Real-Time Systems, 23, 127–141, 2002

Using Control Theory to Achieve Service Level Objectives In Performance Management

S. PAREKH sujay@us.ibm.com
IBM

N. GANDHI gandhin@engin.umich.edu
University of Michigan

J. HELLERSTEIN hellers@us.ibm.com
IBM

D. TILBURY tilbury@umich.edu
University of Michigan

T. JAYRAM jayram@us.ibm.com
IBM

J. BIGUS bigus@us.ibm.com
IBM

Abstract. A widely used approach to achieving service level objectives for a software system (e.g., an email server) is to add a controller that manipulates the target system's tuning parameters. We describe a methodology for designing such controllers for software systems that builds on classical control theory. The classical approach proceeds in two steps: system identification and controller design. In system identification, we construct mathematical models of the target system. Traditionally, this has been based on a first-principles approach, using detailed knowledge of the target system. Such models can be complex and difficult to build, validate, use, and maintain. In our methodology, a statistical (ARMA) model is fit to historical measurements of the target being controlled. These models are easier to obtain and use and allow us to apply control-theoretic design techniques to a larger class of systems. When applied to a Lotus Notes groupware server, we obtain model-fits with R^2 no lower than 75% and as high as 98%.

In controller design, an analysis of the models leads to a controller that will achieve the service level objectives. We report on an analysis of a closed-loop system using an integral control law with Lotus Notes as the target. The objective is to maintain a reference queue length. Using root-locus analysis from control theory, we are able to predict the occurrence (or absence) of controller-induced oscillations in the system's response. Such oscillations are undesirable since they increase variability, thereby resulting in a failure to meet the service level objective. We implement this controller for a real Lotus Notes system, and observe a remarkable correspondence between the behavior of the real system and the predictions of the analysis. This indicates that the control theoretic analysis is sufficient to select controller parameters that meet the desired goals, and the need for simulations is reduced.

Keywords: control theory, system modeling, performance management

1. Introduction

Wide-spread reliance on IT systems has focused increasing attention on service level management, especially achieving response time and throughput objectives. A commonly

used approach is to take an existing target system and add a controller that has access to the metrics and tuning parameters of the system. Based on the feedback information present in the metrics, the controller manipulates the tuning parameters to achieve the desired service level objectives. Examples of such closed-loop software systems abound: the network dispatcher (Iyenger et al., 2000; Hunt et al., 1998), which adjusts load balancing parameters in clusters of web servers; the Multiple Virtual Storage (MVS) workload manager (Aman et al., 1997), which adjusts memory allocations and other operating system tuning parameters to achieve response time and throughput objectives; and fair share schedulers (Essick, 1990), which adjust Unix *nice* tuning parameter to achieve fractional allocations of CPU.

While considerable attention has been focused on the software mechanisms needed to enable closed-loop systems (e.g., instrumentation and tuning control access), much less attention has been paid to a rigorous evaluation of the behavior of the controller. Computer scientists most frequently use simulations to understand and evaluate controller behavior. However, this approach can be time-consuming, expensive, and error prone. The design of feedback control systems is also studied extensively in other engineering disciplines, such as mechanical and aeronautical engineering. Here, designers most often employ linear control theory (Ogata, 1997), which provides sound and rigorous mathematical principles for the design and analysis of closed-loop systems.

The classical controller design methodology consists of two steps:

1. *System identification*: Construct a transfer function which relates past and present input values to past and present output values. These transfer functions constitute a model of the system.

2. *Controller design*: Based on properties of the transfer function and the desired objectives, a particular control law is chosen. Techniques from control theory are used to predict how the system will behave once the chosen controller is added to it.

The goal of our work is to develop a methodology for, and assess the value of, applying control theory to the evaluation of controllers used for service level management of software systems. This is not a novel idea: the application of control theory to analyzing software systems has been prevalent in the networking arena. This work has generally used a first-principles approach to system identification. Chiu and Jain (1989) derive optimal convergence and fairness policies for congestion avoidance based on detailed knowledge of (or assumptions about) the protocol, workload, losses, etc. Keshav (1991) provides a more detailed analysis for performing flow control in a network with a very specific set of assumptions about the networking infrastructure and protocols. More recent work includes Mascolo et al. (1999) and Shor et al. (2000), both of which apply control theoretic ideas to analyzing the congestion control behavior in TCP. Work by Benmohamed and Meerkov (1993) for packet-switched networks, and further extended by Mascolo et al. (1996) for ATM also follow the first-principles approach. Both these papers have a very detailed system model and they do some sophisticated analyses. However, it is not clear how their ideas would generalize to other systems. Researchers have also applied control theory to QoS-oriented systems, such as OS schedulers (Steere

et al., 1999), distributed multimedia systems (Li and Nahrstedt, 1990) and QoS-aware caches (Lu et al., 2001).

