Proceeding of the 2004 American Control Conference
Boston, Massachusetts June 30 - July 2, 2004

Challenges in Control Engineering of Computing Systems

Joseph L. Hellerstein .

Abstract— **Over the last few years, there has been considerable success with applying control theory to computing systems. Our experience has been that there are several commonly occurring control problems in computing systems—translating between service oriented units (e.g., response times) and effector (actuator) units (e.g., the maximum number of connected users); optimizing resource usage; regulating service levels to enforce service level agreements; and adapting to disturbances such as changes in workloads. Developing control systems that address these problems involves challenges related to modeling the managed element (plant); handing sensor data that are noisy, incomplete, and inconsistent; dealing with effectors that have complex effects that often do not correspond well to the control objectives; and designing control systems (especially filters, the choice of measured outputs, and time delays).**

I. INTRODUCTION

The relentless decline in the price of computer hardware and software has led to the widespread use of information technology (IT). As a result, the dependence on and the scale of computing systems has grown dramatically, making it imperative to have stable well-behaved systems.

Despite this imperative, formal control methods are rarely used in practice when developing new capabilities for computing systems. For example, it is uncommon to do system identification of the components to be controlled and almost unheard of to analyze the response of the system to disturbances.

Over the last three years, my colleagues and I at IBM along with researchers elsewhere have had considerable success with using classical control theory (mostly digital control) to analyze and design closed loops in computing systems. This work has resulted in IBM products that are more robust to disturbances and, in some cases, considerable improvements in their performance as well (e.g., lower response times). Further, our work has provided IBM developers with new insights into design choices.

Based on our experience, we believe that control theory has an important role to play in the development of new computing systems, especially complex software systems. This tutorial describes many of the challenges with applying control theory to computing systems. Section II provides an overview of enterprise computing systems. Section III describes various control problems in computing systems. Section IV details challenges in the development of closed loop systems for computing systems. Section V contains a survey of related work. Our conclusions are contained in Section VI.

J.L. Hellerstein is a Research Staff Member at the IBM Thomas J. Watson Research Center, Hawthorne, New York, U.S.A. hellers@us.ibm.com

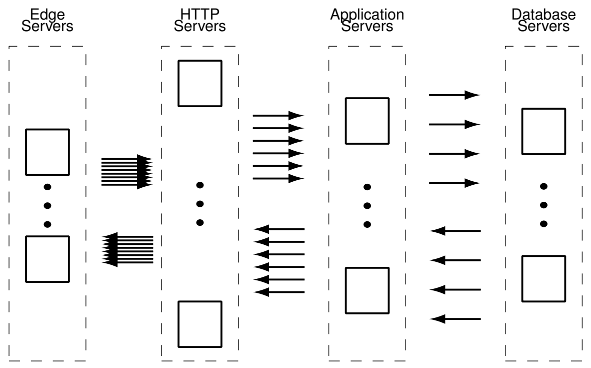

Fig. 1. Example of a multi-tiered eCommerce system. Server tiers are enclosed in dashed lines. The flow and density of requests and responses are indicated by the arrows.

II. STRUCTURE OF COMPUTING SYSTEMS

This section introduces key concepts in computing systems. Our focus is enterprise level computing, especially eCommerce Systems.

Figure 1 displays the general structure of an eCommerce system such as those that provide on-line storefronts on the Internet. End-to-end service levels (e.g., response times) depend on the flow of requests across multiple tiers of servers, each of which has its own complex structure. The system is organized into multiple groups of servers, called tiers. The first tier, the Edge Servers, accept in-coming requests and routes them to the Hypertext Transfer Protocol (HTTP) Servers (the second tier) where requests are interpreted. Some fraction of these requests require more sophisticated processing and so are forwarded to an Application Server (the third tier). The programs that execute here may require access to structured data in a Database Server (the fourth tier).

Each software element (e.g., Edge Server, HTTP Server, Application Server, Database Server) has a complex structure. For example, the Application Server has an operating system (e.g., UNIX^{R}[1]), a Java execution environment (the Java Virtual Machine), a servlet container (an environment in which special servlet programs execute), and one or more servlets. These components of the Application Server require various resources to do their work. Examples of resources are memory, the central processing unit (CPU), and operating system processes.

Contention for resources is a central concern in the design of robust computing systems since the limited availability of resources can result in performance and/or availability problems. For example, too little memory allocated to

[1]UNIX is a registered trademark of The Open Group.

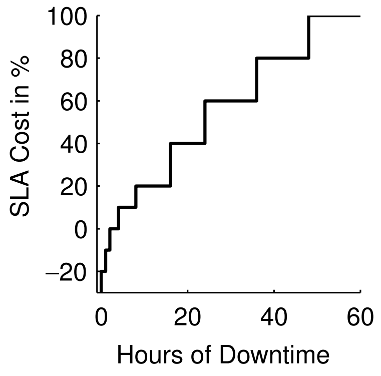

Fig. 2. Staircase function of costs in a service level agreement as reported in [1].

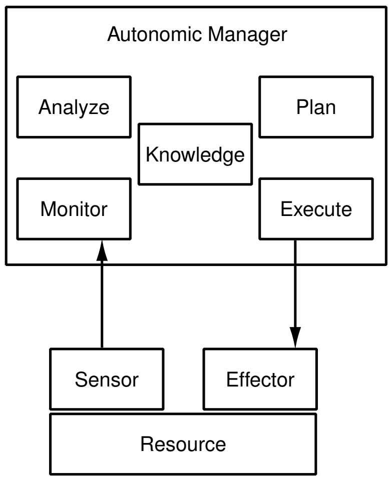

Fig. 3. Components of the Autonomic Computing Architecture.

the Java Virtual Machine can result in excessive "garbage collection" whereby data are moved in order to compact unallocated storage to create larger blocks.

