Practical Application of Control Theory to Web Services *

Tarek Abdelzaher, Ying Lu, Ronghua Zhang
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
{zaher,yl8c,rz5b}@cs.virginia.edu

Dan Henriksson
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden
dan.henriksson@control.lth.se

Abstract

This tutorial describes experiences with applying a control theoretical approach to achieving performance guarantees in Web servers, with emphasis on delay control. A model for the server is formulated and translated into a control problem formulation. Limitations of the control theoretic approach are identified that arise due to system non-linearities and modeling inaccuracies. Solutions are proposed that augment the feedback control framework with elements of scheduling and queueing theory. The theoretical results and QoS control loops presented by the authors are implemented in a middleware package, called ControlWare, which provides the software mechanisms and interfaces that allow control of real server performance. Implementation and performance of ControlWare is described.

1 Introduction

Feedback control theory has recently been applied for performance control in several Web-based applications to achieve quality of service (QoS) guarantees. The fundamental reason feedback control theory is applicable in the computing domain is the fact that performance of computing services is tightly related to the status of various systems queues, such as the CPU scheduling queue and socket queues. At high load, these queues act as integrators of (request) flows, and hence can be described by difference equation models amenable to a control-theoretical analysis. Response time and throughput are two of the most important performance metrics in Internet services. Acceptable performance can be disrupted by several factors the most common of which is system overload. To reduce response time or increase throughput, the fundamental manipulated variable in the computing system is the allocation of resources, such as CPU and network bandwidth, to computing tasks. Resource allocation determines how fast requests are served at different parts of the system, which is equivalent to manipulating (request) flows.

This paper presents a brief tutorial on the performance control problem in contemporary Internet services. Generally, server performance control problems can be divided into rate control problems, delay control problems, and ratio control problems. The former category is the easiest, as it typically results in linear control loops. In contrast, delay control loops offer nonlinear behavior. The non-linearity is due to the inverse relation between flow and delay, which invariably arises in any control loop of a time-related metric. Nonlinearities also arise when the required performance is expressed in relative form. For example, it may be required that the service rate of premium clients be double that of basic clients in some server installation. In this case, a ratio appears in the control loop, which causes non-linear behavior.

In order to accommodate software non-linearities, feedforward control has been used to keep the system in the neighborhood of an operating point around which it can be linearized. Feedforward control requires a model that predicts the effect of system inputs on its performance. Several theoretic foundations can be brought to bear for such prediction, including real-time scheduling theory and queueing theory. Since the feedforward controller keeps the system around the operating point, a linearized small-deviation model becomes sufficient for purposes of feedback control. Moreover, the feedback contoller eliminates the need for accuracy in feedforward models. This tutorial describes several examples of

*The work reported in this paper was supported in part by the National Science Foundation under grants CCR-0093144, ANI-0105873, EHS-0208769, and MURI N00014-01-1-0576.

the aforementioned synergistic use of feedforward prediction in combination with feedback control in Internet servers.

In addition to modeling and control aspects, some practical implementation aspects must be considered in the design of performance control loops. An important practical consideration in server performance control is that application source code is not always available, which makes it difficult to instrument the application with performance control loops. This tutorial describes a non-intrusive approach that allows retrofitting feedback control into the software system without changing application code. Another practical deployment issue is that successful application of feedback control to software systems requires expertise in both control theory and software design. Hence, control engineers and system programmers must communicate to design and implement the control loop. This requirement for interdisciplinary teaming is a challenge to deployment. To solve this challenge, we describe a middleware service, called ControlWare, that allows modular composition of software performance control loops in a way that isolates the control engineer from the systems programmer. In this framework, the software programmer provides sensors and actuators that interface to the target application, while providing a standard API to the controller. The control engineer then designs a controller without consideration to the implementation and semantics of sensors and actuators. ControlWare provides the means for integrating plug-and-play sensors, actuators, and controllers into performance control loops for different software performance control problems.

