Utilization vs. SLO-Based Control for Dynamic Sizing of Resource Partitions

Zhikui Wang, Xiaoyun Zhu, Sharad Singhal
HP Laboratories Palo Alto
HPL-2005-126(R.1)
January 23, 2006*

server virtualization, resource partition, system identification, adaptive control

In this paper we deal with a shared server environment where the server is divided into a number of resource partitions and used to host multiple applications at the same time. In a case study where the HP-UX Process Resource Manager is taken as the server partitioning technology, we investigate the technical challenges in performing automated sizing of a resource partition using a feedback control approach, where certain input variable such as the CPU entitlement for the partition is dynamically tuned to regulate output metrics such as the CPU utilization or SLO-based application performance metric. We demonstrate the importance of obtaining proper models to characterize both the static and dynamic input-output relations, identify the nonlinear and bimodal properties of the models across different operating regions, and discuss their implications for the design of the control loop. We then present various controller designs for either the relative utilization or the mean response time, evaluate the performance of the closed-loop systems while varying certain operating conditions, and discuss their advantages and issues. Finally, we present an adaptive controller that combines the CPU entitlement and utilization information and achieves more robust performance than prior solutions.

* Internal Accession Date Only
Published in and presented at the 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2005), 24-26 October 2005, Barcelona, Spain
Approved for External Publication

Utilization vs. SLO-Based Control for Dynamic Sizing of Resource Partitions

Zhikui Wang Xiaoyun Zhu Sharad Singhal
Hewlett Packard Laboratories
1501 Page Mill Rd, Palo Alto, CA 94304
{zhikui.wang, xiaoyun.zhu, sharad.singhal}@hp.com

Abstract

In this paper we deal with a shared server environment where the server is divided into a number of resource partitions and used to host multiple applications at the same time. In a case study where the HP-UX Process Resource Manager is taken as the server partitioning technology, we investigate the technical challenges in performing automated sizing of a resource partition using a feedback control approach, where certain input variable such as the CPU entitlement for the partition is dynamically tuned to regulate output metrics such as the CPU utilization or SLO-based application performance metric. We demonstrate the importance of obtaining proper models to characterize both the static and dynamic input-output relations, identify the nonlinear and bimodal properties of the models across different operating regions, and discuss their implications for the design of the control loop. We then present various controller designs for either the relative utilization or the mean response time, evaluate the performance of the closed-loop systems while varying certain operating conditions, and discuss their advantages and issues. Finally, we present an adaptive controller that combines the CPU entitlement and utilization information and achieves more robust performance than prior solutions.

Keywords: server virtualization, resource partition, system identification, adaptive control

1. Introduction

Resource partitioning is a type of virtualization technology that enables multiple applications to share the system resources on a single server [13][15][29] while maintaining performance isolation and differentiation among them. On most current systems, partition sizes are pre-determined and allocated to applications by system administrators, posing a challenging configuration problem. On the one hand, each partition has to be provided with enough resources to meet service level objectives (SLOs) of the applications hosted within it in spite of changes in workloads and the underlying system. On the other hand, excessive over-provisioning makes inefficient use of resources on the system. Offline capacity planning or calendar-based scheduling using profiles of past application resource usage are not always accurate or up-to-date and cannot handle unexpected short-term spikes in demand.

To ease management of shared server environments, and to provide "capacity on demand" to enterprise applications, our work aims to develop formal control-theory based techniques to automatically size a resource partition based on its CPU utilization, the SLO and the time-varying workload of its hosted applications.

This paper is organized as follows. In section 2, we describe the technology, the overall architecture and the test bed in our case study. Related work is reviewed in Section 3. Section 4 describes how the input-output behavior of the system was modeled, and discusses implications of the various models for control designs. Section 5 presents a number of different controller designs and their performance evaluation using our test bed. Finally, we summarize our results and conclusions, along with directions for future work in Section 6.

2. A Case Study using A Feedback Control Approach

We conducted a case study where we used the *HP-UX Process Resource Manager* (PRM) [13] as an example of the resource partitioning technology. PRM is a resource management tool that can be used to partition a server into multiple PRM groups, where each PRM group is a collection of users and applications that are joined together and allocated certain amounts of system resources, such as CPU, memory, and disk bandwidth. Under overload conditions, PRM guarantees a minimum entitlement of system resources to each PRM group. Optionally, if CPU or memory capping is enabled, PRM ensures that each PRM group's usage of CPU or memory does not exceed the cap regardless of whether the system is fully utilized. In this paper, we focus on sizing of a PRM group in terms of CPU allocation (with capping enabled). In general, a PRM group can be either a PSET PRM group that is assigned a subset of the system's processors, or an FSS PRM group that is assigned a percentage of the CPU cycles by specifying a number of shares. Here we consider only the FSS PRM group and refer to the CPU percentage allocated as its "CPU entitlement." Because this percentage is enforced by the Fair Share Scheduler (FSS) in the HP-UX kernel, it can be changed at any time thereby enabling dynamic sizing of the PRM group. This is particularly useful for dealing with uncertainties in the resource demands of enterprise applications. It also allows a feedback control approach to be used for automated resource allocation to these resource partitions.

2.1 System architecture

Figure 1 illustrates a shared server that has *m* resource partitions, where each partition can be a PRM group. We consider the scenario where each PRM group is used to host one application. The resource controller interacts with each partition *i* through two modules, A_i and S_i, where S_i is the sensor that periodically measures the performance metric for application *i*, and A_i is the actuator that dynamically sets the CPU entitlement for partition *i* according to the output of the resource controller. The timing of the controller is based on the notion of a "sampling interval". At the beginning of each sampling interval, the controller collects from S_i the measured performance for the last sampling interval, compares it to the desired performance, computes the needed CPU entitlement for the current sampling interval using certain control algorithms and passes it to A_i for actuation. In the remainder of this paper, we focus on resource control for one such partition. The results should be extensible to controlling multiple partitions with multiple resource types using any partitioning technology.

