Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions

Zhikui Wang, Xiaoyun Zhu, and Sharad Singhal

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

Abstract. This paper deals 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 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 identify the nonlinear and bimodal properties of the models across different operating regions, and discuss their implications for the design of the control loops. To deal with these challenges, we then propose two adaptive controllers for tracking the target utilization and target response time respectively. We evaluate the performance of the closed-loop systems while varying certain operating conditions. We demonstrate that better performance and robustness can be achieved with these controllers compared with other controllers or our prior solution.

1 Introduction

Resource partitioning is a type of virtualization technology that enables multiple applications to share the system resources on a single server while maintaining performance isolation and differentiation among them [1][2][3]. 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.

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 work is the continuation of our earlier work in [4] where we used a resource partition to host an Apache Web server and designed and implemented an adaptive PI controller to regulate the mean response time (MRT) of HTTP requests around a target value. The controller self-tunes its gain parameters

J. Schönwälder and J. Serrat (Eds.): DSOM 2005, LNCS 3775, pp. 133 – 144, 2005.

based on online estimation of the dynamic model. In this paper, we describe a new set of modeling experiments and demonstrate how the system's input-output relation changes with various operating conditions of the system. As a result, we show that controlling the MRT using the CPU entitlement alone is 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. We present an alternative design for controlling the relative utilization of the partition, scalable to the time-varying workload. We also propose to incorporate CPU utilization information into the control of SLO-based metrics so that the closed-loop system achieves more robust performance across different operating regions.

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 its implication on control designs. Section 5 presents different controllers and their performance evaluation using our test bed. Finally, we summarize our results, along with directions for future work in Section 6.

2 A Case Study Using a Feedback Control Approach

We conducted the case study where we used the *HP-UX Process Resource Manager* (PRM) [1] 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. 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. We consider an FSS (Fair Share Scheduler) PRM group that is assigned a percentage of the CPU cycles (referred to as "CPU entitlement") by specifying a number of shares. Because this percentage is enforced by the scheduler in the HP-UX kernel, it can be changed at any time thereby enabling dynamic sizing of the PRM group. We describe the architecture and test bed setup in this section.

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.

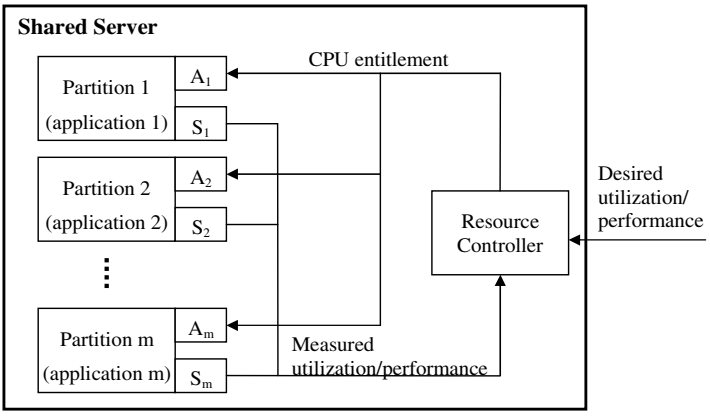


Fig. 1. CPU entitlement control system architecture

In the case study, we took the *Apache Web server* (version 2.0.52) as an example of the hosted applications. We set up an FSS PRM group on an HP-UX server to host the Web server. 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. 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, ensuring that CPU was the only potential bottleneck resource in the system as the workload intensity varied.

3 Related Work

Our approach differs from prior work on operating systems support for server resource reservation and enforcement [5]-[7] or scheduling [8][9] in that it is more generic and can be used on any commodity operating system that supports a resource partitioning technology, and applications that can be hosted inside a partition. In [10] a feedback-driven adaptive scheduler was presented to allocate a percentage of CPU cycles to a thread. 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 [11][12], our approach is distinct in that we rely on classical control theory to guide the design of the algorithms.

