Proceedings of the 39th IEEE
Conference on Decision and Control
Sydney, Australia • December, 2000

Modeling and Performance Control of Internet Servers

Tarek F. Abdelzaher and Chenyang Lu
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
{zaher,chenyang}@cs.virginia.edu

Abstract

The paper describes modeling and performance control of an Internet server using classical feedback control theory. We show that classical feedback control can leverage on well-known real-time scheduling results to resolve one of the fundamental problems in Internet-servers today; namely, achieving overload protection and performance guarantees in the presence of load unpredictability. The research is motivated by the increasing proliferation of a new category of Web-based services, such as online trading, banking, and business transactions, where performance guarantees are required in the face of unpredictable server load. Failure to meet desired performance levels may result in loss of customers, financial damage or liability violations. State-of-the-art Web servers are not designed to offer such performance guarantees.

We show that control theory offers a robust solution to the server performance control problem. We demonstrate that a general web server may be modeled as a linear time-varying system, describe the equivalents of sensors and actuators in that system, formulate a simple feedback loop, describe how it can leverage on real-time scheduling theory to achieve timing guarantees, and evaluate the efficacy of the scheme on an experimental testbed using a real web server (Apache), which is the most popular Internet server today. Experimental results indicate that control-theoretical techniques offer a promising way of achieving desired performance in emerging critical Internet applications.

1 Introduction

Current web servers treat all clients alike in a first-come first-served fashion. As a result, individual performance guarantees cannot be given to individual clients (or client categories). When the server is overloaded, all clients suffer unacceptable delays and connection failures even although enough resources may exist on the server to serve a *subset* of clients efficiently. Computing the size of that subset is a challenge since the load imposed by a single client is unpredictable.

The traditional approach to achieving performance guarantees in computing systems has been to quantify hardware speed and software execution requirements, then apply an appropriate combination of pre-run-time analysis, admission control, and resource allocation algorithms to ensure that the system is not overloaded and that the desired performance is achieved [12]. The approach is open loop in nature. It works well for embedded computing systems, such as avionics and factory automation, in which an accurate model exists for the system *a priori*.

The web has traditionally been considered a non-critical application for which no performance guarantees are required. Recently, we have seen a rapid emergence of a category of applications that require performance guarantees but execute in unpredictable general-purpose environments, such as the Internet. Examples include online trading, e-commerce, and real-time databases. Failure to meet performance specifications may result in loss of customers, financial damage, or liability violations. Existing approaches for designing performance-guaranteed computing systems rely on *a priori* workload and resource knowledge and are therefore inapplicable.

In this paper we show that classical feedback control offers a solution to the problem of achieving performance guarantees for this new category of applications and discuss the challenges involved in realizing this approach. We demonstrate that a web server can be approximated by a time-varying linear model, describe the needed software actuators and sensors, and cast server performance control as a classical feedback control problem. Experimental results derived from testing the scheme on a widely used web server show that our approach is feasible. The computing system is amenable to analysis using feedback control theory. This analysis offers a new mathematical foundation for

0-7803-6638-7/00$10.00 © 2000 IEEE

providing service performance guarantees.

Although our approach can be applied to other services as well, such as multimedia, we have chosen to focus on controlling performance of web servers due to the increasing importance of this application, spurred by the phenomenal growth of the Internet. The web is the largest and most visible client-server application in existence today. It is therefore a perfect vehicle for investigating fundamental problems of distributed client-server computing such as that of overload protection and providing performance guarantees. Web servers are used for a variety of day-to-day functions of various importance levels, including online shopping, trading, banking, and recreation, making it desirable to provide different classes of service to meet the performance demands of different applications.

Support for different classes of (quality of) service on the web has been investigated in recent literature. An important measure of quality is the responsiveness of the server. In the simplest case it is desired to differentiate between two classes of clients; *premium* and *basic*, such that premium clients receive better service at overload. For example, in [13] an architecture is proposed and evaluated where restrictions are put on the amount of server resources (such as threads) available to basic clients. In [6, 1] admission control and scheduling algorithms are investigated to provide premium clients with better service. In [9] a server architecture is proposed that maintains separate service queues for premium and basic clients allowing them to be handled separately in an appropriate order.

