An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services

Ying Lu †, Tarek Abdelzaher †, Chenyang Lu † and Gang Tao ‡
*[0.1in] † Department of Computer Science
‡ Department of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22903
{ying@cs.virginia.edu}

Abstract—**Software mechanisms that enforce QoS guarantees often require knowledge of platform capacity and resource demand. This requirement calls for performance measurements and profiling upon platform upgrades, failures, or new installations. The cost of performing such measurements is a significant hurdle to the wide-spread deployment of open QoS-aware software components. In this paper, we introduce a new QoS-control paradigm based on adaptive control theory. The hallmark of this paradigm is that it eliminates profiling and configuration costs of QoS-aware software, by completely automating the process in a way that does not require user intervention.**

As a case study, we describe, implement and evaluate the control architecture in a proxy cache to provide proportional differentiation on content hit rate. Adaptive control theory is leveraged to manage cache resources in a way that adjusts the quality spacing between classes, independently of the class loads, which cannot be achieved by other cache resource management schemes, such as biased replacement policies [1], LRV [2] or greedy-dual-size [3].

I. Introduction

This paper makes two main contributions. First, we present a case for using adaptive control theory for application-layer QoS-provisioning in QoS-aware servers. We claim that adaptive control theory achieves a property we call *QoS portability* that reduces customization costs of QoS-aware components to new platforms. Second, we implement and evaluate this approach in the context of a differentiated proxy caching service we developed [4]. We believe that while our specific example concerns relative differentiated services guarantees, adaptive control theory can be applied to other types of QoS guarantees as well.

Efficient mechanisms for QoS provisioning have been investigated at length in several contexts. However, the costs of customizing QoS enforcement to a new platform have received less attention. For example, profiling may be needed to determine execution overheads on the new platform. In many QoS-aware algorithms, various control knobs need to be tuned depending on expected load conditions and platform resource capacity. QoS portability is a qualitative measure of the costs involved in configuring the QoS enforcement parameters upon platform changes. Portability, in a general sense, is the ability to move a software system from platform to platform without incurring high cost. Java, for example, achieves portability in a functional sense. However, the correctness of a QoS-aware system depends not only on function but also on performance. If extensive re-configuration is needed to tune the performance of the system for the new platform or new operating conditions we say that the system is not QoS-portable.

Most QoS-sensitive software components are inherently not QoS-portable in that they require additional profiling, identification, tuning, or similar QoS-related initial costs in order to produce the desired QoS on each new platform. Adaptive control theory allows us to build self-tuning performance regulators. In the context of QoS control, such regulators will automatically build a system model and maintain the performance of the system at a level that satisfies QoS requirements, whenever this is feasible. Like other control-theoretic techniques, adaptive control is based on a strong analytic framework that achieves convergence of the computing system to the desired QoS specification despite unpredictable load variations and resource constraints.

Our choice of application is motivated by the heterogeneity of Internet content and clients. Several QoS models have been proposed, such as Integrated Services [5] and Differentiated Services [6] to customize service guarantees. In particular, the proportional differentiation model [7], [8] has received much recent attention. It provides the network operator with "tuning knobs" for adjusting the quality spacing between classes, independently of the class loads. Unlike other differentiated service models, proportional differentiated service provides both *predictable* and *controllable* relative differentiation. It is *predictable* in the sense that 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.

While a significant amount of research went into implementing proportional differentiated services at the network layer, the proliferation of application-layer components that affect client-perceived network performance such as proxy caches and

0-7803-7426-6/02/$17.00 (c) 2002 IEEE.

content distribution servers motivates investigating application layer QoS. The key performance acceleration mechanism in the web infrastructure is web caching and content distribution proxies. Proxy server space is not an "infinite" resource. For example, AOL reports daily traffic on their proxy caches that is in excess of 8 Terabytes of data. With a hit rate of 60%, common to AOL caches, the cache has to fetch $8 * 40\% = 3.2$ Terabytes of new content a day. Even with secondary storage cost of as low as $5/Gigabyte, it would cost about $480,000 to store a month's worth of traffic.[1] This expense makes it reasonable to consider some form of QoS-aware cache resource allocation in which resources are allocated preferentially to content classes that are more important or more sensitive to delays. We use this idea as a case study of how our adaptive control framework may be applied.

The rest of this paper is organized as follows. Section II presents related work. Section III details the adaptive controller design for the proxy cache. In Section IV, the implementation detail of the adaptive control architecture is presented. Section V describes the experimentation and performance evaluation. Finally, Section VI concludes this paper.

II. Related Work

In most open systems such as e-commerce, on-line trading and multimedia, very little is known in advance about resource availability and load. Hence, a *feedback-based* approach is needed for maintaining QoS guarantees. In such an approach, resource allocation is adjusted depending on measured performance to achieve a desired QoS or performance objective. A special subset of feedback-based QoS architectures are those based on a control-theoretic framework. Feedback control theory provides a means for analyzing feedback-based adaptation, setting controller parameters, and bounding the recovery time of the system to the desired performance upon transient load perturbations. Several recent papers [9], [10], [11] 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 [12], [13], [14], 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 [15]. In [16], guarantees were made on power dissipation by applying control-theoretical techniques to microprocessor thermal management. In our previous work [4], we applied feedback control theory to proxy cache QoS, but didn't address the problem of automating controller tuning. At the network layer, control theory was applied to packet flow control in Internet routers [17], [18]. Due to the usefulness of the control-theoretic approach and its versatile applications, middleware frameworks emerged for control-based QoS assurances [19]. The authors of [19], [20], [21] provided tools to help apply control-theoretic design techniques to a larger class of systems.