Resource contention arises in two ways. The first is a result of competition between components of a system. For example, the operating system may compete with the Java Virtual Machine for memory. A second way in which contention arises is a result of the resource requirements of concurrent requests made to the computing system. For example, a "check price" request made by one end-user may compete with an "order status" request made by another end-user for Java threads, memory, and database connections.

Increasingly, **service level objectives (SLOs)** are used to specify the desired response times, throughputs, and/or other metrics desired for different requests to the computing system (e.g., "order status", "buy item", "check price"). For example, a bookstore that outsources its computing systems might have the following SLO: "order status requests should have a response time that is no greater than 1 second." The set of SLOs used by an installation is referred to as a **Service Level Agreement (SLA).** SLAs may specify penalties if the contracted service is not delivered. For example, Figure 2 plots the penalty for excessive downtime (with a reward, or negative cost, for greatly reducing downtime) that is expressed in terms of the percent of the customer's monthly service charge [1]. One can view the provider of computing services as assigning resources (e.g., servers) in a way so as to minimize the sum of resource costs and SLA penalties. Note that not all service requests are covered in the SLA. Those requests that are not covered by the SLA are served on a "best effort" basis. Further, since requests may flow through many servers and components within the server, it can be complicated to determine how to manage resources in a way so that the SLOs are not violated.

One software structure for managing resources to achieve SLOs is the autonomic computing framework [2]. As depicted in Figure 3, an autonomic manager accesses resources through their sensors to obtain measurement data and interacts with resource **effectors** (or actuators) to change the behavior of the resource. The manager contains components for monitoring, analysis, planning, and execution. Common to all of these is knowledge of the computing environment, service level agreements, and other related considerations. The autonomic manager monitoring component filters and correlates sensor data. The analysis component processes these refined data to do forecasting and problem determination, among other activities. Planning constructs workflows that specify a partial order of actions to accomplish a goal specified by the analysis component. The execute component controls the execution of such workflows.

III. CONTROL PROBLEMS

This section describes several control problems in computing systems. Many of these problems arise from key technology and business trends that are shaping the direction of computing. These trends are: (1) the rising cost of operating computing systems; (2) the on-demand model for acquiring IT products; (3) the increasing use of outsourcing to satisfy the IT requirements of businesses; and (4) the wide-spread use of the Internet to connect businesses with their customers. While these trends have been on-going for some time, only now is there emerging a realization of the importance of control engineering in addressing them

We begin with the cost of operating computing systems. For some time, it has been recognized that while hardware and software prices have declined dramatically, the cost of operating computing systems remains large and is increasing. Included here is what is needed to ensure reliability, security, and good performance. Industry analysts estimate that these costs accounts for 60% to 90% of the total cost of ownership (e.g., [3]), largely because of the amount of human involvement required for system operation. One part of the cost of operations is configuring software systems (often consisting of multiple products), a task that often involves adjusting internal characteristics such as buffer pool sizes and number of concurrent threads. In essence, these internal characteristics are the effectors by which the system is controlled. Unfortunately, it is rare to have a direct relationship between the settings of resource effectors and the values of metrics related to SLOs (e.g., response times). Thus, human experts are often needed to translate between the units of resource effectors and SLO metrics. Doing so requires people with considerable expertise and hence increases the cost of delivering IT services. *Control technology can help by automating this translation so that resource effectors operate in the units used in service level objectives*. For example, [4] describes a system in which administrators specify service level objectives instead of details of memory allocations, CPU priorities, and other configuration parameters.

Another reason for the high cost of operating computing systems is that since they are complex non-linear systems it is difficult to optimize their performance. One of the most common ways in which this optimization is done is by **load balancing**, a technique whereby requests are distributed across resources in a way that equalizes loads. For example, the Edge Server in Figure 1 is responsible for distributing incoming requests to HTTP Servers in a way that balances the load on these servers. Load balancing tends to reduce response times and increase throughputs since the performance of computing systems is usually determined by the most heavily loaded resource (often referred to as the bottleneck resource). *This is an optimization problem for which the objective function is to minimize the difference in utilization of the resources.* Control techniques are particularly useful to address the dynamics of the system. For example, [5] describes a system in which memory resources are balanced in a database management system with considerations for changes in the queries made to the system.

A second trend is the interest of larger customers in a new model for acquiring IT. Traditionally, customers purchase computing capacity well in advance of its usage. This provides little flexibility to add capacity if loads increase unexpectedly or to save money by shedding unneeded capacity. Recently, there has been much interest in providing computing capabilities "on-demand" rather than purchasing them outright. This could be done in the context of outsourcing whereby the outsourcer (sometimes referred to as the computing utility) supplies the level of resource needed and the customer is charged accordingly. Alternatively, it could be that the equipment purchased by a customer has metering capability that the seller reads to determine monthly charges to the customer (much like a water or electrical meter). Implementing an on-demand system requires a capability to provision resources dynamically to satisfy new demands as well as an ability to de-provision resources dynamically if they are no longer needed (e.g., [6]). As with load balancing, *this is an optimization problem*. However, it has very different characteristics from load balancing in two ways: (a) a business level optimization is employed in terms of minimizing the sum of the cost of holding costs of servers and penalties for violating SLOs in an SLA; and (b) often provisioning and de-provisioning have dead-times, a characteristic that complicates controller design.

TABLE I
SUMMARY OF THE CONTROL PROBLEMS IN COMPUTING SYSTEMS DISCUSSED IN THIS ARTICLE.

Control Problem	Description
Translate effector units	Translate between SLO metrics and the units used by effectors of computing systems.
Optimize resources	Balance loads to minimize bottlenecks and provision to minimize costs.
Regulate service	Adjust resource usage so that service levels are consistent with SLOs.
Reject disturbances	Regulate in the presence of large changes in loads and resource failures.