Feedback control theory has been applied to solve a number of performance or quality of service (QoS) problems in computer systems in recent years. (See [13][14]

and the references therein.) The effect of this approach depends heavily on the fitness of the mathematical models used to characterize the dynamic behavior of the systems. Much prior work employs a "black-box" approach and uses linear input-output models to capture the dynamic relation between control knobs (inputs) and performance metrics (outputs). However, a single linear model is often insufficient to uniformly capture a system's behavior under all operating conditions. More recent work addresses this issue by applying adaptive control theory to computer systems such as storage systems [15], resource containers [4] and caching services [16]. This approach allows the parameters of the linear models to automatically adapt to changes in operating conditions using online system identification. In [17] 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, while 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 this paper, we focus on the system's nonlinear and bimodal behavior, and present quantitative study on model fitness and variability.

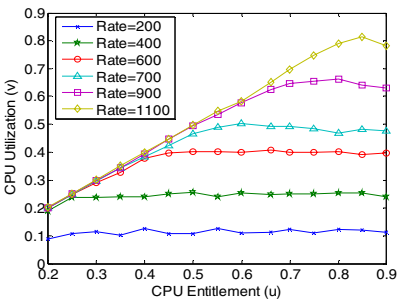
Performance control of Web servers has been studied extensively in the literature. For instance, application-level mechanisms were proposed in [18][19][20] to provide different levels of service to requests of different classes. While these approaches were mainly based on heuristics or queuing models, other work has applied classical control theory to manage Web server delay or server resource utilization using admission control [21] and content adaptation [17], connection scheduling and process reallocation [22], or application parameter tuning [23]. All of these methods require modification to the server application software (with the exception of [20]), which may not be feasible for other enterprise applications. Our focus is not controlling Web server performance in particular, but rather providing a general approach for dynamic sizing of any resource partitions.

4 Modeling of the Input-Output Relation

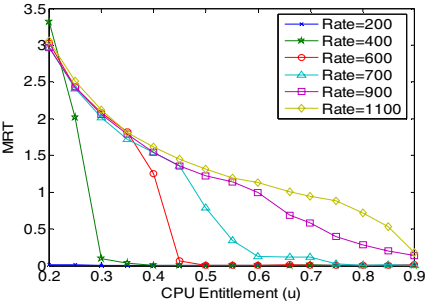
We first describe a set of modeling experiments and demonstrate the nonlinear and bimodal behavior of the system.

4.1 Static Input-Output Relation

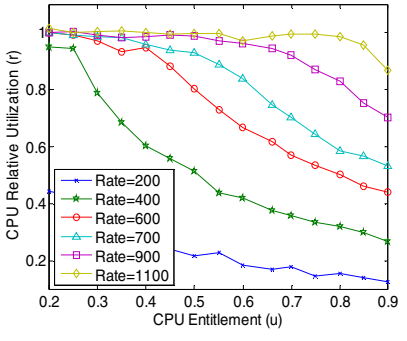
To understand the system's long-term average behavior in the whole operating range, we varied the CPU entitlement (denoted by u) for the Web server partition from 0.2 to 0.9, at 0.05 increments. At each setting, the Web server was loaded for 60 seconds with a fixed workload, while the average CPU utilization (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., $r = v / u$.



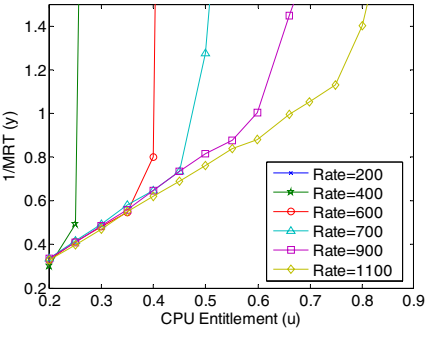
(a) CPU utilization vs. CPU entitlement



(b) MRT vs. CPU entitlement



(c) Relative CPU utilization vs. CPU entitlement



(d) 1/MRT vs. CPU entitlement

Fig. 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:

$$v = \begin{cases} u, & \text{if } u < V, \\ v, & \text{if } u >= V. \end{cases} \tag{1}$$

  Here V is the maximum portion of CPU needed for a given workload. 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:

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

- 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, we expect that the MRT is uncontrollable using the CPU entitlement in this region. In the next Section, we verify this behavior using model identification.

