

Adaptive Control of Multi-Tiered Web Application Using Queueing Predictor

Xue Liu, Jin Heo, Lui Sha
Dept. of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
Email: {xueliu, jinheo, lrs}@cs.uiuc.edu

Xiaoyun Zhu
Internet Systems and Storage Lab
Hewlett-Packard Laboratories
Palo Alto, CA, 94304
Email: xiaoyun.zhu@hp.com

Abstract—How to effectively allocate system resources to meet Service Level Objectives (SLOs) is a challenging problem for Web services providers. In this paper, we propose a scheme for autonomous performance control of Web applications. It uses a queueing model predictor and an online adaptive feedback loop that enforces admission control of the incoming requests to ensure the desired response time target is met. The proposed Queueing-Model-Based Adaptive Control approach combines both the modeling power of queueing theory and self-tuning power of adaptive control. Therefore, it can handle both modeling inaccuracies and load disturbances in a better way. To evaluate the proposed approach, we built a multi-tiered Web application testbed with open-source components widely used in industry. Experimental studies conducted on the testbed demonstrated the effectiveness of the proposed approach.

I. INTRODUCTION

As global E-Commerce continues to grow rapidly [1], the underlying architecture providing the service of E-Commerce is becoming more and more important. The architecture is generally referred to as “Web services”. The term “Web services” describes specific business functionality exposed by a company Web site through Internet connections, for the purpose of providing a way for another entity to use the services it provides [2]. Web services are the building blocks for the future generation of applications and solutions on the Internet.

One of the most pressing problems faced by Web services designers and service providers is how they can provide the quality-of-service (QoS) required by their clients. Current practices typically rely on offline capacity planning to statically determine the resources to be allocated to ensure the QoS. However, Web traffic is highly dynamic and volatile [3], [4]. A carefully planned configuration for a Web site may work well under a specific traffic condition, but the same configuration may make the site go haywire when workload changes. A

well-known example is the frequently referred “Slashdot effect” [5], which is named after the Web site slashdot.org. This occurs when a huge user base is referred to a previously undiscovered Web site which used to operate well. However, overwhelmed by the sudden increase in the traffic volume, the site’s performance degrades or even crashes [6]. Since Web traffic is highly dynamic, offline capacity planning techniques are often not adequate for QoS control in Web applications. Instead, the system must be smarter to react to the changing workloads in an automatic way in order to maintain the desired performance.

Recently, control theory has also been successfully applied to controlling the QoS of computing systems. Chapter 1 of [7] gives an extensive summary of related work in this area. In order to facilitate the application of traditional control theory, most of the previous work on control-theoretic approaches uses linear difference (or differential) models to represent the underlying computing systems. However, computing systems are highly nonlinear [8], [9], except in the limited case of heavy workload that allows for fluid approximations. A good model of the plant is critical to any control design. We believe that the key to success of applying control-theoretic approaches to computing systems is not to force-fit them into linear models. Rather, we should model computing systems for what they are.

During the last 40 years, research has shown that queueing models serve as a fundamental tool to model computing systems [8], [10]. In fact, queueing models have been successfully applied to areas such as capacity planning [9] and performance analysis [9] etc. However, unlike feedback control, queueing theory usually is descriptive, rather than prescriptive. So its traditional application is in capacity planning rather than QoS control. A natural question is: “Can we combine the strength of both queueing theory and feedback control

for performance management in computing systems?”

In this paper, we present Queueing-Model-Based Adaptive Control, a new approach for controlling the performance of computing systems. It utilizes the modeling (descriptive) power of queueing theory and the self-tuning (prescriptive) power of adaptive control. By using an example application of overload control for a multi-tiered Web site, we show that the proposed approach can achieve better performance regulation than previous approaches.

The remainder of this paper is organized as follows. Section II gives the background of overload control for multi-tiered Web applications. Section III provides the formal description of Queueing-Model-Based Adaptive Control approach. Section IV describes our experimental testbed and implementation. Section V shows experimental results in detail. Section VI reviews related work and contrasts our approach to previous proposals. Finally, Section VII summarizes our conclusions and offers possible directions for future work.

II. BACKGROUND

A. Multi-Tiered Web Application Architecture

Modern Web applications use a multi-tiered architecture to provide required services. While some Web applications use two tiers—Web server and database server—high volume sites typically add a third tier: application server to support complex business logic. As a consequence, most deployed Web applications utilize a 3-tiered architecture that is illustrated in Figure 1. This 3-tiered architecture provides a high level of scalability and reliability [11].

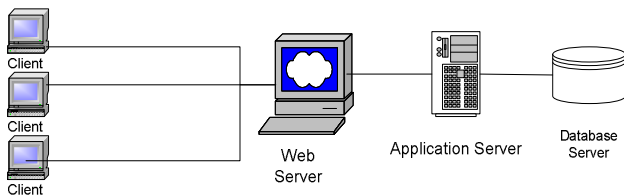


Fig. 1. 3-Tiered Web application architecture.

In the 3-tiered architecture, on the front line of a typical Web site is the Web server that acts as the presentation layer. This tier has three functionalities: (1) receives requests from the clients and service static Web requests; (2) at the same time forwards complex dynamic content requests to the 2nd tier; (3) receives responses from the 2nd tier and sends them back to the clients. Typical Web server includes Apache and Microsoft Internet Information Server (IIS).