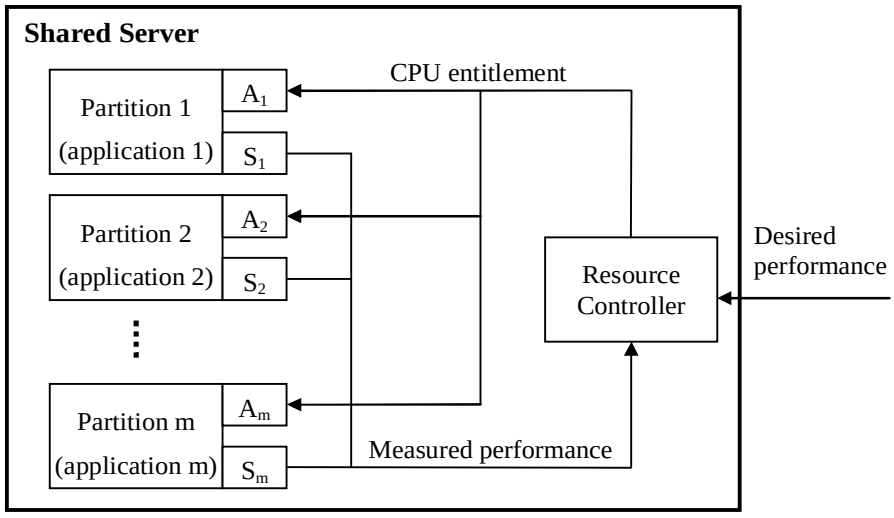


Figure 1. CPU entitlement control system architecture

2.2 Test bed setup

In the case study, we took the *Apache Web server* as an example of the hosted applications. We set up an FSS PRM group on an HP-UX server to host an Apache Web server of version 2.0.52. We refer to this PRM group as the "Web server partition". We used a modified version of *httperf 0.8* (ftp://ftp.hpl.hp.com/pub/httperf) on a Linux 2.4.18-3 client to continuously send HTTP requests to the Web server and to log the response time of every request. We developed a sensor module that parses the httperf log and computes the mean response time (MRT) of all the requests completed during each sampling interval. It is assumed that the Web server has a target value for the MRT based on its SLO. We also used a PRM provided utility *prmmonitor* to measure the average CPU utilization of a partition for every interval. The CPU entitlement (with capping enabled) for the Web server partition can be adjusted at the beginning of every interval to bound the percentage of CPU cycles used by the Web server in that interval. We chose the simplest possible workload where a single static page was repeatedly fetched from the Web server at a fixed rate. This setting ensured that memory and network bandwidth utilization was low and only minimum disk activity was involved. It means CPU was the only potential bottleneck in the system as the workload intensity varied. Therefore, we focus on the problem of dynamically determining the CPU entitlement for the Web server partition using feedback control.

3. Related Work

Our approach differs from prior work on operating systems support for server resource reservation and enforcement [7][17][26][27] or scheduling [11][30] in that it is more generic and can be used on any commodity operating system that supports a resource partitioning technology, and any application that can be hosted inside a partition. In addition, our control loop utilizes a feedback mechanism to determine an application's CPU entitlement in real-time, which is useful in dealing with uncertainties in the resource demands of typical enterprise applications. In [28] a feedback-driven adaptive scheduler was presented to allocate a percentage of CPU cycles to a thread over a period of time. In contrast, our controller allocates a percentage of CPU cycles to a whole application so that the assigned CPU entitlement can be tied directly to the application's SLO. Although the proposed feedback loop is already in use in some existing workload management tools [14][16], our approach is distinct in that we rely on classical control theory to guide the design of the control algorithms.

Feedback control theory has been applied to solve a number of performance or quality of service (QoS) problems in computer systems and services in recent years. (See [1][12] and the references therein.) This approach allows feedback loops to be designed in a systematic fashion and results in well-behaving closed-loop systems with provable properties such as stability and responsiveness. On the other hand, the robustness of these properties depends heavily on the fitness of the mathematical models used to characterize the dynamic behavior of the systems being controlled and how these models are obtained.

Computer systems in general lack common mathematical models that accurately describe their complex, highly-customized and ever-evolving behavior. As a result, much prior work that applies control theory employs a "black-box" approach and uses input-output models to capture the dynamic relation between control knobs (inputs) and performance metrics (outputs). Although much of computer systems' behavior exhibits nonlinearity, most researchers choose linear models to represent the input-output relation for simplicity and tractability. However, a single linear model is often insufficient to uniformly capture a system's behavior under all operating conditions for at least two reasons. First, each linear model may be only a local approximation of the system's nonlinear behavior around an operating point. Second, the system being controlled may face constant changes in workloads and other conditions that lead to changes in the input-output relation. Therefore, a model estimated under one operating condition may not explain the

system's behavior under another condition. Since models are used as the basis for controller design, improper choice of models can lead to instability or poor performance of the closed-loop system. More recent work that applies adaptive control theory to computer systems addresses this issue by allowing the model to automatically adapt to changes in operating conditions using online system identification [19][20][21][24]. However, these papers typically focus on the controller design and do not offer in-depth discussion of the modeling phase of the design cycle. In this paper, we present a detailed quantitative analysis of model fitness and variability through our case study.

Performance control of Web servers has been studied extensively in the literature. For instance, application-level mechanisms or admission-control schemes were proposed in [3][10][18] to provide different levels of service to requests of different classes. While these approaches were mainly based on heuristics or queuing models, other work has applied classical control theory to manage Web server delay or server resource utilization using admission control and content adaptation [2][8], connection scheduling and process reallocation [23], or application parameter tuning [9]. All of these methods require modification to the server application software (with the exception of [18]), which may not be feasible for other enterprise applications. Since our focus is not controlling Web server performance in particular, but rather providing a general approach for dynamic sizing of any resource partitions, it is important that our approach is applicable to any applications that can be hosted inside such partitions.

On the other hand, we do use Apache Web server as a test case to investigate the challenges in both modeling and controller designs. In [2] the authors offered insights into how to obtain appropriate models for the actuator, sensor, and the controlled system using benchmarking and linear regression based estimation techniques, whiling using CPU utilization as the output. The resulting relation between the adaptation level and the CPU utilization is a time-varying static gain with no dynamics but with a time delay. In comparison, we use either CPU utilization or response time as the system output, and evaluate both the static and dynamic input-output relations using experimental data collected at both long and short time scales. We found that both low-order dynamics and nonlinearity are useful in capturing certain characteristics in the input-output relations.

