time Riccati equation. Later, networking researchers proposed new active queue management (AQM) algorithm(s) for Internet routers [19, 20]. Analysis and improvement of AQM using control theory followed. The authors of [21] linearize a non-linear model of TCP and design an AQM control system using the Random Early Detection (RED) scheme. Using standard control theory, they provide a scheme to choose design parameters that can lead to stable operation. In [22], the authors further use classical control (PI control, or Proportional-Integral controller) to develop new controllers for the linearized model of TCP and AQM. The controller is shown to have better theoretical properties than RED. Also, simulation results show that a PI controller outperforms RED.

The control of queueing networks is also an active research area. In [15, 17], Kumar et al. developed important results on the stability and performance guarantees of queueing networks. In the application of control theory to computing systems, Steere et al [23] propose a new scheme of allocating CPU to threads based on proportion and period instead of the traditional scheme based on priority. The scheme allocates to each thread a percentage of CPU cycles over a period of time, and uses a feedback-based adaptive scheduler (controller) to assign automatically both proportion and period. It shows that this new feedback driven scheme has benefits over the old priority based schemes because it has finer-grained control of allocation; lower variance in cycles allocated, and can help avoid accidental priority inversion. In [1], Abdelzaher et al. build a feedback control loop for an Apache web server that enforces desired relative delays among different service classes via dynamic connection scheduling and process reallocation. In [2], a similar control-theoretic approach was used for a Squid proxy server to guarantee cache hit-ratio by dynamically adjusting the disk space allocation. In [8], the parameters (i.e. KeepAlive time, MaxClients) of an Apache web server are dynamically allocated using a feedback controller. The goal is to keep the system's CPU and memory utilization stabilized at the desired reference value. Similar approaches are also used for Lotus Notes Server [7]. These approaches used a pure feedback control scheme with no queueing predictor. In contrast, in this paper we have shown that queueing theory can be leveraged to improve the performance of feedback control of queueing systems.

5 Conclusions

This paper has introduced a combined framework for performance guarantees in unpredictable environments. The framework integrates components from both queueing theory and feedback control. The paper discusses fundamental issues and problems that arise in implementing this framework. It also presents a preliminary performance evaluation, using both simulation and actual experimental prototypes. The results are quite encouraging in that the proposed solution is demonstrated to provide a much tighter control of system QoS in the face of unpredictable load variations.

While this paper presents an initial pilot study, more investigation is needed on several fronts. For example, better queueing models could be used to represent the Internet servers. Such models may represent a network of queues over different server resources. Multivariable control techniques may be used to allocate different resources concurrently. The choice of sampling time for rate estimation and resource reallocation can be solved as an optimization problem. In this problem, it is desired to find the best trade-off between achieving speed of response and reducing measurement noise in the delay control scheme. The fundamental tension between the two is created by the fact that the controlled entity in our scheme is probabilistic in nature, and probability can only be measured accurately over an infinite observation window assuming a stationary process. Of significant interest is the use of stochastic control techniques to account for assumptions regarding the input load distribution in the control scheme. Finally, exploring non-linear control techniques may yield significant benefits in coping with the non-linear relation between rate and delay in Internet servers. These issues are currently under investigation by the authors.

Acknowledgement

We would like to thank C. G. Lee for his helps in the design and development of the simulator. We also want to thank Karl Astrom, Vikram Adve and John Lehoczky for their comments and suggestions. This work is supported in part by ONR N0004-02-0102, MURI program N00014-01-0576, and in part by NSF grants CISE 01-37090, CCR-00-93144 and CCR-00-98269.

References:

1. T. F. Abdelzaher, C. Lu, "Modeling and Performance Control of Internet Servers", Invited Paper, 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000.