All the business logic for a Web site resides in the 2nd tier – application server. Application server receives requests from Web server, looks up information in the database (3rd tier) and processes the information. The processed information is then passed back to the Web server where it is formatted to be displayed on clients’ machines. Typical application servers include Apache Tomcat, Sun Java System Application Server, BEA WebLogic, IBM WebSphere, and JBOSS.

The 3rd tier – database server is the storehouse of a Web site’s information. Everything from user accounts and catalogs to reports and customer orders is stored in the database. Typical database servers used in Web applications include Oracle, Microsoft SQL Server, Sybase, IBM DB2, MySQL, and PostgreSQL.

We built a testbed to emulate an online Web services provider in our Lab. The front-end is using Apache Web server [12], the application server is running Tomcat [13], and the backend database is running MySQL [14]. The reasons why we select this combination are: 1) all of them are open source projects and freely available; 2) their performances are among the highest of all individual components; 3) they are widely used on the Internet, even in commercial sites such as *Amazon.com*, *Mp3.com* and *Yahoo Finance*. Hence this combination is quite representative of the current technology. It is worth noting that the performance control approach proposed in this paper (Section III) does not depend on this specific combination and is general enough for other Web application deployments.

B. Response Time Regulation via Admission Control for an Overloaded Web Site

The QoS of a Web application is often defined as a set of criteria, referred to as Service Level Objectives (SLOs) [15], [16]. One of the most commonly used SLOs is expressed as a maximum average response time guarantee, above which is not acceptable to the clients. In fact, in Web applications, prolonged response times usually lead to lowered usage of a site, and subsequently, reduced revenues [16].

One problem frequently encountered by Web services providers is overload [17], [18], where the volume of requests for transactions at a site exceeds the site’s capacity. Overload causes longer delays to the clients or even denial of service and is a major reason for SLO violations.

Admission control [19] is an effective technique that prevents a system from overload. The idea is reducing the amount of work required when faced with over-

load by dropping a portion of the requests. This way, the server can service the accepted requests faster and meet the response time SLO. However, dropping too many requests should be prevented since this will also cause revenue loss. So the key question is, “what is the minimum portion of requests to be dropped when the Web site is overloaded in order for the accepted requests to meet their response time SLO?” An online feedback based admission control scheme is illustrated in Figure 2. A controller periodically takes performance measurements (measured delay¹ d) of the Web site from a monitoring agent, compares it with the desired performance (reference delay D^{ref}), and adjusts the admitting probability (P_a) to meet the performance goal (D^{ref}). The changes to the admitting probability can be actuated through an admission control (AC) module. For example, the AC module can be implemented through a proxy agent (Section IV). Through the AC module, a request is being accepted with probability P_a , and being dropped with probability $P_d = 1 - P_a$.

III. QUEUEING-MODEL-BASED ADAPTIVE CONTROL

A. Overview of the Queueing-Model-Based Adaptive Control Architecture

In this section, we describe the fundamental elements of the Queueing-Model-Based Adaptive Control. We will use the response time regulation problem discussed in Section II-B as a motivating application example. Figure 3 shows the proposed architecture. In our proposed architecture, there is a feed forward control loop and a feedback adaptive control loop working together to output the control command (admitting probability P_a) necessary to achieve a specified average delay target (D^{ref}).

The feed forward loop is composed of a *Queueing Model Predictor*. It takes measurements through a monitor from the computing system (Web site) to be controlled, and uses classical results from queueing theory to predict a control command (admitting probability) necessary to achieve the specified average delay target given the currently observed average workload. Let’s call the admitting probability produced by this feed forward queueing model predictor as P_a^m . Since queueing model used in the predictor serves only as an approximation of the real Web site, the performance of the Web site (measured average delay d) using the queueing model predicted admitting probability P_a^m may be off from the

targeted delay reference D^{ref} . To correct this “residual error”, we exploit an adaptive feedback loop.

The feedback control loop compares the actual delay achieved to the desired delay reference and adjusts the admitting probability accordingly in an incremental manner to ensure that the desired delay is maintained. Unlike previous approaches [20], [6], which design the controller in the feedback loop based on linearization of the queueing model, we propose to use adaptive control algorithm to design the feedback controller. This is because when the queueing model serves only as an approximation of the underlying Web site to be controlled, the further linearized model could be far off in representing the relationship between control command adjustments (ΔP_a) and the residual error corrections (Δd) well. Hence the feedback controller build on top of the linearized model may lead to poor performances in terms of correcting the residual errors. In our scheme, we propose using adaptive control to design the feedback loop. In the adaptive control design, an online estimator will first estimate an appropriate residual error model based on the measurements of inputs (control command adjustments ΔP_a) and outputs (residual errors Δd). Then the adaptive controller will produce the adjustments of admitting probability ΔP_a based on this online estimated model. The adaptive nature of the feedback loop can help to correct errors due to model inaccuracies and disturbances due to load changes using online measurements. Hence we anticipate the adaptive feedback loop will produce better control performance. A detailed discussion of how we design the adaptive feedback loop is given in Section III-C.