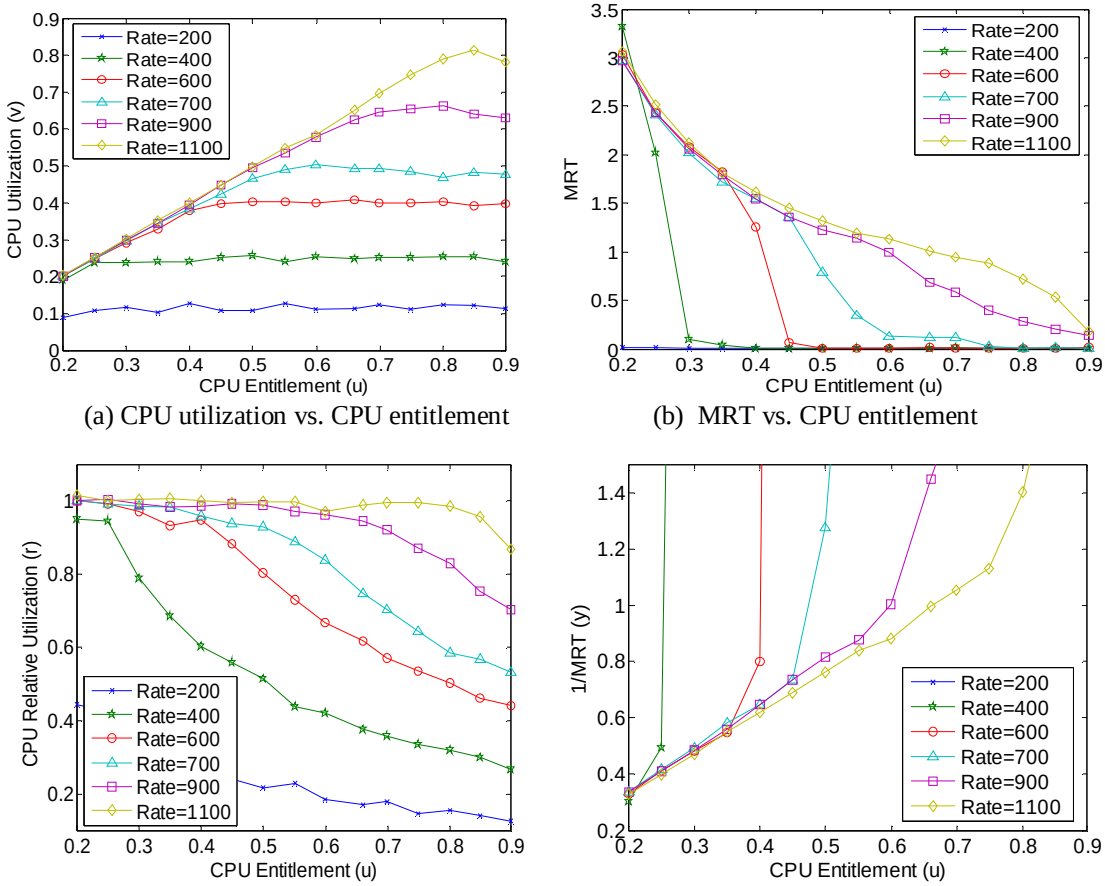
This work is the continuation of our earlier work in [21] where we designed and implemented an adaptive PI controller that regulates the MRT around a target value and self-tunes its gain parameters based on online estimation of the dynamic model. In the next section, we describe a new set of modeling experiments and analysis and demonstrate how the system's input-output relation changes along with various operating conditions of the system. As a result, we show that controlling the MRT using the CPU entitlement alone is only effective when the Web server partition's CPU utilization is close to its CPU entitlement, but may not work well when the application is underutilizing its entitled CPU. In this paper, we present alternative controller designs such as controlling the relative utilization of the partition, or incorporating CPU utilization information into the control of SLO-based metrics such as the MRT so that the closed-loop system achieves more robust performance across different operating regions.

4. Modeling of the Input-Output Relation

4.1 Static input-output relation

First, to understand the system's long-term average behavior in the whole operating range, we varied the CPU entitlement (denoted by u) for the Web server partition from 0.2 to 0.9, at 0.05 increments. At each setting, the Web server was loaded for 60 seconds with a fixed workload, while the average CPU utilization (denoted by v) of the Web server partition was observed and the MRT of all requests returned during this period was computed. Figure 2 shows the static relation between the CPU entitlement, the (absolute and relative) CPU utilization, and the MRT for different workload intensities ranging from 200 to 1100 requests/second (or *r/s*). Note that

each data point is the average of 10 samples obtained from 10 repeated experiments. In addition to *u* and *v*, let *y* denote the inverse of MRT (1/MRT), and *r* denote the relative CPU utilization of the partition, i.e., $r = v / u$.



(a) CPU utilization vs. CPU entitlement (b) MRT vs. CPU entitlement

(c) Relative CPU utilization vs. CPU entitlement (d) 1/MRT vs. CPU entitlement

Figure 2. Long-term relation between CPU entitlement, CPU utilization and MRT

Our key observations from these figures follow:

- As shown in Figure 2(a), for any given request rate, as the CPU entitlement varies, the CPU utilization demonstrates a clear *bimodal* behavior that can be approximated using the following equation:

$$v = \begin{cases} u, & \text{if } u < V, \\ V, & \text{if } u >= V, \end{cases} \quad (1)$$

  where V is the maximum portion of CPU needed for a given workload. For instance, a workload of 600 r/s requires 0.4 CPU. When the CPU entitlement is below 0.4, the system is in the *overload* region where the allocated CPU cycles are fully utilized, therefore making the utilization equal to the entitlement; when the CPU entitlement is above 0.4, however, the system is in the *underload* region where the allocated CPU cycles are underutilized. In the latter case, the CPU utilization stays relatively constant.

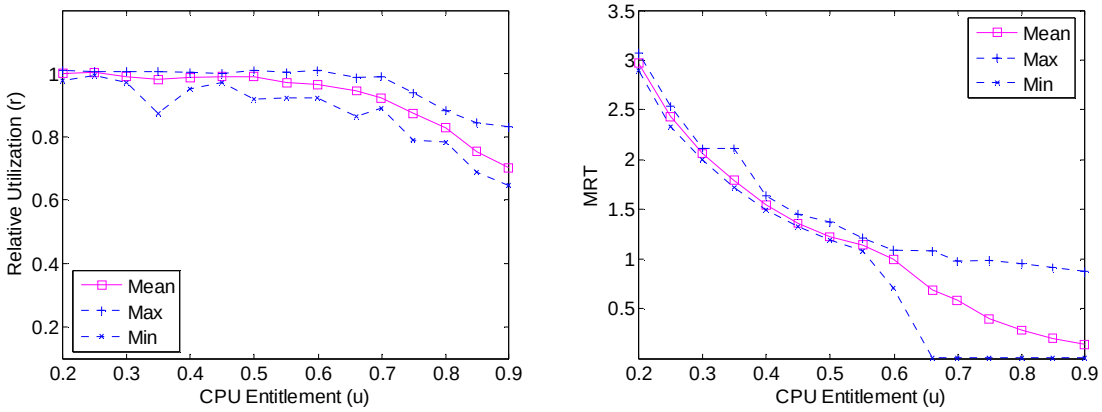
- Figure 2(c) shows a different visualization of the same behavior through the relative CPU utilization. The following equation is equivalent to (1), except expressing the CPU utilization in a relative term:

$$r = \begin{cases} 1, & \text{if } u < V, \\ V/u, & \text{if } u >= V. \end{cases} \quad (2)$$

This nonlinear equation will be useful in the analysis of the controllers in the next section.

- Similarly, the same bimodal behavior is observed in the relation between the MRT and the CPU entitlement in Figure 2(b). Since the MRT is clearly a nonlinear function of the CPU entitlement, we plot 1/MRT vs. CPU entitlement in Figure 2(d) to better illustrate the relation. As we can see, when the system is overloaded (r = 1 in Figure 2(c)), there exists a linear mapping from the CPU entitlement to 1/MRT, and its slope is independent of the request rate. However, when the system is underloaded (r < 1 in Figure 2(c)), 1/MRT increases rapidly with increasing CPU entitlement, indicating a sharp drop in the MRT.
- The linear mapping between the CPU entitlement and 1/MRT for the overload region implies that a linear input-output model is plausible for this region if 1/MRT is chosen as the system output.
- When the system is reasonably underloaded (r < 0.8), the MRT becomes independent of the CPU entitlement setting. Therefore, the MRT is uncontrollable using the CPU entitlement in this region.

Figure 3 shows the mean, maximum and minimum values of the relative CPU utilization and MRT from the 10 experiments for a request rate of 900 r/s. The relative utilization in Figure 3(a) shows a relatively small variation throughout the whole operating region, with the variation being slightly higher in the underload region. For the MRT in Figure 3(b), in the overload region where CPU entitlement is in [0*.2, 0.55], the MRT has a small variance. However, large deviation can be found in the underload region where the CPU entitlement is in [0.65 0.9], which means that MRT may not be taken as a stable metric in this region.