Unfortunately, there are several short-comings with a first-principles approach. First, for complex systems, it is difficult to construct a model from first principles, so often some unrealistic assumptions are made. This difficulty has been a major barrier to applying control theory to computer systems. Second, the first-principles models often employ detailed information about the target system. Since these details may change frequently (e.g., with each software release), a first-principles approach may require expert involvement on an on-going basis. This is expensive and often impractical. Third, the first-principles approach often does not address model validation. Without model validation, it is unclear how the insights obtained using control theory relate to the system being studied.

Rather than proceeding from first principles, we advocate an empirical approach to system identification using statistical techniques. This approach treats the system as a black box, and thus, is not very influenced by system complexity or lack of expert knowledge. We believe that our approach enables the application of these techniques to a wider variety of systems than the traditional approach. As another alternative, Bigus (1993) uses neural networks for system modeling. While this approach also allows us to treat the target system as a black box, the resultant models are not as transparent as the ARMA models. While neural networks are non-linear and they can model more complex systems, we would not able to leverage any of the control-theory analysis tools for designing or analyzing controllers.

In this paper, we demonstrate the appeal and power of a control-theoretic analysis on a controller for doing admission control of a Lotus Notes workgroup server. For the controller law, we analyze the behavior of a saturated integral controller. The particular form of the models allows us to use standard techniques from control theory to perform the analysis. Our results demonstrate that there is a remarkable correspondence between the predictions made by control theory and the observed behavior of an actual Notes server.

The remainder of this paper is organized as follows. Section 2 describes the Notes server and how this target system is embedded into a closed-loop to achieve service level objectives. Section 3 details our approach to system identification. Section 4 discusses controller design and uses empirical studies to access the accuracy of insights obtained from control theory. Finally, we provide a summary and future work discussion in Section 5.

2. Lotus Notes and Its Closed-Loop Control

Architecturally, Lotus Notes is a client–server system. Client software converts high-level user activity (mouse clicks, etc.) into remote procedure calls (RPCs) that are sent to the server. The server maintains a queue of these in-progress RPCs. Once an RPC is serviced, an entry is made in the server log, and the appropriate response is sent to the client. Clients operate in a synchronous manner—waiting for the previous request to complete before sending a new request. The client–server protocol is session-oriented. A new

session is begun after a session-initiating RPC is accepted by the server. We use the term offered load to refer to the load imposed on the server by client requests. In the case of homogeneous clients, offered load is expressed in terms of the number of clients. Our service level metric is the length of the queue of in-progress RPC requests, hereafter just referred to as queue length.

The tuning parameter SERVER_MAXUSERS (hereafter referred to as MAXUSERS) regulates the number of users allowed to access the server at any time. This is a session-level control (as opposed to packet-level RPC controls). It operates by rejecting session-initiating RPCs once the number of connected users exceeds MAXUSERS. As such, this parameter has a somewhat complex effect on queue length. For example, changing MAXUSERS has no effect until a session-initiating RPC arrives, so existing sessions are not affected.

Since the Notes server does not provide a way to obtain direct measurements of RPC rates and queue length, we use a measurement sensor that samples the server log at a rate of once a minute. The queue length computation is performed by counting RPCs that were active in the previous time quantum. However, since RPCs currently waiting in the queue are not present in the log, this approach underestimates the true queue length and true RPC rates. That is, measurement is lossy. We can improve the approximation by delaying one or more time units before performing the measurements since doing so allows more RPCs to complete and hence gives us a more accurate count of the RPCs that were executing.

Notes administrators are keenly interested in controlling queue length since this provides a way to manage trade-offs between response times and throughputs. Figure 1 shows a closed-loop system to control Notes queue length. The administrator specifies a desired value, or reference value, for queue length. This value specifies a management policy that the closed-loop system tries to achieve. Based on the current and past values of the control error (the difference between the reference value and measured queue length), the controller adjusts the value of MAXUSERS. The algorithm for computing this adjustment is called the control law.

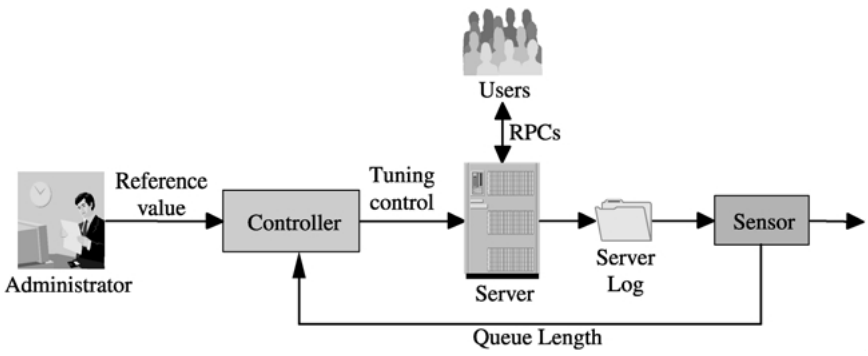

Figure 1. Closed-loop control of Lotus Notes.

3. System Identification and Validation

In system identification, we identify the main components of the system, the data values that flow in and out of them, and the mathematical relationships between these values. This section describes our approach to system identification and its application to Lotus Notes. The particular form of the models we construct (linear transfer functions) is important because it enables us to leverage a large set of analysis techniques that are available in control theory.