4.2 Dynamic Linear Model Identification

We chose the following linear auto-regressive model as the potential one to represent the dynamic relation between the CPU entitlement and the inverse of MRT:

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

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 $\varepsilon(k)$ is the residual term. For convenience, we refer to such a model as "ARX*mnd*" in the following discussion.

In the experiments, the CPU entitlement was randomly varied in [0.2, 0.8]. The sampling interval was fixed at 15 seconds while the rate varied from 200 *r/s* to 1100 *r/s*. The experiment was repeated for each rate. The model in (3) was estimated offline using least-squares based methods [24] in the *Matlab System ID Toolbox* [25] to fit the input-output data collected from the experiments. 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.

Table 1. r^2 values (in percentage) of first-order models

(a) under different workloads

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

(b) for different input-output pairs

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

From the data in Tables 1(a), we can find that a simple linear model does not fit the input-output data when the system is significantly underloaded, i.e., with 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. Other observation can be made on time-varying parameters along with change of workload, different model delays when the sampling interval was changed significantly. For more detailed analysis, see [26].

Similar 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 [4]. 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.

From Figure 2, we know that the MRT is not correlated with the CPU entitlement in the underload region. 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 the two regions, as shown in Table 1(b), under a fixed workload of 900r/s. 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 should be helpful to introduce the CPU utilization into the control loop for the MRT so that more robust performance can be achieved.

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. 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 is that the relation between a given relative utilization level and the client-perceived service level varies with the resource demand of the workload. 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 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. 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.

Define the tracking error at sampling interval k as

$$e(k) = r_{ref} - r(k). \tag{4}$$

We can then use the classical integral (I) controller,

$$u(k) = u(k-1) - K_i e(k-1), \tag{5}$$

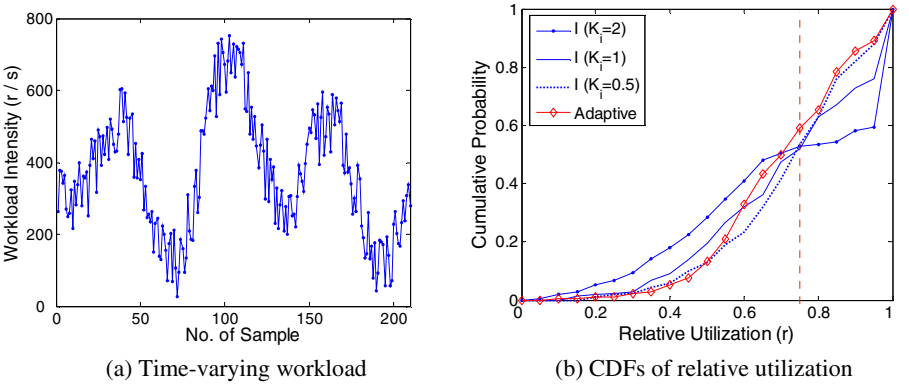
to dynamically tune the CPU entitlement based on the tracking error. In theory, integral control ensures zero steady state error, i.e., the measured relative utilization should converge to r_{ref}, and the integral gain K_i determines the aggressiveness of the tuning. The main challenge here is to choose the right gain parameter such that the closed-loop system is stable, and the relative utilization tracks the reference value as quickly as possible. Although an optimum K_i value may be chosen carefully for certain workload, it may not be applicable to a different workload. Based on the analysis in Section 4, we propose an I controller:

$$u(k) = u(k-1) - K_i(k) e(k-1) \tag{6}$$

with a time-varying adaptive gain:

$$K_i(k) = \begin{cases} \lambda_1 u(k-1)/r_{ref}, & \text{when } u(k-1) = v(k-1), \\ \lambda_2 v(k-1)/r_{ref}, & \text{when } u(k-1) > v(k-1). \end{cases} \tag{7}$$

