RC23459 (W0412-008) December 7, 2004
Computer Science

IBM Research Report

Applying Control Theory to Computing Systems

Joseph L. Hellerstein, Yixin Diao, Sujay S. Parekh
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. Ithas been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distributionoutside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

Applying Control Theory to Computing Systems

Joseph L. Hellerstein, Yixin Diao, Sujay Parekh

IBM Thomas J. Watson Research Center, Hawthorne, NY

Email: {hellers, diao, sujay}@us.ibm.com

A few techniques from control have yielded significant benefits to an IBM software product.

Feedback control is an essential part of computing systems. For example, UNIX[1]® CPU schedulers often use process execution times to adjust CPU priorities, which in turn impact the future execution times of processes. Virtual memory systems use fault rates to allocate physical frames, which in turn affects future fault rates. And, congestion control in networks uses round trip times to adjust window sizes, which in turn impacts future round trip times. Unfortunately, designers of computing systems rarely approach feedback in a systematic way.

In mechanical, electrical, aeronautical and other areas of engineering, control theory is used to systematically design feedback control systems, which are also referred to as closed loop systems. This has motivated us and researchers elsewhere to explore how to apply control theory to computing systems. For example, researchers at the University of Virginia and Hewlett Packard [1] along with the University of Illinois [2] have applied control theory to managing various aspects of the Apache™ Web Server[2] [3], and investigators at the University of Massachusetts [4] have applied control theory to the Internet Transmission Control Protocol

[1] UNIX ® is a registered trademark of The Open Group.
[2] Apache is a trademark of the Apache Software Foundation and is used with tradition.

(TCP). These studies focus on prototype systems in a laboratory setting, a kind of "science experiment".

Like our colleagues elsewhere, we began our explorations with a series of science experiments. Our initial focus was the IBM Lotus Domino® Server[3] [6], a commercial email system. However, our connections with IBM product teams have allowed us to see if the techniques used in the science experiments have value in practice. This has resulted in several control-theory based features in IBM's Universal Database for Linux, UNIX, and Windows (hereafter, just DB2). Having seen the practical value of applying control theory to computing systems, we have commenced a third phase in our work—educating software engineers on the theory and practice of applying control theory to computing systems. Our efforts here have included co-authoring a book on this topic [5] and teaching at Columbia University, Stanford University, and the University of California at Berkeley.

A Control Theory Science Experiment

Our application of control theory to the IBM Lotus Domino Server illustrates the insights that control theory can provide. Early versions of the Domino Server could encounter problems under heavy loads, where load is quantified as the number of remote procedure calls (RPCs) being processed concurrently by the server. To avoid problems, administrators regulated the number of RPCs in the server, a quantity that we abbreviate as **RIS**. Unfortunately, the Domino Server does not provide a configuration parameter to regulate RIS directly. Instead, administrators used the configuration parameter *MaxUsers* that controls the number of connected users. While *MaxUsers* is clearly an upper bound on RIS, the true relationship between the two

[3] IBM Lotus Domino is a registered trademark of the IBM Corporation.

depends on the activity of the end-users. For example, during lunch, RIS may be fairly small even though *MaxUsers* is quite large. However, towards the end of the day when user activity increases, RIS approaches *MaxUsers*. Administrators addressed this by manually adjusting *MaxUsers* to achieve the desired RIS.

Figure 1 depicts the feedback control system in our science experiment that adjusts *MaxUsers* automatically to achieve the desired RIS[4]. The administrator specifies the desired RIS. The controller inputs the control error, the difference between the desired RIS and measured RIS, and computes a new value of *MaxUsers.* This is input to the Domino Server, which in turn affects current RIS. Unfortunately, in our studies, we could not obtain the value of current RIS. Instead, we made use of a log file that is written when an RPC completes. Data collected in this way are called measured RIS, and the mechanism is called the Notes Sensor.

How should the controller compute the next value of *MaxUsers?* Borrowing from other engineering disciplines, we used an integral controller. Integral control is expressed as a difference equation: $u(k) = u(k-1) + K_I e(k)$, where k is the time index of the value, $u(k)$ is the k-th value of *MaxUsers*, $e(k)$ is the k-th value of the control error, and K_I is a constant that is determined by control design. A larger value of K_I results in bigger changes in *MaxUsers* in response to the control error.