A block diagram such as Figure 1 is useful for component identification. We see that RPC rates and `MAXUSERS` are inputs to the Notes server, and queue length is the output. However, since `MAXUSERS` has an indirect effect on RPC rates, the two inputs are not independent and hence they do not combine in a linear fashion. To address this, we divide the operating region of the Notes server into two regions. When `MAXUSERS` $>$ offered load, the tuning parameter has no effect so we can ignore this case. But when `MAXUSERS` $\leq$ offered load, exactly `MAXUSERS` users are allowed onto the system. Here, the offered load value is not relevant (as long as we stay in this region) and hence it can be ignored. In other words, there is no need to consider RPC rates as an input.

This results in the single-input, single-output block diagram of Figure 2.

Can we adequately model the Notes server if `MAXUSERS` is the only input to the transfer function? To answer this question, Figure 3 plots queue length where the offered load is 300 users and `MAXUSERS` increased by 20 every 20 minutes. (These data are obtained using the experimental set-up described in Section 4.4.) The impact of `MAXUSERS` is clear, suggesting that it would be sufficient. A more quantitative assessment is provided later.

Let us now mathematically specify the input–output relationships. Throughout, we assume that time is discrete with uniform interval sizes. For a general linear system with input $x(t)$ and output $y(t)$, we construct an autoregressive, moving average (ARMA) model of the form

$$y(t) = \sum_{i=1}^{n} a_i y(t-i) + \sum_{j=0}^{m} b_j x(t-j) \tag{1}$$

where (n, m) is the order of the model, and the a_i, b_j are constants that are estimated from data. For analysis purposes, we apply the principles of z-transform theory (Ogata, 1997) to this equation. We use the standard notation that the z-transform of $x(t)$ is $X(t)$, and

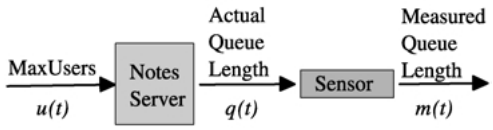

Figure 2. Model of Notes server (open-loop).

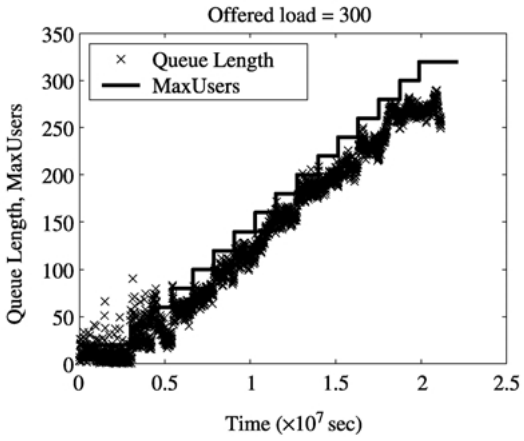

Figure 3. Effect of MAXUSERS on queue length.

similarly for $y(t)$, $u(t)$, etc. Assuming that $n \geq m$ (which is a typical constraint), we obtain a corresponding z-domain transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{j=0}^{m} b_j z^{n-j}}{z^n - (\sum_{i=1}^{n} a_i z^{n-i})} \tag{2}$$

Now refer to the quantities depicted in Figure 2. For $N(z)$, the transfer function for the Notes server, it turns out that a good fit is obtained if $n = 1$ and $m = 0$. For $S(z)$, the transfer function of the sensor, we can use $n = 1$ and $m = 1$ (see Table 1). The transfer functions for these components are:

$$N(z) = \frac{Q(z)}{U(z)} = \frac{zb_0}{z - a_1} \quad \text{and} \quad S(z) = \frac{M(z)}{Q(z)} = \frac{zb_0 + b_1}{(z - a_1)z^d} \tag{3}$$

Recall that to get an accurate estimate of $q(t)$, we could delay d time units so that long-running RPCs present during time t complete their execution. The term z^d in $S(z)$ represents this delay.

Note that in modeling $N(z)$ and $S(z)$, we have treated the components as a black boxes, and have not used any details about their internal operation (e.g., caching policies, buffer sizes, etc.).

Table 1. Model R^2 values and coefficients.

	Delay	R^2	a_1	b_0	b_1
Notes server	N/A	97.6	0.4261	0.4709	0.0
Sensor	0	75.5	0.6371	0.1692	− 0.1057
	1	83.7	0.7991	0.7182	− 0.6564
	2	91.2	0.9237	0.9388	− 0.9092

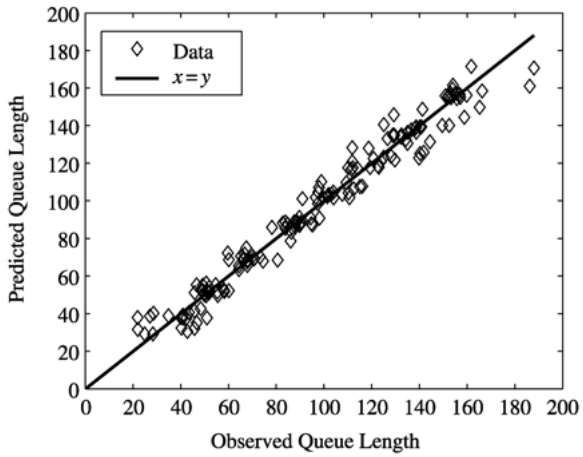

Figure 4. Comparing $N(z)$ model predictions with observed values.