As we see from Figure 3, the sum of queueing model predicted P_a^m and adaptive feedback loop produced ΔP_a will be used as the real admitting probability in the admission control module for the Web site.

B. Queueing Model Predictor Design

Our abstraction for the multi-tier Web application is an M/GI/1 Processor Sharing queue (M/GI/1/PS). There are two reasons to use this simple queueing model in our design. First, modeling computing systems by a single queue is a commonly used simplification, because the performance of a computing systems is often dominated by a bottleneck stage. Hence this abstraction encapsulates the (bottleneck stage) of the multi-tier Web applications. Second, we want to conduct a fair comparison with other approaches in the performance evaluation (Section V). Since other approaches especially the one presented in [6] also uses this simple M/GI/1/PS model,

¹In this paper, we will use the term “response time” and “delay” interchangeably.

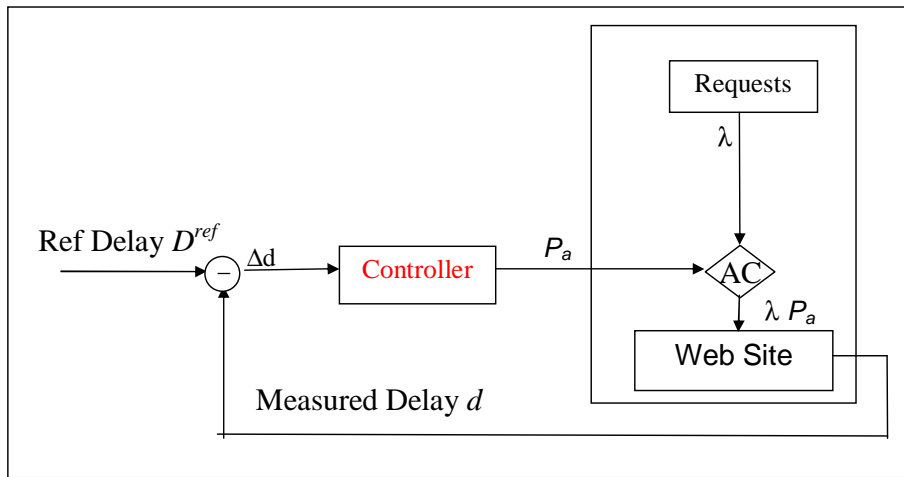


Fig. 2. Online feedback based admission control architecture.

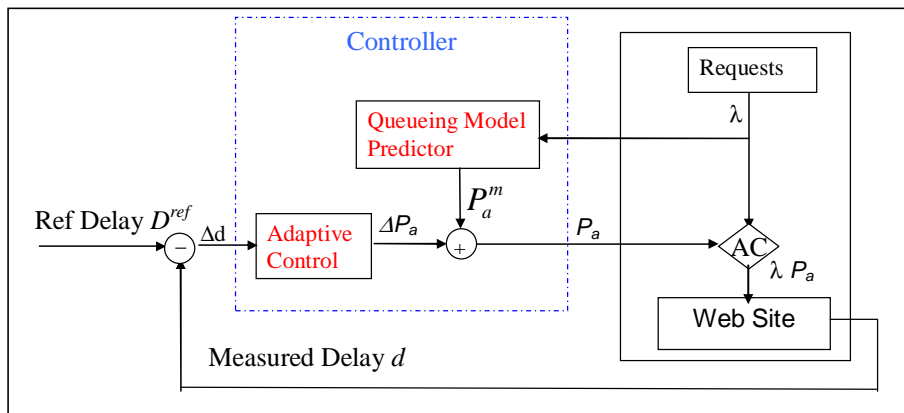


Fig. 3. Architecture of Queueing-Model-Based Adaptive Control.

we opt using it as well. In fact, we will show even with this simple model, the performance of the resulting system is very good (Section V) and out-performs previous approaches. If more accurate models such as queueing network models [21] are used, we anticipate the performance of the resulting system should be further improved. We left this topic as a future research.

We begin our queueing model predictor design by introducing some notations. We denote by $RT(x)$ the mean response time of a job whose job size (or service time) is x . The job size in an M/GI/1 system is an i.i.d. random variable, denoted by X , whose probability distribution function is $F(X)$, with a mean $E(X)$. It is well known that under processing sharing model,

$$RT_{PS}(x) = \frac{x}{1 - \rho}, \quad (1)$$

where ρ is the load of the queue. From Eq. (1), the mean response time for all jobs, is

$$RT_{PS} = \int_0^{\infty} RT_{PS}(t) dF(t) = \frac{E[X]}{1 - \rho}. \quad (2)$$

Let the request arrival rate to the Web site be $\lambda(t)$. When the arrival is modulated by the admission control module with an admission probability $P_a(t)$, the effective arrival rate to the site is $\lambda_a(t) = \lambda(t) P_a(t)$. The mean response time for the admitted requests is simply:

$$RT_{PS}(t) = \frac{E[X]}{1 - \lambda(t) P_a(t) E[X]}. \quad (3)$$

The goal of the performance control is to make the response time of admitted requests as close to the reference delay D^{ref} as possible. Suppose the queueing model is accurate, then from Eq. (3), we know by setting

$$P_a^m(t) = \frac{D^{ref} - E[X]}{\lambda(t) \cdot D^{ref} \cdot E[X]}, \quad (4)$$

we can get the steady state response time to be D^{ref} . Hence Eq. (4) gives the queueing model predicted admitting probability for the admission control.

