Utility-Driven Workload Management using Nested Control Design

Xiaoyun Zhu, Zhikui Wang, Sharad Singhal
Enterprise Software and Systems Laboratory
HP Laboratories Palo Alto
HPL-2005-193(R.1)
March 29, 2006*

virtualization, workload management, utility function, control theory, nested control

The emerging interests within the IT industry on utility computing and virtualization technologies have created a need for more effective workload management tools, one that dynamically controls resource allocation to a hosted application to achieve quality of service (QoS) goals. These goals can in turn be driven by the utility of the service provided, typically based on the application's service level agreement (SLA) as well as the cost of resources allocated to the application. In this paper, we build on our earlier work on dynamic CPU allocation to applications on shared servers, and present a feedback control system consisting of two nested integral control loops for managing the QoS metric of the application along with the utilization of the allocated CPU resource. The control system was implemented on a lab testbed using the Apache Web server as the application and the 90th percentile of the response times as the QoS metric. By testing the system using a synthetic workload based on an industry benchmark, we validate the two important features of the nested control design. First, compared to a single loop for controlling response time only, the nested design is less sensitive to the bimodal behavior of the system resulting in more robust performance. Second, compared to a single loop for controlling CPU utilization only, the new design provides a framework for dealing with the tradeoff between better QoS and lower cost of resources, therefore resulting in better overall utility of the service.

* Internal Accession Date Only
Published in and presented at the American Control Conference, 14-16 June 2006, Minneapolis, MN, USA

© Copyright 2006 American Automatic Control Council

Utility-Driven Workload Management using Nested Control Design

Xiaoyun Zhu, Zhikui Wang and Sharad Singhal
Hewlett-Packard Laboratories
Palo Alto, CA 94304

***Abstract*—The emerging interests within the IT industry on utility computing and virtualization technologies have created a need for more effective workload management tools, one that dynamically controls resource allocation to a hosted application to achieve quality of service (QoS) goals. These goals can in turn be driven by the utility of the service provided, typically based on the application's service level agreement (SLA) as well as the cost of resources allocated to the application. In this paper, we build on our earlier work on dynamic CPU allocation to applications on shared servers, and present a feedback control system consisting of two nested integral control loops for managing the QoS metric of the application along with the utilization of the allocated CPU resource. The control system was implemented on a lab testbed using the Apache Web server as the application and the 90th percentile of the response times as the QoS metric. By testing the system using a synthetic workload based on an industry benchmark, we validate the two important features of the nested control design. First, compared to a single loop for controlling response time only, the nested design is less sensitive to the bimodal behavior of the system resulting in more robust performance. Second, compared to a single loop for controlling CPU utilization only, the new design provides a framework for dealing with the tradeoff between better QoS and lower cost of resources, therefore resulting in better overall utility of the service.**

I. Introduction

THE information technology (IT) industry has in the past few years undergone a major trend of infrastructure consolidation to improve utilization of IT resources, reduce operational cost, and increase return on IT investments. As a result, there has been emerging interest on virtualization technologies that allow common IT resources (such as processors, networks, storage, and software licenses) to be shared across multiple users, organizations, or applications. Examples of these technologies include workload management tools such as *HP-UX workload manager* [1], *IBM enterprise workload manager* [2], and virtual machine technologies such as *VMware* [3] and *Xen* [4]. In this paper, we present two approaches for improving current workload management tools so that they can be used more effectively.

A. Dynamic Feedback-Driven Resource Allocation

A typical workload management tool allows multiple applications to share the system resources (CPU, memory, disk bandwidth) of a single server, while maintaining performance isolation and differentiation between them. The amount of resource allocated to a particular application can be fixed or dynamically adjusted based on certain policies. However, it is a challenging task for the data center operators to provision sufficient resources for enterprise applications running on shared servers using workload management tools. One reason is that many of these applications have time-varying resource demands with a high peak-to-mean ratio.

Fig. 1(a) shows the measured average CPU usage of one server in an enterprise data center on a typical day, with one sample for every five minutes. Because this server has 6 CPUs, the first observation is that the average CPU utilization is very low (< 10%) most of the time. Second, the maximum CPU usage is approximately an order of magnitude higher than the minimum. Therefore, if the application running on this server were consolidated onto a shared server, fixed CPU allocation based on either the peak or the mean would not work because the former would waste a lot of resource while the latter would leave the application overloaded during the demand surge between 10am and 2pm causing performance degradation. Dynamic CPU allocation to this application should be desirable. On the other hand, this application actually has strong periodicity where the pattern shown in Fig. 1(a) repeats itself daily. One might argue that the operator can schedule the CPU resource accordingly once this pattern is learned. However, not every server or application has a resource demand that is predictable.

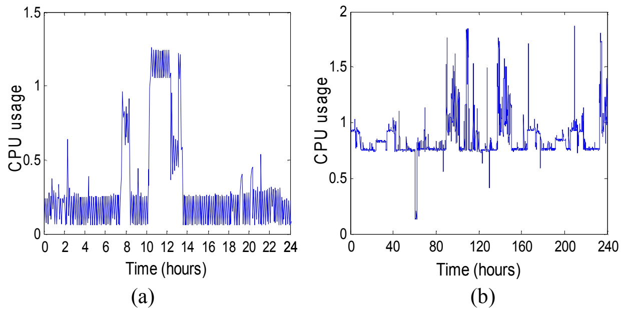


Fig. 1. CPU usage of server A in a data center for one day (a) and CPU usage of server B for ten days (b)

Fig. 1(b) shows ten days worth of data for the CPU usage on another server from the same data center. Clearly there is no daily pattern in this trace, and spectrum analysis on seven weeks of data shows that no other periodicity is present either. For this type of applications, real-time feedback mechanisms are necessary to allocate "capacity on demand" so that the application gets what it needs and resources are utilized more efficiently.