[1]Fault-tolerance measures such as storage redundancy in industrial-strength servers would further increase the price of the cache space, not to mention the additional maintenance and system administration costs.

The feedback control approach requires estimation of a system model to tune the controller. This has two limitations. First, model parameters often depend on the software and hardware configuration (e.g., the total storage space of a web cache server). Thus, system profiling needs to be repeated every time a system is upgraded. Second, model parameters can be a function of workload. Although fixed controllers can be designed to handle nominal workload variations, severe workload changes can degrade control loop performance. For example, in the web cache case presented in this paper, self-similar traffic causes significant variability over a wide range of time scales, which introduces varying process dynamics into the system. Consequently, a fixed controller [4] is not able to provide good performance as we show in Section V-D. This deficiency motivates a more intelligent controller that could work well over a wider range of operating conditions.

Adaptive control theory enables a new generation of QoS services beyond the existing fixed control loops to solve the above limitations and thereby achieve QoS portability. The key advantage of adaptive control is that it can automatically re-tune the controller in response to changes in the system model. Adaptive-control-based QoS services are QoS-portable because they automatically tune themselves to any platform they are installed on. In essence, the evolution from an open loop, to a fixed feedback control loop, to adaptive control approaches represents three generations of QoS services with improving robustness of QoS guarantees with respect to system and workload unpredictability.

We apply adaptive control to differentiated caching services. In recent years, many good cache replacement policies have been proposed. For example, [3] introduces the greedy-dual-size algorithm, which incorporates locality with cost and size concerns in the replacement policy. The authors of [2] propose LRV, which selects for replacement the document with the Lowest Relative Value among those in cache. In [22], a number of techniques are surveyed for better exploiting the bits in HTTP caches. Aiming at optimally allocating disk storage and giving clients better performance, their schemes do not provide QoS guarantees.

In [1], a weighted replacement policy is proposed which provides differential quality-of-service. However, their differentiation model doesn't provide a "tuning knob" to control the performance distance between different classes. Fixed weights are given to each server or URL, but higher weights alone don't guarantee user-perceived service improvement. For instance, the hit rate for the high weight URL may be very low because the proxy cache is over-occupied by many popular low weight URLs. Although the scheme is good in the sense that it saves backbone traffic by caching popular files, there is no predictability and controllability in the differentiated service. In contrast, proportional differentiated caching services described in this paper provide application-layer QoS "tuning knobs" that are useful in a practical setting for network operators to adjust the quality spacing between classes depending on pricing and policy objectives.

III. An Adaptive Control Framework for QoS Guarantees

In order to overcome the limitations of the fixed controller when dealing with varying operating conditions, we apply adaptive pole placement control to QoS-aware web caching. Adaptive pole placement control is one of the adaptive control schemes that could achieve QoS portability in computing systems. In this investigation, we focus on only one adaptive control technique. We leave the performance comparison of different adaptive control schemes as an important future topic.

The goal of our system is to provide proportional differentiation on average hit rate of different content classes. Let us assume there are $N \geq 2$ content classes in the system (for example, WML and regular HTML content). Let the measured average hit rate of $class_i$ be H_i. In a proportional differentiated caching service [4], the quality spacing between classes is guaranteed by imposing constraints of the following form on successive pairs of classes

$$\frac{H_i}{H_j} = \frac{c_i}{c_j}, i, j = 1, \cdots, N \quad (1)$$

where c_i is the QoS specification. If we think of cache space as partitioned among classes, assigning more storage space to a traffic class will increase its hit rate and vice versa. This forms the basis of our feedback architecture. In the following two subsections we describe this architecture and its theoretical underpinnings respectively.

A. The Adaptive Control Architecture

Our adaptive control architecture is described in Figure 1. The core of the architecture is a controller which periodically computes the storage allocation adjustments needed to reach the target relative hit ratio. One controller exists for each successive pair of classes. An automatic model estimator periodically monitors actual system performance and current resource allocation, and derives an analytic model that describes the relation between the two. This model is dynamically fed to a controller design block which in turn fine-tunes the controller function. The controller designer accepts as input a specification of the convergence speed to target performance. Network operators merely supply a platform independent QoS specification.

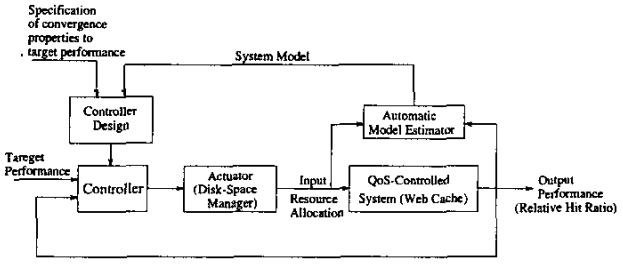


Fig. 1. The adaptive control architecture.

A main step to automate controller design is to produce a model that establishes the cause-effect relation between resource adjustments and measured QoS. In control theory this model is represented by a difference equation. While it might seem at first that such a relation would be too approximate for practical purposes, a key insight is that the current performance of a computing system (such as current experienced delay) often depends on a finite history of past measurement (such as a finite history of past arrival rates and queue fill levels). This type of relation is what a difference equation expresses, which makes it a useful modeling tool. The main appeal of control theory, however, lies in its robustness. Simple control loops are often quite tolerant to modeling errors. Thus, controllers designed based on approximate linear difference equation models are generally sufficient as shown in several prior publications [9], [14], [4], [15], [16], [13], [12]. Hence, in this paper, we demonstrate the use of a linear adaptive controller based on a linear approximation of the intrinsically nonlinear web cache system. We experimentally show that this approach is robust and leads to a significant system performance improvement over non-adaptive schemes.