C. Adaptive Feedback Loop Design

In this section, we present the controller design of the adaptive feedback loop. The purpose of the adaptive control loop is to correct the “residual errors” of the response time (Δd) by tuning the adjustment of admitting probability ΔP_a .

In adaptive control theory, an adaptive controller is formed by combining an online parameter estimator, which provides estimates of unknown parameters at each control instant, with a control law that is motivated from the known parameter case. The way the parameter estimator is combined with the control law gives rise to two different approaches. In the first approach, referred to as indirect adaptive control, the plant parameters are estimated online and used to calculate the controller parameters. In the second approach, referred to as direct adaptive control, the plant model is parameterized in terms of the controller parameters that are estimated directly without intermediate calculations involving plant parameter estimates [22].

In this paper, we apply direct adaptive control in the adaptive feedback loop design for its simplicity. In particular, we use the scheme presented in [23]. In order to construct the control law, the adaptive controller first needs to estimate a model for the system whose parameters can be used in the controller. In the following discussion, we describe the scheme using a general model:

$$A(q^{-1})y(k) = q^{-d}B_0(q^{-1})u(k), \quad (5)$$

where:

$$\begin{aligned} A(q^{-1}) &= 1 + a_1q^{-1} + \dots + a_nq^{-n}, \\ B_0(q^{-1}) &= b_0 + b_1q^{-1} + \dots + b_mq^{-m}, b_0 \neq 0, \end{aligned} \quad (6)$$

and $y(k)$ is the control output, $u(k)$ is the control input (command). In our admission control application, $y(k)$ corresponds to response time residual error $\Delta d(k)$, and $u(k)$ corresponds to admitting probability adjustment $\Delta P_a(k)$.

The model parameters of Eq. (5) are estimated using a Recursive Least-Squares (RLS) estimator [22], which is an online version of the well-known least-square estimator.

Let

$$\begin{aligned} \phi(k) &= [y(k), y(k-1), \dots, y(k-n+1), u(k), \\ &\quad u(k-1), \dots, u(k-m-d+1)]^T, \end{aligned} \quad (7)$$

and

$$\begin{aligned} \theta(k) &= [\theta_1(k), \dots, \theta_n(k), \theta_{n+1}(k), \\ &\quad \theta_{n+2}(k), \dots, \theta_{n+m+d}(k)]^T, \end{aligned} \quad (8)$$

then the RLS algorithm works as follows:

$$\begin{aligned} P(k-1) &= P(k-2) - [1 + \phi(k-d)^T P(k-2) \phi(k-d)]^{-1} \\ &\quad P(k-2) \phi(k-d) \phi(k-d)^T P(k-2), \end{aligned} \quad (9)$$

$$\begin{aligned} \theta(k) &= \theta(k-1) + P(k-1) \phi(k-d) [y(k) \\ &\quad - \phi(k-d)^T \theta(k-1)], \end{aligned} \quad (10)$$

Finally, the control law is given by solving:

$$\phi(k)^T \theta(k) = y^*(k+d), \quad (11)$$

where $y^*(k)$ is the reference input at time instance k . In our case, since we want to make the “residual errors” to diminish, we set $y^*(k) = 0$. The above algorithm begins with initial condition $P(-1) = p_0I$ and $p_0 > 0$.

Figure 4 illustrates the direct adaptive control scheme we in the Queueing-Model-Based Adaptive Control architecture. It replaces the dummy adaptive control loop in Figure 3.

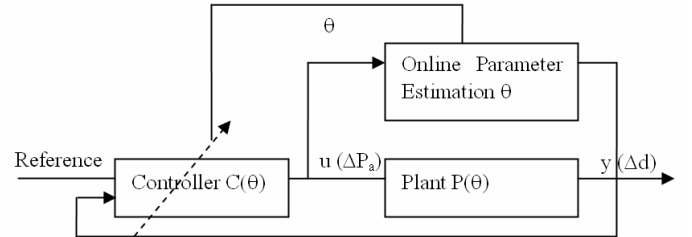


Fig. 4. Adaptive feedback loop design using direct adaptive control.

IV. EXPERIMENT SETUP AND IMPLEMENTATION

To validate the effectiveness of the proposed Queueing-Model-Based Adaptive Control, we built a testbed for response time regulation of a 3-tiered Web application using the proposed approach. In this section, we describe the testbed set-up and the hardware and software environment.

Figure 5 shows our testbed infrastructure. The testbed is composed of four machines. One of them is used as client workload generator and the other three machines are used as Web server, application server and

database server respectively. They are connected via 100 Mbps Ethernet connections. We use a proxy server for intercepting the requests and implementing the sensor, controller and actuator (Section IV-B). Though these modules can also be implemented in the Web server, we opt for using a separate proxy because this will make the implementation more modular and the modifications are non-intrusive to the Web application components. Non-intrusive approach is usually preferred by Web hosting companies because it can avoid possible bugs introduced by the new modules into the original components. The proxy server we use is very lightweight, so in our testbed, we put it on the same machine with the Web server to minimize communication overheads between the proxy and Web server.