Email: {xiaoyun.zhu, zhikui.wang, sharad.singhal@hp.com}

B. *Utility-driven QoS goals*

A second reason why it may be hard to configure workload management solutions is that, although the tools usually allow the users to specify goals (fixed value or a range) for certain quality of service (QoS) metrics, such as throughput or latency, operators often find it hard to define such goals that are sensible to a particular application. One way to resolve this is to relate these goals to a meaningful utility function. Let *y* be the QoS metric for an application, then the payoff *z* defined in a service level agreement (SLA) may be a convex or concave function (depending on what the metric is) or piecewise-linear function of *y*. Fig. 3 shows an example where *y* is the latency, so higher value of y means poorer QoS, and the payoff function $z(y) = \max\{\alpha(1 - e^{y - y_m}), 0\}$. This means that the service provider gets a positive payoff from the user only when y is below a threshold y_m, and the payoff decreases more and more rapidly as the latency approaches the threshold, after which the payoff becomes zero. Let us also assume that the cost of resources allocated to the application, *c*, is a monotonically decreasing function of *y* because higher cost is usually involved in providing better QoS (smaller y). Fig. 3 gives an example of a cost function where c(y) = β/y. Now let the utility of the service be h(y) = z(y) – c(y). Fig. 3 shows that h(y) is a concave function for y < y_m. Ideally, we would like to find the optimum operating point for y with maximum utility, but it may be hard to maintain y exactly at the optimum. An alternative is to find a range [Y_L, Y_H] such that h(y) is above a threshold, or close to the optimum. (See Fig. 3 for an example.) In this paper, we assume that both the payoff function and the cost function are known so that such a range can be computed. We then focus on designing a closed-loop control system that maintains the QoS metric within this range.

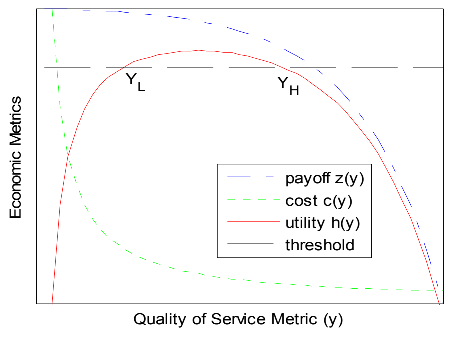


Fig. 3. Economic metrics (payoff/cost/utility) vs. quality of service (QoS) metric (e.g., latency) for a hosted application

C. *Related Work*

Feedback control theory has been applied to solve a number of performance or quality of service problems in computing systems in the past few years [5][6]. However, the intrinsic complexity of computing systems poses many challenges on control system designs [7].

Much of the early work assumed that the system under control has linear dynamics, and the model parameters can be identified offline, e.g., in [8], where a MIMO controller was used for automatic parameter tuning of a Web server. However, due to the wide variation of demands observed in computing systems, the parameter values or even the structure of the models may change slowly or rapidly over time. To deal with time-varying parameters in linear models, adaptive control theory has been applied to systems in the context of, for instance, caching services [9], storage systems [10] and resource containers [11]. Nevertheless, it is usually not enough to consider only the linear behavior since much of systems' behavior exhibits strong nonlinearity, for instance, actuator saturation and bounded buffer sizes [12], and the nonlinear relation between the content adaptation level and the CPU utilization [6]. It is usually difficult to handle these nonlinearities, among which is the notoriously challenging problem of response time regulation. In this paper, we study this issue based on our prior work described in [13], where the nonlinear relation between the client-perceived response times and CPU allocation to a Web server is studied quantitatively. When the server is switching between the underload and overload operating regions because of the time-varying workload, this relation shows a strong bimodal property.

Single-loop feedback control is usually not satisfactory in most cases of computing systems control. More complex techniques are introduced, for instance, MIMO control [8] and predictive control with multiple constraints [14]. Recently, combination of reactive and proactive control techniques are applied, where the workload and its effect on the performance metrics can be predicted so that the system can react proactively, and the predictive errors can be corrected through feedback schemes [15][16] [17]. In our earlier work [13], an adaptive controller was developed to deal with the bimodal behavior of the Web server, where one extra measurement, i.e., CPU usage, was incorporated into the response time control loop so that more accurate model parameters and more robust controllers can be obtained. In this paper, we consider the nested (or cascaded) control technique [18] since we have two measurements, the response time and the CPU usage, but only one actuator, the CPU allocation. This scheme helps to improve the robustness of response time control, which is in the primary (outer) loop, by controlling CPU utilization in the secondary (inner) loop.