(a) Relative CPU utilization (b) Mean Response Time

Figure 3. Properties of relative utilization and MRT as metrics when rate is 900

Next, we describe how to obtain a linear dynamic model for controlling MRT through CPU entitlement in the overload region using standard system identification techniques.

4.2 Dynamic linear model identification

Similar to prior work, we chose the following linear auto-regressive model as the potential model to represent the dynamic relation between the CPU entitlement and the inverse of MRT:

$$y(k) = \sum_{i=1}^{n} a_i y(k-i) + \sum_{j=0}^{m-1} b_i u(k-d-j) + e(k), \quad (3)$$

where the parameters a_i, b_j , the orders *m*, *n*, and the delay *d* characterize the dynamic behavior of the system. *y(k)* is the inverse of MRT for sampling interval *k*, *u(k)* is the CPU entitlement for sampling interval *k*, and $e(k)$ is the residual term. For convenience, we refer to such a model as "ARX*mnd*" in the following discussion.

We performed two series of system identification experiments for various request rates and sampling intervals, where the CPU entitlement was randomly varied in [0.2, 0.8], and the resulting 1/MRT was calculated. The model in (3) with different structure and parameters was estimated offline using least-squares based methods [22] in the *Matlab System ID Toolbox* [25] to fit the input-output data. The models are evaluated using the r^2 metric defined in Matlab as a goodness-of-fit measure. In general, the r^2 value indicates the percentage of variation in the output captured by the model. In the first series, the sampling interval was fixed as 15 seconds while the rate was varied from 200 r/s to 1100 r/s, and the experiment was repeated for each rate. The results are shown in Tables 1(a), 1(b) and 1(c). In the second series, the rate was fixed at 900 r/s but the sampling interval was varied from 3 seconds to 20 seconds. The r^2 values (in %) of the resulting models are shown in Table 1(d).

Table 1. Model fitness and parameter values for the ARX input-output model

Model	Rate (r/s)					
	200	400	600	700	900	1100
ARX110	-10.2	12.8	2.8	63.1	70.3	78.3
ARX111	-1	6.7	2.7	-5	0.09	6.4

(a) r^2 of first-order models under different workloads

Rate (r/s)	Model			
	ARX110	ARX220	ARX330	ARX440
900	70.3	71.7	70.8	71.5
1100	78.3	79.9	80.3	80.2

(b) r^2 of models with different orders

Param.	Rate (r/s)				
	700	800	900	1000	1100
a	-0.013	0.11	0.17	0.22	0.3
b	1.21	1.21	1.21	1.12	1.03
b/(1-a)	1.2	1.36	1.46	1.43	1.47

(c) Parameters of ARX110 under different workloads

Model	Sampling Interval (seconds)				
	3	5	10	15	20
ARX110	5.4	25.9	77.9	74.8	79.5
ARX111	59	31.9	9.6	-33.1	-179.4

(d) r^2 of models under different sampling intervals

The observations we made from the above tables are the following:

- **The linear ARX model fits the input-output data for heavier workloads, not lighter workloads.** When the system is significantly underloaded, a simple linear model does not fit the input-output data as shown in Table 1(a) by the r^2 value for ARX110 (first-order model with no delay) for a rate below or equal to 600 r/s. This is consistent with our earlier observation from Figure 2(d). In contrast, when the request rate is above 600 r/s, the ARX 110 model fits quite well, providing a good basis for controller design. Moreover, ARX111 (first-order model with one-step delay) does not explain the system behavior for any request rate, showing that no significant delay is observed in the system dynamics for a sampling interval of 15 seconds.

- **Under all conditions where an ARX model is a good fit, a first-order model is sufficient.** Table 1(b) shows, using rates of 900 and 1100 r/s as examples, that increasing the order of the ARX model does not increase its fitness. Additionally, experience has shown that simpler models offer better robustness for the resulting control system.

- **Under overload conditions, the parameter values of the dynamic model changes with the workload intensity.** The evolution of the parameters *a* and *b* values of the ARX110 model as shown in Table 1(c) reveals that, with a heavier workload, the system contains more inertia and responds slower to changes in the CPU entitlement. However, the steady state

gain, $b/(1-a)$, remains relatively constant at around 1.4 (except for 700 r/s). This is consistent with the constant slope in Figure 2(d).

- **Models with different delay values should be chosen for different sampling intervals.** Table 1(d) shows how the fitness of the ARX110 (no delay) or ARX111 (one-step delay) model changes as we change the sampling interval T_s, under a workload of 900 r/s. For a short sampling interval (T_s=3 seconds), the ARX111 model provides a much better fit ($r^2 = 59.0\%$) than the ARX110 model ($r^2 = 5.4\%$), indicating a delay from the input to the output of around 3 seconds. It is mainly caused by the actuation delay in PRM. For longer sampling intervals (T_s >5 seconds), this delay can be ignored, and an ARX110 model provides a fairly good fit ($r^2 > 70\%$). For a sampling interval of 5 seconds, neither model provides a good enough fit. Because smaller T_s results in noisier data, and larger T_s causes slower response in the controller, both the sampling interval and the delay in the model should be chosen carefully before the controller design.

The above experiments and analysis were repeated for a different server, and the same qualitative results were observed. Our main conclusion is that, due to the existence of first-order ARX models for the dynamic relation between the CPU entitlement and 1/MRT when the system is overloaded, the MRT should be controllable using simple controllers such as the adaptive PI controller used in [21]. On the other hand, it will be quite challenging to regulate the MRT in the underload region because our observations from the modeling exercise suggest that the MRT is simply uncontrollable using the CPU entitlement as the only input. Because variation in the workload intensity may frequently move a resource partition between the two regions, it is desirable to design a controller that can manage the partition size across both operating regions.

From Figure 2, we know the essential reason that MRT is uncontrollable when the Web server partition is underutilized is that the MRT is no longer correlated with the CPU entitlement. However, the MRT should always be dependent upon the real CPU utilization of the Web server process. This was confirmed from the following exercise, where offline identification experiments were repeated when the CPU entitlement was randomly varied in two regions, as shown in Table 2. Under a fixed workload of 900 r/s, the system worked in the overload or underload region when the CPU entitlement was varied in [0.2, 0.5] or [0.5, 0.8], as can be seen from Figure 2(b). Consider the ARX110 models between the CPU entitlement and 1/MRT, the CPU utilization and 1/MRT, and the CPU entitlement and utilization. In the underload case where the entitlement range is [0.5, 0.8], 1/MRT is only weakly correlated with the CPU entitlement with r^2=26%. However, the r^2 value of the models between the CPU utilization and 1/MRT is always much higher. Therefore, it may be helpful to introduce the CPU utilization into the control loop for the MRT such that more robust designs can be achieved.