Note that the success of the feedback loop in achieving its QoS goal is contingent on the feasibility of the specified target relative hit ratio. Assume the average hit rate of the unmodified cache is $X\%$. In general, when space is divided equally among classes, the maximum multiplicative increase in space that any one class can get is upper-bounded by the number of classes N. Since hit rate increases logarithmically with cache size, in a cache of total size S, the maximum increase in hit rate for the highest priority class is upper-bounded by $ln\ N$. After some algebraic manipulation, this leads to $X_{max} \leq X/(1 - ln\ N/ln\ S)$. If the relative hit ratio between the top and bottom classes is q, the hit rate of the bottom class is upper bounded by X_{max}/q. This gives some orientation for specifying the desired relative hit ratio q.

B. Theoretical Underpinnings

We cast the proportional hit rate differentiation into an adaptive pole placement control problem as shown in Figure 2. For simplicity, we focus on a system with two classes. A larger number of classes would entail a control loop per pair. For a given pair of classes, the desired ratio, $\frac{c_i}{c_j}$ is the target performance reference in the corresponding adaptive control loop. The control objective is to make the system output $y(k)$ (which is the measured relative hit ratio $\frac{H_i}{H_{i+1}}$) track a reference trajectory $y_m(k)$, which converges to $\frac{c_i}{c_{i+1}}$. The ratio of storage space S_i assigned to the respective classes is the plant (proxy cache) input $u(k) = \frac{S_i(k)}{S_{i+1}(k)}$. The controller design problem is to find a function that computes the value of $u(k)$ such that the control objective is achieved.

As a valid approximation verified by our experimental results (Section V), we model the web cache as a linear and time-

invariant system as

$$y_i(k)=\sum_{j=1}^{n} -p_{n-j}^{i} y_i(k-j) + \sum_{j=1}^{n} r_{n-j}^{i} u_i(k-j), 1 \le i \le N-1 \quad (2)$$

where y_i is the ratio of average hit rate, u_i is the ratio of storage space, p_{n-j}^{i}, r_{n-j}^{i}, $j = 1, \cdots, n$, are unknown but constant parameters, and N is the number of classes. For our results reported in this paper, we have taken $n = 2$, that is, we consider a second-order case. In our experiments, we divide the content into two classes, that is $N = 2$, and we only need one equation to describe the QoS web cache

$$y(k)=\sum_{j=1}^{n} -p_{n-j} y(k-j) + \sum_{j=1}^{n} r_{n-j} u(k-j) \quad (3)$$

where $y(k) = \frac{H_0(k)}{H_1(k)}$, $u(k) = \frac{S_0(k)}{S_1(k)}$, $p_{n-j} = p_{n-j}^{1}$, and $r_{n-j} = r_{n-j}^{1}$. Taking the z-transform of the above equation, we have

$$y(z) = \frac{R(z)}{P(z)} u(z) \quad (4)$$

where

$$P(z) = z^n + p_{n-1} z^{n-1} + \cdots + p_1 z + p_0 \quad (5)$$

$$R(z) = r_{n-1} z^{n-1} + r_{n-2} z^{n-2} + \cdots + r_1 z + r_0. \quad (6)$$

Since in our application we want the system output $y(k)$ to converge to a constant level, we specify the reference output signal as $y_m(k) = c \neq 0$, which is characterized in the z-domain as

$$Q_m(z) y_m(z) = 0, Q_m(z) = z - 1. \quad (7)$$

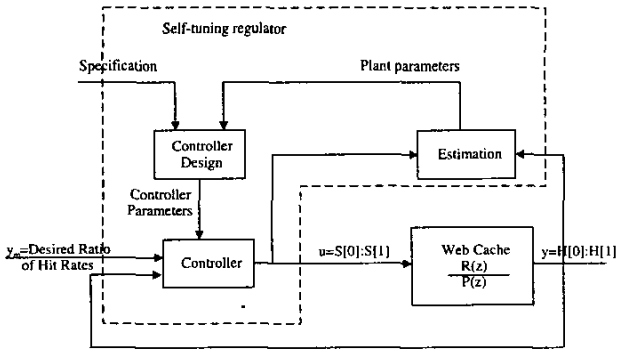


Fig. 2. Adaptive QoS web cache control system.

The adaptive controller [23] applied to our QoS proxy cache system (2) is

$$u(k) = \sum_{j=1}^{l} a_j(k) u(k-j) + \sum_{j=0}^{l} b_j(k)(y_m(k-j) - y(k-j)) \quad (8)$$

where l is the controller order to be specified, $a_j(k)$, $b_0(k)$, and $b_j(k)$, $j = 1, \cdots, l$, are controller parameters that are adjusted automatically online.

In our adaptive control system shown in Figure 2, the self-tuning regulator is fed with the output y, the reference y_m and the plant input u signals at each sampling instant. The adaptive controller automatically produces the new plant input u for the next sampling instant, which will ensure that the system output y tracks the reference output y_m asymptotically.

Unlike those of a fixed controller, the parameters of the adaptive controller are updated online to adapt to system parameter uncertainties, for achieving desired closed-loop system performance. At each sampling instant, the system output y, and input u are fed to the cache model estimator which generates new estimates of the parameters p_{n-j}, r_{n-j}. Based on the estimated model parameters, the controller parameters $a_j(k)$ and $b_j(k)$ are updated accordingly. In the next two subsections we describe how the cache parameters and controller parameters are estimated and updated respectively.