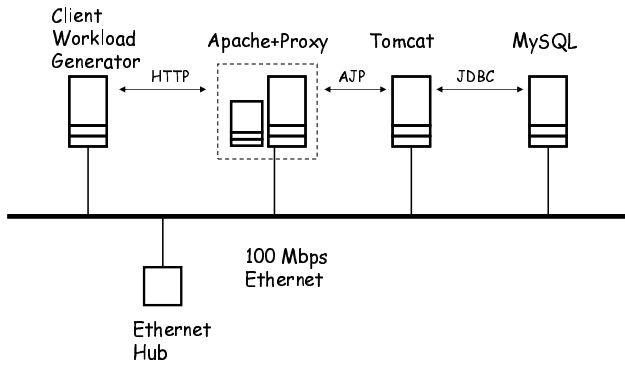


Fig. 5. Testbed infrastructure.

The client machine is equipped with a 2.8 GHZ Intel Pentium IV processor and 512MB RAM. TPC-W client emulator [24] is used as synthetic workload generator on the client (Section IV-A). The Apache Web server (and the proxy) machine has a 1GHZ Intel Pentium III processor and 256MB RAM, which runs Apache 2.0.7 [12]. The application server machine has a 1.5GHZ Intel Pentium III processor and 256MB RAM, which runs Tomcat 5.0 [13]. The database server machine has a 1GHZ Intel Pentium III processor and 512MB RAM, which runs MySQL 4.1.7 [14].

A. Client Workload Generator

We use TPC-W [24], an industry standard e-Commerce benchmarking tool for our E-commerce Web site testbed. TPC-W from the Transaction Processing Council (TPC) is a transactional Web benchmark specifically designed for evaluating e-commerce systems. Thus, it implements all functionalities that typical e-commerce Web sites provide, including multiple online browser sessions, dynamic Web page generations, database trans-

actions, authentications and authorizations. TPC-W specifies 14 unique Web interactions, which are different from each other in the sense that they require different amount of server side work. Most interactions require generation of dynamic pages and database queries, range from simple select SQL statements to complicated transactions.

We modified a Java implementation of TPC-W from the PHARM group at the University of Wisconsin [25] to make it compatible with the newest version of Tomcat and MySQL installed in our testbed. It implements all functionalities in TPC-W specification. The database is configured to contain 10,000 items and 288,000 customers. In our tests, we use 12 Web interactions, except two of the original 14 defined in the TPC-W specification, since these two are only used for administrative purpose. The 12 Web interactions are rotationally generated to be sent to the Web server.

B. Controller, sensor and actuator implementation

Tinyproxy 1.6.1 [26] is modified and used to implement the sensor (monitor), controller and actuator modules. Tinyproxy is a lightweight and fast HTTP proxy that consumes less resource than fully equipped proxies such as Squid [27]. In our testbed, all HTTP requests from clients flow through Tinyproxy and are forwarded to Apache Web server. Responses from the Web server also return back to clients through Tinyproxy. In our modifications, we first implement a sensor (monitor) module. It monitors both HTTP requests from clients and responses from the Web server. The monitor also calculates parameters such as client request rates and the performance metric – average response time of requests. These parameters and metrics are then sent to the controller module via shared memory.

The controller module implements the main algorithms of Queueing-Model-Based Adaptive Control. It is composed of two parts: The Queueing Model Predictor takes the measured client request rates from the sensor module and produces the model predicted admitting probability P_a ; The Adaptive Feedback Controller takes the measured average response time as input and produces the admitting probability adjustment ΔP_a . Then the combined dropping probability $P_d = 1 - (P_a + \Delta P_a)$ is sent to the actuator. In our implementation, we confine P_a , ΔP_a to be within range (0.1, 1.0) and P_d to be within range (0.0, 0.9).

The actuator is implemented in the following way within Tinyproxy. It randomly drops a request from

clients based on the dropping probability P_d got from the controller module.

C. Determination of other parameters

The mean service time $E[X]$ to be used in the queueing model predictor is measured offline. To this end, a very light workload is applied to the Web site. Under the light workload, since there is no queueing delay the measured mean response time approximates the mean service time $E[X]$. To remove possible measurement noises, we conducted the measurement test 5 times and averaged the measured response time to get $E[X] = 0.035$ sec.

In order to implement the adaptive controller, we need to determine the model orders d, m and n . Due to the digital implementation of the feedback controller, the effect of control command determined on time interval k can only be measured in interval $k+1$, so we set delay order $d = 1$. For computing systems, the model orders n and m are usually fixed and low [7] and can be estimated offline. In practice, we use the following method to determine them. We first disable the adaptive feedback controller and collect offline data Δd by making use only the queueing model predictor and a white noise input of ΔP_a . Under different combinations of n and m , we use model identification method – least square estimator to find the corresponding model parameters θ , then we test which θ got from these combinations gives good fitting. By “good fitting”, we mean using the θ , a new data group of collected data pairs $\{(\Delta P_a, \Delta d)\}$ produces high r^2 value [28]. In our system, we find $n = 1$ and $m = 0$ give relatively good fitting.