Table 2: r^2 (in %) of ARX110 models between different input-output pairs for different regions

Range of Entitlement	[0.2, 0.5]	[0.5, 0.8]
Entitlement --> 1/MRT	77.6	26
Utililzation --> 1/MRT	84.3	65.5
Entitlement --> Utilization	86	31.5

5. Controller Design and Performance Evaluation

The CPU utilization of a Web server is a common metric that is monitored to determine whether more or less CPU resource should be allocated to the server. The usage-based operation mode in the HP-UX Workload Manager [14] allows the relative CPU utilization of a PRM group to be controlled within a user-specified range, e.g., 50%-75%. Compared to SLO-based metrics such as response times, the relative utilization of the resource partition is easier to measure on the server

side, more directly related to the CPU entitlement and its control is more intuitive. The downside lies in that the relation between a given relative utilization level and the client-perceived service level varies with the demand of the workload. Therefore, no guarantees can be given to metrics such as the MRT for an arbitrary workload when only the relative utilization is being controlled. This is in contrast to using the MRT as the controlled output that is more directly related to the SLO but its relation with the CPU entitlement is rather complex. In addition, regulating the MRT using the CPU entitlement is more difficult in the underload region, as explained in Section 4. In this section, we present controller designs for dynamic sizing of the Web server partition using both output metrics, and discuss possible ways to combine these two metrics to provide more effective control across the whole operating region.

5.1 Control of relative utilization

We first consider dynamic sizing of the Web server partition using its relative utilization, *r(k)*, as the output and the CPU entitlement, *u(k)*, as the input. The goal is to maintain dynamically the relative utilization at a reference value, r_{ref}. This value can be chosen higher for more predictable workloads, and lower for more variable workloads. Prior work on applying control theory to regulating Web server CPU utilization used different input variables, such as the content adaptation level [2] or the application parameters [9].

From offline identification experiments, we observed that *r(k)* responds quickly to changes in *u(k)* with negligible delay and inertia when the sampling interval is set at 15 seconds. Therefore, the nonlinear static model (2) can be used to represent the input-output relation. i.e.,

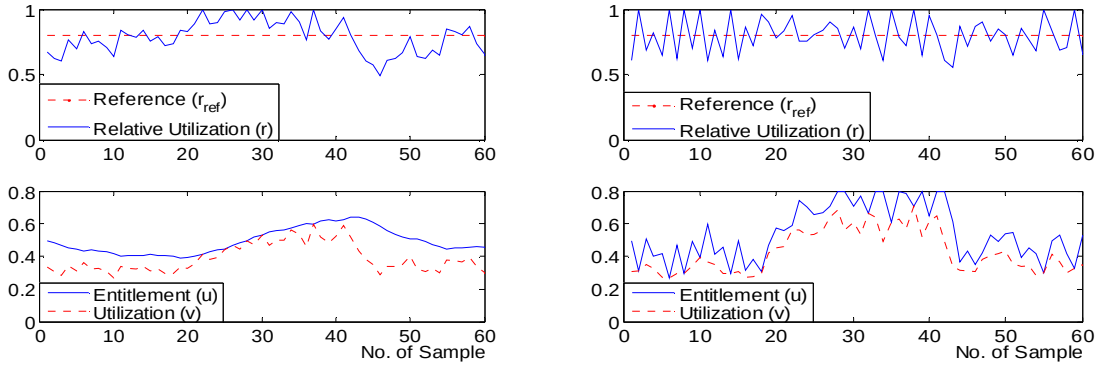
$$r(k) = \begin{cases} 1, & \text{if } u(k) < V; \\ V / u(k), & \text{if } u(k) >= V. \end{cases} \quad (2^*)$$

Define the tracking error at sampling interval k as $e(k) = r_{ref} - r(k)$. We can then use the classical integral (I) controller to dynamically tune the CPU entitlement based on the tracking error:

$$u(k) = u(k-1) - K_i e(k-1). \quad (4)$$

In theory, integral control ensures zero steady state error, i.e., the measured relative utilization should converge to r_{ref}, and the integral gain K_i determines the aggressiveness of the tuning. Note that *e(k)* is a non-decreasing function of *u(k)*. The minus sign in (4) is required to ensure a negative feedback loop for a positive K_i.

The main challenge here is to choose the right gain parameter K_i such that the closed-loop system is stable, and the relative utilization tracks the reference value as quickly as possible. This can be illustrated using an example. Let r_{ref} =80%. We started from a workload of 500 r/s, changed it to 800 r/s at the 20th sampling interval, and back to 500 r/s at the 40th interval. Figure 4 demonstrates the behavior of the closed-loop system with a K_i value of 0.1 and 1.0, respectively. The two top figures show both the measured relative utilization and its reference value. The two bottom figures show the allocated CPU entitlement and the measured (absolute) CPU utilization. As we can see, when the gain is too small as in Figure 4(a), the response to the workload change is very sluggish; when it is large, however, the response is unstable as shown in Figure 4(b). In this experiment and all those followed, the CPU entitlement was upper-bounded by 0.8. The x-axes in the figures on the control system performance are labeled in the number of sampling interval.



(a) $K_i = 0.1$ (b) $K_i = 1.0$

Figure 4. Performance of fixed I Controller from CPU entitlement to relative utilization

Although an optimum K_i value may be chosen carefully for certain workload, it may not be applicable to a different workload or a different operating region. Therefore, it should be desirable to design an adaptive controller with an adjustable gain parameter. Since the process is nonlinear, the controller design approach for linear systems such as pole placement cannot be applied. Instead, we can perform the following analysis. It does not solve the problem completely, but we can have some analytical direction on the parameter configuration.

We rely on the static models defined in (1) and (2*). Assume that at steady state, $u(\infty) = u_0$, $v(\infty) = V$. Consider the problem in the two operating regions respectively. In the overload case, $u(k-1) \le V$, $v(k-1) = u(k-1)$, then $e(k-1) = r_{ref} - 1$, and $u_0 = V / r_{ref} \ge u(k-1)/r_{ref}$. A safe but conservative option for $K_i(k)$ could be $u(k-1)/r_{ref}$ such that $u(k) = u(k-1)/r_{ref} \le u_0$. Note that this adaptive gain will lead to exponential increment of the CPU entitlement in the overload region since $r_{ref} < 1$. In the underload case, $u(k-1) > V$, and $u(k) = u(k-1) - K_i(r_{ref} - v(k-1)/u(k-1))$. Let us consider the local stability of this nonlinear system. Linearizing the equation around u_0, we derive that $K_i V / u_0^2 < 2$ is required for the local stability. This is a necessary condition. But we can still consider one candidate for K_i in this region as $K_i(k) = \mathrm{l}_1 v(k-1)/r_{ref}{}^2$, with some $0 < \mathrm{l}_1 < 2$. Figure 5(a) shows the performance of such a nonlinear adaptive controller where $\mathrm{l}_1 = 0.5$. The relative utilization is maintained around the reference, and the CPU entitlement tracks the changes in the workload fairly quickly.

