

What Does Control Theory Bring to Systems Research?

Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang,
Sharad Singhal, Arif Merchant
Hewlett-Packard Laboratories
Palo Alto, CA 94304, USA
{firstname.lastname}@hp.com

Pradeep Padala,
Kang Shin
University of Michigan
Ann Arbor, MI 48109, USA
{ppadala, kgshin}@umich.edu

ABSTRACT

Feedback mechanisms can help today's increasingly complex computer systems adapt to changes in workloads or operating conditions. Control theory offers a principled way for designing feedback loops to deal with unpredictable changes, uncertainties, and disturbances in systems. We provide an overview of the joint research at HP Labs and University of Michigan in the past few years, where control theory was applied to automated resource and service level management in data centers. We highlight the key benefits of a control-theoretic approach for systems research, and present specific examples from our experience of designing adaptive resource control systems where this approach worked well. In addition, we outline the main limitations of this approach, and discuss the lessons learned from our experience.

Categories and Subject Descriptors

D.2.10 [SOFTWARE ENGINEERING]: Design – Methodologies.

General Terms

Algorithms, Management, Performance, Design, Experimentation, Theory.

Keywords

Control theory, systems research, model, dynamics, and stability.

1. INTRODUCTION

Control theory provides a powerful mechanism for dealing with unpredictable changes, uncertainties, and disturbances in systems using feedback. Feedback mechanisms can be found in many engineering as well as biological systems. Although feedback-based techniques have also been developed for computing systems [25, 26] and networks [24] in the past, formal control theory is rarely used. In conventional systems research, feedback algorithms are typically designed based on the system designer's domain knowledge and intuition instead of a quantitative model for the behavior of the system being controlled, and parameter values in these algorithms are often chosen in an ad-hoc fashion. What control theory brings to systems research is a rigorous methodology for modeling, analysis, design, and evaluation of feedback systems.

Figure 1 illustrates a standard feedback control loop. We refer to the system being controlled as the *target system*, which has a set of metrics of interest (referred to as *measured output*) and a set of control knobs (referred to as *control input*). The controller periodically adjusts the value of the control input such that the measured output can match its desired value (referred to as *reference input*) specified by the system designer. That is, it aims to maintain the difference between the two (referred to as *control error*) at zero, in spite of the *disturbance* in the system, something affecting the measured output that is not under control.

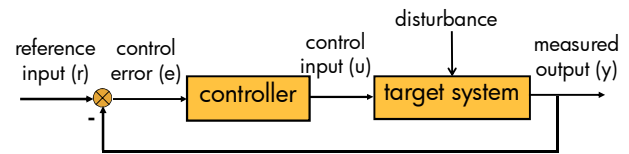


Figure 1. Standard feedback control loop.

In the past decade, control theory has been applied to performance control in computing systems. A number of good overviews can be found in [2, 4, 9]. The types of systems that have benefited from such an approach include multi-media systems [14], Web servers [1, 8], proxy caches [18], database servers [22], multi-tier Web sites [10], and real-time systems [17]. There has also been growing interest in applying control theory to power management of data center servers and clusters due to the increasing energy cost in recent years [7, 13].

In particular, researchers at HP Labs and University of Michigan have applied a control-theoretic approach to automated resource and service level management in data centers. Areas of successful applications include performance isolation and differentiation in storage systems [12], CPU utilization control for resource partitions [28] and response-time-driven workload management [30] on HP-UX systems, performance control of a three-tier application running in distributed virtual machines [27], performance assurance and differentiation for multiple co-hosted multi-tier applications [15, 20, 21], and coordinated power management for data centers [23].

In this article, we summarize the lessons we learned from this experience. In particular, we describe key strengths of the control-theoretic approach that make it a powerful tool for building feedback loops in computing systems, and present specific examples from our own work. In addition, we discuss some limitations in this approach that remain open problems.

2. OVERVIEW

Next-generation enterprise data centers and cloud computing environments are being designed with a utility computing paradigm in mind, where all hardware resources are pooled into a common shared infrastructure and applications share these resources as their demands change over time. In this new paradigm, multiple applications share dynamically allocated resources. These applications are also consolidated to reduce infrastructure and operating costs while simultaneously increasing resource utilization – virtualized infrastructures are becoming commonplace in consolidated environments. As a result, data center administrators are faced with growing challenges to meet service level objectives (SLOs) in the presence of dynamic resource sharing and unpredictable interactions across many applications. These challenges are:

Complex SLOs: It is nontrivial to convert individual application SLOs to corresponding resource shares in the shared virtualized platform. For example, determining the amount of CPU and disk shares required for a financial application to achieve a specified number of transactions per second is difficult.

Time-varying workload demands: The intensity and the mix of typical application workloads change over the lifetime of an application. As a result, the demands for individual resources also change over time. This implies that static resource allocation can meet application SLOs only when the resources are allocated for peak demands, wasting resources and increasing operating costs.

Distributed resource allocation: Multi-tier applications spanning multiple nodes require resource allocations across all tiers to be at appropriate levels to meet end-to-end application SLOs.

Resource dependencies: Application-level performance often depends on the application’s ability to simultaneously access multiple system-level resources.