A third trend is IT outsourcing. Because specialized skills are required to operate complex computing environments, many customers are choosing to outsource their IT needs. Outsourcing means that an organization such as EDS or IBM provide the IT equipment and skilled professionals to operate these equipment. This requires that the outsourced customer specify a service level agreement (SLA) that quantifies the service expected from the outsourcing organization. Thus, the outsourcer needs a way to enforce SLAs so that customers who pay for higher grades of service (e.g., lower response times) receive their expected level of service. That is, there must be a way to regulate the level of service delivered. *In control terms, this is a regulation problem.*

The last trend is that businesses have a presence on the Internet, both to supply information about the companies goods and services and to sell these goods and services. While the broad reach of the Internet has great appeal for both of these goals, it creates challenges as well. In particular, web sites are subject to flash-crowds, a phenomena whereby loads grow dramatically (e.g., as a result of political or weather events) [6]. Thus, it has become essential for IT systems to deal with loads that can increase (or decrease) within seconds or minutes while still complying with SLAs. *In control terms, this is a disturbance rejection problem.*

Our characterization of control problems in summarized in Table I.

IV. CONTROL ANALYSIS AND DESIGN

The design and analysis of control systems for computing systems requires models of the resource operation, appropriate sensors and effectors, and considerations for the full set of elements in the control loop (e.g., controller, filter, parameter estimator). This section addresses each of these topics.

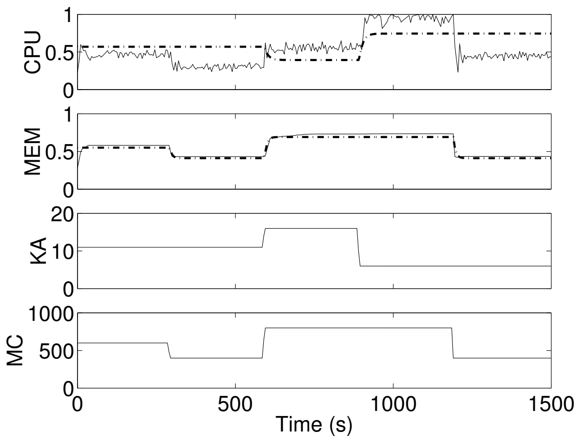

Fig. 4. Predictions of CPU and memory utilizations using two SISO models.

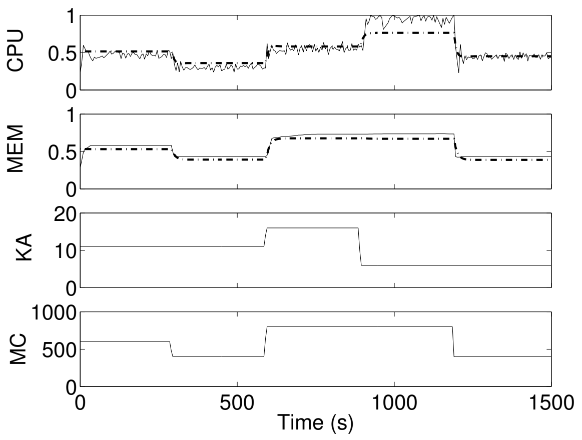

Fig. 5. Predictions of CPU and memory utilizations using a single MIMO model.

A. Modeling the Managed Element

The construction of system models remains a significant challenge in the successful application of control theory to computing systems. Four approaches are used in practice. The first is a purely empirical approach that employs curve fitting to construct models; these models do not address dynamics. This has been very effective in IBM's mainframe systems [4]. Its application to systems such as Figure 1 is being investigated.

The second approach to modeling is a black box methodology that has been applied to SISO and MIMO (multiple input multiple output) systems (e.g., [7], [8]). This approach requires: choosing an operating point, designing appropriate experiments, and developing empirical models. Typically, ARX models are used such as $y(k) = a_1 y(k-1) + \cdots + a_n y(k-n) + b_1 u(k-1) + \cdots + b_m u(k-m)$, where $m \leq n$. For example, in the Apache HTTP Server, there are two control inputs, the maximum number of clients (denoted by `MaxClients`) that controls the level of concurrency, and the keep alive timeout (denoted by `KeepAlive`) that specifies how long a connection to the server persists after

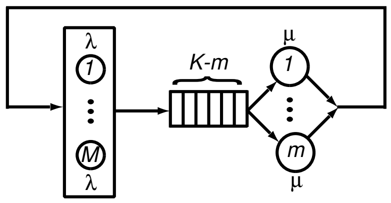

Fig. 6. $M/M/m/K/M$ queueing model. The think time of the M customers is exponentially distributed with the rate λ. The service times of the m servers is exponentially distributed with the rate μ.

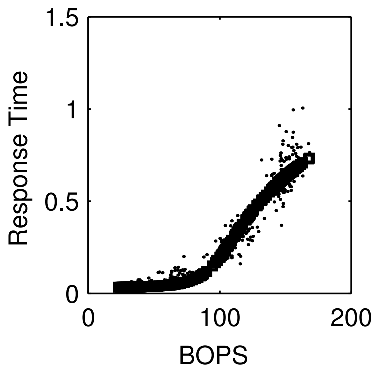

Fig. 7. Fit of $M/M/3/140/140$ model to data from a three server testbed running an eCommerce workload.