1) Parameter Estimation: To estimate the web cache system parameters p_{n-j} and r_{n-j}, we use a standard gradient algorithm from control literature [23]. The input $u(k)$ and output $y(k)$ measurements are fit to the model described in equation (2). Define the vector

$$\phi(k) = [u(k-n), \cdots, u(k-1), y(k-n), \cdots, y(k-1)]^T. \quad (9)$$

Then the cache model(2) can be expressed as

$$y(k) = \theta_p^{*T} \phi(k), \quad (10)$$

where

$$\theta_p^* = [r_0, \cdots, r_{n-1}, -p_0, \cdots, -p_{n-1}]^T. \quad (11)$$

Let

$$\theta_p(k) = [\hat{r}_0(k), \cdots, \hat{r}_{n-1}(k), -\hat{p}_0(k), \cdots, -\hat{p}_{n-1}(k)]^T \quad (12)$$

be the estimate of the model parameter vector θ_p^*, which is generated from an estimator with initial estimates taken as some arbitrary values less than 1. Let $\Gamma \in R^{2n \times 2n}$ be the adaptation gain matrix which is chosen as a diagonal matrix with its diagonal elements set to 0.5. The estimator equations at sampling instant k are

$$\epsilon(k-1) = \theta_p^T(k-1)\phi(k-1) - y(k-1) \quad (13)$$

$$m(k-1) = \sqrt{\kappa + \phi^T(k-1)\phi(k-1)}, \ \kappa > 0 \quad (14)$$

$$\theta_p(k) = \theta_p(k-1) - \frac{\Gamma \phi(k-1)\epsilon(k-1)}{m^2(k-1)}. \quad (15)$$

At each sampling instant, the estimator updates the parameter estimate $\theta_p(k)$ using new measurements of the plant input $u(k)$ and output $y(k)$. Then these estimated plant parameters will be used to calculate the parameters of a chosen controller structure, based on a design equation.

2) Controller Design: With the estimated cache parameters $\hat{p}_i(k)$ and $\hat{r}_j(k)$, we define the estimates of $P(z)$ and $R(z)$ (equation (3)-(5)) as

$$\begin{aligned}\hat{P}(z) &= z^n + \hat{p}_{n-1}(k)z^{n-1} \\ &\quad + \cdots + \hat{p}_1(k)z + \hat{p}_0(k) \qquad (16)\\ \hat{R}(z) &= \hat{r}_{n-1}(k)z^{n-1} + \hat{r}_{n-2}(k)z^{n-2} \\ &\quad + \cdots + \hat{r}_1(k)z + \hat{r}_0(k). \qquad (17)\end{aligned}$$

An adaptive controller is designed to assign our system a desired closed-loop characteristic polynomial $A^*(z) = z^{2n+n_q-1}$, where n is the cache model order and n_q is the order of the reference polynomial $Q_m(z)$. The $A^*(z)$ polynomial corresponds to the specification input to the controller designer and it characterizes the system convergence speed to the target performance (see Figure 1). The design equation

$$\hat{C}(z)Q_m(z)\hat{P}(z) + \hat{D}(z)\hat{R}(z) = A^*(z) \qquad (18)$$

is solved to find the polynomials $\hat{C}(z)$, $\hat{D}(z)$ in the form

$$\begin{aligned}\hat{C}(z) &= z^{n-1} + c_{n-2}(k)z^{n-2} \\ &\quad + \cdots + c_1(k)z + c_0(k) \qquad (19)\\ \hat{D}(z) &= d_{n_q+n-1}(k)z^{n_q+n-1} \\ &\quad + \cdots + d_1(k)z + d_0(k). \qquad (20)\end{aligned}$$

Finally, choosing $\Lambda_c(z)$ as monic stable polynomial of the form $\Lambda_c(z) = z^{n+n_q-1}$ and letting $l = n + n_q - 1$, we design the adaptive controller for our QoS web cache as

$$u(z) = \frac{B(z)}{A(z)}(y_m(z) - y(z)) \qquad (21)$$

where

$$\begin{aligned}A(z) &= \frac{\hat{C}(z)Q_m(z)}{\Lambda_c(z)} \\ &= 1 - a_1(k)z^{-1} - \cdots - a_{l-1}(k)z^{-(l-1)} \\ &\quad - a_l(k)z^{-l} \qquad (22)\\ B(z) &= \frac{\hat{D}(z)}{\Lambda_c(z)} \\ &= b_0(k) + b_1(k)z^{-1} + \cdots + b_{l-1}(k)z^{-(l-1)} \\ &\quad + b_l(k)z^{-l}, \qquad (23)\end{aligned}$$

which is the same as that given in (7). In our study, $n = 2$ (see equation (24)) and $n_q = 1$ (see equation (6)).

This completes the adaptive controller design. Note that, in our design, the ideal output tracking property: $\lim_{k\to\infty}(y(k) - y_m(k)) = 0$, is ensured if the presented adaptive pole placement control scheme is applied to a linear time-invariant system of the form (2). For our web cache system, its dynamic model is only approximated by a linear system, and there are possible nonlinear characteristics and/or parameter variations in the dynamics. Therefore, in theory, the ideal tracking property may not be guaranteed for the web cache system. On the other hand, from system identification results, the linear approximation of the web cache system is valid from a practical point of view and its approximation error has been shown to be small in the average sense (see Figure 4 in Section V). It is therefore reasonable to apply an adaptive linear control scheme to the web cache system. As it will be shown in Section V, such an adaptive control scheme results in a good system tracking performance, despite the uncertainties of the system parameters to the applied adaptive control design. Motivated by this favorable and encouraging result, we believe that using adaptive control designs for web cache system applications is a promising direction. Modifications to system modeling and control designs to handle system nonlinearities and parameter variations may further improve system performance. This should be an important topic for future investigation.