We use a statistical approach to estimate the parameters (the a_i, b_j) of $N(z)$ and $S(z)$. First, measurements of the target system are obtained while varying the input parameters in a controlled way, such as the data in Figure 3. Then, we use least-squares regression to estimate the a_i, b_j for different values of (n, m). In general the fit of the model improves as n, m are increased. We seek a model that has adequate fit and a low order.

As mentioned before, a first order model provides a good fit for both $N(z)$ and $S(z)$. Table 1 reports values of the a_i, b_j and R^2 for these transfer functions. For the Notes server, R^2 is 98%, which is an excellent fit. The quality of this model can be further assessed by plotting observed values of queue length versus those predicted by the model, shown in Figure 4. Note that almost all observations lie close to the line of unit slope where the predicted value equals the actual value. For the Sensor transfer function, R^2 is smaller, although still acceptable. Note that as d increases so does R^2, and a_1 approaches 1. Both effects are expected since with longer delays, measured values approach the actual value.

4. Controller Design and Assessment

The next step is to design and assess one or more controllers. In this section, we describe how to construct the controller from a control law. We assume that the system is linear, but since this assumption does not always hold for real systems, we first determine the conditions for linearity. We then use control theory to gain insights into controller behavior, especially the presence of controller-induced oscillations. These predictions are assessed using measurements of a real Notes server.

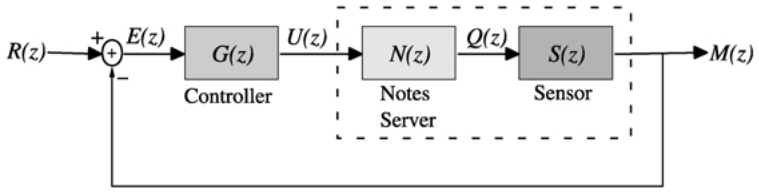

Figure 5. System with controller.

4.1. Control Law and Closed-Loop Analysis

Based on the transfer function of the open-loop target system of Figure 2, we construct a transfer function of a closed-loop system, as shown in Figure 5. For the control law, we focus on integral control (Ogata, 1997), a widely used technique that is a reasonable approach for the Notes server. Only one control law is considered since our objective is to demonstrate the value of our methodology. A time-domain expression of the integral control law is $u(t) = u(t-1) + K_i e(t)$, where $u(t)$ is the new control value at time t, and $e(t) = r(t) - m(t)$ is the control error. The parameter $K_i > 0$ is called the *gain*. Intuitively, this control law dictates that `MAXUSERS` be adjusted incrementally based on its previous value and the gain-weighted control error. Higher K_i values lead to a faster response. However, care is required since larger values of K_i can cause oscillations or even instabilities. The controller transfer function is

$$G(z) = \frac{U(z)}{E(z)} = K_i \frac{z}{z-1} = K_i D(z) \tag{4}$$

Solving the equations that are implied in Figure 5, we get the following closed-loop transfer function for the system in Figure 5:

$$\begin{aligned} M(z) &= E(z)*K_i D(z) N(z) S(z) \\ E(z) &= R(z) - M(z) \\ \frac{M(z)}{R(z)} &= \frac{K_i D(z) N(z) S(z)}{1 + K_i D(z) N(z) S(z)} \end{aligned} \tag{5}$$

4.2. Linearity Analysis

The problem with directly implementing this control law into software for controlling Lotus Notes is that if $|K_i e(t)|$ is too large, `MAXUSERS` is set to a value that exceeds its legal range. To avoid such situations, we limit the range of `MAXUSERS` by extending the control law: $\forall t : u_{\text{Min}} \leq u(t) \leq u_{\text{Max}}$. Such saturated controllers are not linear, which causes difficulty in the analysis. Hence, we perform a preliminary analysis to determine the values of K_i for which the control value stays in the linear range, and restrict our subsequent analysis to this range.