the completing of the last request on that connection. We consider two measured outputs, the utilization of the CPU (denoted by `CPU`) and the utilization of memory (denoted by `MEM`). One way to model the effect of `MaxClients` and `KeepAlive` on `CPU` and `MEM` is to construct two SISO models. While `MaxClients` affects both `CPU` and `MEM` `KeepAlive` only affects `CPU`. Thus, we use the models

$$\begin{aligned} y_{CPU}(k) &= a_{CPU} y_{CPU}(k-1) + b_{CPU} KA(k-1) \\ y_{MEM}(k) &= a_{MEM} y_{MEM}(k-1) + b_{MEM} MC(k-1) \end{aligned}$$

Figure 4 shows the results. We see that the fit for `MEM` is quite good. However, the model of `CPU` is poor as `MaxClients` is changed from its operating point. An alternative is the MIMO model

$$\mathbf{y}(k) = \mathbf{A}\mathbf{y}(k-1) + \mathbf{B}\mathbf{u}(k-1)$$

where $\mathbf{y}^T = \begin{bmatrix} CPU & MEM \end{bmatrix}$, $\mathbf{A}$ is a 2×2 matrix, $\mathbf{B}$ is a 1×2 matrix, and $\mathbf{u}^T = \begin{bmatrix} KA & MC \end{bmatrix}$. Figure 5 displays the results of the MIMO model. We see that the MIMO model is considerably more accurate than the multiple SISO model.

A third approach to modeling is based on stochastic processes, especially queueing theory. By making assumptions about the distribution of inter-arrival times and service times, queueing theory provides a way to calculate the effect of control inputs such as buffer size, number of servers, and service rates on measured outputs such as response times and throughputs. Figure 6 depicts an $M/M/m/K/M$ queueing system in which there are M customers, a buffer of length $K-m$, and m servers. The M customers think for an exponentially distributed period of time with rate λ, at which point they submit a request. The request is assigned to an idle server, if one exists, and service takes an exponentially distributed time with rate μ. If there is no idle server, the request is placed at the end of the queue where it waits until those requests in front of it have been served. If the queue is full, the request is discarded and the customer waits for another exponentially distributed time with mean $\frac{1}{\lambda}$.

We assessed the adequacy of this model in the context of the HotRod system for an eCommerce workload [6]. The intensity of the workload is measured in business operations per second (BOPS), and service quality is quantified in terms of response time. Figure 7 compares the measured response times obtained from the HotRod system with those estimated by an $M/M/m/K/M$ queueing system, where $m = 3$ and $M = 140$. (With first-come-first-served scheduling, it is straightforward to estimate response times from the Markov state model of $M/M/m/K/M$.) The dots are measured response times, and the squares are the response times estimated by the model. The fit is quite good in that the model accounts for over 90% of the variability in the data. Further, it is relatively easy to model transient behavior using the Markov chain, which has appeal for characterizing the effects of control actions. A shortcoming of the model is that it is not closed form in that its solution requires constructing and analyzing a Markov chain. Simpler open queueing models that assume an infinite number of customers (i.e., $M = \infty$) often have a closed form solution. Unfortunately, these models have a poor fit to the HotRod data.

A fourth approach to modeling is to develop special purpose representations of specific systems. An example of this is the first principles analysis done for adaptive queue management in network routers [9]. This approach involves a detailed understanding of the TCP/IP protocol and the development of differential equations to estimate a transfer function.

B. Sensors

A significant focus in the management of computing systems is the choice of sensors, especially standardizing interfaces to sensors. The most widely used protocol for accessing sensor data in computing systems is the Simple Network Management Protocol (SNMP) [10]. While this allows for programmatic access, it has not addressed various issues that are of particular concern for control purposes. Among these are the following:

1) Typically, there are multiple measurement sources (even on a single server) that produce both interval and event data. Unfortunately, the intervals are often not synchronized (e.g., 10 second vs. 1 minute vs. 1 hour), and missing data are common. Even worse, data from different servers often come from clocks that are unsynchronized (or worse still, are synchronized via some complex protocol that converges over a longer window).
2) Often, the metric that it is desirable to regulate is not available. For example, end-to-end response times are notoriously difficult and expensive to obtain. Thus, surrogate metrics are often used such as CPU queue length. Hence, it may well be that the surrogate is well regulated but the desired metric is not.
3) There can be substantial overheads associated with metric collection. For example, it can be quite informative to collect information about the resource consumption of individual requests to a web server. However doing so may consume a substantial fraction of the server CPU. This results in another kind of control problem—determining which measurements to collect and at what frequency.
4) Often, the measurement system has built-in delays. For example, response times cannot be reported until the work unit completes. Sometimes, the mean response time is about the same as the control interval, which can lead to instabilities. Unpredictable delays are common as well since measurement collection is typically the lowest priority task and so is delayed when high priority work arrives (which can be a critical time for the controller).
5) An on-going challenge for developers of instrumentation for computing systems is that there is a wide variation in the semantics of supposedly standard metrics. For example, the metric "paging rate" could mean any of the following: (a) the rate at which pages are written to disk; (b) the rate at which pages are read from disk; and (c) the rate at which page-ins are requested (not all of which result in accessing secondary storage). Because of these disparities, an attempt has been made to standardize the definition of metrics [11]. However, this effort is limited to UNIX Operating Systems since they have a similar structure and hence similar metrics.

C. *Effectors*

One of the more challenging problems in the control engineering of computing systems is that the set of available effectors (actuators) often has a somewhat complex relationship with the measured output, especially in terms of dynamics. We illustrate this problem by giving several examples.

Consider the IBM Lotus Domino Server, an email server. One objective of considerable interest to administrators is to control internal queueing, both for reasons of reliability (e.g., so that certain load-dependent exceptions do not occur) and performance. That is, administrators want to limit the number of users whose requests are being processed concurrently. We refer to this as the number of concurrent users. Commonly, administrators use the effector `MaxUsers`, a parameter that limits the maximum number of users that are connected to the system. However, the number of *connected* users is not the same as the number of *concurrent* users. For example, during lunch time, there may be many connected users, very few of whom submit requests. Under these circumstances, `MaxUsers` could be much larger than the number of concurrent users. On the other hand, during busy periods (e.g., close to an end-of-month deadline), almost all connected users may have submitted requests. In this case, `MaxUsers` may be very close to the number of concurrent users. In essence, the gain associated with this effector is load dependent.