We address these challenges by using control theory as the basis for the modeling, analysis, and design of feedback-driven resource management systems. We have built various controllers towards this end over the years, all of which culminated in an automated resource control and adaptation system called *AutoControl* [20]. In *AutoControl*, operators can specify the SLO for an application in a tuple (priority, metric, target), where *priority* represents the priority of the application, *metric* specifies the performance metric in the SLO (e.g., average throughput, 90th percentile of response time), and *target* indicates the desired value for the metric. *AutoControl* can manage any resources that affect the application performance metrics and that can be allocated between the applications.

Figure 2 shows a three-node subset of a virtualized infrastructure shared by multiple multi-tier applications, where each tier of an application is hosted in a virtual machine (VM). *AutoControl* can automatically adjust resource allocations to all the VMs in real time to meet the SLOs of the hosted applications. If this is mapped to the feedback loop in Figure 1, the control inputs are the resource allocations, the measured outputs are the measured values of the performance metrics, and the reference inputs are the performance targets in the SLOs. The application workloads are considered “disturbances” since they are not under our control.

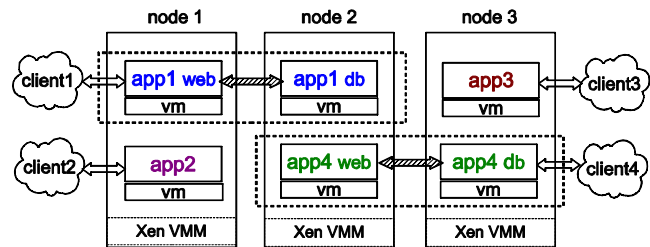


Figure 2. Example of a shared virtualized infrastructure with three nodes hosting multiple multi-tier applications.

AutoControl consists of a set of application controllers (AppControllers) and a set of node controllers (NodeControllers). There is one AppController for each hosted application, and one NodeController for each physical node. For each application, its AppController periodically polls an application performance sensor for the measured performance. We refer to this period as the *control interval*. The controller compares this measurement

with the application performance target, and based on the discrepancy, automatically determines the allocation needed for each resource type in each application tier for the next control interval, and sends these resource requests to the NodeControllers for the nodes that host the individual tiers of the application.

For each node, based on the collective resource requests from all the relevant AppControllers, the NodeController determines whether it has enough resources of each type to satisfy all the demands and computes the actual resource allocations. The resource allocations are then fed into the resource schedulers in the virtualization layer for actuation, which allocates portions of the node’s resources to the VMs.

The high-level goal of *AutoControl* is to meet application-level SLOs as much as possible while increasing resource utilization in the shared computing environment. More specifically, our controller design has the following three main objectives:

- **Guaranteed application performance:** When system resources are shared by multiple multi-tier applications, it is desirable to maintain performance isolation and to ensure that each application can achieve its SLO if possible.
- **High resource utilization:** It is also desirable to increase overall utilization of the shared resources so that fewer nodes are required to host a given set of applications, reducing the operating cost of the data center. One way to achieve this is to maintain a high-enough utilization in individual virtual machines such that there is more capacity for hosting other applications. There is a fundamental tradeoff between this goal and the previous goal, and a systematic approach in required to reach an appropriate balance between the two.
- **Performance differentiation during resource contention:** Whenever a bottleneck is detected in the shared resources, the controller needs to provide a certain level of performance differentiation where higher-priority applications experience lower performance degradation, or none. For example, one can aim to maintain a certain ratio of response times when the system is overloaded based on the priority values of the respective applications.

3. STRENGTHS OF A CONTROL-THEORETIC APPROACH

We have identified six key strengths of the control-theoretic approach based on our experience building adaptive resource control systems. We now discuss these strengths in detail and illustrate using examples from our own research.

3.1 Quantitative Input-Output Models

Traditional performance models used in systems problems are inadequate for feedback controller designs for two reasons: 1) They may be in a form that is hard to analyze - say, in a look-up table; 2) They are rarely expressed in the form of an *input-output model*, a specific type of model that is required for designing and analyzing feedback controllers. In control theory, the notion of *input* and *output* refers to the control input and measured output as defined in Figure 1. Note that they may have nothing to do with the input and the output of the system. For example, a queuing system has the incoming and the outgoing requests as the input and the output, whereas a queue control system may view resource allocation to the backend process as the input and the resulting queue length as the output.

Although feedback-based techniques have been used extensively in systems work, most algorithms are not designed based on an input-output model. For instance, the adaptive scheduler in [25] dynamically allocates proportions and periods of CPU time to threads based on measurements of their progress, where a PID (Proportional, Integral, and Derivative) controller was used directly without first identifying an input-output model. Similarly, the admission controller for Internet servers in [26] manages performance under overload; its design was based on intuitive understanding of the server behavior rather than a quantitative model of the relationship between the control input and the performance.

Input-output models are not commonly used in systems research for feedback control design because first-principle models are in general unavailable for computing systems. In our research, we have found a *black-box* approach, where models are inferred from experimental data, extremely useful [12, 15, 20, 21, 28]. For example, Figure 3 illustrates an input-output model for an application running in a VM, for instance, app2 in Figure 2. The control inputs are the resource allocations to the VM (u). The measured outputs include the measured application performance (y) such as response time and throughput, as well as the resource utilization within the VM (r). Note that, for each type of resource, *utilization* is defined as the ratio of the absolute amount of resource consumed by the VM (v) and the amount of resource allocated (u), i.e., $r = v / u$. The incoming workload (d) to the hosted application is viewed as a “*disturbance*” because it is not directly under control, although it has an impact on the measured outputs. Typically as the workload changes, the quantitative input-output relationship changes accordingly, increasing the difficulty in modeling as well as controller design.