Designing a controller requires considering several properties of the resulting closed loop system. First, the system should be stable. This means that a bounded change in workload or desired RIS should not cause an unbounded change in measured RIS. A second property is accuracy. By this we mean that in steady state, measured RIS should be the same as desired RIS.

[4] Technically, the values used in control diagrams are normalized around an operating point. Typically, normalization is achieved by subtracting the desired value from the original value.

Short settling times are also of importance so that the system adapts quickly to changes in workload and other considerations. Last, we want to avoid large overshoot, a situation in which measured RIS increases (or decreases) beyond what is needed to achieve desired RIS. In our class at Columbia University and elsewhere, we use the acronym **SASO** to refer to the properties of stability, accuracy, settling times, and overshoot.

Designing an integral controller means selecting a K_I that achieves the desired SASO properties. One approach is to conduct experiments with different values of K_I. Figure 2 depicts the results of three such experiments for a Domino Server with a standard workload. Desired RIS, which is denoted by *r(k)* in Figure 1, is initially 15 and is then changed to 25. We see that if K_I is 0.1, measured RIS, which is denoted by *y(k)*, settles slowly. Settling times are considerably shorter if K_I is 1. However, if K_I is 5, the system oscillates wildly and never settles.

While an empirical approach provides insights, it is a very time consuming way to do control design. Control theory provides a model-based approach to selecting K_I that addresses the SASO properties. The technical details require a modest understanding of discrete time linear systems, which is discussed in [5] among other sources. However, the basics are easily explained. There are two steps: (1) system identification and (2) control design. In system identification, experiments are conducted (either online or offline) to construct difference equations that describe the relationship between inputs to and outputs from components of the closed loop system. For example, two components in Figure 1 are the Domino Server and the Notes Sensor. The Domino Server has control input *MaxUsers* and its output is current RIS (denoted by *w*). From our experiments, these are related by the difference equation $w(k) = 0.43w(k-1) +$

0.47u(k−1). A similar approach is used to construct a model of the Notes Sensor. This results in the difference equation $y(k)=0.8y(k-1)+0.72w(k-1)-0.66w(k-2)$.

Control design makes use of these models by using transfer functions, a straight forward way of representing difference equations that provides an easy way to assess SASO properties. Transfer functions are expressed in terms of the complex variable z. For example, the transfer function of $w(k) = 0.43w(k-1) + 0.47u(k-1)$ is $W(z)=0.47(z-0.43)^{-1}$. It turns out that this representation tells us that the time domain solution of $w(k)$ has the term 0.43^k. Indeed, for transfer functions that have higher order polynomials in z, their solution at time k includes a sum of terms of the form p^k . Here, p is a pole, which is a value of z for which the denominator of the transfer function is 0. A system is stable if the poles of its transfer function lie within the unit circle. This make sense since if $|p|<1$, then p^k converges. Further, a stable system settles quickly if its poles are close to 0, and it settles slowly if its poles are close to 1. Last, for transfer functions with denominators that have higher order polynomials in z, there may be complex poles. Having a time domain solution with terms of the form p^k means that the system has oscillations if the poles are negative or imaginary. If there are multiple poles, then settling times are largely determined by the pole with the largest absolute value, which is called the dominant pole. As with the foregoing discussion of settling time, the accuracy and overshoot of a system can also be estimated from its transfer function.

We take a simple approach to control design called pole placement design. Pole placement design determines the value of controller constants such as K_I based on the desired poles the closed loop system. For example, we may want the absolute value of the largest pole of the closed loop system to be no greater than 0.44 so that the system settles in approximately 5

time steps. (See [5] for how to calculate this.) To find the poles of the closed loop system, we must first construct its transfer function. This is done by making use of the transfer functions of all the components of the system. In Figure 1, these components are the controller, the Domino Server, and the Notes Sensor. K_I is selected based on its effect on the closed loop poles. For example, a settling time of approximately 20 time steps is achieved if the dominant pole is approximately 0.8, which happens if K_I =1 in Figure 2. For comparison purposes, the dominant closed loop pole for K_I =0.1 is approximately 0.99, which explains the long settling times. For K_I =5, the poles become imaginary, which is consistent with the pronounced oscillations in Figure 2.