There is still another complication with `MaxUsers`. The mechanism employed does not maintain a queue of waiting requests to connect to the server. That is, if `MaxUsers` is increased, there is no effect until the next request arrives. If requests are of short duration and are made quickly, there is little delay. However, if requests occur at a lower rate, then this effector introduces a dead time that makes control more challenging.

Another example of a complex effector is the *nice* command used in UNIX systems. *nice* provides a way to adjust the priority of a process, something that is especially important if there is a mixture of CPU intensive and non-CPU intensive work in the system. In theory, *nice* can be used to enforce SLOs dealing with the fraction of the CPU that a process receives. However, this turns out to be complicated to do in practice because of the way *nice* affects priorities. As shown in [12], this is non-linear relationship that depends on the number of processes competing for the CPU as well as the range of priority numbers used. Recognizing the limitations of using *nice*, special purpose schedulers have been developed (e.g., [13]). In essence, these approaches create a new, more rational set of effectors.

A final example is the start-time fair queueing (SFQ) algorithm. This resource management algorithm controls the service delivered by a resource by controlling the priority assigned to incoming requests [14]. Specifically, SFQ operates by tagging incoming work by class, and the tags determine the priority by which the request is processed. Unfortunately, this mechanism has some subtle, load-dependent characteristics that create challenges for designing control systems. In particular, changing the tag assigned to a new request has no effect until the requests ahead of it have been processed. If load is light, there will be few such requests, and so little dead time. However, if loads are heavy, dead times could be substantial. If dead times can be predicted, then compensation might be possible. Otherwise, the control performance of this effector can be impaired, possibility even resulting in stability problems.

D. Control Systems

This section describes issues often encountered in the design of closed loops for computing systems.

We begin by observing that where control theory has been applied to computing systems very simple controllers have been used. For example, a PI controller is used in [15], [16], and [17]. An even simpler integral controller is used in [7].

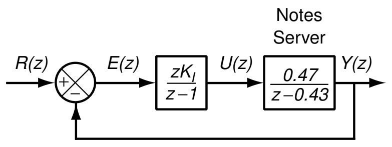

Fig. 8. Block diagram for integral control of the IBM Lotus Domino Server.

In almost all cases, the closed loop system is single input single output (SISO).

A natural way to apply control theory is to a regulation problem, such as maintaining service level objectives (SLOs). For example, an eCommerce site such as Figure 1 may provide different SLOs for response time depending on the type of interaction (e.g., “buy” versus “browse”). A first thought is to regulate response times directly. However, there is an issue here. If load is light, the eCommerce site can provide service that is much better than the SLO. And, if load is heavy, then it may be that none of the SLOs can be satisfied.

One way to avoid this conundrum is to regulate the *relative* performance of the different kinds of requests. That is, non-negative fractions $H_1, \cdots, H_n$ are selected for each kind of request, so that $H_1 + \cdots + H_n = 1$. The regulation problem is to make $D_i/(D_1+\cdots+D_n) = H_i$, where D_i is the delay incurred by the $i-th$ kind of request. This idea is developed in [18] and applied to differentiated caching services. Such an approach works best in overload situations (which is where control is most important). This being the case and assuming that service times are small compared to delays, it can be a reasonable approach for regulating relative response times as well.

The remainder of this section describes two examples of designing closed loop systems for computing systems. The first example relates to the nature of measurement sensors. The dynamics of the measurement system are almost never considered in queueing analyses of computing systems, in large part because the focus is on steady state. However, these dynamics can be play a major role in control performance.

To illustrate the foregoing, consider the IBM Lotus Domino Server. The control problem we address is regulating the number of concurrent users by manipulating the `MaxUsers` effector (which controls the number of connected users). System identification of the IBM Lotus Domino Server determined that the transfer function from `MaxUsers` to concurrent users is

$$N(z) = \frac{0.47}{z-0.43}$$

(See [7] for details.) Figure 8 displays the control system considered in which integral control is used. The transfer function from the reference input to the measured output is

$$F(z) = \frac{Y(z)}{R(z)} = \frac{zK_I(0.47)}{(z-1)(z-0.43)+zK_I(0.47)} \quad (1)$$

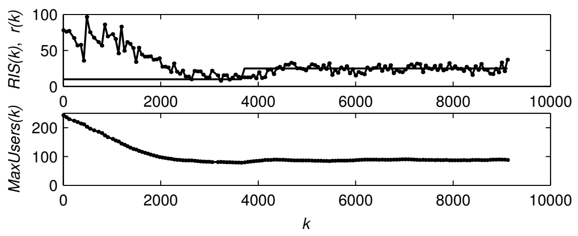

Fig. 9. Step response of testbed for the control system for the IBM Lotus Domino Serverdescribed in Figure 8.

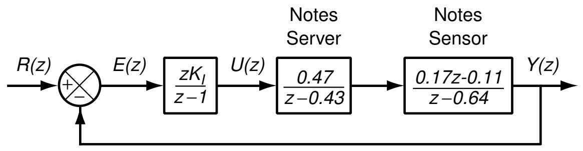

Fig. 10. Block diagram of integral control of the IBM Lotus Domino Server that explicitly models the sensor.