IV. Implementation

We modified Squid, a high-performance proxy caching server, to validate and evaluate our adaptive control scheme for proportional differentiated service. We added five modules in our implementation to the original Squid proxy, namely: *timer*, *self-tuning regulator*, *classifier*, *actuator* and *output sensor*. The timer sends signals to output sensor and self-tuning regulator to let them update their outputs periodically. The classifier is responsible for request classification and the actuator is in charge of cache space allocation and deallocation.

In order to make the adaptive control loop work at a fixed time interval, we add a module to Squid that regulates the control loop execution frequency. Every certain period, the output sensor is invoked to measure the smoothed average hit rate [4] and calculate the ratio of average hit rate between classes; the self-tuning regulator estimates the cache system parameters (equation (12)-(14)), and changes the controller parameters by solving the design problem (equation (17)-(22)) and the new parameterized controller (equation (7)) outputs the new ratio of desired space for classes, which is then used by the actuator to adjust the cache space assignment.

In Squid, the cache disk space management includes two separate processes: deallocation and allocation. The desired space for each class calculated by the controller will be used to guide them. In the space deallocation process, the entries of the LRU list are scanned from the bottom. When scanning one entry, the cache first checks which class it belongs to and then decides whether to remove it or not. Let $realspace_i$ be a running counter of the actual amount of cache space used by $class_i$. If the cache space occupied by the class is less than the desired cache space for it ($realspace_i < S_i[k]$), the entry will not be removed. Otherwise it will. Similarly, in the space allocation process, whenever a page is fetched from its origin server, the decision of whether to save it in the cache disk or not is based on which class the page belongs to and the current

cache space of the class. If the cache space occupied is greater than the desired cache space for the class ($realspace_i > S_i[k]$), the page will not be saved. That is, we change the status of the page to be not cachable.

V. Experimentation

In this section, we present evaluations of the QoS adaptive pole placement controller in an instrumented cache prototype named Squid. Both synthetic and log-based workloads are used.

A. Experiments Setup

The testbed for our experiments consists of three parts, the workload generator, a proxy cache and the web servers. We configure the proxy cache to be 10MB. The reason for choosing such a small size proxy cache is to simulate the real world in a fast rhythm. In our experiment, we choose 30 seconds as the sampling time. While in reality, we probably will not change the cache disk space allocation among traffic classes so frequently; once every 5 minutes may be a better choice. Therefore, in order to simulate the similar relative server conditions in 30 seconds instead of 5 minutes, we choose small proxy cache size. We conduct experiments with both synthetic workload and log-based workload as follows

- *Synthetic Trace Generation*
  To generate a synthetic trace of web accesses, we use six copies of Surge 2.2 (Scalable URL Reference Generator) [24] running on different machines that send URL requests to the cache. The main advantage of Surge is that it generates realistic web workloads that faithfully mimic client access patterns, request distributions, think time distributions, document popularity, embedded object references, and locality of reference properties. We modify Surge such that three of the copies send requests for type A files ($class_0$) and the other three send requests for type B files ($class_1$). To test the performance of the cache under saturation, we configure the file population to be 33 times the cache size. As mentioned in Section III-A, in order to apply the proportional differentiation model in practice, we have to make feasible QoS specifications. Hence, we set the reference ratio to $\frac{H_1}{H_0} = 2:3$ in all the synthetic trace experiments, which can be shown to be a feasible constraint. The QoS web cache and the Apache [25] web server are started on two other machines.
- *Empirical Trace Generation*
  A URL reference generator is developed, which reads URLs from a proxy trace and generates the corresponding requests to our instrumented proxy that implements proportional differentiated service. The proxy trace used is extracted from the NLANR (National Laboratory for Applied Network Research) sanitized access logs available at URL: ftp://ircache.nlanr.net/Traces/. At reception of a request, the proxy cache will contact the actual web server mentioned in the request across the Internet for the reference. For example, if a request for "www.cnn.com" is found in the data set, our proxy cache will actually contact the CNN web server. The reason we set our testbed this way is that we want to investigate how the QoS web cache performs in the real life. Only using such "real" data can we prove that our architecture works well on the Internet. However, in order to simulate the real world, either we have to carry out the experiment for the real time scale, for example, days or we need to change the scale of the simulation. As we have mentioned, we choose a small web cache for that reason and we should only expect the small web cache to service relatively reasonable file sizes. Based on this belief, out of the 89000 URLs in our log trace, we removed 27 URLs, which request files that are 4MB or larger. For example, one of the requests removed is http://mssjus.www.conxion.com/download/win2000ddk/install/combined-8-00/nt5/en-us/win2kddk.exe, whose size is 66MB. We remove it because in reality, we don't expect a file whose size is six times that of the proxy cache.
  Based on the referenced URLs, we divide the requests into two traffic classes of similar volume, as we believe similar volume traffic classes are comparable and only for those classes, the application of the proportional differentiation model makes sense. Since it is not hard for the proxy cache administrators to get the average hit rate information for the classes, as discussed in Section III-A, they can easily derive a suitable value for the desired relative hit ratio. In our experiments, the natural ratio of average hit rate between the two classes (with no forced differentiation) was $\frac{H_0}{H_1} = \frac{2}{1}$. To favor $class_0$, we set the desired value for $\frac{H_0}{H_1}$ to be $\frac{3}{1}$.