Value in Practice

Just as we were completing our studies of the Domino Server (and science experiments with the Apache Web Server), we were contacted by the IBM database development team to help with a problem they face with managing administrative utilities.

First, some background. The primary functions provided by database management systems are SQL parsing, construction of query plans, query execution, and run time management of database resources. However, administrative utilities are also needed to address recoverability (BACKUP, RESTORE), data reorganization (REBALANCE), statistics collection (RUNSTATS), and other concerns as well. Such administrative utilities have the following characteristics: (1) their execution is essential to the integrity of the system; but (2) they can severely impair the performance of production work if executed concurrently with that work. While administrative utilities used to run in off-peak periods, the advent of 24-by-7 operation as a result of globalization has forced administrators to run utilities concurrently with production work.

Our task was to develop a control system for throttling utilities in accordance with administrative policies. Before we could apply control theory, we first had to formulate the utilities throttling problem as a block diagram similar to Figure 1. Our first challenge was to specify the reference input. To do this, we had to examine more closely what policies administrators wanted. Our assessment was that policies should be expressed in terms of degradation of production work. A general form for such a policy is

`There should be no more than an` *x*`% performance degradation of production work as a result of executing administrative utilities.`

So, percent degradation corresponds to *r(k)* in Figure 1.

Now that we had a reference input, our second challenge was to find a knob comparable to *MaxUsers* in the Domino Server that would allow us to regulate the degradation of production work due to running utilities. The setting of this knob corresponds to the value of *u(k)* in Figure 1. Initially, we tried using operating system priorities. This proved to be a poor choice. Foremost, UNIX based operating systems typically only have CPU priorities, but many of the database utilities are I/O intensive, especially BACKUP. Second, operating system priorities only affect operating system resources, which do not include application resources such as database locks.

Ultimately, we were forced to modify utility codes to incorporate an appropriate control. The control we used is self-imposed sleep (**SIS)**. SIS assumes that the utility being throttled is structured as a repetitive loop (e.g., backup the next set of rows in a table). Implementing the SIS control involved inserting code at the beginning of the loop to do the following: (1) compute the time since beginning the last sleep by the utility; (2) compare this to the value of the desired

"duty cycle" for the utility; and (3) if the desired duty cycle is shorter, then sleep for a controller specified period of time that is referred to as sleep time. Our experience has been that it is relatively easy to incorporate SIS into utility codes.

With the SIS codes in place, our next challenge was to obtain measured values of the percent degradation of production work, which corresponds to *y(k)* in Figure 1. This proved particularly problematic. The issue is that both the production work and the resource demands of utilities vary with time. So, a degradation in the performance of production work may be due to giving too much resource to the utilities by not making sleep time sufficiently large. Or, it may be a result of a change in the nature of the production work.

One way out of this conundrum is to have a detailed characterization of the resource demands of the production work. However, such detailed knowledge is rarely available in practice. In the end, we employed a model-based approach that estimates the relationship between sleep time and the performance of production work. We establish the baseline level of production work by using the model to estimate performance if the utilities are always sleeping. Given this and current performance measurements, we can estimate the measured value of percent degradation.

The foregoing allowed us to implement a closed loop system using control theory, as detailed in [7]. In particular, we used control theory in controller design to understand the trade-off between adapting quickly to changes in workloads and avoiding oscillations in the performance of production work. More importantly, control theory provided a systematic approach to solving the throttling problem in that we separately considered the reference input, control input, measured output, and controller design. Utilities throttling was first released as part

of IBM's Universal Database Server version 8.1 for throttling a single utility, and has been enhanced to simultaneously throttle multiple utilities in version 8.2.

Recently, we have been asked to study another problem with database management systems—managing the allocation of memory pools. Examples of memory pools are buffer pools where data are cached, sort pools, lock lists, and temporary pools. Traditionally, the size of these pools has been specified statically by configuration parameters. However, as workloads change, performance can be degraded radically if pool sizes are not adjusted, a task that takes considerable knowledge of database systems. In essence, this is an optimization problem, which is quite different from the regulation problem we solved for utilities throttling. Even so, our recent results show that by properly changing the regulatory control architecture, we can address this optimization. Our preliminary studies show that compared to an expert tuning of memory pools, the control theory based approach yields almost a 60% improvement for complex query processing [8].