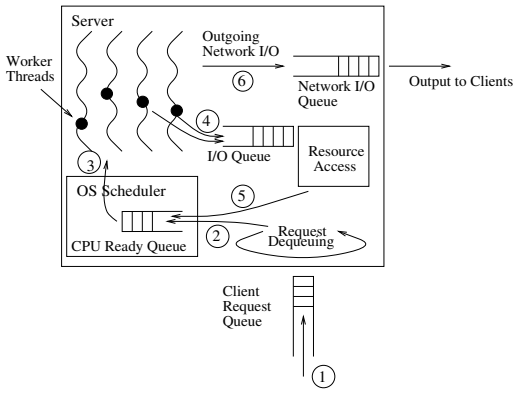
The rest of this paper is organized as follows. Section 2 describes the basic server model. Section 3 quickly describes the simplest (linear) performance control loop. Section 4 describes performance control and the use of prediction to alleviate non-linearities. Practical considerations in implementing the feedback schemes are introduced in Section 5. Related work for further reading is summarised in Section 6. The paper concluses with Section 7.

2 Basic Server Model

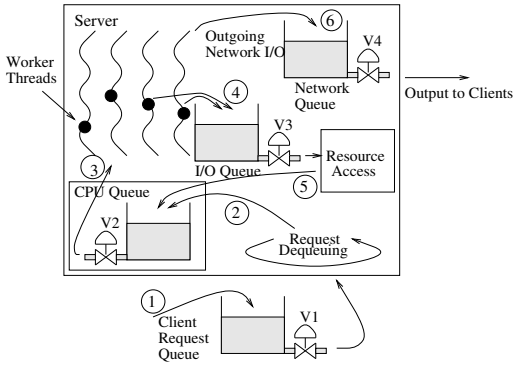
We begin the tutorial by elaborating on the basic server model. Consider a high-performance server, such as one used for hosting popular commercial web sites. Such a server can typically be approximated by a liquid task model in which the progress of requests through server queues is represented by a fluid flow. A detailed description of Web server internals and their control-theoretic model is found in [6]. Briefly, the server is composed of several stages of processing. The service rate, $dN_k(t)/dt$, of stage k, defines the amount of flow through that stage, where $N_k(t)$ is the total number of requests served by this stage by time t. The server queues up requests in several queues, shown in Figure 1(a). These queues include the CPU ready queue, the socket accept queue, the disk I/O queue and network output queue. They can be modeled as capacities that accumulate the corresponding flows. The number of requests queued at stage k, denoted V_k, is a quantity akin to volume, given by $V_k = \int_{-\infty}^{t}(F_{in} - F_k)dt$, where F_k is the service rate of stage k (i.e., $F_k = dN_k(t)/dt$) and F_{in} is the request arrival rate to that stage. Queues also offer points at which flows, F_k, can be manipulated. Figure 1(b) depicts the server from a control perspective where capacities are represented by water tanks. Observe that "valves" in Figure 1(b) represent points of control (i.e., actuators, which manipulate the service rates F_k). We assume in this analogy that flow through the valve depends only on valve opening and not on the liquid level. Thus, the arrangement is perhaps more akin to a pump.

A performance feedback control loop typically manipulates one or more of the valves in Figure 1(b). In many cases, only one of the aforementioned queues is the bottleneck. Hence, the system is most effectively controlled by manipulating the valve associated with that bottleneck. Manipulating the valve can typically be done in one of two ways. The first is to change the amount of resources allocated to the request flow. For example, a larger fraction of the CPU can be allocated to a particular client class using appropriate operating systems mechanisms. Consequently, these requests are served faster (i.e., F_k is increased). Alternatively, the server can change the amount of work needed per request. For example, it might substitute high-resolution images with low-resolution ones, that take less resources to serve. From a modeling perspective, this actuator has a gain that describes the relation between actuator input and the corresponding flow F_k.

To close the loop, performance sensors must be employed. These sensors measure the actual value of the performance metric controlled. For example, in a loop controlling response-time, the server might time-stamp requests when they arrive and when they have been served completely, then average the differences over a window of choice. This average will constitute a delay sensor. The most important performance metrics in Internet servers can be categorized based on how they relate to time. We call them time-based, rate-based, or



(a) the computing model



(b) a control-oriented representation

Figure 1. Server architecture

ratio based, depending on whether they are measured in units of time (e.g., seconds), inverse units of time (e.g., requests per second), or are unitless (ratio metrics). In the next section, we shall focus on rate and delay control as examples, offering various options for handling the non-linear behavior of the delay control loop.