The intuition behind this adaptive gain is from the bimodal property of the relative utilization w.r.t. CPU entitlement. When the system is overloaded, the CPU utilization is actually capped by the entitlement. The system needs to react fast to the increase of the workload. When underloaded, it is desirable for the system to be conservative to avoid going into the overload region, which may lead to unavailability and large response times. The controller (6-7) can act as expected with different gains in the two regions. It is scalable and adaptive to the workload, and shows good convergence performance when λ_2 is in (0, 2). More analysis can be found in [26].



(a) Time-varying workload (b) CDFs of relative utilization

Fig. 3. Performance of controllers from entitlement to relative utilization

The performance of the adaptive controller (6-7) was tested in an experiment where a synthetic workload as shown in Figure 3(a) was applied. For comparison, our adaptive controller was used along with an I controller with different fixed gains to regulate the relative utilization at around 75%. The target (relative utilization) tracking performance for different controllers is shown in Figure 3(b) by the cumulative distributions of the resulting relative utilization, where the vertical dashed line indicates the ideal distribution. Other performance measures such as average CPU entitlement, throughput, and mean and 95th percentile of response times are compared in Table 2. It can be observed that the adaptive controller achieves better tracking performance upon change of the workload, lower CPU consumption, higher throughput and smaller response times compared to the I controller with fixed gains.

Table 2. Average performance of the utilization controllers

Controller	CPU Ent	Throughput (r/s)	MRT (sec)	95-p RT (sec)
I (Ki=2)	0.43	359	1.02	4.15
I (Ki=1)	0.38	368	0.51	2.49
I (Ki=0.5)	0.38	363	0.64	2.71
Adaptive	0.37	370	0.31	1.69

5.2 Control of Mean Response Time

In this section, we highlight the challenges in controlling the mean response time (MRT). Using examples, we show that even the adaptive PI controller presented in [4] 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 4, we consider an ARX110 model to represent the dynamic relation between the CPU entitlement (u) and the inverse of MRT (y), which is estimated online as done in [4]. Define the tracking error

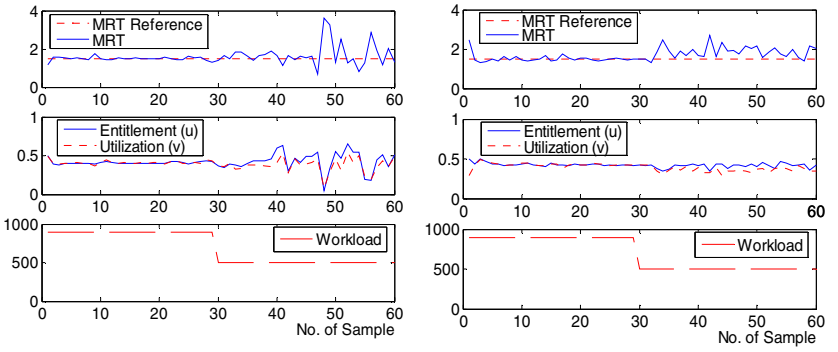
$$e(k) = y_{ref} - y(k), \tag{8}$$

where $y_{ref}(k)$ is the target value for 1/MRT. Then a PI controller implements the following algorithm:

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

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 [28].

The controller (9) with adapted parameters was tested in an experiment where 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 4(a) shows the performance of the closed-loop system, where we can see that both the CPU entitlement and the resulting MRT became unstable because the loss of controllability of the MRT by the CPU entitlement leads



(a) Adaptive PI Controller (b) Improved Adaptive PI controller

Fig. 4. Performance of controllers from CPU entitlement to MRT

to over-provisioning of the CPU resource. This is 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 5, where G_2 is the mapping from CPU entitlement to CPU utilization, and G_1 is the mapping from CPU utilization to 1/MRT.