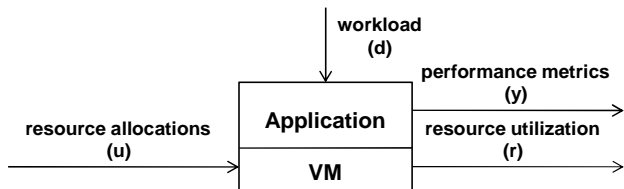


Figure 3: Input-output model for a VM-hosted application.

To build a black-box input-output model, the design of the modeling experiments is critical because they need to cover a rich set of test conditions to discover the various operating regions. For example, in [28], we varied the CPU allocation (u) for a container running an Apache Web server application from 0.2 to 0.9 of a CPU, in 0.05 CPU increments. At each setting, the Web server was loaded for 60 seconds with a fixed workload, while the average CPU consumption (v) of the container was observed and the relative CPU utilization (r) within the container was computed. In addition, the mean response time (MRT) of all the requests completed during this period was computed. The experiment was repeated at different workload intensities ranging from 200 to 1100 requests/s.

Figure 4(a) shows the utilization of allocation as a function of the CPU allocation, and Figure 4(b) shows the measured MRT as a function of the CPU allocation. Each data point is the average of 10 samples obtained from 10 repeated experiments. The empirical data provide a basis for inferring analytical models that describe both the static and the dynamic input-output relationships.

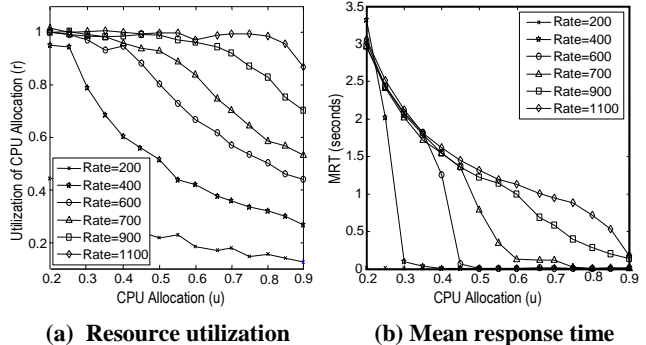


Figure 4: Resource utilization and mean response time vs. CPU allocation for a Web server under different workloads.

In general, for a fixed workload with an average resource demand D , the relationship between the resource allocation (u) and the utilization (r) can be approximated by:

$$r = \begin{cases} 1, & \text{if } u \leq D; \\ \frac{D}{u}, & \text{otherwise.} \end{cases} \quad (1)$$

We make two observations about this model. First, it is bimodal. The system is overloaded when the allocation is below the demand of the workload, where the utilization is a constant 100%, and is underloaded when the allocation exceeds the demand, where the utilization is inversely proportional to the allocation. Second, when the system is underloaded, the exact function depends on the workload demand. These observations are consistent with the result of the modeling experiment as shown in Figure 4(a) [28]. A feedback controller for managing resource utilization has to take into account both operating regions.

The relationship between the MRT and the resource allocation is more complicated. From Figure 4(b), we first observe that the steady-state relationship between the two is nonlinear, and likely multi-modal. For example, for a workload of 600 requests/s, the relationship curve can be classified into three regions: a smooth region where CPU allocation u is below 0.35, a highly sensitive region where u is between 0.35 and 0.45, and a highly insensitive region where u is above 0.45.

If a feedback controller is designed to adjust the input without an input-output model, the following problems may occur:

1. The controller may not converge to equilibrium, if the system does not have a monotonic relationship between a single input and a single output. This occurred when the two Web tiers of two applications a and b shared a single node, and each application is a two-tier implementation of the *RUBiS* online bidding benchmark [3]. We defined l_a as the number of lost connections for application a , and the normalized loss ratio between the two applications as $l_a / (l_a + l_b)$. We used the CPU allocation to Web tier a as the input. Figure 5 shows the non-monotonic input-output relationship when the loss ratio is chosen as the output; and the monotonic relationship when a better output metric of the response time ratio is used instead [21]. For example, for a target ratio of 40% (dotted line), the controller for the former may oscillate between the two operating points (circles), whereas the controller for the latter can easily find the operating point (triangle) using a simple control algorithm (e.g., an integral controller).

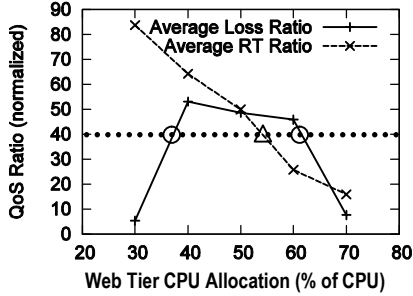


Figure 5. A non-monotonic input-output relationship between loss ratio (output) and Web tier CPU allocation (input), and a monotonic input-output relationship between response time ratio and Web tier CPU allocation.