V. EXPERIMENTAL RESULTS

In this section we present detailed experimental results to show the effectiveness of Queueing-Model-Based Adaptive Control approach. To this end, we compare the performance of four different approaches with each other. They are:

- (1) Queueing model only;
- (2) Adaptive control only (approach proposed in [29] and [30]);
- (3) Queueing model + PI control (approach proposed in [20] and [6]);
- (4) Queueing model + Adaptive control (approach proposed in this paper).

Among them, approach (2) is proposed in work by Lu et al. [29] and Karlsson et al. [30] which only uses an adaptive control loop. Approach (3) is proposed by

Sha et al. [20] and [6] which uses both a queueing predictor and a PI control loop. Approach (4) is the Queueing-Model-Based Adaptive Control proposed in this paper. The comparisons are based on the measurements obtained from the testbed. In all the experiments, the reference delay was set to 0.10 sec. The length of control interval is set to 3 sec. At each control period, the average request arrival rate and average response time of that interval are measured. These values are used in the controllers’ calculations for different approaches to produce the corresponding dropping probabilities P_d . The resulting dropping probability is then set in Tinyproxy’s actuator module to take into actuation.

Two sets of workloads were used in our tests. The first set, Workload A (WLA) is a simple workload which has exponentially distributed inter-arrival time with mean 0.025 sec. The second set, Workload B (WLB) is a more complicated workload which is changing. From time $t = 0$ sec to $t = 150$ sec, WLB is the same as WLA. At time $t = 250$ sec, a second workload with the same mean inter-arrival time (0.025 sec) joins in, which makes the total request rate twice as much as that of WLA.

A. Comparisons of Different Approaches under Workload A

First, we tested the four approaches under workload A. Each test runs for 300 seconds. In order for the readers to better distinguish between the result graphs of different approaches, we plot the response time measurements under workload A in two figures. Figure 6 compares the response time measurements of Queueing model only, Adaptive control only and Queueing model + Adaptive control. Figure 7 compares the response time measurements of Queueing model + PI control versus Queueing model + Adaptive control. In these figures, each point shows the averaged response time of all the requests during the past 3 seconds. 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. Table I summarizes the results.
- Using Queueing model only approach, a large steady state error develops (Figure 6). This is attributed to the fact that the multi-tiered Web application is not exactly a single queue system. Thus, the model derived in this paper is only an

TABLE I
THE AGGREGATE ERRORS UNDER WORKLOAD A

Workload A	Queueing model only	Adaptive control only	Queueing model + PI control	Queueing model + Adaptive control
Aggregate error	0.474	0.233	0.218	0.181

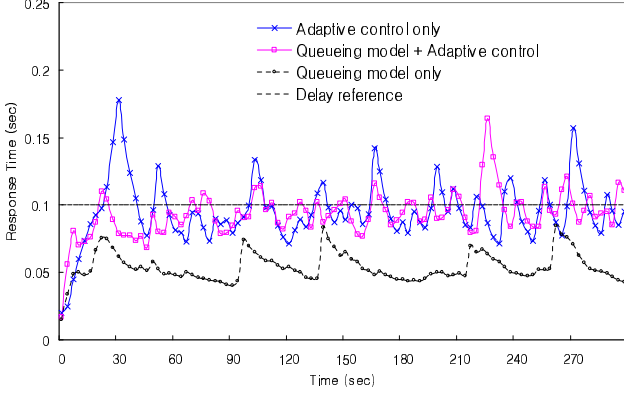


Fig. 6. Performance of Queueing model only, Adaptive control only and Queueing model + Adaptive control approaches under workload A.

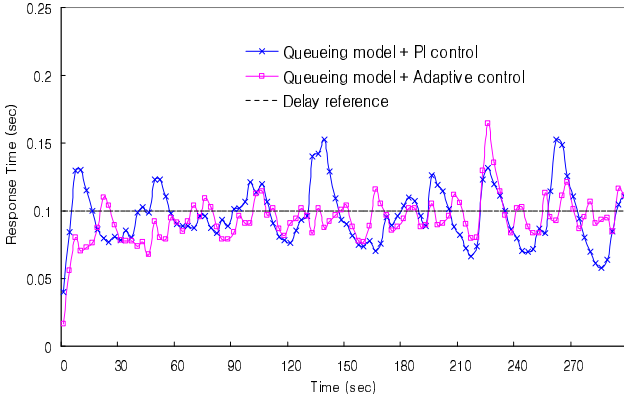


Fig. 7. Performance of Queueing model + PI control and Queueing model + Adaptive control approaches under workload A.

approximation of the true Web site. The fluctuation is also quite large.

- Queueing model + PI control approach's performance is a little better than that of Adaptive control only approach. This is attributed to the fact that Adaptive control uses linear model to approximate the real system, the online estimator need to run for some time before settling near the true value. However, in the case of Queueing model + PI control approach, the queueing model predictor is an approximation of true Web site, it can help jump to the vicinity of the true control value faster. And

the PI controller will help to reduce the residual errors.