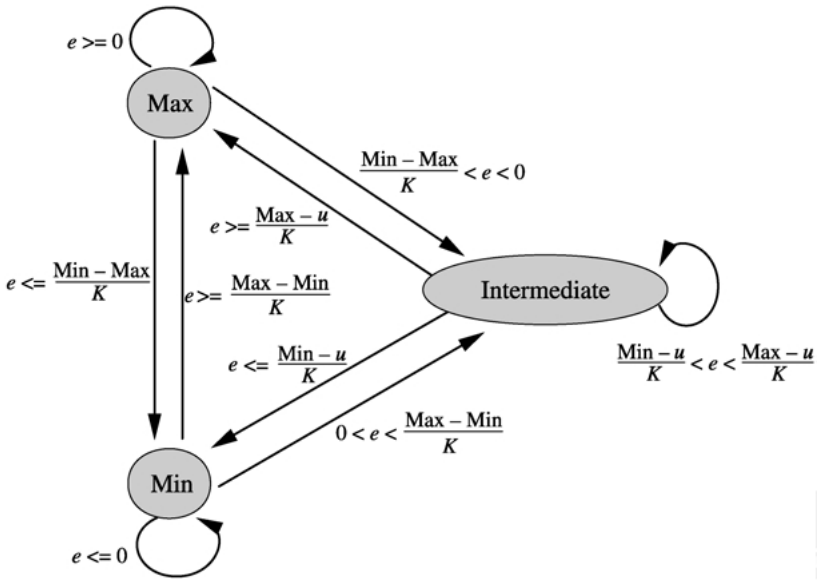

Figure 6. Controller state transitions.

We divide the control region into three parts: $u(t) = u_{\text{Max}}$, $u(t) = u_{\text{Min}}$ and $u_{\text{Max}} < u(t) < u_{\text{Min}}$. We designate these states as Max, Min, and Intermediate, respectively. We seek to understand the conditions under which control values will be in states Min and Max;. If we stay away from these regions, then the assumptions of our analysis should hold.

Figure 6 shows the state transitions obtained from the control law. We see that as $K_i \to \infty$, all transitions are between states Min and Max. Clearly, we want to avoid large K_i. How big can K_i be without encountering states Min or Max? Let ε be the largest error that occurs once the closed-loop system is in operation. Then, if $K_i < (\text{Max} - \text{Min})/\varepsilon$, we never transition into the extreme states. In our empirical studies of an uncontrolled Notes system, queue lengths range from approximately 20–100 if $d = 0$ and 60–140 if $d = 2$. So, if there is no bias, then in either case, $\varepsilon = 40$. We set Max = 200 so that it equals offered load, and Min = 1. That gives us: $K_i < 5$.

4.3. Analytical Studies

We now use classical control theory to evaluate the closed-loop system. We know from control theory that K_i should be as large as possible to provide a fast response. The issue addressed here is to predict when K_i will be so large that there are controller induced oscillations. Such a prediction is made by studying properties of the transfer function for the closed-loop system, that is Equation (5).

First, some background. The transfer functions we consider can be expressed as a ratio of two polynomials in z, e.g., $H(z) = A(z)/B(z)$. The roots of the numerator $A(z)$ are

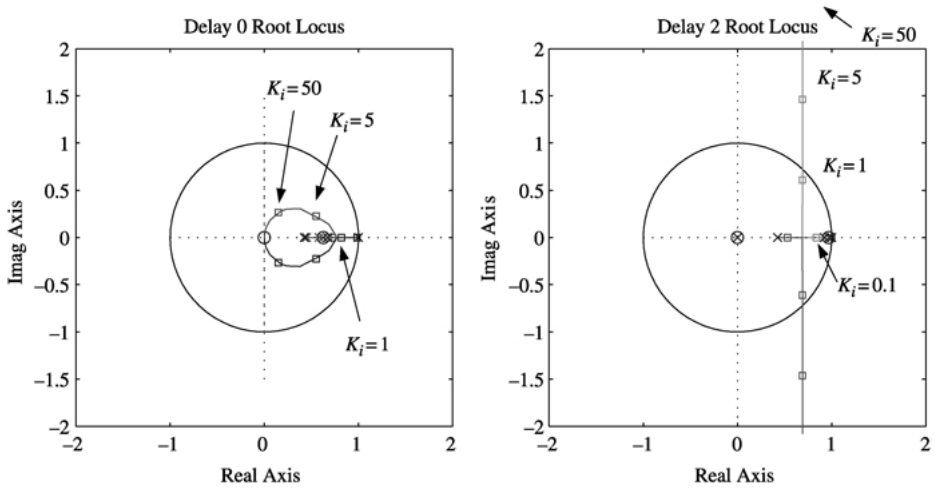

Figure 7. Root-locus plots for delay = 0 and delay = 2.

called its zeros and the roots of the denominator $B(z)$ are its poles. The poles and zeros of a transfer function provide insight into stability and controlled-induced oscillations. First, if any of the poles of $H(z)$ lie outside the unit circle, then $H(z)$ is unstable. That is, a bounded input produces an unbounded output. Second, poles for which $\mathrm{Im}(z) \neq 0$ imply time-domain terms of the form $e^{j\omega t}$, where $j = \sqrt{-1}$. This is a sinusoid and so oscillations are present in the output that increase its variability.