While the above architectures offer better service to premium clients, no individual performance guarantees are given to served requests. This may not be acceptable to critical online applications such as stock-trading. Our work [4, 5, 2, 14] differs from other approaches in that we offer performance guarantees. Server response time, i.e., the time it takes to serve a client's request, is probably the most visible performance metric from the client's perspective. Thus, in this paper, we consider guarantees on server response time. Real-time (deadline-monotonic) scheduling theory [7] makes response-time guarantees possible if server utilization is maintained below a pre-computed bound. In the absence of exact knowledge of per-client load, utilization can be maintained around the bound via feedback control to enforce the specified response-time bounds.

In the rest of this paper we detail our performance control architecture and describe issues in applying control-theoretical techniques to a computing server. Section 2 describes the computing system being controlled. Section 3 introduces the control problem, the issues involved with sensors and actuators, their modeling, and closing the feedback control loop around a popular web server. Section 4 evaluates the performance of the resulting feedback loop on an experimental testbed. Section 5 concludes the paper with a summary of contributions and suggests avenues for future work.

2 The Computing Server

We consider a distributed client-server system in which clients send a succession of requests to the web server across a communication network. Each service request has an associated soft deadline by which it must be served (perhaps determined from the importance, preferences, or subscription fees of the client). The delay seen by the client includes the time a request spends in the network and the time it spends on the server. Research in the networking community addressed the problem of bounding network delays, as in diff-serv [10] and int-serv [11] architectures. We address the complementary problem of bounding the *server-side* delay. With the dramatic increase in the number of Internet clients, servers are becoming potentially significant bottleneck points making this problem particularly important.

Serving a request consumes different server resources, such as memory, disk bandwidth, communication bandwidth, and processing cycles. The capacity of the server is limited by that of the bottleneck resource. We assume that the ith request has a deadline D_i and consumes an amount C_i of the bottleneck resource. Since each request must be executed by the deadline, the utilization of the bottleneck resource is $U = \sum_i C_i/D_i$, where the summation is carried out over all current requests. A request is *current* if it has arrived but its deadline has not yet expired.

3 The Control Problem

The objective of our performance control loop is to (i) avoid server overload, and (ii) meet the individual deadlines of all served requests. A recent result in deadline-monotonic scheduling theory [3] states that a group of aperiodic tasks with individual deadlines, scheduled in order of increasing response-time bounds, will always meet their deadline constraints as long as $U < 5/8$. This result leads to a simple implementation of a server

that guarantees meeting all request deadlines; the server simply needs to ensure that its utilization does not exceed the aforementioned bound. Since it is also desired to maximize server throughput, the utilization should be maintained exactly at the upper bound, if possible. The key difficulty in implementing an algorithm that would observe the utilization bound is the lack of proper estimates of resource load imposed by an individual client.

A classical linear feedback control problem can be formulated to keep server utilization at $5/8$. Server load can be approximated by a liquid flow model. Since a typical server can serve thousands of clients concurrently, each client contributes an unknown, but small, amount to this flow, represented by a series of requests. The control loop, depicted in Figure 1, measures server utilization and determines (based on load conditions) a subset of clients that may receive service at the current time. The size of this subset is adjusted to keep the utilization at the desired level. The three main challenges in implementing the control loop are (i) the choice of a proper actuator that can affect server utilization, (ii) the choice of a monitor that can measure current utilization reliably, and (iii) appropriate modeling of the server, actuator and monitor. The above three items are elaborated upon below. Section 3.1 describes issues in designing the actuator. Section 3.2 describes the monitor. Section 3.3 discusses system modeling for the purposes of feedback control and presents the procedure we used for controller tuning.

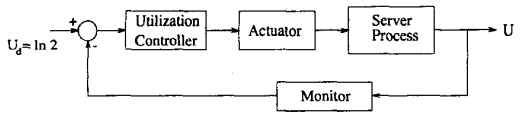


Figure 1: The Utilization Control Loop

3.1 The Actuator

A fundamental problem in applying control to a computing server is an appropriate choice of the actuator. In general, *admission control* can be used as an actuator in computing servers. In the simplest case, admission control limits the number of clients who access the server concurrently by denying service to some of them. An algorithm decides, on a request-per-request basis, whether service should be granted (request admitted) or denied (request rejected), depending on client identity. If the client is not admitted, a short message may be returned to the client informing them that the service is temporarily unavailable. Client rejection is acceptable because no scheme can provide guarantees on admission to all web clients, unless one limits the total number of clients on the Internet to the maximum number than can be served by a single server concurrently. The challenge a control loop accomplishes is that of providing performance guarantees to the clients whom the "actuator" chooses to admit.

In general, the actuator can also offer intermediate "degraded" service levels. In a web server, a degraded level of service can correspond to providing an abbreviated version of content, or a lower quality version of images and sound. The degraded content is prepared *a priori*. Such a scheme is described in [5]. Content degradation has also been discussed at length in multimedia applications. A good survey is found in [8].