B. System Identification

We begin our experimental evaluation by making the case for adaptive control, as opposed to using fixed controllers. The case will be made by observing experimentally that system model parameters may vary depending on the workload generation process. Hence, no fixed controller is universally applicable. We use system identification in each case to estimate the model of the web cache, whose structure is given by the equation

$$y(k) = \sum_{j=1}^{n} -p_{n-j}y(k-j) + \sum_{j=1}^{n} r_{n-j}u(k-j) \qquad (24)$$

where $y(k) = \frac{H_0}{H_1}$, $u(k) = \frac{S_0}{S_1}$, and $n = 2$ in our study. We use a pseudo-random digital white noise generator as actuator input. This actuator switches the space allocation ratio accordingly between the two classes. White noise input has been commonly used for system identification [26] because it facilitates input/output correlation and hence model parameter estimation. In the following we present the identification results of two different experiments. In one a synthetic workload generator is used, and in the other, the cache is fed with requests from an actual proxy-log.

- *Synthetic Trace Data*
  For the experiment with the synthetic workload, Figure 3 depicts the output of the estimator. It can be seen that the estimated parameters are changing very slowly and their approximate values are $p_1 = -1.13$, $p_0 = 0.22$,

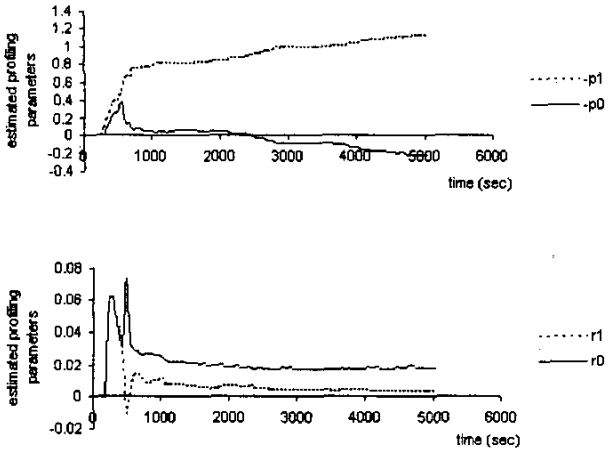


Fig. 3. Estimated system parameters (with synthetic log).

$r_1 = 0.0027$, and $r_0 = 0.0177$. Thus from equation (2), we have

$$y(k) = 1.13y(k-1) - 0.22y(k-2) + 0.0027u(k-1) + 0.0177u(k-2). \quad (25)$$

To verify the accuracy of the model, we re-run the experiment by simulating the proxy cache with a different white noise input (i.e., with a different random seed) and compare the actual relative hit ratio with that predicted by the estimated model (24). The result is illustrated in Figure 4. We can see that prediction of the estimated model is consistent with the actual relative hit ratio throughout the experiment. It shows that our estimation of the proxy cache as a second order linear and time-invariant system is acceptable.

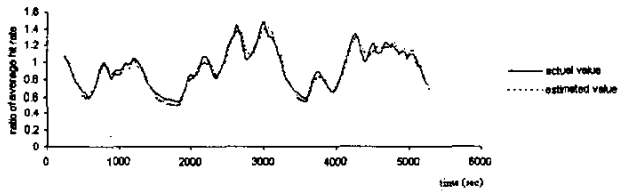


Fig. 4. System modeling validation (with synthetic log).

Taking the z-transform of equation (24), we have the transfer function representation of the proxy cache

$$y(z) = \frac{R(z)}{P(z)}u(z) = \frac{0.0027z + 0.0177}{z^2 - 1.13z + 0.22}u(z). \quad (26)$$

As shown by equation (25), the zero $-\frac{r_0}{r_1} = -6.556$ of $R(z)$ is unstable, i.e. outside the unit circle $|z| \leq 1$. It indicates that $P(z)$ and $R(z)$ are co-prime (they do not have any common factors in z) because the zeros of $P(z)$ are all in $|z| \leq 1$. This is a necessary condition for the use of adaptive pole placement control.

- *Empirical Trace Data*
  Next, we carry out the same system identification experiment for the log-based workload. By comparing the results with that of the synthetic workload, we want to check if the system parameters or the system order change.

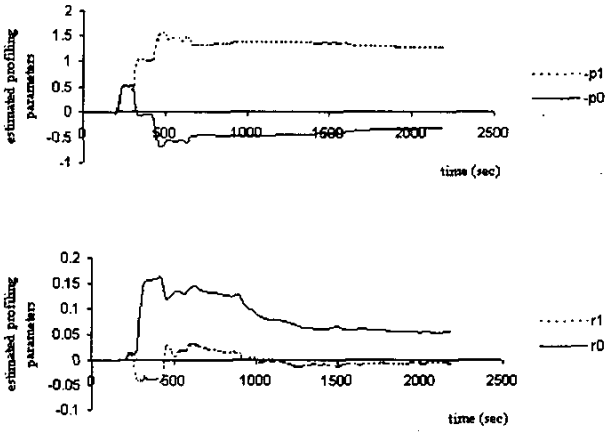


Fig. 5. Estimated system parameters (with empirical log).

Figure 5 shows the new parameter estimates: $p_1 = -1.26$, $p_0 = 0.34$, $r_1 = -0.0082$, and $r_0 = 0.0517$, which are different from the estimates with a synthetic workload (see Figure 3). The transfer function representation of the proxy cache is

$$y(z) = \frac{R(z)}{P(z)}u(z) = \frac{-0.0082z + 0.0517}{z^2 - 1.26z + 0.34}u(z). \quad (27)$$

We can see that $P(z)$ and $R(z)$ are still co-prime. The web cache can be still modeled as a second-order linear system, as shown by the model validation results in Figure 6. Hence, the assumptions required for the adaptive pole placement control still hold, even when traffic changes.