How well does this control model describe the behavior of the real system? To answer this question, a testbed was developed and experiments were conducted to assess the step response to a change in the reference input. Figure 9 displays the step response to a change in the reference input that occurs around time 3800 for $K_I = 0.1$. The figure contains two plots, one for the number of concurrent users (denoted by $RIS(k)$) and a second plot that shows the associated value of $\texttt{MaxUsers}(k)$. While the stochastics of the system make it difficult to estimate settling times precisely, it seems that the settling time is about 300. However, if $K_I = 0.1$ the dominant pole of Equation (1) is 0.91. Using a first order approximation of Equation (1) and defining steady state as being within 2% of the final value of the step response, then the settling time of Equation (1) is 43. This is almost a factor of ten difference from the actual settling time.

Why is the foregoing estimate so inaccurate? The answer lies in the fact that we did not consider the complete control system. In particular, we did not model the sensor. Figure 10 extends Figure 8 to include the effect of the sensor. The transfer function from the reference input to the measured output of this system is

$$G(z) = \frac{Y(z)}{R(z)} = \frac{zK_I(0.47)(0.17z - 0.11)}{D(z)} \qquad (2)$$

where

$$D(z) = (z-1)(z-0.43)(z-0.64)+zK_I(0.47)(0.17z-0.11)$$

At $K_I = 0.1$, the dominant pole is 0.99. Again using a first order approximation, this corresponds to a settling time of 287, a result that is consistent with Figure 9.

Our second example of a control system addresses the use of filters as described in [19]. Figure 11 displays a block diagram of an adaptive control scheme for the Apache HTTP Server. A filter with Z-transform $\frac{1-\alpha}{z-\alpha}$ is inserted after the server to smooth the stochastics of the system. There are two reasons for this. The first is to avoid having the controller respond to noise. The second is that adaptation provided by the parameter estimator should based on long-term changes in the measured output, not short-term transients.

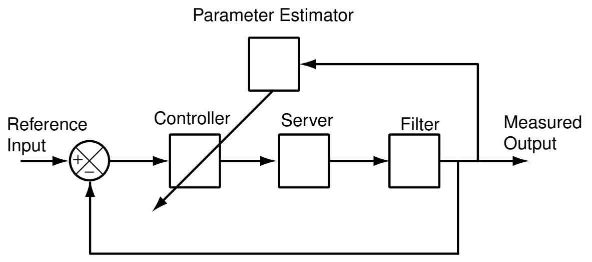

Fig. 11. Block diagram of an adaptive controller for the Apache HTTP Server.

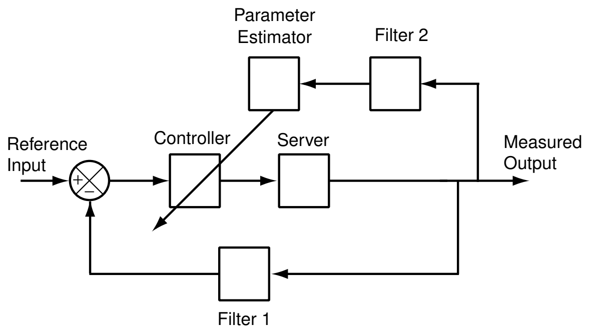

Fig. 12. Block diagram of an adaptive controller for the Apache HTTP Server in which separate filters are used for the parameter estimator and the calculation of the control error.

However, there is a problem with this design. If we use a moderate value of α, say $\alpha = 0.5$, then settling time is long, 3.5 minutes. Reducing α to 0.3 reduces settling time considerably, but it also increases the variability of the measured output due to excessively rapid adaptation of parameters. The issue is that the controller needs to operate in seconds or minutes, which requires a smaller α. However, the time constant of the parameter estimator should be in tens of minutes, which demands a larger α. The result is that we either end up with very long settling times or highly variable parameter estimates.

These considerations led to the design in Figure 12 in which separate filters are used for the controller and the parameter estimator. Filter 1, which is used to smooth controller output, has $\alpha = 0.3$. Filter 2, which is used for parameter estimation, has $\alpha = 0.5$. The new design reduces settling times without a substantial change in output variability.

V. RELATED WORK

Since the early 1990s, there has been broad interest in the application of control theory to computing systems, especially in the areas of data networks operating systems, middleware (e.g., web servers, database servers), multimedia, and power management. Below, we summarize these efforts.

In the area of data network, there has been considerable interest in applying control theory to problems of flow control. One of the first, [20], develops the concept of a Rate Allocating Server that regulates the flow of packets through queues. Others have applied control theory to short-term rate variations in TCP (e.g., [21]) and some have consider stochastic control [22]. More recently, there have been detailed models of TCP developed in continuous time (using fluid flow approximations) that have produced interesting insights into the operation of buffer management schemes in routers (see [17], [9]). The area of Asynchronous Transfer Mode (ATM) Networks has been an area of intensive exploitation of control theory in the 1990s (e.g., [23], [24], [25], [26], [27], [28]). However, the limited success of ATM technology and the use of continuous time and/or advanced control techniques (e.g., stochastic control), meant that there was little adoption of control theory by computing practitioners.

Although not nearly as prodigious, there has been considerable interest in applying control techniques to operating systems as well. [4] describes the details of control techniques widely used in IBM's Multiple Virtual Storage (MVS) operating system to achieve several kinds of service level objectives. The foregoing is primarily based on detailed knowledge of the operating system's control inputs and measured outputs. Others have proposed approaches that require little knowledge of details, relying instead on learning algorithms (e.g., [29]).