Consider a general actuator with M discrete service levels. These levels are numbered $1, ..., M$ from lowest quality to highest quality. The level 0 is added to denote the special case of request rejection. The actuator accepts as input the control variable m in the range $[0, M]$. The manipulated variables are the fractions of clients served at each service level. An essential problem in actuator design is that of unique mapping from m to these manipulated variables. One option is suggested below.

In general, m is a fractional number composed of an integral part I and a fraction F, such that $m = I + F$. We let the two nearest integers to m (namely, I and $I + 1$, where $I < m < I + 1$) determine the two most appropriate service levels at which clients must be served under the given load conditions. The fractional part F determines the fraction of clients served at each of the two levels. In effect, m is interpreted to mean that a fraction $1 - F$ of clients must be served at level I, and a fraction F at level $I + 1$. This specification constrains only the total number of clients to be served at each level without dictating *which* clients should be chosen. The latter is a separate policy that may operate within the confines of the former.

Server utilization will increase (possibly nonlinearly) when m is increased and vice versa. At the upper extreme, $m = M$, all requests are given highest quality service. At the lower extreme, $m = 0$, all requests are rejected. The actuator changes the amount of load on a server with discrete service levels in a continuous fashion, depending on its input m.

3.2 The Monitor

Most operating systems provide means for monitoring server resource utilization, such as percentage of consumed CPU, disk and network bandwidth. Unfortunately, utilization measurements in practical systems tend to be extremely noisy. One option would be to use a filter to smooth these measurements. The filter will introduce an additional lag that may reduce the tightness of utilization control. A problem we encountered was obtaining "good" utilization measurements.

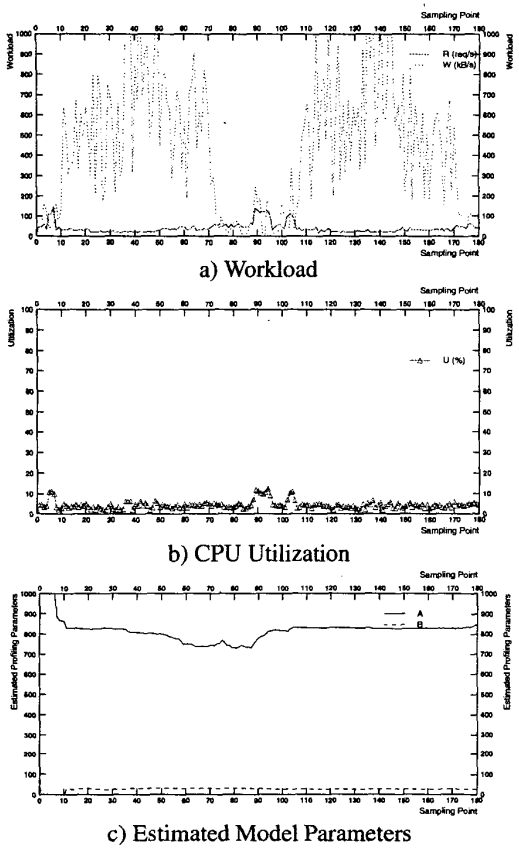
In the common case where the web server serves mostly static files (web pages), one can express server utilization U as a function of the served request rate R and delivered byte bandwidth W. Both of these variables can be measured very accurately in a computing server. In [5], we prove that:

$$U = aR + bW \quad (1)$$

where a and b are constants which can be computed by a linear regression. The intuition behind Eq. (1) comes from the fact that the resource requirements C_i of serving a file are composed from a fixed overhead plus a variable overhead that depends on the length x_i of the file. Thus, $C_i = a + bx_i$, where a and b are constants that depend only on the type of platform used. Aggregating resource consumption over multiple requests and averaging over time we arrive at Eq. (1). The expression gives a noise-free utilization estimate which we use for feedback in the control loop of Figure 1. If only a fraction f of requests are admitted, server utilization is given by:

$$U = aRf + bW + cR(1 - f) \quad (2)$$

where c is the cost of rejecting one request. In Eq. (2), R is the total request rate received by the server (including requests that will be rejected), f is the fraction of admitted requests, and W is the total byte rate sent by the server. An interesting challenge is to compute parameters a, b, and c, and determine their sensitivity to variations in the operating conditions of the computing system. In a preliminary investigation, we compute c experimentally by instrumenting the server to reject all requests and obtaining the reciprocal of the maximum attained rejection rate. This results in $c = 0.55ms$. To estimate a and b, we fit the web server with a recursive least squares estimator that measures R, W, and U and infers the coefficients of Eq. (1). In [2], we present an evaluation of this estimator; a and b are computed using data collected in real-time on an Apache web server subjected to a varying request rate. Figure 2 illustrates