Utility functions have been used for resource sharing and SLA enforcement in self-managed systems. For instance, Internet congestion control was modeled as a distributed control problem to maximize the aggregate utility [19]. In computing systems, the low-level resource allocation and high-level SLA enforcement can be formulated as a utility optimization problem. In [20] the utility of e-commerce servers' performance was defined in terms of business metrics such as revenue and profit. In [21], the feedback control of the performance of an email server was designed to maximize SLA profits. Discrete resource allocation in a utility data center was formulated as a utility maximization problem in [22]. Utility functions for response time and throughput were defined in [23], based on which hierarchical control structure was developed to optimize the global utility of a data center.

D. *Main Contributions*

The main contributions of this paper compared to prior work are the following. We proposed a new design of two nested control loops for dynamic resource allocation to applications hosted on virtualized servers. The control target is driven by the service utility, which is in turn defined by certain payoff function in an SLA and cost of resources allocated. We implemented the new design on a real system and evaluated its performance through experiments. Compared to a single loop for controlling response time only, the nested design is less sensitive to the bimodal behavior of the system resulting in more robust performance. On the other hand, compared to a single loop for controlling CPU utilization only, the new design provides a framework for dealing with the tradeoff between better QoS and lower cost of resources, therefore resulting in better overall utility of the service.

The remainder of the paper is organized as follows. We describe the testbed and workloads used in Section II, and review the single-loop control of response time or utilization in Section III. In Section IV, we present the new design of a nested control system, performance of which is evaluated in Section V. Finally, we offer conclusions and discuss future work in Section VI.

II. Testbed Setup and workload design

A. *Testbed Setup*

Our testbed is based on the testbed used in our earlier work [11][13]. Its setup is shown in figure 4.

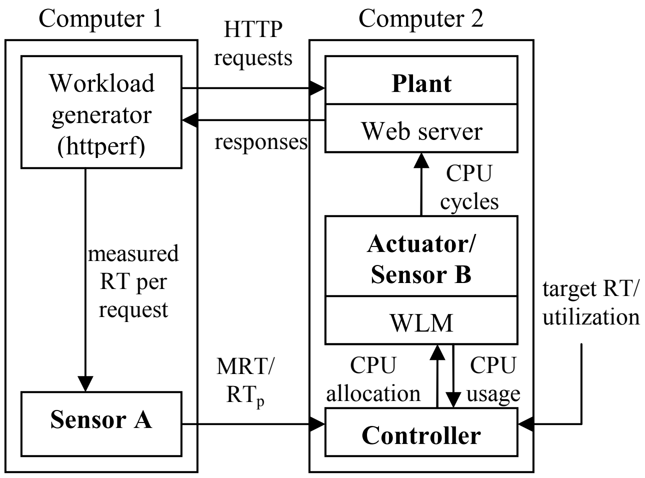


Fig. 4. Setup of the experimental testbed

The testbed consists of two computers connected with 100Mbps Ethernet. One computer (dual-processor HP9000-L server running HP-UX B11.11) contains the plant to be controlled, which can be any application that is CPU intensive, utilizes resources on a shared server, and has a target range for a QoS metric as described in Section I. Here we choose the *Apache Web Server* version 2.0.52 [24] as an example of the application, and use the *HP-UX Workload Manager* (WLM) [1] to configure one partition of the server for hosting the Web server, leaving the remaining resources for other processes on the same system. The WLM implements a weighted Fair Share Scheduler (FSS) that runs inside the HP-UX kernel and assigns a portion of the CPU to a process based on its weight relative to other competing processes. We define the number of CPUs (can be fractional) allocated to the Web server during a time interval as our control variable $u(k)$. The WLM acts as the actuator in our control loop, which allows $u(k)$ to be changed dynamically. It also serves as a sensor (Sensor B in Fig. 4) that measures the Web server's average CPU usage, $v(k)$, for each sampling interval. We define another variable $r(k)$ to be the relative utilization of the CPU allocation, i.e., $r(k) = v(k) / u(k)$. For instance, if the Web server is allocated 0.8 CPU in one interval but only uses 0.6 CPU on average, then $u(k) = 0.8$, $v(k) = 0.6$, and $r(k) = 75\%$.

A scalable client workload generator, *httperf* [25], runs on another computer (HP LPr Netserver running Red Hat Linux 7.3 with kernel version 2.4.18) to continuously send HTTP requests to the Web server. We modified httperf 0.8 to record the response time of every request in a log file. A sensor module (Sensor A in Fig. 4) runs on the same machine that computes some statistical measure of the response times for all the requests completed during a sampling interval. Here we assume that either the mean response time (MRT) or the p^{th} percentile of the response times (RT_p) is the QoS metric defined in the SLA. Let y(k) denote its value for the k^{th} sampling interval.

At the beginning of every sampling interval, the controller pulls the measurements of the QoS metric and the CPU usage for the last interval from both sensors, computes the CPU allocation for the current interval using some control law, and passes it on to WLM for actuation.

B. *Design of Workload*

The workload used in our experiments is chosen to create varying amount of CPU demand for the Web server, while capturing statistical properties of real-world workloads.

Here we use the static portion of the workload model from the *SPECweb99* benchmark [26], an industry standard for testing Web server performance. We populated our Web server with the SPECweb99 data set that contains 418 directories and 36 files in each directory with sizes ranging from 100 Bytes to 900 KB, with a total working data set of 2.3 GB. We use httperf to generate a workload emulating many concurrent users accessing the Web server simultaneously. The number of user sessions per second is generated using a *Poisson* distribution with an average rate λ. Each session has a burst length with a *Normal* distribution N(5,3). Each file accessed is chosen using a combination of a series of distributions (e.g., a *Zipf* distribution for the directory accessed) specified in the SPECweb99 documentation [26]. The session rate allows us to control the average intensity of the workload, while the file distributions and access patterns are used to add stochastic fluctuations to the resource demand the workload places on the Web server.