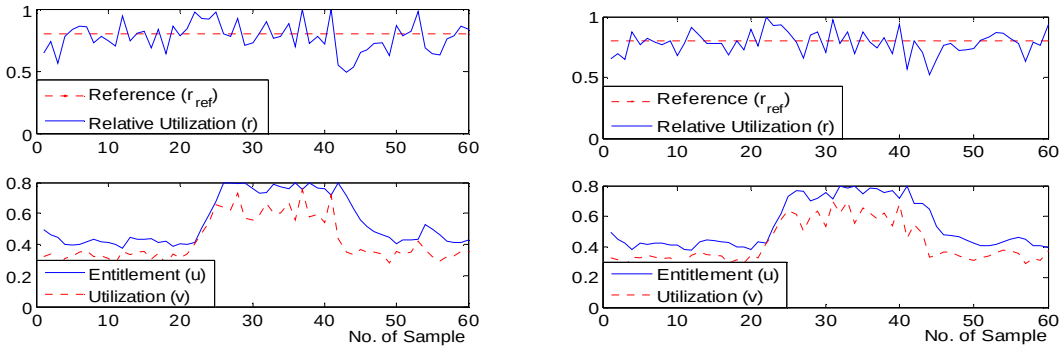
Because the nonlinearity in equation (2*) in the underload region, the analysis of the global stability for the controller in (4) is challenging. We perform the following transformation to linearize the system here. Let $\tilde{r}(k) = 1/r(k)$ be the output of the system. Then $\tilde{r}(k) = u(k)/V$ when $u(k) >= V$. Now define the tracking error at time k as $\tilde{e}(k) = \tilde{r}_{ref} - \tilde{r}(k)$, and use the following negative feedback integral controller:

$$u(k) = u(k-1) + \tilde{K}_i \tilde{e}(k-1). \tag{5}$$

We may set the gain $\tilde{K}_i(k)$ adaptive to the workload as follows:

$$\tilde{K}_i(k) = \begin{cases} u(k-1), & when\ u(k-1) = v(k-1), \\ \mathrm{l}_2 v(k-1), & when\ u(k-1) > v(k-1). \end{cases} \tag{6}$$

Then in the overload and underload cases, respectively, we can still have the exponential change of the entitlement when the workload changes. Moreover, global stability can be guaranteed in the latter case since the closed-loop system is linear. One example for such a controller with $l_2 = 0.5$ is shown in Figure 5(b), whose performance is similar to that in Figure 5(a).



(a) Nonlinear adaptive controller (b) Linear adaptive controller

Figure 5. Performance of adaptive I Controller from CPU entitlement to relative utilization

5.2 Control of mean response time

In this section, we demonstrate the challenges in controlling the mean response time compared to controlling the relative utilization. We first consider both a fixed PI controller and the adaptive PI controller presented in [21] that dynamically adjust the CPU entitlement for the Web server partition to meet its SLO-based MRT target. Using examples, we show that this adaptive controller is effective in handling changes in the target performance by online estimation of the dynamic model, but may not work well when a sudden change in the workload pushes the system into the underload region. We then describe a new controller design by introducing the CPU utilization measurement into the control loop.

Based on the analysis in Section 3, we consider the following ARX110 model to represent the dynamic relation between the CPU entitlement (u) and the inverse of MRT (y):

$$y(k) = ay(k-1) + bu(k) + \varepsilon(k). \quad (7)$$

Define the tracking error $e(k) = y_{ref}(k) - y(k)$, where $y_{ref}(k)$ is the target value for 1/MRT. Then a PI controller implements the following algorithm:

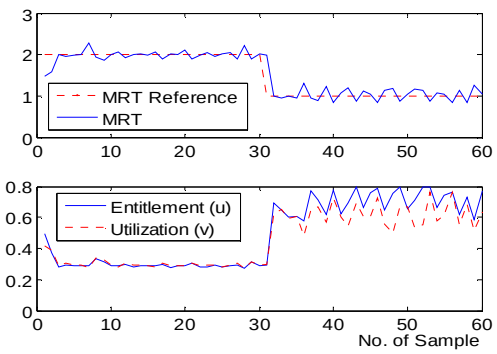
$$u(k) = u(k-1) + (K_p + K_i)e(k-1) - K_p e(k-2). \quad (8)$$

The closed-loop system is of second order and the gain parameters, K_p and K_i, can be chosen using the pole placement algorithm according to design specifications such as overshoot, rising time and settling time [5]. We applied both the fixed PI controller and the adaptive PI controller to regulate the MRT around its target. For the fixed PI controller, we used the model identified offline under a fixed rate 900 r/s with (a, b) = (0.20, 1.14), from which the best gain parameters were chosen. For the adaptive PI controller, the model parameters were estimated online and the gain parameters were also computed online to maintain the desired closed-loop poles.

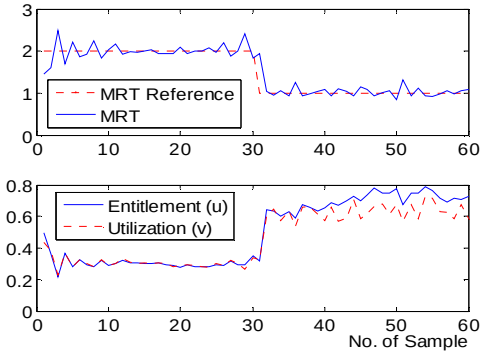
In the first experiment, a fixed workload with 900 r/s rate was sent to the Web server for a duration of 60 sampling intervals (15 minutes), while the target of MRT was changed from 2 seconds to 1 second at the 30th sampling interval. From Figure 2(b), we can see that these two steady states are located in the overload and underload regions, respectively. Figure 6(a) and 6(b) show the performance of the closed-loop system for both controllers for the duration of the experiment. The top figures compare the measured MRT to its target value, while the bottom figures show both the CPU entitlement actuated and the measured CPU utilization. Both controllers worked well when the system was overloaded. However, as in Figure 6(a), the CPU entitlement was oscillating heavily (hitting the upper bound periodically) when the model

parameters were fixed. This is because the gain parameters for the PI controller were chosen using the model estimated over the whole input region, and were too aggressive when the system became underloaded. The adaptive PI controller fixes this mismatch of the parameters and achieves more stable results as shown in Figure 6(b).

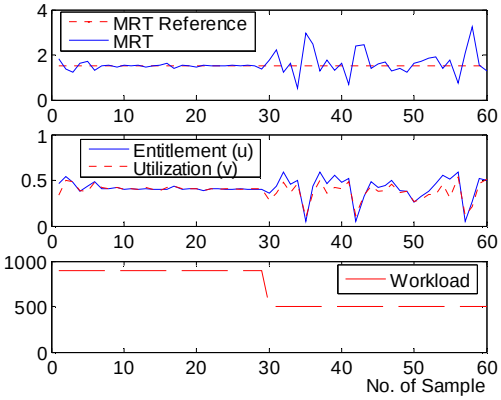
In the second experiment, the target MRT was fixed at 1.5 seconds, but the rate of the workload was changed from 900 r/s to 500 r/s at the 30th sampling interval, which pushes the system suddenly into the underload region. Figure 6(c) and 6(d) show the performance of the closed-loop system for both controllers. We can see that both the CPU entitlement and the resulting MRT became unstable because of the over-tuning actions of the controller. In this scenario, the adaptive controller only improves the performance very marginally.