a) Workload

b) CPU Utilization

c) Estimated Model Parameters

Figure 2: Online Estimation of Apache Server Model, Low Load

the conversion of the resulting estimates in a representative experiment. Figure 2-a shows the workload that we applied to the server for parameter estimation purposes. In particular, it depicts the request rate R on the server and the resulting bandwidth W delivered as a function of time during the experiment. Requests for short web pages where interleaved with those for long pages to offer load points with different proportion of R to W. The horizontal axis depicts the sampling count (one sample of R, W, and U was taken every 3 seconds). Figure 2-b depicts the measured CPU utilization, U, during this experiment due to the applied workload. Figure 2-c depicts the output of the least squares estimator. In can be seen that the estimator converges to the values $a = 830\mu s$, and $b = 33\mu s/kB$. Thus, from Eq. (2), $U = 0.83Rf + 0.035W + 0.55R(1 - f)$, where R is in $requsts/ms$ and W is in kB/ms. We call the above procedure "profiling". Architectures for automated profiling are needed for performance-guaranteed services

to facilitate monitor design. During normal operation the monitor will use results of profiling to estimate the measured variable periodically at some period, T.

3.3 The Controller and System Model

The choice of controller may be an interesting problem in computing systems. As a first step towards system utilization control, we use a digital approximation of a linear continuous PI controller given by the equation $G(s) = K_p(1 + K_i/s)$. A PID is not used due to the noisy nature of the process which may be amplified by a derivative action or lead controllers. The controlled software system (including the web server, actuator, and monitor) is modeled by a transfer function $P(s)$. Modeling introduces a plethora of future challenges that we summarize in the conclusions of this paper. In the simplest case, we assume that $P(s)$ is a static gain, p (i.e., process dynamics are negligible). The gain is linearized by obtaining the derivative of the output (utilization) with respect to the input, m. Thus, $p = dU/dm$ around the operating point (which typically is $U = 5/8$). Intuitively, the maximum gain, p_{max}, occurs when the range of m is minimum for the same range in output U, i.e., when the actuator supports only client rejection, but no intermediate service levels. In this case, $m = f$, the fraction of admitted clients, and $p_{max} = dU/df$. Let U_{accept} be the utilization that would result if all clients were admitted. If presently a fraction f of the clients is admitted, the actual utilization is $U = U_{accept}f + c(1-f)R$. Differentiating with respect to f, we obtain $p_{max} = U_{accept} - cR$. It may be more convenient to express the gain in terms of actual current utilization, U. Algebraic manipulation yields:

$$p_{max} = (U - cR)/f \tag{3}$$

Designing the controller for the maximum process gain will guarantee stability for all other gain values. While the server has additional dynamics, they are of the order of milliseconds and thus will be neglected given the range of human perception. In addition to the gain, digital sampling introduces an effective dead-time of half the sampling period, $T_d = T/2$. Under the above general assumptions, the simplified transfer function of the computing system, $P(s)$, is given by the Laplace transform:

$$P(s) = p_{max}.e^{-sT/2} \tag{4}$$

The natural frequency of oscillation of this loop, w, is obtained by setting $s = jw$ in the poles of the closed loop transfer function, where j is the imaginary unit vector. This yields:

$$p_{max}e^{-jwT/2}K_p(1 + K_i/jw) = -1 \tag{5}$$

For stable control, a gain margin G is specified as a design parameter. From Eq. (5), the closed loop gain and phase at the natural frequency of oscillation are:

$$p_{max}|e^{jwT/2}|K_p|1 + K_i/jw| = 1/G \tag{6}$$

$$wT/2 + tan^{-1}(K_i/w) = \pi \tag{7}$$

Incidentally, if the damped frequency of oscillation is not far from w, successive peaks of the closed loop oscillations will have a ratio of approximately $1/G^2$. It is a common practice in industrial PI controller tuning to set the controller phase to $-\pi/6$ [16]. In this case:

$$tan^{-1}(K_i/w) = \pi/6, \tag{8}$$

Eq. (6), Eq. (7), and Eq. (8) are three equations in three unknowns, K_p, K_i, and w, two computed model parameters, p_{max} and T, and one design parameter, namely the gain margin G. They can be solved for K_p and K_i for a particular gain margin to achieve a specified transient response. The system is time-varying. Controller settings depend on process gain which changes with the incoming request rate, utilization, and fraction of admitted requests, as seen from Eq. (3). For purposes of controller tuning, utilization in Eq. (3) can be substituted by its set point value.