- Without an estimate of the sensitivity of the outputs with respect to the inputs, the controller may become too aggressive (or even unstable) or too slow. For example, if the slope of the linear relationship between the average response time ratio and the Web tier CPU allocation in Figure 5 is p , and if an integral controller is applied to maintain the response time ratio at the target value, then the integral gain of the controller, K_I , has to be below $1/p$ to ensure stability in the closed-loop system; otherwise the controller’s adjustments to the CPU allocation will be too big, causing oscillations in the response time ratio.
- The controller cannot adapt to different operating regions in the input-output relationship. One example is given in Figure 4(a) and Equation (1) as we have discussed formerly. Another example can be seen in Figure 4(b) where the mean response time is *controllable* using CPU allocation only when the CPU consumption is close to the allocated capacity and *uncontrollable* when the CPU allocation is more than enough. Here the notion of “uncontrollable” refers to the condition where the output is insensitive to changes in the input, as we can see when the CPU allocation is above 0.45 for a workload of 600 requests/s. Without explicit modeling of this behavior, the closed-loop system may not behave properly when the controller switches between different operating regions. In the next section, we will discuss how to build a dynamic input-output model for the operating region where the system is controllable.

3.2 Dynamics and Transients

The study of system *dynamics*, as defined in control theory, is unused by most computer systems researchers. In essence, dynamics represent the transient effects of *memory* in the system, i.e., the correlation between a system’s current and past states. Most computing systems have dynamics because of the presence of queues in these systems.

While queuing theory characterizes long-term averages, it does not capture short-term, transient behaviors. The latter can only be assessed quantitatively when the dynamics of a system are explicitly modeled. Control theory provides methods for estimating dynamic models. Although dynamics in traditional continuous-time systems are represented by differential equations, dynamics in discrete-time systems, such as those computing systems we are interested in, are often better characterized by difference equations. In these models, we use $x(k)$ to represent the value of variable x in the k th time interval. A dynamic, linear

relationship between the input, $u(k)$, and the output, $y(k)$, in a system is often represented using the following auto-regressive-moving-average (ARMA) model:

$$y(k) = \sum_{i=1}^n a_i y(k-i) + \sum_{j=0}^{m-1} b_j u(k-d-j) + e(k). \quad (2)$$

The coefficient a_i captures the degree of correlation between the current output and the past output, and the coefficient b_j captures the degree of correlation between the current output and the current or past input. The parameter d is referred to as “*delay*” in the system, indicating the number of time intervals one has to wait to observe a change in the output $y(k)$ after a change in the input $u(k)$ has occurred. Such delays can be caused by actuator or sensor delays, or delay in the system itself. The parameters m and n are referred to as “*orders*” of the model, and they capture the length of memory in the system and its relationship to the time interval. For example, a system with fully occupied queues may require a model with higher orders if the time interval is small. Finally, $e(k)$ represents the noise and disturbances that affect the measured output and that are not captured by the model. Note that, for a multiple inputs, multiple outputs (*MIMO*) system, both $u(k)$ and $y(k)$ are vectors and a_i and b_j will be matrices.

These dynamic input-output models can be estimated using data collected from system identification experiments. In designing these experiments, it is essential that the inputs be stimulated sufficiently to capture the full spectrum of the dynamic response in the system. At the same time, it is usually beneficial and hence a common practice to ignore nonessential high-order dynamics and to use low-order models as approximations of the system behavior. In our research, we have found that first- or second-order models, representing system memory that lasts one or two time intervals, are usually good enough approximations of the dynamics in the computing systems we have tested [12, 15, 20, 28]. This can also simplify the controller design, and offer better robustness in the resulting control system. However, ignoring critical dynamics of a system, e.g., ignoring dynamics altogether or ignoring actuator or sensor delay may lead to instability of the closed-loop system.

We have done detailed system identification experiments in [28] to capture the dynamic relationship between the CPU allocation to a Web server and its measured mean response time (MRT). From Figure 4(b), we know that the MRT is a nonlinear function of the CPU allocation in steady-state. However, we observe a linear dependency between the inverse of MRT (y) and the CPU allocation (u) within certain operating regions. Therefore, we varied the CPU allocation randomly between 0.2 and 0.8 in contiguous time intervals, and calculated the resulting $1/\text{MRT}$ for each interval. The time interval was fixed as 15 seconds, while the experiment was repeated for different request rates ranging from 200 to 1100 requests/s. The model in Equation (2) with different structures and parameters was estimated offline by using the least-squares method [16] in the *Matlab System Identification Toolbox* [19] to fit the input-output data. For convenience, we refer to such a model as “*ARMA nd* ” in the following discussion. We evaluated the fitness of the model using the *coefficient of determination* (R^2) metric. In general, this metric indicates the percentage of variation in the output captured by the model. The results are shown in Tables 1(a) and 1(b).