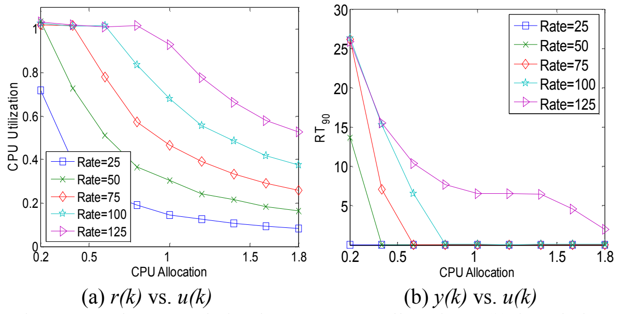
III. Single-Loop Control of Utilization or MRT

The Web server application can be modeled as a discrete-time input-output system that maps the CPU allocation, *u(k),* to the QoS metric, *y(k),* or the relative utilization of the allocation, *r(k).* Because the CPU allocation is not the only factor potentially affecting the QoS metric, other factors such as the workloads are considered as disturbance to the system.

In this section, we review the results of the modeling experiments and the single control loops designed in our earlier work [13] for controlling the MRT or CPU utilization and discuss their features and problems.

A. Results from Modeling Experiments

Fig. 5 shows the static mapping between *u(k), r(k),* and the 90th percentile of response times, RT_{90}, or *y(k)*, at steady state. The experiments were described in detail in [13], where we basically set the sampling interval at 60 seconds (long enough to pass transients), and measured *r(k)* and the mean response time (MRT) for different values of *u(k)* with different workload intensities. We repeated the experiments with the workload described in Section II.B, in varying number of sessions per second, and measured RT_{90} instead of MRT as the output metric.



(a) *r(k)* vs. *u(k)* (b) *y(k)* vs. *u(k)*

Fig. 5. Steady-state relation between CPU allocation *u(k),* its relative utilization *r(k),* and RT_{90} *y(k)*

It is easy to see that the system displays a clear bimodal behavior. For example, the relative utilization *r(k)* can be approximated using the following model:

$$r(k) = \begin{cases} 1, & if\ u(k) < V \\ V/u(k), & if\ u(k) >= V \end{cases} \qquad (1)$$

where V is the average number of CPUs needed for a given workload (unknown). When *u(k)* < *V*, the Web server is *overloaded* therefore it always uses 100% (*r=1*) of its allocated CPU; when *u(k)* > *V*, the Web server is *underloaded*, then the relative utilization is inversely proportional to the CPU allocation. Note that in this relation, the transient queueing dynamics are not considered.

Similarly, the mapping between *u(k)* and *y(k)* also demonstrates a bimodal behavior. Extensive system identification experiments in [13] verified that the dynamic relation between *u(k)* and the inverse of the mean response time can be modeled using a first-order autoregressive model in the overload region. However, when increasing *u(k)* pushes the system into the underload region, *y(k)* immediately drops to very small values.

B. Adaptive Integral Controller for regulating utilization

The CPU utilization of an application is a common metric that is monitored to determine whether more or less CPU resource should be allocated to the application. To regulate the relative utilization *r(k)* of the Web server's CPU allocation at around a target value r_{ref}, the following adaptive-gain integral (I) controller was designed in [13]:

$$u(k) = u(k-1) - K_1(k)(r_{ref} - r(k-1)), \qquad (2)$$

where

$$K_1(k) = \alpha v(k-1)/r_{ref}, \qquad 0 < \alpha < 2, \qquad (3)$$

and *v(k)* means the measurement of CPU usage in the k_{th} interval. Note that the gain parameter K_1 self-tunes based on the resource usage in the last sampling interval. The intuition behind this design is from the bimodal behavior of the system. The controller aggressively allocates more CPU when the system is overloaded, and slowly decreases CPU allocation in the underload region. (See the technical report [27] version of [13] for stability analysis for this controller.) Experimental results in [13] showed that this adaptive controller resulted in lower response time and higher throughput with less average CPU allocation compared to a fixed-gain integral controller when the Web server was subject to a time-varying stochastic workload.

This controller is fairly easy to implement where the only measurement needed is the CPU usage of the controlled application, which is widely available on many systems. Its disadvantage is that no guarantees can be given to SLA-related QoS metrics such as the response time metrics for an arbitrary workload when only *r(k)* is being controlled. The next subsection reviews another adaptive controller we designed in [13] for regulating the MRT directly.

C. Adaptive PI controller for regulating MRT

The existence of a first-order linear dynamic model between the CPU allocation, *u(k),* and 1/MRT, in the overload region makes it relatively easy to regulate the MRT by dynamic control of the CPU allocation when the system is overloaded. The adaptive PI controller we presented in [11] works reasonably well in this region. In [13] we presented an improved version of this PI controller that incorporates the measured CPU usage of the application, v(k), so that the closed-loop control system can more robustly track a target value for the MRT in or close to the overload region. However, when the system is significantly underloaded ($r < 0.8$), the MRT becomes independent of the CPU allocation, therefore MRT is uncontrollable using *u(k)* in this region.