One of the most recent areas in which control theory has been applied is to middleware. Middleware are software systems that facilitate the development of robust, enterprise level applications. Examples include application servers (e.g., the Apache HTTP Server), database management systems (e.g., IBM's Universal Database Server), and email servers (e.g., the IBM Lotus Domino Server). There are three types of control problems that are typically addressed. The first is to provide a capability for enforcing service level agreements in that customers receive the service levels for which they contracted. Often referred to as service differentiation, this is achieved by enforcing relative delays [15], preferential caching of data [18], or in special cases modifying application codes to insert effectors (e.g., [30]). A second problem is to regulate resource utilizations so that they are not excessive, either because of reliability considerations (e.g., some software systems become fragile at heavy loads) or because of system design (e.g., to allow spare capacity for fail overs). Examples here include a mixture of queueing and control theory used to regulate the Apache HTTP Server [31], regulation of the IBM Lotus Domino Server [7], and multiple-input, multiple-output control of the Apache HTTP Server (e.g., simultaneous regulation of CPU and memory resources) [8]. The third problem that is often addressed is to optimize the system configuration, such as to minimize response times [32].

Management of multi-media streams has also been an area of focus for applying control theory to computing systems. The challenge here is that end-user performance is related to receiving an regular flow of correlated streams of data (e.g., voice and video) whereas the underlying systems operate more on a contention basis (e.g., execution priority). One solution to this is to regulate process priorities in accordance with the desired service levels (e.g., [33]). Another approach is to develop a control framework in which to build the capabilities for providing these service levels (e.g., [34]).

There is one last area we mention in passing—dynamic power management. The expense and engineering complications associated with supplying power to computational elements has motivated intensive investigations into how power can be managed within computing elements. Considerations here include addressing nonstationary service requests [35], the success of which largely depends on being able to model dynamics. More extensive discussions of power-aware computing can be found in [36] and related articles in the same issue.

There is a vast literature on load balancing, including its use in multiple source routing [37], implementations for L4 switches [38], techniques for balancing loads in data warehouses [39], and redirection algorithms for web-server systems [40]. There have also been studies that analyze general strategies, especially static load balancing (which makes use of long-term trends) versus dynamic load balancing (which exploits current changes in state) [41].

We close this discussion by pointing to an overview of the application of control techniques to computing in [42] and related articles in the same issue.

VI. CONCLUSIONS

There are several commonly occurring control problems in computing systems—translating between service oriented units (e.g., response times) and effector units (e.g., the maximum number of connected users), optimizing resources, regulating service levels, and rejecting disturbances (e.g., variations in workloads). Developing control systems to address these problems involves a number of challenges. Modeling the managed element (plant) requires dealing with stochastics and non-linearities. Our experience (which is consistent with many other researchers) is that simpler models work better in that they are easier to construct and tend to be more robust. A second challenge is that sensor data is noisy, incomplete, and inconsistent. In practice, we find that substantial effort is often required to change sensors to correct these shortcomings. Third, effectors have complex effects that often do not correspond well to the control objectives. Again, our experience is that these behaviors must be changed, which often results in product modifications. Last, the computer science community is often naive about what constitutes a control system in that the focus is entirely on the controller. That little attention is paid to filters, the choice of measured outputs, and time delays can lead to poor control performance, especially long settling times and/or significant oscillations.

Over the last two years, we have had considerable success in IBM with applying control theory to computing systems, often resulting in systems that operate more consistently and avoiding extreme behaviors such as limit cycles. This success has motivated us to evangelize the application of control theory to computing systems through tutorials (e.g., [43] and [44]), a book on control theory for computer scientists [45], and a class we are teaching at Columbia University on control theory for computer scientists (CS 6998-4). Our experience to date has been that some radical revisions are needed in the way control theory is taught in order to make it accessible to the computer science community. Beyond using examples drawn from computing systems, we work entirely in discrete time, do no frequency analysis, and almost exclusively use pole placement design. Further, we include material on system identification since this is one of the most challenging aspects of applying control theory in practice. Thus far, the response to this approach has been quite good. We find that students with only a modest mathematical background quickly grasp the key concepts and are able to apply control theory to design problems in computing systems.

References

[1] C. Ward, M. J. Buco, R. N. Chang, and L. Z. Luan, "A generic sla semantic model for the execution management of e-business," in *Third International Conference on E-Commerce and Web Technologies*, Sept. 2002.

[2] I. B. M. Corporation, "An architectural blueprint for autonomic computing," Tech. Rep. http://www-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf, IBM Corporation, 2004.

[3] J. Humphreys and M. Melenovsky, "Enabling business agility through server blade technology," Tech. Rep. P508.872.8200, IDC, 2003.

[4] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive algorithms for managing a distributed data processing workload," *IBM Systems Journal*, vol. 36, no. 2, 1997.

[5] Y. Diao, J. L. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano, "Using MIMO Linear Control for Load Balancing in Computing Systems," in *American Control Conference*, June 2004.

[6] E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. Hellerstein, L. Hsiung, M. R. T. Mummert, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye, "Dynamic surge protection: An approach to handling unexpected workload surges with resource actions that have lead times," in *IFIP Distributed Systems Operations and Control*, Oct. 2003.

[7] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, J. Bigus, and T. S. Jayram, "Using control theory to acheive service level objectives in performance management," *Real-time Systems Journal*, vol. 23, pp. 127–141, 2002.

[8] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server," in *IEEE/IFIP Network Operations and Management*, April 2002.

[9] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "A control theoretic analysis of RED," in *Proceedings of IEEE INFOCOM '01*, (Anchorage, Alaska), Apr. 2001.

[10] A. Tannenbam, *Computer Networks*. Prentice Hall, 4th ed., 2002.

[11] T. O. Group, *Systems Management: The Universal Measurement Architecture*. The Open Group, 1997.

[12] J. L. Hellerstein, "Achieving service rate objectives with decay usage scheduling," *IEEE Transactions on Software Engineering*, vol. 19, no. 8, pp. 813–825, 1993.