Table 1. R^2 values (%) of different input-output models

Model	Workload Rate (r/s)					
	200	400	600	700	900	1100
ARMA110	-10.2	12.8	2.8	63.1	70.3	78.3
ARMA111	-1	6.7	2.7	-5	0.09	6.4

(a) First-order ARMA models for different workloads

Workload Rate (r/s)	Model			
	ARMA110	ARMA220	ARMA330	ARMA440
900	70.3	71.7	70.8	71.5
1100	78.3	79.9	80.3	80.2

(b) ARMA models with different orders

We make the following observations from these tables:

- According to Table 1(a), the simple linear models (ARMA110 and ARMA111) do not fit the input-output data for a workload below or equal to 600 requests/s. This corresponds to the condition where the Web server is underloaded. In contrast, when the request rate is above 600 requests/s, ARMA110 (first-order model with no delay) fits the data quite well, providing a good basis for controller design. Moreover, ARMA111 (first-order model with one-step delay) does not explain the system behavior even with the heavier workload. This means, when there is significant correlation between the CPU allocation and the MRT, 15 seconds is long enough to observe that correlation.
- Under all conditions where an ARMA model is a good fit, a first-order model is sufficient. Table 1(b) shows, using rates of 900 and 1100 requests/s as examples, that increasing the order of the ARMA model does not increase its fitness value.

3.3 Correlation between Multiple Metrics

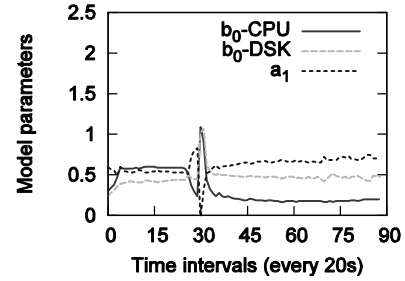
In a MIMO system, there are different degrees of correlations between various inputs and outputs. Although system designers may have intuition for some of the correlations, it is often hard to quantify them, making the design of a MIMO controller difficult.

Using control theory, we can develop MIMO models (often empirically) to capture the correlations between different inputs and outputs. For example, the dependency of an application's throughput (y) on two critical control variables, CPU allocation (u_{cpu}) and disk allocation (u_{disk}), can be represented using the following linear, second-order, multiple-input, single-output model [20]:

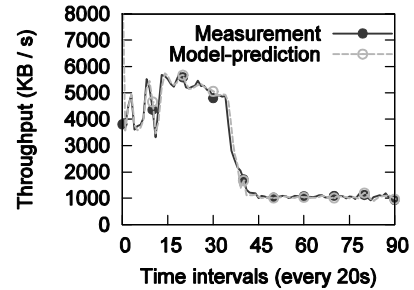
$$y(k) = \sum_{i=1}^2 a_i y(k-i) + \sum_{j=0}^1 (b_{j,cpu} u_{cpu}(k-j) + b_{j,disk} u_{disk}(k-j)) + e(k). \quad (3)$$

The model coefficient $b_{0,cpu}$ (or $b_{0,disk}$) captures the degree of correlation between the measured application throughput in the k th time interval and the CPU allocation (or disk allocation) in the $(k-1)$ th time interval. The other coefficients in the model can be interpreted in a similar fashion. This model can be easily estimated offline using the least squares method [16] based on the measured throughput, the CPU allocation, and the disk allocation for the same set of time intervals. It can also be continuously updated online using the recursive least squares (RLS) algorithm [5] every time a new set of measurements is obtained.

Figure 6(a) shows an example for an estimated model from [20]. It shows the model parameters ($b_{0,cpu}$, $b_{0,disk}$, and a_1) for an smedia application, a synthetic benchmark for secure media server [20], as functions of the time interval. The second-order parameters are not shown due to lack of space. During the first 30 time intervals, we can see that $b_{0,cpu} > b_{0,disk}$. This is consistent with the application being CPU-bound. After the 30th interval, we notice that the value of $b_{0,cpu}$ dropped below the value of $b_{0,disk}$, indicating that the disk had become a more critical resource for the application. This demonstrates the model's ability to capture the correlations between different metrics appropriately. A comparison between the measured application throughput and the model-predicted throughput for the same time period is shown in Figure 6(b). The coefficient of determination (R^2) of this model is 92.2%, and the mean absolute percentage error (MAPE) is 6.9%. These indicate that the model matches the measured data fairly well.



(a) Model parameter values from online estimation



(b) Measured and model-predicted throughput

Figure 6. A linear model capturing correlations and the comparison between measurement and model prediction.

3.4 Well-Studied Control Algorithms

Threshold-based algorithms are commonly used in computing systems for maintaining a measured output at certain target value because such algorithms are intuitive and easy to implement. However, it is nontrivial to choose values for parameters associated with such algorithms, including the step size for changing the control input when the measured output violates the threshold, and the size of the dead zone (if any) around the threshold where no control action is taken. Such parameters are usually chosen by trial and error, and can result in oscillations in the system, if not chosen carefully.

Control theory provides a whole suite of well-studied control algorithms that can be used in practice. For example, PID controllers are widely used for SISO (single input, single output)

systems. For MIMO systems, standard algorithms such as state or output feedback and Linear Quadratic Regulator (LQR) allow us to manage multiple inputs and multiple outputs simultaneously [29]. Designers of feedback control systems can draw from decades of research that provides guidelines for setting key parameter values in these algorithms to ensure stability and good performance of the closed-loop system. In our research, we have used an adaptive integral controller in [21, 27, 28, 30], a direct self-tuning regulator in [12], and a linear quadratic optimal controller in [15, 20] and all of them have worked well.