From a practical point of view, no operators would allow their systems to be running in an overloaded mode because that usually implies degraded performance such as longer latency or lower throughput. In the next section, we present the design of a control system that keeps the application reasonably underloaded while meeting QoS goals such as bounds on response times.

IV. A NESTED CONTROL DESIGN

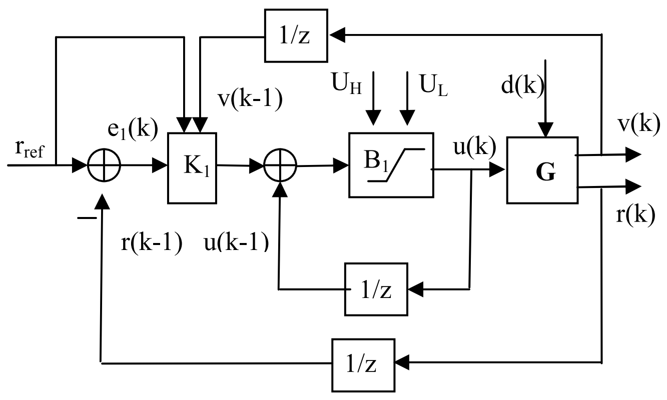
Consider the control loop in Section III.B that maintains the relative utilization *r(k)* at a target value r_{ref}. The relation between a given r_{ref} value and the resulting QoS metric *y(k)* varies with the workload. For example, $r_{ref}=50\%$ may be necessary for highly interactive and bursty workloads, while $r_{ref}=80\%$ may be appropriate for less variable workloads. It is desirable to have the r_{ref} value automatically driven by the application's QoS goals rather than manually chosen by an operator for each application. Given this insight, we design another control loop outside of the utilization control loop to adjust the r_{ref} value dynamically to ensure that the QoS metric, *y(k),* is within the desirable range, [Y_L, Y_H] . Fig. 6 shows the block diagram of the two nested control loops in detail.

Fig. 6(a) illustrates the inner loop as described in Section III.B that contains an adaptive I controller with a self-tuning gain parameter K_1 to regulate the measured relative utilization, r(k), at a target value, r_{ref}, by dynamically adjusting the CPU allocation *u(k)* to the controlled application G(z). The workload to the application acts as a disturbance, *d(k),* to the control system. The block B_1 implements an anti-windup rule to ensure that *u(k)* is always within the specified minimum (U_L) and maximum (U_H) CPU allocation allowed for an application.

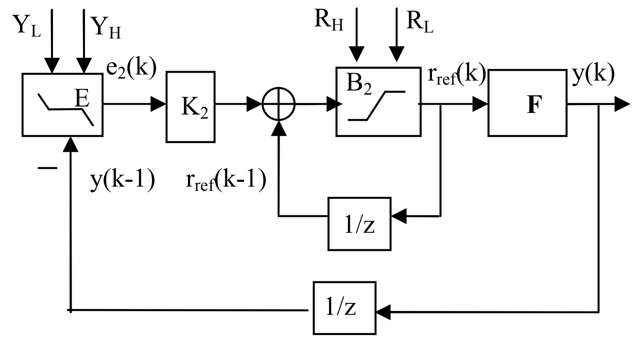
The outer loop as shown in Fig. 6(b) implements another I controller to periodically tune the r_{ref} value so that the measured QoS metric, y(k), can be maintained within [Y_L, Y_H]. As was discussed in Section III, if the inner loop is effective in regulating the relative utilization r(k), then the whole closed-loop subsystem can be viewed as a black-box. Therefore, we can model it using a discrete-time operator F(z) that maps the reference signal $r_{ref}(k)$ to the measured the QoS metric y(k). Note that although the same time index *k* is used in the two loops, the functions G(z) and F(z) may not be based on the same sampling frequency. Typically the inner loop uses a shorter sampling interval than what the outer loop uses because the former usually has faster dynamics. The block E implements the following error function:

$$e_y(k) = \begin{cases} Y_L - y(k), & y(k) < Y_L \\ 0, & Y_L \le y(k) \le Y_H \\ Y_H - y(k), & y(k) > Y_H \end{cases} \qquad (4)$$

This means when the measured *y(k)* is within [Y_L, Y_H], $r_{ref}(k)$ remains the same; otherwise, $r_{ref}(k)$ is reduced when *y(k)* is too large and $r_{ref}(k)$ is increased when *y(k)* is too small. The gain parameter K_2 of the integral control law determines how aggressively $r_{ref}(k)$ is adjusted. Again the block B_2 ensures that the computed $r_{ref}(k)$ value is always within a specified range [R_L, R_H], e.g., [50%, 90%]. The reason for setting the lower bound R_L is to ensure that, as long as the QoS goal for y(k) is met, the CPU cycles allocated to the application should be utilized at least 50%, if not more. On the other hand, the allocated CPU should never be utilized more than 90% on average so that there is always some headroom to prevent the application from becoming overloaded. We will see the impact of these parameters in the experimental results in the next section.