Learning Control Theory

Having seen the value of applying control theory to IBM products, we eagerly encouraged others to adopt these techniques. However, there are some significant barriers to software engineers learning control theory. Foremost, we found that existing books on control theory are poorly suited for software professionals, a problem we struggled with as well in trying to learn control theory. First, the examples in these books focus on design problems in mechanical and electrical engineering such as dashpots, inverted pendulums, and electrical circuits. Second, the books start with an analysis in continuous time, forcing software engineers

to recall college courses on differential equations. Third, the books are not oriented towards the manner in which software people learn most effectively—by working in an interactive programming environment.

We decided to write a book on control theory for software engineers. Knowing that we needed help with both the content and the pedagogy, we recruited our colleague Dawn Tilbury, a professor in the Department of Mechanical Engineering at the University of Michigan. Our starting point was to construct a set of eight examples of feedback control in computing systems (including the Domino Server example discussed earlier).

We structured the book so that it focuses on problems relevant to computing systems. The book only considers discrete time, it only addresses the SASO properties, and the only design technique is pole placement. As a result, the only mathematical concept needed is the *z*-transform. Subsequently, we developed intuitive approaches to describing *z*-transforms, and incorporated this material into a control theory short course that can be found at the URL in [5]. This has been very effective in our classes at Columbia, Stanford, and Berkeley.

Last, we needed a simple laboratory environment in which software engineers can explore control theory concepts. The requirements here are: (a) specifying difference equations; (b) observing immediately the effects of changes in system characteristics such as K_I and the model of the Domino Server; and (c) plotting variables over time. It turns out that spreadsheets work extremely well for this. Figure 3 displays a spreadsheet-based simulation of the system in Figure 1. Columns A through F correspond to the variables used in Figure 1. The calculations done within a cell correspond to the difference equation for the components in Figure 1. For example, cell D6 is the sum of D5 and C6 times G5 since column D corresponds to the controller

which has the difference equation $u(k)= u(k-1) + K_I e(k)$. The plot displays the values of $y(k)$ over time. Thus, if we change the system characteristic K_I by modifying cell G5, we see immediately the effect on the convergence of $y(k)$.

Conclusions

Control theory provides a systematic way to design feedback systems to ensure that they are stable and to understand the trade-offs between accuracy, short settling times, and minimal oscillations and overshoot. Applying control theory to computing systems has resulted in considerable benefit to IBM based on our experience to date. Our hope is that properly structured classes and texts will greatly lower the barrier for software engineers to learn the basics of control theory. At Berkeley, we communicated this material in approximately 6 one hour lectures. We are now attempting to do the same within IBM.

References

1. Abdelzaher, T.F. and N. Bhatti. Adaptive Content Delivery for Web Server QoS. International Workshop on Quality of Service (1999).
2. Sha, L., X. Liu, Y. Lu, and T. Abdelzaher. Queueing Model Based Network Server Performance Control. IEEE RealTime Systems Symposium (2002).
3. The Apache Software Foundation, http://www.apache.org.
4. Hollot, C. V., V. Misra, D. Towsley, and W.B. Gong. A Control Theoretic Analysis of RED. Proceedings of IEEE INFOCOM '01 (2001).
5. Hellerstein, J. L., Y. Diao, S. Parekh, and D. M. Tilbury. Feedback Control of Computing Systems. Wiley Inter-Science (2004). Also see http://www.research.ibm.com/fbcs/.
6. Gandhi, N., J. L. Hellerstein, D. M. Tilbury, T. S. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. Real Time Systems Journal (2002), vol. 23, no. 1-2.
7. Parekh, S., K. Rose, Y. Diao, V. Chang, J. L. Hellerstein, S. Lightstone, and M. Huras. Throttling Utilities in the IBM DB2 Universal Database Server. American Control Conference (2004).
8. Diao, Y., J. L. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Incorporating Cost of Control Into the Design of a Load Balancing Controller. Real-Time and Embedded Technology and Application Systems Symposium (2004).