3.5 Stability Guarantees

In conventional systems research, a design is considered satisfactory if it meets the design goals (e.g., maintaining system utilization below a threshold) under the conditions that are tested in the experiments. Stability is rarely considered an explicit criterion. However, for systems with feedback, stability becomes an important metric because a poorly designed feedback controller can introduce large oscillations to an otherwise stable system.

Control theory offers analytical methods for assessing the stability of the closed-loop system, given models of the target system and the controller, and provides guidelines for choosing the controller parameter values to ensure stability. For example, for the nonlinear relationship between the resource allocation (u) and the utilization (r) shown in Equation (1), we have designed the following self-tuning, integral controller [28] to regulate the utilization within a VM such that it can be maintained at a desired level specified by the reference input r_{ref} :

$$u(k) = u(k-1) - \alpha \frac{v(k-1)}{r_{ref}} (r_{ref} - r(k-1)). \quad (4)$$

We have proven that this adaptive controller is globally stable for $0 < \alpha < 1/r_{ref}$. In other words, the closed-loop system using this controller is always stable under any workload demands. In practice, it is almost impossible to completely eliminate oscillation in the measured output in a computing system due to actuator and sensor noises. A stabilizing controller helps minimize the oscillation around the reference input, therefore improving predictability in the system performance.

Moreover, there is a fundamental tradeoff between stability and responsiveness. The latter can be characterized by the amount of time it takes for the measured output to track a step change in the reference input (e.g., a change in the SLO) or for it to go back to the normal state after a disturbance occurs (e.g., a change in the workload). Heuristic designs either do not address this issue or deal with it in an ad-hoc manner. This tradeoff is explicitly handled in control theory. For example, in standard LQR designs, such tradeoff is dealt with using weighting matrices [29]. In another example, the AppController in [20] periodically minimizes a quadratic cost function. The optimizer in the AppController determines the resource allocation required in order for the application to meet its performance target. It does so without causing large oscillations in the resource allocation by including a stability factor (q) that allows system designers to make tradeoffs between stability and fast responses in the output. The general form of the optimization takes the form $J_a = J_p + q \times J_c$, where the performance cost (J_p), a penalty for the deviation of the application's measured performance from its target, is jointly taken into account in the optimization with the

control cost (J_c), which is used to curtail the amount of resource allocation changes the controller can make in a single interval.

Figure 7 shows the achieved throughput of a TPC-W application, a multi-tier e-commerce benchmark, where the CPU and the disk allocations for the VMs hosting the application were managed by the AppController. Multiple experiments were run with the stability factor q set to different values. In each experiment, at the 30th time interval, the workload for the application surged to a higher level, and the throughput target was adjusted accordingly. As we can see, $q = 1$ results in faster and more aggressive response in the controller, with oscillations in the throughput around the target; for $q = 10$, the controller is sluggish and does not track the change in the target fast enough; among the three values tested, $q = 2$ offers the best tradeoff between fast tracking and stability in the delivered application performance.

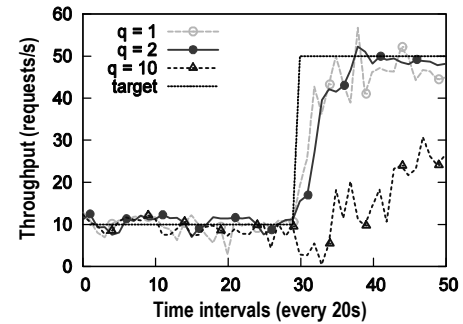


Figure 7. TPC-W throughput (requests/s) for different values of the stability factor $q = [1, 2, 10]$.

3.6 Nonlinear and Time-Varying Behavior

Adaptive control theory [5] offers a well-studied methodology for designing controllers for systems whose behavior (encoded in the input-output model) varies over time, or systems whose input-output relationships (such as the one shown in Figure 4) can be approximated using linear models around an operating point. It does this by online estimation of model parameters and automatic adaptation of controller parameters based on the model. Ad hoc controllers rarely adapt their parameters automatically.

To illustrate this behavior, we studied a combination of a multi-tier application (RUBiS) and multiple instances of a media server (smedia) sharing a virtualized infrastructure across multiple nodes. For RUBiS, we used the default browsing mix workload with 600 threads emulating 600 concurrent clients connecting to the RUBiS server, and used 100 requests per second as the throughput target. Each of the smedia applications was driven with 40 threads emulating 40 concurrent clients downloading media streams at 350KB/sec. We then ran an experiment for 90 time intervals and varied the percentage of encrypted streams requested from smedia to create a shift of resource bottleneck in each of the virtualized nodes. For the first 29 intervals, smedia1 and smedia2 on node 1 were CPU-bound, whereas smedia3 and smedia 4 on node 2 were disk-bound. At interval 30, smedia1 and smedia2 on node 1 were switched to disk-bound, and at interval 60, smedia3 and smedia4 on node 2 were switched to CPU-bound. Figure 8 shows that AutoControl was able to achieve the targets for all the applications (only RUBiS is shown) in spite of the fact that, (i) resource bottleneck occurs either in the CPU or in the disk (Figure 9 and Figure 10 show the CPU and disk allocations over time); (ii) both tiers of the RUBiS application distributed across two physical nodes experienced resource contention.