- The proposed Queueing-Model-Based Adaptive Control out-performs other approaches. This is because first, the queueing model predictor is able to supply an approximate control value that achieves a response time close to the set point. Secondly, in terms of correcting the residual errors, the PI control loop is based on the linearization of the approximated queueing model. On the other hand, adaptive feedback loop can find better control value adjustments based on online measurements and adaptation.

B. Comparisons of Different Approaches under Workload B

Now we present the experiments under changing workload (WLB). Since approaches using queueing model predictor give better results, we will only show the comparison of Queueing model + PI control (approach 3) and Queueing model + Adaptive control (approach 4). Figure 8 compares the response time measurements of these two approaches. Table V-B shows the aggregate errors under workload B.

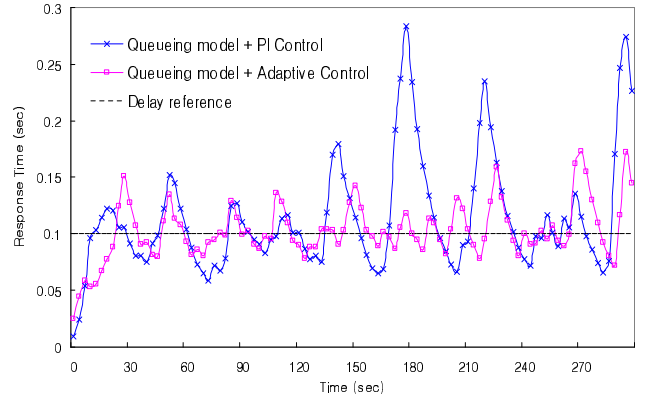


Fig. 8. Performance of Queueing model + PI control and Queueing model + Adaptive control approaches under workload B.

As we can see from both Figure 8 and Table V-B, the proposed Queueing-Model-Based Adaptive Control approach has smaller error and achieves better response time regulation.

TABLE II
THE AGGREGATE ERRORS UNDER WORKLOAD B

Workload B	Queueing model + PI control	Queueing model + Adaptive control
Aggregate error	0.521	0.252

Our experimental evaluation demonstrates the advantages of integrating a queueing model predictor with an adaptive feedback loop to achieve performance guarantees in Web applications. The two components have complementary strengths, jointly offering more robust tracking of performance set points in the presence of widely unpredictable load.

While queueing theory and adaptive control theory 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 computing systems.

VI. RELATED WORK

Much related work has been done in the areas of Quality of Service (QoS) control for computing systems. Due to space limitations, in this section, we only provide a brief overview of the research most related to our work, i.e. control-theoretic approaches.

Feedback control theory was invented more than 50 years ago; since then many powerful results and tools have been obtained and deployed. Recently, Control theory has been successfully applied to controlling the performance for computing systems including Web application systems. In [31], Abdelzaher *et al.* build a PI (Proportional-Integration) control loop for Apache Web server that enforces desired relative delays among different service classes via dynamic connection scheduling and process reallocation. In [32], a similar approach is used for Squid proxy server to guarantee cache hit-ratio by dynamically adjusting the disk space allocation. In [33], the parameters (i.e. KeepAlive time, MaxClients) of an Apache Web server are dynamically allocated using a MIMO feedback controller. The goal is to keep the system's CPU and memory utilization stabilized at a desired reference value. A similar approach is also used in Lotus Notes Server [34]. These approaches view the server system as a transfer function/state space model and use linear feedback control schemes.

However, due to the fundamental difference between a

mechanical system and computing system, transfer function/state space models cannot be built directly to reflect the internal dynamics of the server studied. Instead, these models are usually constructed offline using black-box approaches such as model identification [35] under certain predefined workloads. Due to the stochastic nature of the Web traffic with temporally and spatially variations, models thus obtained are not accurate. Furthermore, since computing systems are highly nonlinear [8], these models are at best a linear approximation of the real system. This leads to the problem of poor robustness when directly applying classical control theory to controlling computing system's performance, especially in presence of dynamic traffic loads. To solve this problem, researchers [28], [29], [30] propose using adaptive control techniques for controlling the performance of computing systems. The introduction of adaptive controller helps adjusting the model to the traffic dynamics to some extent. However, the model and control are still intrinsically linear; and the adaptive controller usually responses slowly to abrupt traffic changes.

In [20] Sha *et al.* propose queueing model based feedback control in Web servers. It combines a queueing model predictor and a PI controller to achieve response time regulation. Recently, Kamra *et al.* [6] uses the similar approach for overload control in a 3-tiered Web sites. Though combing the power of queueing model and feedback control achieves better performance for the underlying systems, the design of the feedback loop in these proposals is based on linearization of the queueing models around the operating point. Since the queueing models applied are only approximations to the original system, the so obtained linearized residual error model may be far off from the system residual dynamics. Hence the performance of the system is undermined.