4 Experimental Evaluation

We now evaluate controlling server utilization using the loop described in the previous section. We consider an Apache web server on the Linux operating system. The combination of Apache over Linux is representative of many web server configurations today. The experimental server platform was an AMD-based PC connected via a local area network to client machines. Several machines were used to run client software that tests the server with a synthetic workload. We used a web-load generator, called httperf [15], on the client machines to bombard the server with web requests. The server code was modified to implement the discussed control loop.

Figure 3 depicts the achieved utilization. In this experiment, the request rate on the server was increased suddenly, at $time = 0$, from zero to a rate that overloads the server. Such a sudden load change approximates a step function. It is more difficult to control than small incremental changes, thereby stress-testing the responsiveness of our control loop. The target utilization, U_t,

was chosen to be just under $5/8$. The actuator was using admission control. Figure 3 compares the open loop server utilization to the closed loop utilization (with gain margin values of $G = 4$ and $G = 10$). As shown in Figure 3, the controller was successful in reducing server utilization to remain successfully around the target for the duration of the experiment. Utilization control was achieved by admitting the "right" number of clients. One can observe that the exponential decay in tracking error follows a damped profile that converges quickly to the set point. A zero steady state error is achieved.

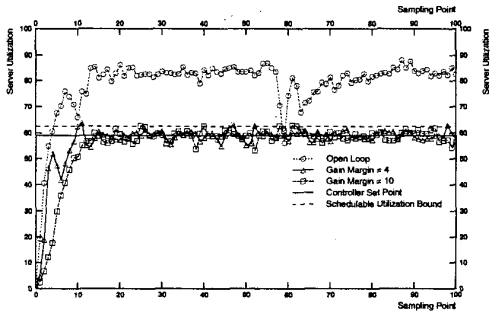


Figure 3: Utilization Control Performance

With server utilization steady ar or below $5/8$, we could easily verify by instrumenting the client software that individual request deadlines were met. At the expense of rejecting excess clients, all admitted clients received their requested pages within their respective timing constraints.

5 Conclusions and Future Work

In this paper, we demonstrated the application of control theory to Internet server performance control. All measurements in this paper were done on an experimental platform running the popular Apache web server which we have modified to incorporate the suggested control loop. There are several outstanding issues that the authors hope to address in future interdisciplinary collaborations. For example, how to model non-linearities peculiar to computing systems? How can these nonlinearities be accounted for in control? How efficient are adaptive control and robust control techniques in dealing with parameter variation and load uncertainty? Can automatic identification and estimation techniques be applied to model servers, software sensors, and actuators? How to implement control-theoretical resource management in the operating system? Examples, theoretical foundations, experimental evidence, and practical experience are needed in applying feedback performance control to different computing systems. This is an important focus of our research group at the present time.

References

[1] T. Abdelzaher and K. G. Shin. Qos provisioning with *q*contracts in web and multimedia servers. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

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

[3] T. F. Abdelzaher. A schedulable utilization bound for aperiodic tasks. Technical Report CS-2000-21, University of Virginia, Charlottesville, VA, August 2000.

[4] T. F. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. In *International World Wide Web Conference*, Toronto, Canada, May 1999.

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

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

[7] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Deadline monotonic scheduling theory. In *Proceedings of 18th IFAC Workshop on Real Time Programming*, pages 55–60, Bruges, Belgium, June 1992.

[8] C. Aurrecoechea, A. Cambell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems Journal*, 6(3):138–151, May 1998.

[9] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, September 1999.

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

[11] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: An overview. *Request for Comments RFC 1633*, July 1994. Xerox PARC.

[12] G. C. Buttazzo and J. A. Stankovic. *Hard Real-Time Computing Systems*. Kluwer Academic Publishers, September 1997.

[13] L. Eggert and J. Heidemann. Application-level differentiated services for web servers. *World Wide Web Journal*, 3(2):133–142, March 1999.

[14] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. The design and evaluation of a feedback control edf scheduling algorithm. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[15] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *WISP*, pages 59–67, Madison, WI, June 1998. ACM.

[16] F. G. Shinskey. *Process control systems: application, design, and tuning*. McGraw-Hil, New York, 4th edition edition, 1996.