3 Rate Control

Consider a server that supports some actuation mechanism that can influence the processing rate (referred to as *server speed* in the sequel) of the incoming web requests. Let us denote by C, the average number of resource cycles required to process a typical request. The actual execution time of the request is then given by $\frac{C}{\mu}$, where μ is the current server speed. Server speed can be changed by techniques such as dynamic voltage scaling (DVS), or scheduling algorithms where only a fraction of CPU capacity is allocated to the server.

An example of a rate control loop is one that controls server utilization. Consider a server in which a controlled fraction of capacity must be allocated to a given class of clients. The objective is to control the rate at which clients are admitted such that the desired utilization is achieved for the class under consideration. This loop is generally linear in that the utilization is proportional to the number of admitted clients. The only dynamics in this loop come from the sampling delay and from the accumulative effect that relates flow to utilization. Utilization, measured as an average load over a given time interval, is proportional to the integral of the difference between the input and output flows. A standard P or PI controller is typically sufficient to maintain utilization at a desired value. This linearity and simplicity is characteristic of rate control loops in computing systems. These loops will thus not be covered any further in this tutorial.

4 Delay Control

Delay is inversely proportional to flow, which introduces non-linear behavior in control loops. One way to deal with the nonlinearity is to use feedforward control in combination with feedback control as mentioned in the introduction. The purpose of feedforward control is to reduce system excursions away from the operating point, hence making it possible to apply a linearized model in the neighborhood of the operating point.

In addition to the standard feedback loop, a feedforward control action, μ_{ff}, is computed from the reference delay and information regarding the past arrival pattern. This feedforward signal is then adjusted by a feedback term, $\Delta\mu$. In essence, the feedforward signal uses a predictive model to decide how flow should be manipulated such that the desired delay is achieved, given the nonlinear delay-flow relationship in the server. The derivation of the feedforward predictor and the design of the feedback controller will be described below.

4.1 A Queueing Predictor

Queueing theory offers expressions that relate service rate and server response time. These expressions can be used to determine the service rate, μ_{ff}, needed for a particular response time to be met. For simplicity of explanation, assume the request stream is Poisson distributed, with an arrival rate of λ. From queueing theory, we know that for an M/M/1 queue with arrival rate λ and service rate μ, the long-term average queueing time (i.e. connection delay) for the clients is

$$D = \frac{\lambda}{\mu(\mu - \lambda)}, \; where\, \lambda < \mu \qquad (1)$$

The above equation can be solved for μ that achieves the desired delay D, given the current arrival rate λ. This value of μ is given for enforcement by the actuator. Should the resulting delay be different from D, an error is generated which drives a controller (e.g., a PI controller) to correct actuator input until the error is eliminated. This loop is eventually successful in bringing average server delay of each class to the desired set point.

One problem with the above approach is that queueing theory describes relations between long-term averages only. In the short term, delay may deviate from the set point. Also, it is hard to measure the long term average arrival rate, λ, quickly enough. Hence, if the request rate changes suddenly, some time may elapse before the queueing predictor is able to account for this change. To address these limitations, a different type of predictor can be employed that uses instantenous queue measurements instead of queueing theory equations.

4.2 An Enhanced Feedforward Predictor

The enhanced feedforward predictor computes the needed service rate from actual measurements of recent delay and queue length. Consider the situation at some arbitrary time, t_{now}. At this time, there are a number of requests queued up waiting to be processed. Let us consider N of these requests and compute their average delay as a function of the processing speed, μ. The situation is depicted in Figure 2.