2. Y. Lu, A. Saxena, T. F. Abdelzaher, "Differentiated Caching Services; A Control-Theoretical Approach", International Conference on Distributed Computing Systems, Phoenix, Arizona, April 2001.
3. C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "A Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers," University of Virginia Tech Report CS-2001-06, January 2001.
4. Z. Liu, N. Niclausse, C. Jalpa-Villanueva, S. Barbier, "Traffic Model and Performance Evaluation of Web Servers", Technical Report, INRIA, France, <http://www.inria.fr/rrrt/rr-3840.html>, December, 1999
5. V. Jacobson, "Congestion avoidance and control", Proceedings of SIGCOMM' 88, ACM, August, 1988
6. T. F. Abdelzaher, K. G. Shin, "QoS Provisioning with qContracts in Web and Multimedia Servers", IEEE Real-Time Systems Symposium, Phoenix, Arizona, December, 1999
7. N. Gandhi, S. Parekh, J. Hellerstein, D.M. Tilbury, "Feedback Control of a Lotus Notes Server: Modeling and Control Design", American Control Conference, 2001.
8. Y Diao, N. Gandhi, J. Hellerstein, S Parekh, D. M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Serve", To appear in Network Operations and Management, 2002
9. P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance evaluation", Measurement and Modeling of Computer Systems, 151-160, 1998
10. D. Mosberger, T. Jin, "httpperf: A Tool for Measuring Web Server Performance", Performance Evaluation Review, Volume 26, Number 3, December 1998, 31-37. (Originally appeared in Proceedings of the 1998 Internet Server Performance Workshop, June 1998, 59-67.)
11. The Apache Software Foundation, <http://www.apache.org>.
12. L. P. Slothouber, "A Model of Web Server Performance", <http://www.geocities.com/webserverperformance/modelpaper.html>, June, 1995
13. K. J. Astrom, B. Wittenmark, "Computer-Controlled Systems: Theory and Design", 3rd edition, Prentice Hall, November, 1996
14. V. Jacobson, "Congestion avoidance and control", Proceedings of SIGCOMM' 88, ACM, August, 1988
15. P. R. Kumar, "A tutorial on some new methods for performance evaluation of queueing networks", IEEE Journal on Selected Areas in Communications: Special Issue on "Advances in the Fundamentals of Networking-Part I: Bridging Fundamental Theory and Networking", pp. 970-980, vol. 13, no. 6, August 1995
16. G. I. Winograd and P. R. Kumar, "The FCFS Service Discipline: Stable Network Topologies, Bounds on Traffic Burstiness and Delay, and Control by Regulators", Mathematical and Computer Modelling, vol. 23, no. 11/12, pp. 115-129, 1996
17. P. R. Kumar and S. P. Meyn, "Stability of Queueing Networks and Scheduling Policies", IEEE Transactions on Automatic Control, pp. 251-260, vol. 40, no. 2, February 1995.
18. R. Srikant, "Control of Communication Networks", in "Perspectives in Control Engineering: Technologies, Applications, New Directions," Tariq Samad (Ed.), IEEE Press, 2000, pp. 462-488.
19. S. Floyd, V. Jacobson, "Random early detection gateways for congestion avoidance", IEEE/ACM Trans. on Networking, Vol.1, No.4, August 1993
20. B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Floyd etc. "Recommendations on Queue Management and Congestion Avoidance in the Internet", Internet RFC 2309, April, 1998.
21. C. Hollot, V. Misra, D. Towsley, W. Gong, "A Control Theoretic Analysis of RED", IEEE Infocom 2001.
22. C. Hollot, V. Misra, D. Towsley, W. Gong, "On designing improved controllers for AQM routers supporting TCP flows", IEEE Infocom 2001.
23. D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, J. Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling", Operating Systems Design and Implementation, 1999
24. N. Gandhi, S. Parekh, J. Hellerstein, D.M. Tilbury, "Feedback Control of a Lotus Notes Server: Modeling and Control Design", American Control Conference, 2001.
25. Y Diao, N. Gandhi, J. Hellerstein, S Parekh, D. M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Serve", To appear in Network Operations and Management, 2002
26. W. N. Mills III, L. Krueger, W. Chiu, N. Halim, J. L. Hellerstein, M. S. Squillante, "Metrics for Performance Tuning of Web-Based Applications", The Computer Measurement Group, 2000.
27. E. Altman, T. Basar, R. Srikant, "Congestion Control as a Stochastic Control Problem with Action Delays", Automatica Special issue on Control Methods for Communication Networks, Anantharam and Walrand Eds., Dec. 1999
28. J. Almedia, M. Dabu, A. Manikntty, P. Cao, "Providing Differentiated levels of Service in Web Content Hosting", First Workshop on Internet Server Performance, Madison, Wisconsin, June, 1998