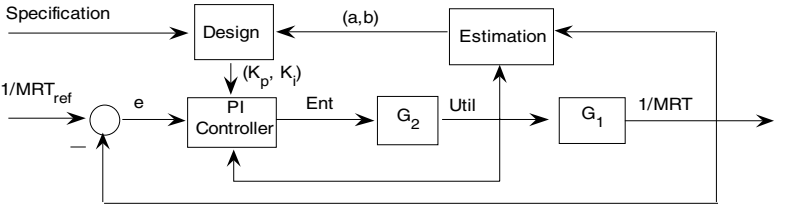


Fig. 5. An adaptive control loop with incorporation of measured CPU utilization

From Table 1(b) in Section 4.2, 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 PI controller in (9) is replaced by *v(k-1)* as follows:

$$u(k) = v(k-1) + (K_p + K_i)e(k-1) - K_p e(k-2) \tag{10}$$

The parameters were chosen according to the same specification as in the prior designs. The previous experiments with varying workload intensity were repeated using the new controller in (10). The closed-loop performance is shown in Figure 4(b), which shows that the stability of the system is maintained, even with a significantly reduced workload. In this control design, introducing the CPU utilization into the model estimation leads to a more truthful and stable model. Moreover, 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 $v(k)=u(k)$. Offset exists in the underload region between the expected value of the utilization and its measurement. That is why, as shown in Figure 4(b), the measured MRT is above the target value when the system is underloaded. This steady-state error can be estimated approximately and fixed partially as suggested in [26]. Therefore, the proposed solution can improve the robustness of the controller (9) significantly only in or close to the overload region.

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. We recognize that this input-output 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 closed-loop performance of an adaptive integral controller for controlling relative utilization of a resource partition. We also 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 the system and develop a better way to integrate the control of relative utilization (using controller (6-7)) and the response time (using controller (10)) in possibly different regions. 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] HP Process Resource Manager, http://h30081.www3.hp.com/products/prm/index.html

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

[3] SUN Solaris Resource Manager, http://www.sun.com/software/resourcemgr/index.html

[4] 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.

[5] 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.

[6] 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.

[7] 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.
[8] 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.
[9] C. Waldspurger and W. Weihl, "Lottery Scheduling: Flexible proportional-share resource management," 1st USENIX Symposium on Operating System Design and Implementation, 1994.
[10] D.C. Steere, et al., "A feedback-driven proportion allocator for real-rate scheduling," 3rd USENIX Symposium on Operating System Design and Implementation, 1999.
[11] HP-UX Workload Manager, http://h30081.www3.hp.com/products/wlm/index.html
[12] IBM Enterprise Workload Manager, http://www.ibm.com/developerworks/autonomic/ewlm/
[13] J.L. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, Feedback Control of Computing Systems, Wiley-Interscience, 2004.
[14] 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.
[15] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," 12th IEEE International Workshop on Quality of Service, 2004.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] V. Kanodia and E. Knightly, "Multi-Class latency-bounded Web services," 8th IEEE International Workshop on Quality of Service, June, 2000.
[21] P. Bhoj, S Ramanathan, and S. Singhal, "Web2K: Bringing QoS to Web servers," HP Labs Technical Report, HPL-2000-61, May 2000.
[22] 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.
[23] 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.
[24] L. Ljung, System Identification: Theory for the User (2nd Edition), Prentice Hall, 1999.
[25] Matlab System Identification Toolbox, http://www.mathworks.com/products/sysid/
[26] Z. Wang, X. Zhu, S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions", HP Labs Technical Report, HPL-2005-126, July 2005.
[27] Apache Web server, http://www.apache.org/
[28] K. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning (2nd Edition),* Instrument Society of America, 1995.