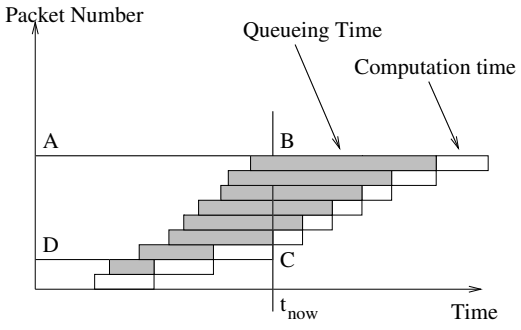


Figure 2. Visualization of queuing and processing

Assuming that requests have an average nominal processing time, $\hat{C}$, their individual processing time in next sample will be, $\hat{C}/\mu$. Let $\hat{D}$ denote the average delay experienced by the N requests. We will now provide a geometric derivation of the equation relating the average delay, $\hat{D}$, and the server speed, μ.

The total delay experienced by the N requests, $N\hat{D}$, can be computed geometrically from Figure 2 as the area $BECF$. This is given as the area of the rectangle $ABCD$, plus the area of the triangle BEC, minus the lightgray shaded area ($ADFB$). Noting that the area $ADFB$ is the sum of the arrival times of the requests, we arrive at the equation

$$N\hat{D} = N \cdot t_{now} + \frac{N \cdot (N\hat{C}/\mu)}{2} - \sum_{k=i}^{i+N-1} A_k \quad (2)$$

Dividing by N, we get:

$$\hat{D} = t_{now} - \hat{A} + \frac{N\hat{C}}{2\mu} \quad (3)$$

Now introduce $\hat{A}_i = \frac{1}{N}\sum_{k=i}^{i+N-1} A_k$ as the average arrival time, D_i as the average delay, C_i as the average computation time, and μ_i as the server speed for requests being dequeued in the ith sample. We also see that $t_{now} - \hat{A}_i = Q_i$ is the average queuing time for the requests being dequeued in the ith sample. Solving for μ_i then gives:

$$\mu_i = \frac{NC_i}{2(D_i - Q_i)} \quad (4)$$

which is a predictor equation telling us how to choose the server speed in order to obtain an average delay, D_i, of the next N requests. The feed-forward controller is invoked after each departure and computes the new processing speed according to Equation (4). D_i is chosen as the current delay set-point, D_r. the average queuing time, Q_i, is measured exactly. The average nominal processing time, C_i, is estimated from past measurements. N will be chosen as the current queue length at each sampling instant.

Figure 3 compares the aforementioned mechanisms in terms of ability to maintain delay guarantees in an Apache web server. It is seen that the delay set point is most closely tracked when the control loop is augmented with the enhanced predictor.

5 Practical Considerations

One challenge in implementing feedback control in web servers is that fact that server code does not support QoS guarantees. In particular, there is no mechanism to allocate separate resources for different classes of clients sharing the same server.

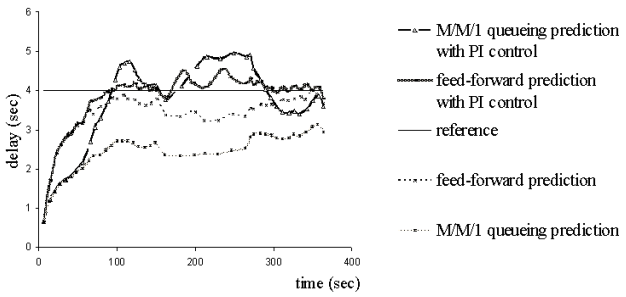


Figure 3. Control of average delay

Consider a common best effort server in which one or more identical worker threads or processes serve incoming client requests. Connection requests to the server arrive on a common port (e.g., port 80 for web servers). These connection requests are dequeued from the server port, either directly by the worker threads themselves, or separately by a dedicated dequeue thread, which will pass the requests to a worker thread. We assume that the server is *multi-instance safe*. In other words, if we run multiple instances of the server, each instance should work well in presence of others. This assumption holds for many Internet server applications, such as web servers, mail servers, and FTP servers.

To convert a legacy application into a QoS-aware one, the classifier, sensors, and actuators must be added. One approach is to add generic versions of these components as a general kernel-level mechanism. A classifier seprates incoming requests in the kernel by class. After classification, input request traffic of each class is queued separately on a different, per-class, kernel port (socket). To the legacy server application, these ports look like ordinary sockets. A separate instance of the best-effort server is instantiated to listen to each port. Hence, a separate server instance is assigned to each class of clients. This design greatly facilitates the design of resource allocation mechanisms. Namely, operating system support may be used to to allocate the resources among different servers.