(a) The inner loop controls the CPU allocation u(k) to regulate the relative utilization r(k)



(b) The outer loop controls r_{ref} to regulate the QoS metric y(k)

Fig. 6. Block diagrams of the two nested control loops

V. Performance Evaluation

To evaluate the performance of the nested control loops, we implemented the above design on our testbed described in Section II.A. For comparison, we also implemented the single utilization control loop shown in Fig. 6(a) with r_{ref} being a range $[R_L, R_H]$ instead of a single value, as well as a single integral control loop that regulates the QoS metric y(k) to be within $[Y_L, Y_H]$. The latter is similar to the loop shown in Fig. 6(b) except that the control variable is the CPU allocation u(k) (bounded by $[U_L, U_H]$) instead of $r_{ref}(k)$.

We tested the three control loops using a SPECweb99 workload shown in Fig. 7(a), with its key parameters and statistical properties described in Section II.B. The workload lasts 30 minutes in time. The mean session rate was set to 50 sessions/sec in the first 10 minutes, increased to 90 sessions/sec in the second 10 minutes, and decreased to 50 sessions/sec again in the last 10 minutes. We attempt to use this workload to mimic the real-world demand pattern shown in Fig. 1 that stays at a low level the majority of the time but has a square-wave surge in demand for certain period of time. In these experiments, we use RT_{90}, the 90th percentile of all the response times during a sampling interval, as our QoS metric *y(k),* and assume that its utility-driven target range is [0.2, 1] seconds. We also set the bounds on the relative utilization *r(k)* to be [50%, 90%], and the bounds on the CPU allocation to be [0.1, 1.6] CPU (i.e., 5% to 80% of the CPU cycles). The parameter α in the utilization controller (2)(3) is set to 1.5 in both the inner loop of the nested control design and the single utilization control loop. The gain K_2 in the nested loops is 0.1, the same as that used in the single control loop on response time. We set the sampling interval of the utilization controller (in both the inner loop and the single loop) to be 5 seconds, and the sampling interval of the response time controller (in both the outer loop and the single loop) to be 20 seconds.

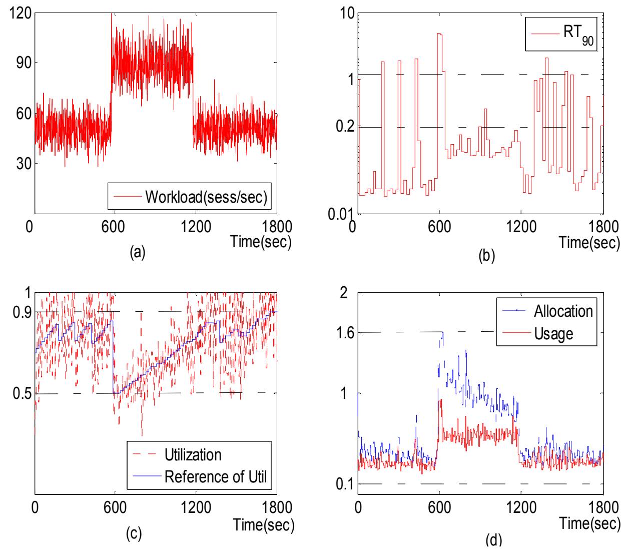


Fig. 7. Performance of the nested control loops

The performance of the nested control loops is shown in Fig. 7(b), (c), and (d). Fig. 7(b) shows in logarithmic scale the resulting QoS metric RT_{90} sampled at 20 seconds intervals. The control of the target utilization r_{ref} in the outer loop as shown in Fig. 7(c) presents strong fast-decrease-slow-increase bimodal behavior. Similarly, the control of CPU allocation as shown in Fig. 7(d) shows a clear fast-increase-slow-decrease pattern. (All the horizontal dash-dot lines in each figure indicate the respective bounds on that variable.) As a result, the Web server is underloaded most of the time, where RT_{90} is below the 0.2 second lower bound (Y_L). In this case, the outer loop controller is activated and starts pushing up r_{ref} (shown in Fig. 7(c)), which in turn causes the inner loop to reduce the CPU allocation (shown in Fig. 7(d)). However, the increase of r_{ref} in each control interval is bounded by the product of the gain K_2 and the bound Y_L, which is 0.02 in this case. This causes the value of r_{ref} to increase gradually so that the system does not fall into the overload region immediately. Eventually, whenever r_{ref} gets closer to 90%, the relative utilization *r(k)* may become 100% sometimes causing the application to be overloaded, then the response time measure RT_{90} increases rapidly due to heavy queueing in the system, Once RT_{90} exceeds the 1 second upper bound (Y_H), the outer loop controller starts to reduce r_{ref} sharply so that the inner loop can allocate significantly more CPU to the application to bring it out of the overload region.

The result from the single-loop response time control is shown in Fig. 8(a) and (c). This controller is clearly even more aggressive in quickly allocating more CPU under overload conditions, which immediately pushes the response time measure RT_{90} below the 0.2 seconds lower bound. It then takes a long time to recover because it reduces allocation very slowly in the underload region. As a result, the Web server spends even more time producing lower-than-needed response times due to significant over-allocation by the control system. By comparing these results with those in Fig 7(b) and (d), we observe that by placing a 50% lower bound on the relative utilization, the nested control design is able to reduce overshooting of the controller therefore increasing resource utilization.