[13] J. Kay and P. Lauder, "A fair share scheduler," *Communications of the ACM*, vol. 31, no. 1, pp. 44–55, 1988.

[14] P. Goyal, H. M. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integratred servicepacket switching networks," in *ACM SIGCOMM*, 1996.

[15] T. F. Abdelzaher and N. Bhatti, "Adaptive content delivery for Web server QoS," in *International Workshop on Quality of Service*, (London, UK), June 1999.

[16] S. Mascolo, "Classical control theory for congestion avoidance in high-speed internet," in *Proceedings of the 38th Conference on Decision & Control*, Dec. 1999.

[17] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *Proceedings of IEEE INFOCOM '01*, (Anchorage, Alaska), Apr. 2001.

[18] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated caching services: A control-theoretic approach," in *International Conference on Distributed Computing Systems*, Apr. 2001.

[19] R. Zhang, S. Parekh, Y. Diao, and M. Surendra, "Improving the adaptation speed of a self-managing web server," in *Personal Communication*, 2004.

[20] S. Keshav, "A control-theoretic approach to flow control," in *Proceedings of ACM SIGCOMM '91*, Sept. 1991.

[21] K. Li, M. H. Shor, J. Walpole, C. Pu, and D. C. Steere, "Modeling the effect of short-term rate variations on tcp-friendly congestion control behavior," in *Proceedings of the American Control Conference*, pp. 3006–3012, 2001.

[22] E. Altman, T. Basar, and R. Srikant, "Congestion control as a stochastic control problem with action delays," *Automatica*, vol. 35, pp. 1936–1950, 1999.

[23] L. Benmohamed and S. M. Meerkov, "Feedback control of congestion in packet switching networks: the case of a single congested node," *IEEE Transactions on Networking*, vol. 1, Dec. 1993.

[24] C. E. Rohrs, R. A. Berry, and S. J. O'Halek, "A control engineer's look at ATM congestion avoidance," in *GLOBECOM*, pp. 1089–1094, 1995.

[25] P. Johansson and A. Nilsson, "Discrete time stability analysis of an explicit rate algorithm for the abr service," in *IEEE ATM Workshop*, pp. 229–350, 1997.

[26] N. S. Shankar and A. Shivaprasad, "An Instantaneous Control model for Flow Control in an ATM Network," in *International Conference on Information, Communications and Signal Processing*, pp. 878–882, 1997.

[27] A. Pitsillides, Y. A. Sekercioglu, and G. Ramamurthy, "Effective control of traffic flow in atm networks using fuzzy explicit rate marking (ferm)," *Journal on Selected Areas in Communications*, vol. 15, no. 2, pp. 209–225, 1997.

[28] S. Mascolo, D. Cavendish, and M. Gerla, "ATM rate based congestion control using a smith predictor: an EPRCA implementation," in *Proceedings of IEEE INFOCOM '96*, 1996.

[29] J. Bigus, "Applying neural networks to computer system performance tuning," in *IEEE International Conference on Neural Networks*, pp. 2442–2447, 1994.

[30] S. Parekh, K. Rose, J. L. Hellerstein, S. Lightstone, M. Huras, and V. Chang, "Managing the performance impact of administrative utilities," in *IFIP Conference on Distributed Systems Operations and Management*, 2003.

[31] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing model based network server performance control," in *IEEE RealTime Systems Symposium*, 2002.

[32] Y. Diao, J. L. Hellerstein, and S. Parekh, "Optimizing quality of service using fuzzy control," in *Distributed Systems Operations and Management*, 2002.

[33] S. Selvakumar and S. V. Raghavan, "Differential priority-based adaptive rate service discipline for qos guarantee of video stream," *Computer Communications*, vol. 20, no. 13, pp. 1160–1174, 1997.

[34] B. Li and K. Nahrstedt, "Control-based middleware framework for quality of service applications," *IEEE Journal on Selected Areas in Communication*, 1999.

[35] E. Chung, L. Benini, A. Bogliolo, Y. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests,"

IEEE Transactions on Computers, vol. 51, no. 11, pp. 1345–1361, 2002.

[36] M. R. Stan and K. Skadron, “Power-aware computing,” *IEEE Computer*, vol. Dec., pp. 35–38, 2003.

[37] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. Yang, “Load balancing of multipath source routing in ad hoc networks,” in *International Conference on Communications*, 2002.

[38] J. Hyun, I. Jung, J. Lee, and S. Maeng, “Content sniffer based load distribution in a web server cluster,” *IEICE Transactions on Information and Systems*, vol. E86-D, no. 7, 2003.

[39] H. Marteins, E. Rahm, and T. Stohr, “Dynamic query scheduling in parallel data warehouses,” *Concurrency Computation Practice and Experience*, vol. 15, no. 11-12, 2003.

[40] V. Cardellini, M. Colajanni, and P. S. Yu, “Request redirection algorithms for distributed web systems,” *IEEE Transactions on Parallel and Distribued Systems*, vol. 14, no. 4, pp. 355–368, 2003.

[41] H. Kameda, E. S. Fathy, I. Ryu, and J. Li, “A performance comparsion of dynamic vs. static load balancing policies in a mainframe – peronal computer network model,” in *Proceedings of the 39th IEEE Conference on Decision and Control*, IEEE, 2000.

[42] R. Sanz and K. Arzen, “Trends in software and control,” *IEEE Control Systems Magazine*, vol. June, pp. 12–15, 2003.

[43] J. Hellerstein and S. Parekh, “An introduction to control theory with applications to computer science,” in *ACM Sigmetrics Tutorial*, 2001.

[44] Y. Diao, J. Hellerstein, and S. Parekh, “Fast track introduction to control theory for computer scientists,” in *IEEE/IFIP Integrated Management*, 2003.

[45] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004 (expected).