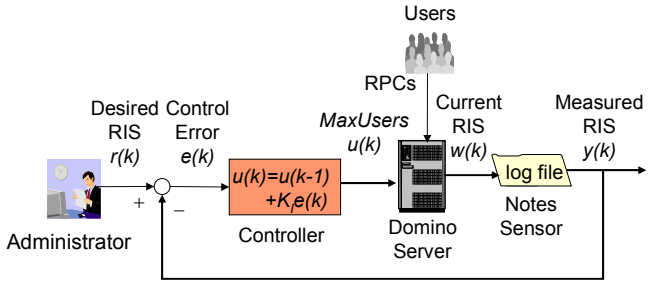

Figure 1: Feedback system that controls RPCs in System in the Domino Server.

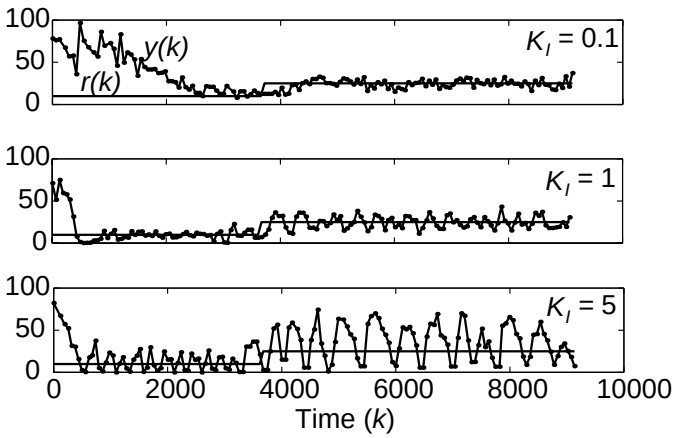

Figure 2: Testbed experiments comparing feedback systems for the Domino Server with three settings of the integral control parameter K_I.

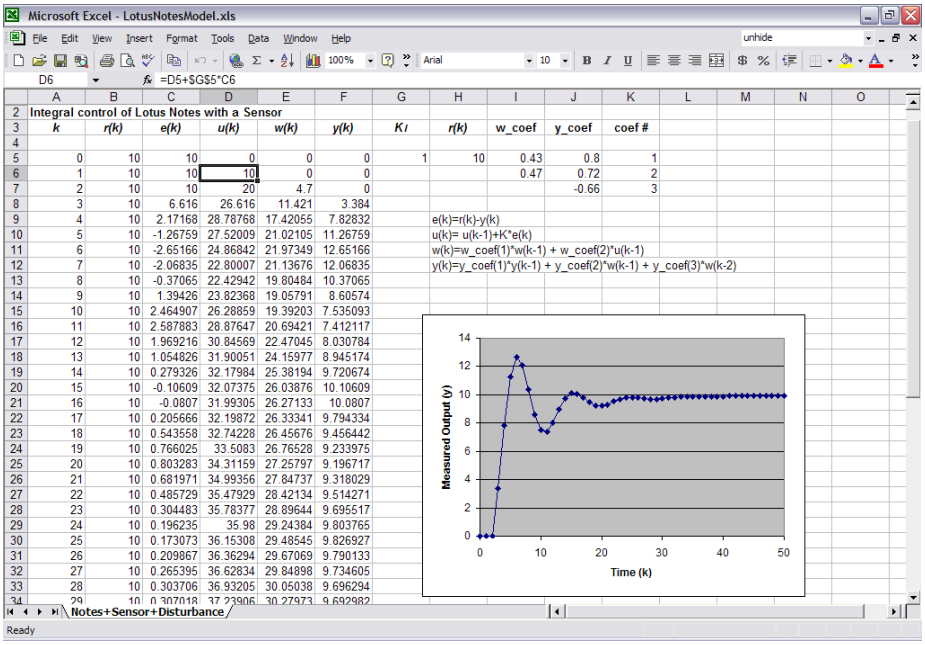

Figure 3: Spreadsheet simulation of the feedback system for the Domino Server.