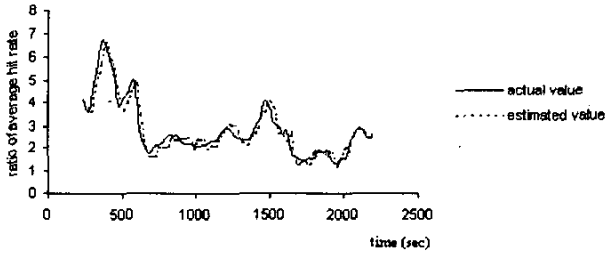
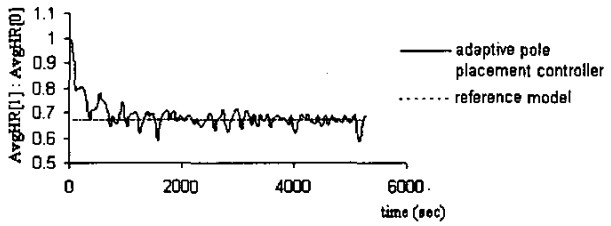


Fig. 6. System model validation (with empirical log).

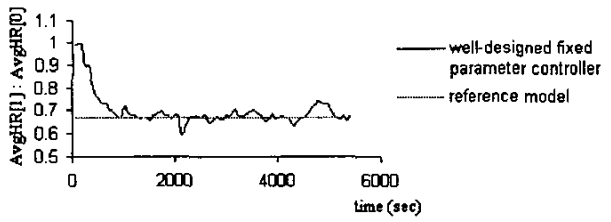
The system identification results reveal that the change of traffic leads to the change of parameters. That explains why the fixed parameter controller doesn't work well for the empirical workload (see Section V-D). The fixed parameter controller is designed based on the information obtained with a synthetic workload. Thus, there is no guarantee that it will work well for other traffic patterns. In contrast, in the adaptive control case, since the structure of the system doesn't change (a second-order linear system), the adaptive control scheme can still effectively use system parameter estimation to adjust the controller parameters in real time to adapt to the changed traffic.

C. Synthetic Trace QoS-Control Experiments

In this section, we report on performance experiment using a synthetic workload. We compare the adaptive controller with a well-designed fixed parameter controller. Performance results are shown in Figure 7. In both cases, the measured ratio of average hit rate converges to the reference value after an initial transient.



a) Web cache response with an adaptive pole placement controller.

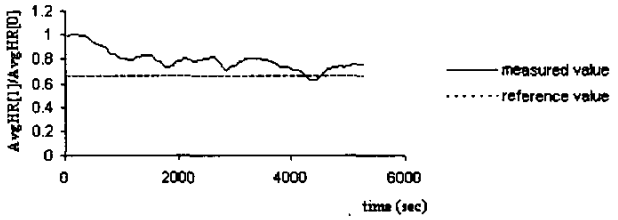


b) Web cache response with the fixed parameter controller.

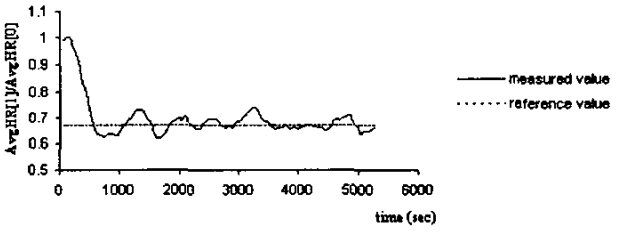
Fig. 7. Ratio of average hit rate (experiment result for a synthetic log).

Although both controllers achieve the same control objective, they are based on different approaches. For the fixed parameter controller, we first carefully modeled the web cache and estimated its model parameter, which reflected the relation between cache size and cache hit probability. Based on this model, we then derived the desired controller that will meet the control objective. While for the adaptive controller, we do not need such detailed knowledge of the system. Instead, we only assume some knowledge of the system structure, i.e., consider the web cache as a second-order linear system, then apply the adaptive control scheme to automatically estimate the cache parameters and complete the controller design online. The good performance achieved by our adaptive controller proves the applicability of such an automatic approach to a real computing system.

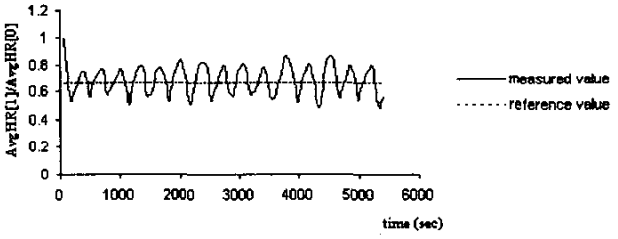
An advantage of adaptive control is that the controller is retuned on-line continuously to catch up with system parameter changes. This prevents the controller from growing poorly tuned. The effect of a poorly tuned controller on system performance are illustrated in Figure 8. It shows the behavior of our prototype, as compared to the desired performance differentiation, in the presence of a proportional feedback controller, whose gain K is varied. The figure indicates that with small K, the convergence speed is too slow; while with large K, the system becomes unstable. An adaptive controller finds the right parameters for controlling the system and adjusts them as system parameters become uncertain.



a) K = 3000



b) K = 10000

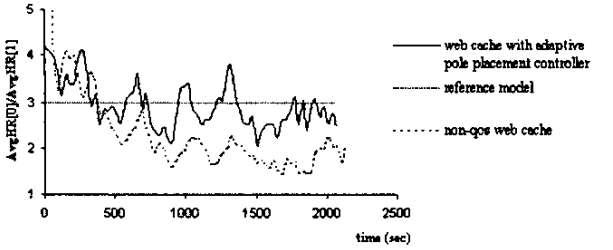


c) K = 100000

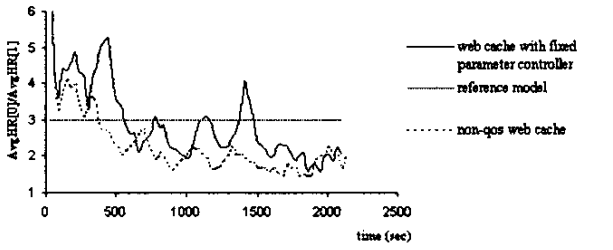
Fig. 8. Ratio of average hit rate for a proportional feedback controller.