Another practical problem in applying control to computing services lies in the interdisciplinary nature of the expertise needed to close the control loop. To alleviate this problem, the authors designed a middleware package called, ControlWare [20], which allows the user to express QoS specifications offline, maps these specifications into appropriate feedback control loop sets, and connects loops to the right performance sensors and actuators in the application such that the desired QoS is achieved. A main novelty of this middleware lies in isolating the software application programmer from control-theoretic concerns while utilizing this theory to achieve the desired QoS guarantees. At the same time, ControlWare isolates the control engineer from the software task of interfacing the controller to the controlled software system and designing software performance sensors and actuators.

ControlWare contains a library of macros written in a topology description language, each formulating a particular type of QoS guarantee as a feedback control problem. The library is extensible in that a control engineer can transform a new guarantee type into a macro that describes the corresponding loop interconnection topology and stores that macro in the middleware's library. Currently, the library includes macros for absolute convergence guarantees, relative differentiated service guarantees, prioritization, and optimization guarantees. Each macro, like a block diagram, includes components such as sensors, actuators, and controllers. ControlWare contains a library of common sensors and actuators that can be used in these software control loops. These sensors and actuators are written by software engineering who are familiar with the application, but do not needed to understand the control-theoretic principles behind the feedback loop. The library is extensible in that it is possible for an application programmer to add new sensor and actuator types. A control engineer can therefore quickly assemble loops that connect the controller to the controlled process by virtue of the sensors and actuators imported. The controller is then tuned, the software is compiled, and run-time guarantees are enforced.

6 Related Work

Several recent papers [4, 1, 5] 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 [19, 18, 12], 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 [17], guarantees were made on power dissipation by applying control-theoretical techniques to microprocessor thermal management. At the network layer, control theory was applied to packet flow control in Internet routers [9, 7]. Due to the usefulness of the control-theoretic approach and its versatile applications, middleware frameworks emerged for control-based QoS assurances [20]. The authors of [20, 10, 8] provided tools to help apply control-theoretic design techniques to a larger class of systems.

Several predictive frameworks were designed to augment the feedback control component. In addition to the ones we mentioned in this paper, a theory for using utilization bounds was developed for predicting deadline misses, when worst case latency guarantees must be attained. These bounds are an extension of the original work by Liu and Layland [11]. The original bounds were limited to the assumption that requests arrive periodically [14, 16] or to simple extensions of the periodic arrival model [13]. Abdelzaher's work [2] is the first to relax the periodicity assumption completely, thus generating results that are compatible with arbitrary queuing systems such as web servers where requests arrive randomly with nothing known a priori on their arrival pattern. In [3], the notion of utilization bounds for schedulability of aperiodic tasks is generalized to the case of distributed resource services.

7 Conclusions and Future Work

In this paper, we demonstrated the application of control theory to web server performance control. The use of prediction was described to account of the non-linear behavior of delay control loops. Practical issues in the application of performance control were described. A brief summary of related performance feedback control efforts was presented. In conclusion, with the increasing complexity of computing systems, feedback control is increasingly important to provide performance assurances in the presence of growing system complexity.

References

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

[2] T. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services, 2001.

[3] T. Abdelzaher, G. Thaker, and P. Lardieri. A feasible region for meeting aperiodic end-to-end deadlines in resource pipelines, 2004.

[4] 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.

[5] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[6] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3), June 2003.

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

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

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

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

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of ACM*, 20(1):46–61, 1973.

[12] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *IEEE Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.

[13] A. K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE transactions on Software Engineering*, 23(10):635–645, October 1997.

[14] M. Pandya and M. Malek. Minimum achievable utilization for fault-tolerant processing of periodic tasks. *IEEE Transactions on Computers*, 47(10):1102–1112, October 1998.

[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] D.-W. Park, S. Natarajan, and A. Kanevsky. Fixed-priority scheduling of real-time systems using utilization bounds. *Journal of Systems and Software*, 33(1):57–63, April 1996.

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

[18] J. A. Stankovic, T. H. 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.

[19] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Operating Systems Design and Implementation*, pages 145–158, 1999.

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