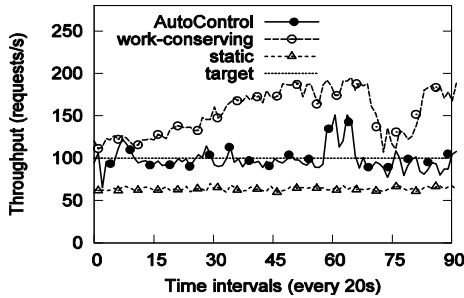


Figure 8: Measured RUBiS throughput.

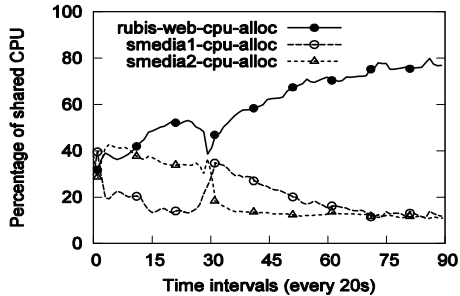


Figure 9: CPU allocations at node 1 using AutoControl.

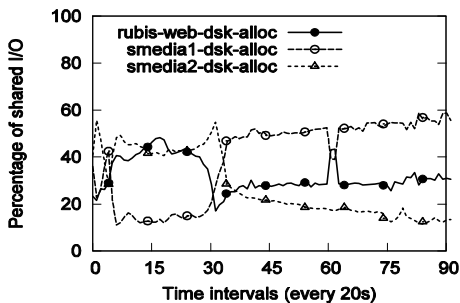


Figure 10: Disk allocations at node 1 using AutoControl.

Figure 8 also shows a comparison between AutoControl and two other resource management policies commonly used in data centers – work-conserving and static allocation. It is clear that the other two policies cannot ensure application performance or adapt to changes in the workload (see a more detailed study in [20]). We are convinced from our experience that adaptive control is an effective approach for feedback designs in computing systems with nonlinear relationships and time-varying behavior, because it allows the controller to automatically adapt to changes in workloads or operating conditions.

4. LIMITATIONS OF CONTROL THEORY

We have also discovered a number of limitations in the control-theoretic approach as applied to systems research.

- 1) Most inter-relationships in computing systems are nonlinear. This makes modeling difficult. Unlike classical linear control theory that is more intuitive and accessible to people outside the field of control, nonlinear control theory and adaptive control theory are much harder to understand and to apply in practice. Adaptive control also imposes limitation on how fast workloads or system behavior can change.
- 2) The lack of first-principle models requires an empirical approach to inferring input-output models for computing

systems. However, this approach requires controlled experiments. Existing data collected from production systems are hard to use for modeling purposes because it often lacks sufficient excitation to identify all relevant correlations.

- 3) Most classical control problems are formulated as tracking problems, i.e., the controller maintains the outputs at certain reference values. However, this may not be appropriate for all design problems in systems. For example, a meaningful design objective for a Web server may be to minimize the mean or percentage response time instead of keeping it at a certain value. For such problems, design methodologies that provide stability guarantees are not generally available.
- 4) Classical control theory only deals with continuous inputs. Many computing systems have input variables that only take on discrete values. For the systems where input values are discrete and numerical (e.g., number of servers allocated or P-states in a processor with dynamic voltage scaling enabled), it is possible to design controllers assuming continuous input variables. This approach may suffer from instability or inefficiency due to quantization errors. For systems with input values that are logical (e.g., Linux kernel version), discrete-event control theory [6] can be explored.
- 5) Many computing systems were not designed to be controllable in the first place. For example, many application configuration parameters (e.g., number of threads) cannot be changed at runtime, prohibiting online control of these parameters. Moreover, until recent years, most performance sensors produce measurements at the time granularity of minutes, making it impossible to design controllers that respond to changes at shorter time scales. For the computing systems to become amenable to dynamic feedback control, both actuator and sensor designs need to become an integral part of the computing systems design itself. In [11], we presented a set of necessary and sufficient conditions for a system to be controllable by a family of adaptive controllers.

5. CONCLUSIONS

In summary, feedback-based techniques, when applied to data center resource and service level management, can help computing systems adapt to changes in workloads or operating conditions. However, if not designed properly, such adaptation may become too aggressive or overly sensitive to small noises in the measurements or transients in the workloads, resulting in large oscillations in the controlled metrics. In our own research, we have used control theory to guide the modeling and designing of these feedback loops to achieve a proper balance between faster responses to changes and better stability in the system. For computing systems, better stability implies lower variance, therefore more predictability in the system performance.

Traditional systems research relied heavily on experimental validation of innovative design ideas. We believe that by combining such an experimentally oriented approach in systems research with the rigorous methodology in control theory, we can build better and more robust feedback systems.

6. REFERENCES

- [1] Abdelzaher, T.F., Shin, K.G. and Bhatti, N. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems* (Vol. 13, No. 1, January, 2002).