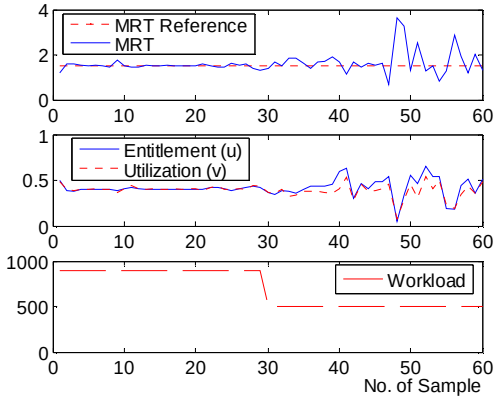
(a) Fixed PI controller upon change of reference



(b) Adaptive PI controller upon change of reference



(c) Fixed PI controller upon change of workload



(d) Adaptive PI controller upon change of workload

Figure 6. Performance of PI controllers from CPU entitlement to MRT

In summary, in the overload region, both the fixed and adaptive PI controllers work reasonably well, while the adaptive controller offers better performance in the slightly underload region (as in Figure 6(b)) by self-tuning of its parameters. However, neither controllers are applicable when the system is significantly underloaded (as in Figure 6(c) and 6(d)), where the loss of controllability of the MRT by the CPU entitlement leads to over-provisioning of the CPU resource. This is also consistent with our observation from Figure 3(b) that the MRT is not a stable metric in the underload region. Given the suggestion of Figure 3(a) that the CPU utilization is a more stable metric, we propose one design that attempts to incorporate the measured CPU utilization into the control loop to extend the controllable region, as illustrated in Figure 7.

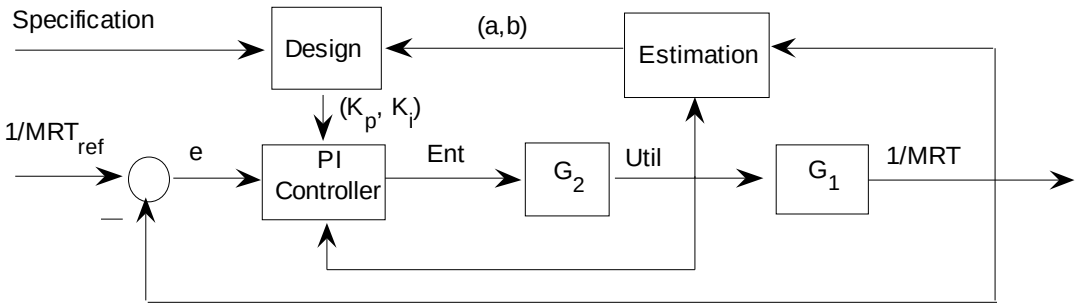


Figure 7. Block diagram of an adaptive control loop with incorporation of measured CPU utilization

From Section 3.2, we know that the CPU utilization has a tighter relation with the MRT than the CPU entitlement does in the underload region. In the following design, the ARX110 model was estimated online between the measured CPU utilization (*v(k)*, as the input) and 1/MRT (*y(k)*, as the output). Moreover, the term *u(k-1)* in the controller (8) is replaced by *v(k-1)* as follows:

$$u(k) = v(k-1) + (K_p + K_i)e(k-1) - K_p e(k-2). \tag{9}$$

The parameters were chosen according to the same specification as in the prior designs. The previous experiments with varying target MRT or workload intensity were repeated using the new controller in (9) and the closed-loop performance is shown in Figure 8. In both cases, the measured MRT is kept near the reference values. In the second case, even with a significantly reduced workload, the stability of the system is maintained.

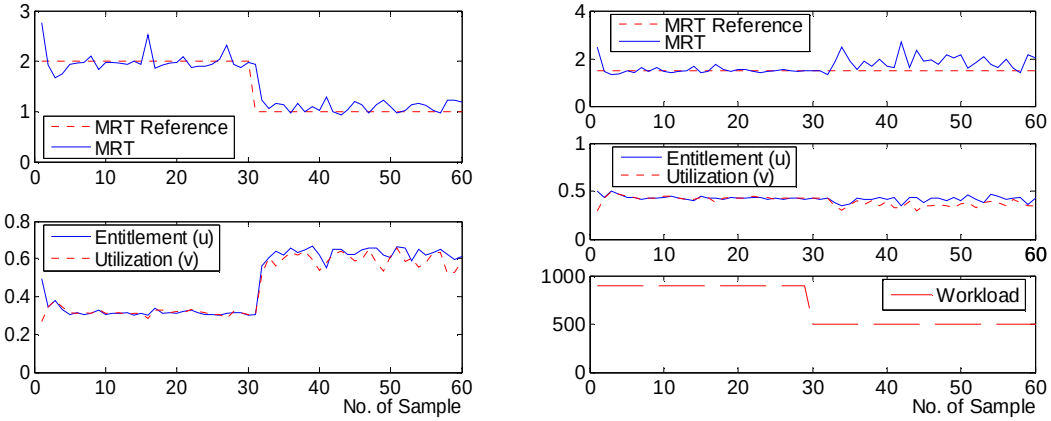
In this control design, introducing the CPU utilization into the model estimation leads to a more truthful and stable model. With the controller in (9), over-tuning of the CPU entitlement can be avoided since it is based on the measured utilization. However, one implicit assumption in this solution is that the utilization measurement tracks the entitlement immediately, that is, $G_2 = 1$. This is satisfied in the overload region where $v(k) = u(k)$. When the system works close to the overload region, it is an approximation of the relation between the entitlement and the utilization. Error exists in the underload region between the expected value of the utilization and its measurement. That is why, as shown in Figure 8, the measured MRT is above the target value when the system is underloaded. This steady-state error can be estimated approximately as follows. Assume that $v(k) = \beta u(k)$ with $\beta < 1$ and the closed loop is stable. Then the transfer function is, by omitting the estimation and design blocks,

$$G_c(z) = \frac{\beta b(K_p + K_i)(z - \frac{K_p}{K_p + K_i})}{(z-\beta)(z-a) + \beta b(K_p + K_i)(z - \frac{K_p}{K_p + K_i})}. \tag{10}$$

According to the Final Value Theorem, the DC gain of $G_c(z)$ is $G_c(1) < 1$ since $\beta < 1$. Note that the output is the inverse of the MRT. The MRT in steady state is larger than its reference on average.

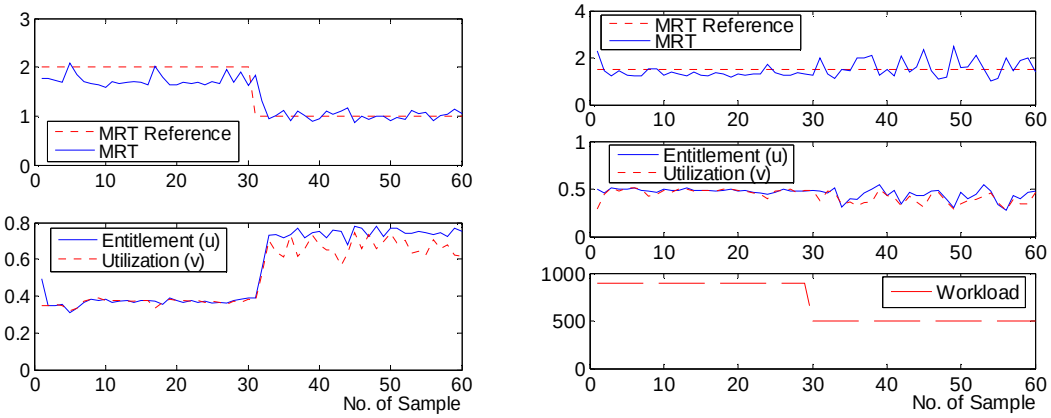
It is possible to eliminate this average error, for instance, by setting the entitlement appropriately larger than the expected utilization, or setting the reference proportional to $1/G_c(1)$ to compensate the steady state error. However, it is challenging to choose the right ratio β due to variations in the workload and measured utilization. Figure 9 shows the performance of the controller in the two scenarios when β is assumed to be a fixed value of 0.9, and the output of the controller is multiplied by $1/0.9$ so that $G_2 = 1$ is satisfied. In the first case when the reference value changes during the process, the steady-state error in the underload region is almost removed as shown in Figure 9(a). In the second case as shown in Figure 9(b), the error in the underload