In this paper, we solve the above problem by using adaptive controller to design the feedback loop working together with the queueing model. The adaptive feedback loop can build the residual error model and corresponding controller based on online measurements of the system. It can handle inaccuracies in the queueing model and changes in workload in a dynamic fashion by adjusting the controllers accordingly. Together with the queueing model predictor, the proposed Queueing-Model-Based Adaptive Control scheme provides a better performance regulation under a wide range of workloads and conditions.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we propose Queueing-Model-Based Adaptive Control approach for performance regulation of computing systems. In this approach, the queueing model predictor and adaptive feedback controller operate concurrently in a complementary manner to make the performance of the system meet the desired target. The feed forward queueing model predictor controls the system state near an equilibrium operation point, in spite of changes in the arrival process. The adaptive controller corrects errors due to the inaccuracies in the queueing model and disturbances by using online estimation and adaptation. To evaluate the effectiveness of the proposed approach, we build an *example* multi-tiered Web application testbed with open-source components widely used in industry. We also implemented Queueing-Model-Based Adaptive Control in the testbed. Experimental studies show it achieves better response time regulation than previous proposals.

We have identified several avenues for future work. First, while the M/GI/1/PS model used in the queueing model predictor works well in our multi-tiered Web application, we plan to investigate if more sophisticated models such as queueing network model can help to improve the performance. Secondly, we plan to conduct studies using other adaptive schemes in designing the feedback loop to see if they work better with the queueing model predictor than the currently deployed direct adaptive controller. Finally, it is interesting to see the application of the proposed approach in other general computing systems.

ACKNOWLEDGEMENT

The authors would like to thank Prof. Tarek Abdelzaher and Ying Lu for the discussion on the adaptive control algorithm.

REFERENCES

- [1] census.gov, "Quarterly retail e-commerce sales, 3rd quarter 2004," 2004.
- [2] UDDI, "Universal description, discovery, and integration of business for the web," 2004.
- [3] J. C. Mogul, "Operating systems support for busy internet servers," in *Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, Orcas Island, WA, 1995.
- [4] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Measurement and Modeling of Computer Systems*, 1998, pp. 151–160.
- [5] S. Adler, "The slashdot effect, an analysis of three internet publications," *Linux Gazette*, 1999.

- [6] A. Kamra, V. Misra, and E. Nahum, "Yaksha: A controller for managing the performance of 3-tiered websites," in *The Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004)*, Toronto, Canada, 2004.
- [7] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.
- [8] L. Kleinrock, *Queueing Systems, Vol. 2, Applications*. John Wiley, 1976.
- [9] D. Menasce and V. Almeida, *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall PTR, 2001.
- [10] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, Eds., *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.
- [11] D. K. Barry, *Web Services and Service-Oriented Architectures: The Savvy Manager's Guide*. Morgan Kaufmann, 2003.
- [12] The Apache Group, "Apache http server project," 2004.
- [13] The Apache Jakarta Project Group, "Tomcat," 2005.
- [14] M. AB, "Mysql," 2005.
- [15] L. Jin, V. Machiraju, and A. Sahai, "Analysis on service level agreement of web services," HP Labs, Tech. Rep. HPL-2002-180, 2002.
- [16] B. J. Hill and K. Thomson, "System performance management: Moving from chaos to value," Sun Microsystems, Tech. Rep., 2001.
- [17] H. Chen and P. Mohapatra, "Session-based overload control in qos-aware web servers," in *IEEE INFOCOM*, 2002.
- [18] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, 2003.
- [19] L. Cherkasova and P. Phaal, "Session-based admission control: A mechanism for peak load management of commercial web sites," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 669–685, 2002.
- [20] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *23rd IEEE Real-Time Systems Symposium (RTSS02)*. IEEE Computer Society, 2002, p. 81.
- [21] X. Liu, J. Heo, and L. Sha, "Modeling 3-tiered web applications," in *13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, Atlanta, Georgia, 2005.
- [22] K. J. Astrom and B. Wittenmark, *Adaptive Control (2nd Edition)*. Prentice Hall, 1994.
- [23] G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*. Prentice-Hall, 1984.
- [24] The Transaction Processing Council, "Tpc-w," 2002.
- [25] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, "Characterizing a java implementation of tpc-w," in *The 3rd Workshop On Computer Architecture Evaluation Using Commercial Workloads (CAECW)*. Toulouse, France: <http://www.ece.wisc.edu/pharm/tpcw.shtml>, 2000.
- [26] R. J. Kaes and S. Young, "tinyproxy," 2005.
- [27] A. Chadd, R. Collins, H. Nordstrom, A. Rousskov, and D. Wesels, "Squid web proxy cache," 2005.
- [28] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control to resource containers on shared servers," in *9th IFIP/IEEE International Symposium on Integrated Network Management (IM 2005)*, Nice, France, 2005.

- [29] Y. Lu, T. F. Abdelzaher, and G. Tao, "Direct adaptive control of a web cache system," in *American Control Conference*, Denver, CO, 2003.
- [30] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," in *The Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004)*, 2004.
- [31] T. F. Abdelzaher and C. Lu, "Modeling and performance control of internet servers," in *39th IEEE Conference on Decision and Control*, Sydney, Australia, 2000.
- [32] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated caching services: A control-theoretical approach," in *International Conference on Distributed Computing Systems*, Phoenix, Arizona, 2001.
- [33] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Mimo control of an apache web server: Modeling and controller design," in *American Control Conference*, 2002.
- [34] N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Managing the performance of lotus notes: A control theoretic approach," in *the Computer Measurement Group*, 2001.
- [35] L. Ljung, *System Identification: Theory for the User (2nd Edition)*. Prentice Hall PTR, 1998.