Root-locus plots provide a systematic way to study the location of poles in the Complex plane. Figure 7 shows root-locus plots for a unit step response (unit change in the tuning parameter) of the system described by Equation (5). Consider the left-most plot, which addresses $d = 0$. To provide a frame of reference, there is a unit circle centered at 0. The x's indicate poles in $G(z)N(z)S(z)$, and the o's indicate its zeros. The root-locus is the curve inside the unit circle that traces the poles as K_i increases from 0 to ∞. Since all poles lie within the unit circle, there is no problem with stability. Further, observe that for small K_i (e.g., 0.1 and 1), the poles lie on the real axis. Thus, there is no sinusoidal component associated with the step response for these gains. However, for larger K_i (e.g., 5 and 50), there is a non-zero imaginary component to the poles. This suggests the presence of controller-induced oscillations that increase the variance of queue length.

Now consider that the root-locus plot for $d = 2$, which is the right plot in Figure 7. While $K_i = 0.1$ lies on the real axis, poles for the other gains have non-zero imaginary components. Hence, we expect controller-induced oscillations that result in higher variability for queue length. We further observe that for $d = 2$, the pole at $K_i = 5$ lies outside the unit circle. This suggests a stability problem. Of course, the system cannot really become unstable since we have bounded the range of values that `MAXUSERS` is assigned. As we showed in Section 4.2 large gains can cause another problem—a limit cycle in which the tuning parameter only takes on values in $\{u_{\mathrm{Min}}, u_{\mathrm{Max}}\}$. We discuss this further in Section 4.4. The analysis thus reveals that if we wish to introduce a delay in

order to obtain more accurate queue length information, it severely limits the range of gain values that we may use, and thus limits the responsiveness of the control system. Thus, we have a rigorous way of trading off accuracy for recency in the sensor data.

4.4. Empirical Assessments

Here, we present empirical results for various values of K_i used for an integral controller of a real system with a synthetic workload. We study how the predictions made by control theory compare with the behavior of the real system.

The testbed for our experiments consists of a workload generator, product level Notes server, a sensor running on the Notes server, and a controller running on a third machine (so as not to perturb the Notes server). The workload generator simulates the activity of multiple clients by running copies of an identical script that sends RPCs to the server. These scripts are executed repeatedly with a one-minute delay between executions. The script was selected from the NotesBench suite, a standard for such workload generation. During the experiment, the offered load to the server (i.e., the number of users trying to issue requests) is kept constant at 200 users. The reference queue length is initially set to 10, and after 60 minutes it is changed to 25.

First consider the behavior of the open-loop system. Figure 8 displays the result for both $d = 0$ and $d = 2$. In the former, queue length hovers around 80. In the latter, it's around 100. These results are consistent with the fact that $d = 0$ is more lossy than $d = 2$. We also see substantial variability in both cases, with changes in queue length of 40 being common. This variability comes, in large part, from the client side.

When we close the loop, Figure 9 shows the effect of the integral controller with $K_i \in \{0.1, 1, 50\}$ and delays of 0 and 2. The figure consists of 12 plots presented in two columns. The left column is $d = 0$ and the right column is $d = 2$. There are three parts to

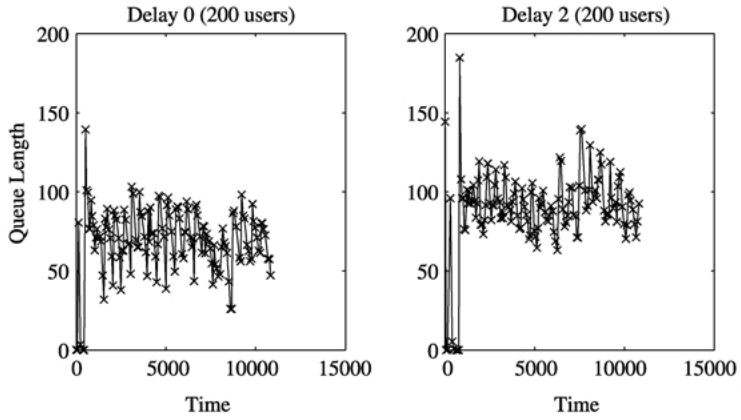

Figure 8. Uncontrolled system run (MAXUSERS > offered load).