- [2] Abdelzaher, T.F., Stankovic, J.A., Lu, C., Zhang, R., and Lu, Y. Feedback performance control in software services. *IEEE Control Systems Magazine* (Vol 23, No. 3, June, 2003).
- [3] Amza, C., Cecchet, E., Chanda, A., Cox, A., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K., and Zwaenepoel, W. Specification and implementation of dynamic Web site benchmarks. In *Proc. of the 5th IEEE Workshop on Workload Characterization (WWC-5, November, 2002)*.
- [4] Arzen, K-E., Robertsson, A., Henriksson, D., Johansson, M., Hjalmarsson, H., and Johansson, K.H. Conclusions from the European roadmap on control of computing systems. The 1st IEEE/IFIP International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'06, April, 2006).
- [5] Astrom, K.J. and Wittenmark, B. *Adaptive Control*. Prentice Hall. 1994.
- [6] Cassandras, C.G., and Lafortune, S. *Introduction to Discrete Event Systems*. Kuwer Academic Publishers. 1999.
- [7] Chen, Y., Das, A., Qin, W., Sivasubramaniam, Wang, Q., Gautam, N. Managing server energy and operational costs in hosting centers. In *Proc. of the ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'05, June, 2005)*.
- [8] Gandhi, N., Tilbury, D.M., Diao, Y., Hellerstein, J.L., and Parekh, S. MIMO control of an Apache Web server: Modeling and controller design. In *Proc. of the American Control Conference (ACC'02, May, 2002)*.
- [9] Hellerstein, J.L., Diao, Y., Parekh, S., and Tilbury, D.M. *Feedback Control of Computing Systems*. John Wiley & Sons. 2004.
- [10] Kamra, A., Misra, V., and Nahum, E.M. Yaksha: A self-tuning controller for managing the performance of 3-tiered Web sites. In *Proc. of the 12th IEEE International Workshop on Quality of Service (IWQoS'04, June, 2004)*.
- [11] Karamanolis, C., Karlsson, M., and Zhu, X. Designing controllable computer systems. In *Proc. of the 10th Workshop on Hot Topics in Operating Systems (HotOS X, June, 2005)*.
- [12] Karlsson, M., Karamanolis, C., and Zhu, X. Triage: Performance differentiation for storage systems using adaptive control. *ACM Transactions on Storage* (Vol. 1, No. 4, November, 2005).
- [13] Lefurgy, C., Wang, X., and Ware, W. Server-level power control. In *Proc. of the 4th IEEE International Conference on Autonomic Computing (ICAC'07, June, 2007)*.
- [14] Li, B. and Nahrstedt, K. Control-based middleware framework for quality of service applications. *IEEE Journal on Selected Areas in Communication* (Vol. 17, No. 9, September, 1999).
- [15] Liu, X., Zhu, X., Padala, P., Wang, Z., and Singhal, S. Optimal multivariate control for differentiated services on a shared hosting platform. In *Proc. of the 46th IEEE Conference on Decision and Control (CDC'07, December, 2007)*.
- [16] Ljung, L. *System Identification: Theory for the User* (2nd Edition). Prentice Hall. 1999.
- [17] Lu, C., Wang, W., and Koutsoukos, X. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems* (Vol. 16, No. 6, June, 2005).
- [18] Lu, Y., Lu, C., Abdelzaher, T.F., and Tao, G. An adaptive control framework for QoS guarantees and its application to differentiated caching services. In *Proc. of the 10th IEEE International Workshop on Quality of Service (IWQoS'02, May, 2002)*.
- [19] Matlab System Identification Toolbox. <http://www.mathworks.com/products/sysid/>
- [20] Padala, P. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Hou, K., and Shin, K. Automated control of multiple virtualized resources. HP Labs Technical Report (HPL-2008-123, October, 2008).
- [21] Padala, P. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K., and Shin, K. Adaptive control of virtualized resources in utility computing environments. In *Proc. of EuroSys'07 (March, 2007)*.
- [22] Parekh, S., Rose, K., Diao, Y., Chang, V. Hellerstein, J.L., Lightstone, S., and Huras, M. Throttling utilities in the IBM DB2 universal database server. In *Proc. of American Control Conference (ACC'04, June-July, 2004)*.
- [23] Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., and Zhu, X. No power struggles: Coordinated multi-level power management for the data center. In *Proc. of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08, March, 2008)*.
- [24] Shenker, S. A Theoretical analysis of feedback flow control. In *Proc. of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM'90, September, 1990)*.
- [25] Steere, D.C., Goel, A., Gruenberg, H., McNamee, D., and Pu, C., and Walpole, A. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI'99, February, 1999)*.
- [26] Welsh, M., and Culler, D. Adaptive overload control for busy Internet servers. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03, March, 2003)*.
- [27] Wang, Z., Liu, X., Zhang, A., Stewart, C., Zhu, X., Kelly, T., and Singhal S. AutoParam: Automated control of application-level performance in virtualized server environment. The 2nd IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'07, May, 2007).
- [28] Wang, W., Zhu, X., and Singhal, S. Utilization and SLO-based control for dynamic sizing of resource partitions. In *Proc. of the 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM'05, October, 2005)*.
- [29] Zhou, K., and Doyle, J.C. *Essentials of Robust Control*. Prentice Hall. 1998.
- [30] Zhu, X., Wang, Z., and Singhal, S. Utility-Driven workload management using nested control design. In *Proc. of the American Control Conference (ACC'06, June, 2006)*.