region is decreased, but the output is noisier than before. In the overload region, new errors appear in both cases. This is because b is actually 1 in this region. These errors may be removed if the operating regions can be identified. However, there is no obvious boundary between the two regions and it is difficult to find the exact value for b. Therefore, the proposed solution can improve the robustness of the controller (8) significantly only in or close to the overload region.



(a) Varying target MRT (b) Varying workload intensity

Figure 8. Performance of the adaptive PI controller with incorporation of measured CPU utilization



(a) Varying target MRT (b) Varying workload intensity

Figure 9. Performance of the adaptive PI controller with incorporation of measured CPU utilization

b is assumed to be 0.9 and u is set to $1/0.9$ of the controller output.

6. Conclusions

This paper identifies challenges in applying control theory to dynamic sizing of a resource partition using CPU entitlement as the input and the mean response time or the relative CPU utilization as the output metric. We emphasize that the input-output relation has to be accurately modeled for the theoretical properties of the closed-loop control system to be meaningful. We recognize that this relation varies significantly as the resource partition moves between the overload and the underload regions, which has a noticeable impact on the performance of any controller design. We evaluate the performance of the closed-loop system using either relative utilization or the mean response time as the controlled output, and discuss their respective advantages and issues. Finally, we present a new adaptive controller design for regulating the

mean response time that incorporates information on measured CPU utilization and improves the robustness of prior adaptive algorithms.

To make the system work well across all operating regions, we need to respect the bimodal behavior of system and develop a better way to integrate the control of relative utilization (using controller (4-5)) and the response time (using controller (8)) in possibly different regions. Note that the targets on the relative utilization and the MRT may not be tracked simultaneously in most regions. To make the combination possible, one scheme is to give the regulation of the relative utilization a higher priority than that for the regulation of the MRT, and to design an intelligent switching scheme between these two controllers as the system state moves across different operating regions. It could be applied to the control of multiple resource containers, where the objectives on relative utilization can be guaranteed when there is no contention for resource among the containers, and then the controller for MRT comes into play when the system is overloaded. This is one topic of our ongoing work.

Another interesting direction is to apply the same approach to dynamic sizing of a resource partition in terms of its physical memory allocation. The distinct interaction between application performance and its memory may make it much more challenging to design a sensible controller that works under all operating conditions.

References

[1] T.F. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson, ``Practical application of control theory to Web services," invited paper, *American Control Conference*, June 2004.

[2] T.F. Abdelzaher, K.G. Shin, and N. Bhatti, "Performance guarantees for Web server end-systems: A control-theoretical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, 2002.

[3] J. Almeida, M. Dabu, A. Manikutty and P. Cao (1998), "Providing differentiated levels of service in Web content hosting," *SIGMETRICS Workshop on Internet Server Performance*, June 1998.

[4] Apache Web server, http://www.apache.org/

[5] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning (2nd Edition)*, Instrument Society of America, 1995.

[6] K.J. Astrom and B. Wittenmark, *Adaptive Control (2nd Edition)*, Prentice Hall, 1994.

[7] G. Banga, P. Druschel, and J.C. Mogul, "Resource Containers: A new facility for resource management in server systems," *3rd USENIX Symposium on Operating Systems Design and Implementation*, Feb. 1999.

[8] P. Bhoj, S Ramanathan, and S. Singhal, "Web2K: Bringing QoS to Web servers," *HP Labs Technical Report*, HPL-2000-61, May 2000.

[9] Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury, "MIMO control of an Apache Web server: Modeling and controller design," *American Control Conference*, 2002.

[10] L. Eggert and J. Heidemann, "Application-Level differentiated services for Web servers," *World Wide Web Journal*, Vol. 3, No. 1, pp. 133-142, March, 1999.

[11] P. Goyal, X. Guo, and H. Vin, "A hierarchical CPU scheduler for multimedia operating systems," *2nd USENIX Symposium on Operating System Design and Implementation*, October, 1996.

[12] J.L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*, Wiley-Interscience, 2004.

[13] HP Process Resource Manager, http://h30081.www3.hp.com/products/prm/index.html

[14] HP-UX Workload Manager, http://h30081.www3.hp.com/products/wlm/index.html

[15] IBM Application Workload Manager, http://www.ibm.com/servers/eserver/xseries/systems_management/director_4/awm.html

[16] IBM Enterprise Workload Manager, http://www.ibm.com/developerworks/autonomic/ewlm/

[17] M.B. Jones, D. Rosu, and M.-C. Rosu, "CPU reservations and time constraints: Efficient, predictable scheduling of independent activities," *16th ACM Symposium on Operating Systems Principles*, 1997.

[18] V. Kanodia and E. Knightly, "Multi-Class latency-bounded Web services," *8th IEEE International Workshop on Quality of Service*, June, 2000.

[19] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," *12th IEEE International Workshop on Quality of Service*, 2004.

[20] A. Kamra, V. Misra, and E.M. Nahum, "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites," *IEEE International Workshop on Quality of Service*, June, 2004.
[21] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource partitions on shared servers," *9th International Symposium on Integrated Network Management*, May, 2005.
[22] L. Ljung, *System Identification: Theory for the User (2nd Edition)*, Prentice Hall, 1999.
[23] C. Lu, T.F. Abdelzaher, J. Stankovic, and S. Son, "A feedback control approach for guaranteeing relative delays in Web servers," *IEEE Real-Time Technology and Applications Symposium*, 2001.
[24] Y. Lu, C. Lu, T. Abdelzaher, and G. Tao, "An adaptive control framework for QoS guarantees and its application to differentiated caching services," *IEEE International Workshop on Quality of Service*, May, 2002.
[25] Matlab System Identification Toolbox, http://www.mathworks.com/products/sysid/
[26] C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating system support for multimedia applications," *International Conference on Multimedia Computing and Systems*, 1994.
[27] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource Kernels: A resource-centric approach to real-time and multimedia systems," *ACM Conference on Multimedia Computing and Networking*, 1998.
[28] D.C. Steere, *et al.*, "A feedback-driven proportion allocator for real-rate scheduling," *3rd USENIX Symposium on Operating System Design and Implementation*, 1999.
[29] SUN Solaris Resource Manager, http://www.sun.com/software/resourcemgr/index.html
[30] C. Waldspurger and W. Weihl, "Lottery Scheduling: Flexible proportional-share resource management," *1st USENIX Symposium on Operating System Design and Implementation*, 1994.