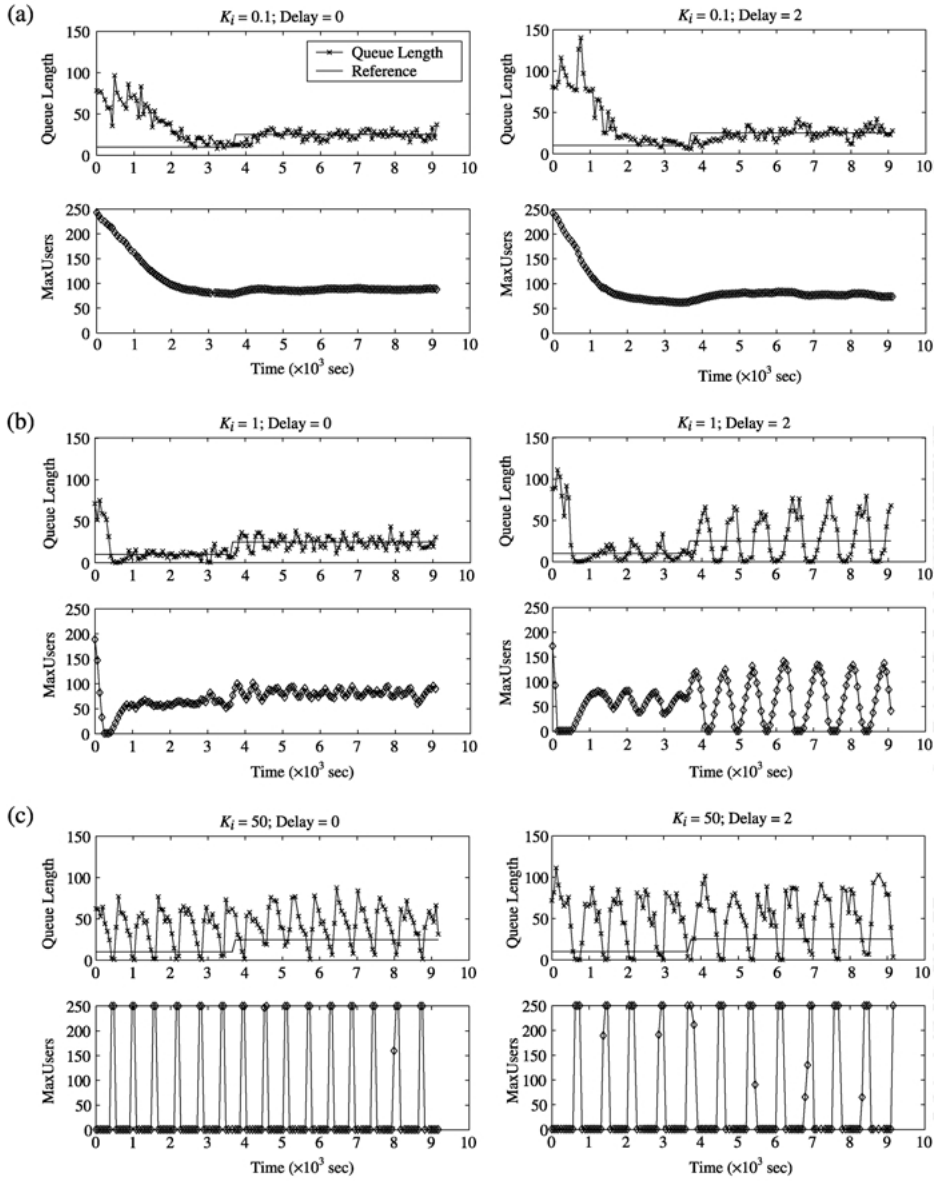

Figure 9. Effect of controller on a real system.

the figure, corresponding to each K_i. For example, part (a) consists of the first two rows of plots. The first row plots queue length (and the reference value) versus time. The second row shows the value of the control at the same time as the queue length plot. Part (b) does the same for $K_i = 1$, and Part (c) displays $K_i = 50$.

With $K_i = 0.1$, while there is an initial transient, the queue length converges to the reference value. There is some variability, but variance is considerably smaller than in Figure 8. This observation holds for both $d = 0$ and $d = 2$. These results are consistent with the root-locus analysis that found $\mathrm{Im}(z) = 0$ for $K_i = 0.1$, where z is a pole of the transfer function of the closed-loop system.

In part (b), gain has increased by a factor of ten. For $d = 0$, while variability is substantially larger than with $K_i = 0.1$, there does not appear to be a controller-induced oscillation. Also, queue length values remain centered on the reference value suggesting that bias is small. The situation for $d = 2$ is not so good. There is a pronounced cycle in the queue length values, which corresponds to a cycle in the values of the control. This suggests a controller-induced oscillation. We note that root-locus analysis predicted both of these results in that (a) there is no pole with a non-zero imaginary component for $K_i = 1$, $d = 0$ and (b) there is such a pole for $K_i = 1$, $d = 2$. The oscillations in queue length are the result of overcompensation. That is, a positive $e(t)$ causes the controller to increase `MAXUSERS` and this in turn causes $e(t + 1)$ to be so negative that `MAXUSERS` is greatly reduced, and so on.

In Part (c), we have an extreme example, one that clearly violates the constraint of $K_i < 5$ that was established in our preliminary analysis. There is a strong limit cycle for the control value, and the resultant queue length plot shows large oscillations. Changing the reference value has no apparent impact on the system's behavior.

Table 2 quantifies these results for the region where the reference value $r(t) = 25$. Note that standard deviations are small for those values of (K_i, d) that root-locus identified as only having real-valued poles. On the other hand, standard deviations are large for those values of (K_i, d) that do have complex poles. We also observe that for larger K_i and d, there is a problem with bias. This is indicated by large values of RMS and the difference between average queue length and the reference value of 25. This validates our analysis that for larger delays, we cannot get a fast-responding system (large K_i), otherwise the system quickly becomes unstable.

Table 2. Controller performance statistics.