The performance of the single-loop utilization control with a fixed target utilization range is shown in Fig 8(b) and (d). Fig. 8(b) shows the resulting RT_{90} values that are below the lower bound of 0.2 seconds most of the time. A spike reaching 4 seconds occurred at around 600 seconds due to the surge in the workload intensity, but then the controller quickly increased CPU allocation to accommodate the increased demand. This controller shows the same fast-increase-slow-decrease behavior that results in lower response times in general, which is desirable without considering the cost of resource allocation. In comparison, the added outer loop in the nested control design compares the measured RT_{90} with its target and adjusts the target utilization level accordingly, thereby providing a tradeoff between better QoS and lower infrastructure cost.

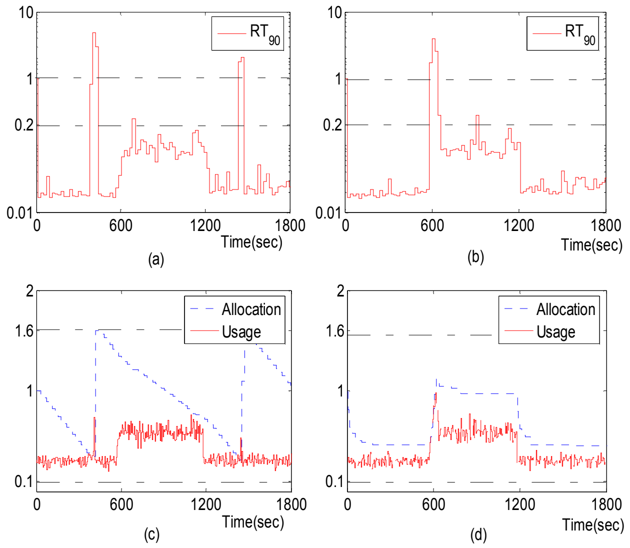


Fig 8: Performance of single-loop response time control ((a) and (c)) and single-loop utilization control ((b) and (d))

This tradeoff is more visible in the overall statistics of these experimental results shown in Table 1. To compare the three designs, we computed the following five metrics: the percentages of time when RT_{90} is below Y_L (P_{low}), within $[Y_L, Y_H]$ (P_{good}) and above Y_H (P_{high}), as well as the average CPU allocation and the average utilization of the allocated CPU cycles. As we can see, the single-loop response time control allocates 24% more CPU cycles to the Web server than the nested control does (0.84 vs. 0.60 CPU), resulting in the system spending 90% of the time in the low response time, high cost region, and only 3% of the time in the good response time, lower cost region. On the other hand, the single-loop utilization control spends even more time (94%) in the low response time, high cost region, but requires only 0.65 CPU on average. In comparison, the nested control implementation guides the system to spend more time (11%) in the good response time, lower cost region, requires the minimum CPU allocation (0.6 CPU), and results in the maximum CPU utilization (68%) overall.

Table 1: Overall statistics of results from the three control systems

Control system design	P_{low}	P_{good}	P_{high}	mean allocation	mean utilization
Nested control	0.78	0.11	0.11	0.60	68%
RT control	0.90	0.03	0.07	0.84	48%
Util.control	0.94	0.03	0.03	0.65	62%

To see how sensitive the above results are to the value of the gain K_2 in the nested control loops and in the single-loop response time control, we ran more experiments with different values for K_2. Due to the different dynamics in the two operating regions, we set different values (K_{2L}/K_{2H}) for the gain K_2 for $y<Y_L$ and $y>Y_H$, respectively. Fig. 9 shows the performance of the single-loop response time control, when K_{2L} ((a) and (c)) or K_{2H} ((b) and (d)) is increased from 0.1 to 0.5. We can see that the system is more sensitive to the value of K_{2L} than to that of K_{2H}. Fig. 9(a) shows that when K_{2L}=0.5, the system oscillates heavily between over-allocation and under-allocation, causing the measured RT_{90} to swing between large values and very small values. We observe that it is difficult to manage the bimodal behavior by tuning the control gain when a single loop is used for direct control of response times. The results of the nested control with the same setup of the gains are shown in Fig. 10, which shows less visible bi-modal behavior, and the resource allocation is less sensitive to the gain values. In summary, the nested control design meets QoS goals more effectively and utilizes resources more efficiently.

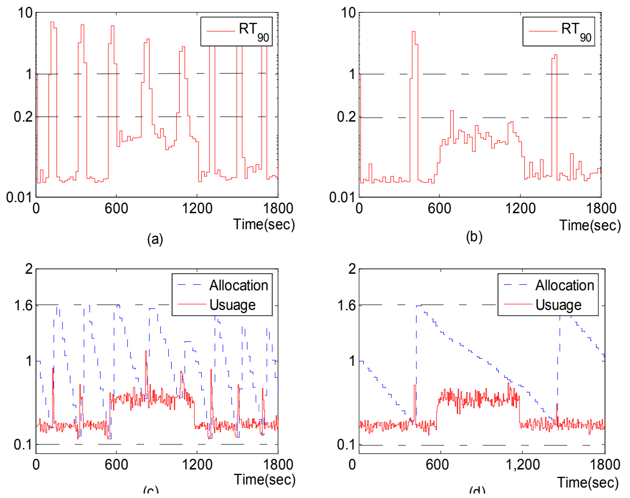


Fig. 9: Performance of single-loop response time control with different gains K_2=(0.5, 0.1) ((a) and (c)) and gains K_2=(0.1, 0.5) ((b) and (d))