We have demonstrated the performance of the controllers under a synthetic workload. In order to test the flexibility of

the controllers, we use an empirical workload in the second part of the experiment to check if the performance is still good.



a) Response of a web cache with an adaptive pole placement controller vs. that of a regular web cache.



b) Response of a web cache with a fixed parameter controller vs. that of a regular web cache.

Fig. 9. Ratio of average hit rate (experiment result for a real log).

D. Empirical Trace QoS-Control Experiments

In these experiments, we send a log-based URL request sequence to the regular Squid cache, a web cache with a fixed parameter controller and a web cache with an adaptive pole placement controller respectively. Figure 9 shows the resulting ratio of average hit rate. From the figure, we can see that the adaptive pole placement controller (Figure 9-a) achieves the QoS goal and brings the ratio to the desired value. While our fixed parameter controller (Figure 9-b) is not so effective. This proves in dynamic environments, the adaptive controller is more flexible than the fixed-parameter controller. The result is not surprising. As shown in Section V-B, when the traffic changes, cache system parameters change, while the system order and structure remain the same. Therefore, our adaptive controller works well in the dynamic environments, while the fixed parameter controller doesn't.

VI. Conclusions

The application of adaptive control theory for QoS-aware computing is a promising direction. As a preliminary investigation, this paper made two contributions. First, we demonstrated an adaptive control methodology for constructing a QoS-aware proxy cache. While previous work used offline controller design and tuning methods which require human intervention for parameter re-estimation and controller re-configuration, using adaptive control theory, we automated the parameter estimation and controller design process. We believe this approach can be applied more generally to achieve QoS portability of QoS-sensitive software components. Second, we proposed a dynamic resource management scheme which provides proportional hit rate differentiation in proxy caches. Cache space allocation is adjusted at run-time to the dynamics of Internet traffic, which guarantees predictable and controllable differentiation among content classes.

VII. Acknowledgements

The work reported in this paper was supported in part by the National Science Foundation under grants ANI-0105873, CCR-0093144 and CCR-0098269. We would like to thank Xidong Tang for his insightful feedback on the earlier manuscripts of this paper.

References

[1] Terence P. Kelly, Yee Man Chan, Sugih Jamin, and Jeffrey K. MacKie-Mason, "Biased replacement policies for web caches: Differential quality-of-service and aggregate user value," in *4th International Web Caching Workshop*, San Diego, CA, March 1999.

[2] Luigi Rizzo and Lorenzo Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 158–170, 2000.

[3] Pei Cao and Sandy Irani, "Cost-aware www proxy caching algorithms," in *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997, pp. 193–206.

[4] Ying Lu, Avneesh Sexana, and Tarek Abdelzaher, "Differentiated caching services; a control-thoeretical approach," in *Proceedings of the 2001 International Conference on Distributed Computing Systems*, 2001, pp. 615–622.

[5] P. White, "Rsvp and integrated services in the internet: A tutorial," *IEEE Communications Magazine*, pp. 100–106, May 1997.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," in *IETF RFC 2475*, December 1998.

[7] Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan, "Proportional differentiated services: Delay differentiation and packet scheduling," in *SIGCOMM*, 1999, pp. 109–120.

[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] Tarek F. Abdelzaher and Nina Bhatti, "Web server QoS management by adaptive content delivery," in *International Workshop on Quality of Service*, London, UK, June 1999.

[10] Tarek Abdelzaher, "An automated profiling subsystem for qos-aware services," in *IEEE Real-Time Technology and Applications Symposium*, Washington, D.C., June 2000.

[11] T. Abdelzaher, K. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, June 2001.

[12] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Operating Systems Design and Implementation*, 1999, pp. 145–158.

[13] 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.

[14] Chenyang Lu, Tarek Abdelzaher, Jack Stankovic, and Sang 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] 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.

[16] Kevin Skadron, Tarek Abdelzaher, and Mircea 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.

[17] S. Keshav, "A control-theoretic approach to flow control," in *Proceedings of the conference on Communications architecture & protocols*, 1993, pp. 3–15.

[18] Nicolas Christin, Jörg Liebeherr, and Tarek F. Abdelzaher, "A quantitative assured forwarding service," in *IEEE Infocom*, NEW YORK, NY, June 2002.

[19] Ronghua Zhang, Chenyang Lu, Tarek F. Abdelzaher, and John 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.

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

[21] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole, "SWiFT: A feedback control and dynamic reconfiguration toolkit," Tech. Rep. CSE-98-009, Oregon Graduate Institute, Portland, OR, June 1998.

[22] J. Mogul, "Squeezing more bits out of http caches," *IEEE Network*, pp. 6–14, May/June 2000.

[23] Goodwin, G. C., and K. S. Sin, *Adaptive Filtering Prediction and Control*, Prentice Hall, 1984.

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

[25] R. T. Fielding and G. Kaiser, "The apache http server project," *IEEE-Internet-Computing*, vol. 1, no. 4, pp. 88–90, July 1997.

[26] Karl J. Astrom and Bjorn Wittenmark, *Adaptive Control*, chapter 2, Addison Wesley, 2nd edition, 1995.