		Queue length		
K_i	Delay	Mean	St. Dev.	RMS error
0.1	0	23.95	5.58	5.64
0.1	2	23.80	7.37	7.43
1	0	24.66	7.57	7.54
1	2	29.70	25.67	25.96
5	0	35.39	20.94	23.27
5	2	59.99	41.63	54.17
50	0	42.73	21.41	27.70

5. Summary and Future Work

In this paper, we have demonstrated a methodology for constructing and analyzing closed-loop systems using a statistical approach to system identification. This approach is more generally applicable than the conventional first-principles approach, and also has the potential to adapt to changes in the underlying system (such as new software releases). The fit for our models of a Lotus Notes email server is quite good: R^2 is no lower than 75%, and is as high as 98%.

Further, we illustrate the value of a control-theoretic analysis by applying it to an integral controller for the Notes server, where the controller manipulates the Notes `MAXUSERS` tuning parameter. Our control-theoretic analysis provides useful insights into the following trade-off of the gain value K_i: while a large gain makes the system more responsive, too large a gain can cause instabilities in the closed-loop system.

Since classical control theory requires that we restrict ourselves to linear regions of the controller's operation, we use a simple state analysis to restrict the gain values to a linear region. Root-locus analysis then allows us to predict which values of K_i cause controller-induced oscillations. Our empirical studies using a real Notes system confirm these predictions. Thus, we can analytically choose a value for K_i that allows the system to be responsive and yet not be subject to controller-induced oscillations. Moreover, the analysis clarifies the effect of sensor-induced delays.

Note that both system identification and controller design are performed off-line, based on data that has been collected from either controlled or production runs of the target system. This allows us to use a large amount of sample data, perform more time-consuming analyses and even consult domain experts. We assume that the system evolves slowly, if at all, so the model does not need to be estimated often. An online changepoint detection (Basseville and Nikiforov, 1993) scheme can be employed to actively monitor the system and trigger the parameter re-estimation when required. Online adaptation of the target that is within the bounds of the estimated model is performed by the controller.

Much work remains. Our approach to identifying linear regions of operation is approximate at best. A better approach would employ describing functions, a technique used in non-linear control theory. In this paper we have restricted ourselves to a simple control law in order to demonstrate the value of this approach. We plan to study more complex controllers to assess if control theory provides useful insights as to their operation. More broadly, we are interested in applying our methodology to other service level management situations both to refine our methodology and to asses its value.

References

Aman, J., Eilert, C. K., Emmes, D., Yocom, P., and Dillenberger, D. 1997. Adaptive algorithms for managing a distributed data processing workload, *IBM Systems Journal* 36(2): 242–283.

Basseville, M., and Nikiforov, I. 1993. *Detection of Abrupt Changes: Theory and Applications*. Prentice Hall.

Benmohamed, L., and Meerkov, S. M. 1993. Feedback control of congestion in packet switching networks: the case of a single congested node. *IEEE Transactions on Networking* 1: 693–708.

Bigus, J. P. 1993. *Adaptive Operating System Control using Neural Networks*. Phd Thesis, Lehigh University.

Chiu, D.-M., and Jain, R. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems* 17.

Essick, R. B. 1990. An event-based fair share scheduler. In *Proceedings of ACM USENIX*, pp. 147–161.

Hunt, G., Goldszmidt, G., King, R., and Mukherjee, R. 1998. Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, April 1998 (published as Computer Networks and ISDN Systems, Vol. 30, pp. 347–357.

Iyenger, A., Challenger, J., Dias, D., and Dantizig, P. 2000. High-performance web site design techniques, *IEEE Internet Computing* 4: 17–26.

Keshav, S. 1991. Proceedings of ACM SIGCOMM 91 (September 1991), Zurich, Switzerland (published as Computer Communication Review, Vol. 21, No. 4, pp. 3–15).

Li, B., and Nahrstedt, K. 1990. Control-based middleware framework for quality of service applications. *IEEE Journal on Selected Areas in Communication*, pp. 1632–1650.

Lu, Y., Saxena, A., and Abdelzaher, T. F. 2001. Differentiated caching services: A control-theoretic approach. In *International Conference on Distributed Computing Systems*, Phoenix, Arizona.

Mascolo, S., Cavendish, D., and Gerla, M. 1996. ATM rate based congestion control using a smith predictor: an EPRCA implementation. In *Proceedings of IEEE INFOCOM'96*, pp. 569–576.

Mascolo, S. 1999. Classical control theory for congestion avoidance in high-speed internet. In *Proceedings of the 38th Conference on Decision & Control*, pp. 2709–2714, Phoenix, Arizona.

Ogata, K. 1997. *Modern Control Engineering*. Prentice Hall, 3rd edn.

Shor, M. H., Li, K., Walpole, J., Steere, D. C., and Pu, C. 2000. Application of control theory to modeling and analysis of computer systems. In *Proceedings of the Japan-USA-Vietnam RESCCE Workshop*, Ho Chi Mihn City, Vietnam.

Steere, D. C., Goel, A., Gruenberg, J., McNamee, D., Pu, C., and Walpole, J. 1999. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, pp. 145–158.