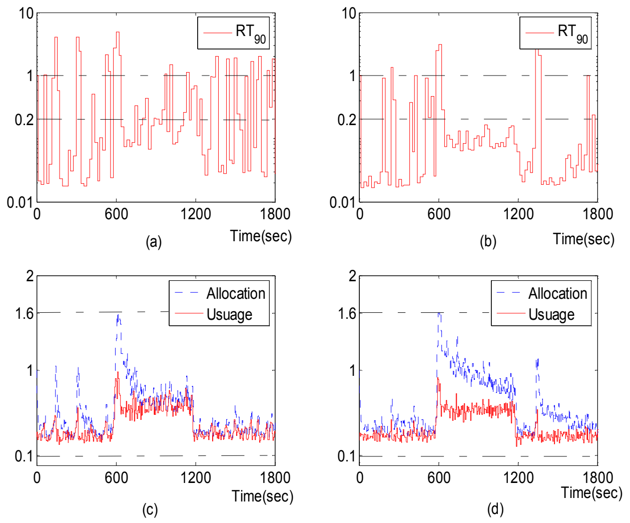


Fig. 10: Performance of nested control with different gains K_2=(0.5, 0.1) ((a) and (c)) and gains K_2=(0.1, 0.5) ((b) and (d))

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented a nested feedback control design for effectively managing the QoS metrics of applications hosted on virtualized servers. Experimental results on a lab testbed showed benefits of using this design over our earlier designs of single control loops for either the QoS metric or the CPU utilization. This design can potentially be used to create more efficient workload management solutions for dynamic resource provisioning on virtualized servers and to provide tradeoffs between better QoS and lower cost of ownership.

As ongoing work, we are applying the same closed-loop design to resource control of virtual machines where we expect to achieve similar results. In addition, stability analysis of the outer loop is lacking due to absence of an analytical model for the inner loop and the gain parameter K_2 still requires hand-tuning to strike a balance between stability and efficiency. Although for a given workload and given utility function, an offline optimization may be done to find the best value for K_2, it may not be the optimum for other workloads. We attempt to find a self-adapting rule for K_2 that is effective under a wide range of system conditions.

This work only deals with one resource bottleneck while any computer system typically has many potential constraints that can affect an application's QoS measures. In our future research, we plan to develop algorithms to determine which control knob should be turned to correct a problem when multiple knobs are available.

REFERENCES

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

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

[3] VMware, www.vmware.com.

[4] Xen Virtual Machine, http://www.xensource.com/xen/about.html.

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

[6] 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.

[7] J. L. Hellerstein, "Challenges in Control Engineering of Computing Systems", Proceeding of the 2004 American Control Conference, Boston, MA, June 30-June 2, 2004.

[8] 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.

[9] 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.

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

[11] 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.

[12] A. Robertsson, B. Wittenmark, M. Kihl, M. Andersson, "Design and evaluation of load control in web server systems", in Proceeding of the 2004 American Control Conference, Boston, MA, June 30 – july 2, 2004.

[13] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-based control for dynamic sizing of resource partitions," 16th IFIP/IEEE Distributed Systems: Operations and Management, October, 2005, Barcelona, Spain, in press.

[14] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," IEEE Transactions on Parallel and Distributed Systems, 16(6):550-561, June 2005.

[15] L. Sha, X. Liu, Y. Lu and T. F. Abdelzaher, "Queueing Model Based Network Server Performance Control", the 23rd IEEE International Real-Time Systems Symposium, Austin, Texas, December 2002.

[16] A. Chandra, W. Gong and P. Shenoy, "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements", the Eleventh IEEE/ACM International Workshop on Quality of Service (IWQoS 2003), Monterey, CA, June 2003.

[17] D. Henriksson, Y. Lu, T. Abdelzaher, Improved Prediction for Web Server Delay Control, 16th Euromicro Conference on Real-Time Systems, Catania, Italy, July 2004.

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

[19] F. P. Kelly, "Models for a self-managed Internet", in Philosophical Transactions of the Royal Society, A358, 2000, 2335-2348.

[20] D. Menasce, V. F. Almeida, R. Fonsece, M. Mendes, "Business-oriented resource management policies for e-commerce servers", Performance Evaluation, Sept., 2000.

[21] Y. Diao, J.L. Hellerstein, S. Parekh, "A business-oriented approach to the design of feedback loops for performance management", Distributed Operations and Management, 2001.

[22] T. Kelly, "Utility-directed allocation", in Proceedings of the first workshop on algorithms and architectures for self-managing systems, June 11, 2003, San Diego, CA.

[23] M. N. Bennani, D. A. Menasce, "Resource Allocation for Autonomic Data Centers Using Analytic Performance Models," in *Proc. 2005 IEEE International Conference on Autonomic Computing,* Seattle, WA, June 13-16, 2005.

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

[25] D. Mosberger and T. Jin, "Httperf --- A tool for measuring Web server performance," *Performance Evaluation Journal*, Vol. 26, No. 3, pp. 31-37, December 1998.

[26] Standard Performance Evaluation Corporation, SPECweb99 benchmark, http://www.spec.org/web99/docs/whitepaper.html.

[27] Z. Wang, X. Zhu, S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions", HP Labs Technical Reportd, HPL-2005-126, July 2005.