# Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform

Xue Liu*, Xiaoyun Zhu* Pradeep Padala†, Zhikui Wang*, Sharad Singhal*
*Hewlett Packard Laboratories, Palo Alto, CA 94304, USA
{xue.liu, xiaoyun.zhu, zhikui.wang, sharad.singhal}@hp.com
†EECS, University of Michigan, Ann Arbor, MI 48109, USA
ppadala@eecs.umich.edu

*Abstract*— **Today's shared hosting platforms often employ virtualization to allow multiple enterprise applications with time-varying resource demands to share a common infrastructure in order to improve resource utilization. Meeting application-level quality of service (QoS) goals becomes a challenge in such an environment as enterprise applications often have a multi-tier architecture and complex interactions and dependencies among individual tiers. In addition, when the shared infrastructure becomes overloaded, appropriate resource control needs to be performed at these individual tiers in a coordinated fashion in order to provide differentiated services to co-hosted applications. In this paper, we present an adaptive multivariate controller that dynamically adjusts the resource shares to individual tiers of multiple applications in order to meet a specified level of service differentiation. The controller parameters are automatically tuned at runtime based on a quadratic cost function and a system model that is learned online using a recursive least-squares (RLS) method. To evaluate our controller design, we built a testbed hosting two instances of the RUBiS application, a multi-tier online auction web site, using Xen virtual machines. Our results indicate that our controller is able to meet given QoS differentiation targets between co-hosted applications while the total demand from these applications exceeds the capacities of the shared systems.**

## I. Introduction

Data centers today play a major role in providing on-demand computing power to various enterprise applications supporting different business processes such as supply chain, e-commerce, human resource, payroll, customer relationship management, etc. These applications often have different needs for computing resources, and time varying resource demands driven by changes in workload intensity and mix. Data center operators face the challenge of provisioning sufficient resources to these applications such that they can meet their service level objectives (SLOs) while maintaining high resource utilization. As a result, shared hosting platforms are becoming more widely developed and deployed as an alternative to the traditional silo-oriented architecture where each application has its own dedicated servers. In a shared hosting environment, all hardware resources are pooled into a common shared infrastructure and applications share these resources as their demands change over time [1]. The rapid development of virtualization technologies in the past few years has pushed this trend further by allowing multiple virtual containers to be created on a single physical server, each running its own operating system and applications.

When multiple enterprise applications share a common infrastructure, meeting the application-level QoS goals becomes a nontrivial task due to the usual multi-tier architecture of these applications and complex interactions among individual tiers. For example, three-tier web applications consist of a web server tier, an application server tier, and a database server tier. On a shared hosting platform, these individual tiers will be hosted inside different virtual containers spread across multiple servers, which poses the following challenges. First, the resource demands placed on these separate tiers vary from one tier to another; e.g., the web tier may consume mainly CPU and network bandwidth, whereas the database tier consumes more I/O bandwidth the web tier does. Second, the resource demands across tiers are dependent and correlated to each other; for example, a database tier only serves connections established through the web tier. Finally, resource demands vary from one application to another; e.g., for the same number of user sessions served in the web tier, we may be seeing vastly different resource demand profiles at the database tier for different applications. As a result, dynamically adjusting resources to an application component has to take into account not only the local resource demands in the node where that component is hosted, but also the resource demands in other nodes hosting all the other components of the same application.

In this paper, we address the problem of dynamically allocating resources to individual application components of multiple, multi-tier enterprise applications in a shared hosting environment. In particular, we focus on the goal of service differentiation between co-hosted applications when multiple shared servers are being overloaded. We develop an optimal multivariate controller for allocating shared resources to individual tiers of multiple application stacks in a coordinated fashion that accounts for the dependencies and interactions among these tiers. The controller is designed to adapt to varying workloads such that a specified level of QoS differentiation can be maintained. The controller parameters are automatically tuned at runtime based on a quadratic cost function and a system model that is learned online using a recursive least-squares (RLS) method.

This work is built on top of our earlier work in [2] where an adaptive resource controller is designed to dynamically adjust resource shares for individual application components sharing virtualized servers. The goal of that controller is to maintain high resource utilization in the virtualized servers while meeting application-level QoS objectives if possible. If any of the nodes hosting multiple application components becomes saturated, a SISO controller is used for arbitration among these application components such that a desired level of service differentiation can be achieved. Compared to the techniques presented in [2], the main contributions of this paper are two-fold. First, the SISO controller in the earlier work does not deal with the situation where more than one hosting nodes become saturated, yet our MIMO controller here handles this situation by adjusting resource allocations to individual tiers of multiple applications in a coordinated manner in order to meet the differentiation goals for end-to-end QoS metrics. Such goals will be hard to achieve if multiple SISO controllers are run independently of one another. Second, by using online estimation of the input-output model for the controlled system, we eliminate the need for extensive offline system characterization as was done in [2]. Our controller parameters are also adapted in real-time based on the estimated model parameters as opposed to being manually tuned offline.

The adaptive optimal controller design we use here is an extension of a similar design in [3], where it was used for admission control of $M$ different classes of worklaods to a shared computing service. In this paper, the earlier controller has been generalized to deal with dynamic resource allocation to components in $M$ of $N$-tier applications. The system model identified using the RLS algorithm not only has to recognize interactions among the $M$ components sharing a single node, it also has to account for dependencies among the $N$ components for a single application. These interactions and dependencies are used in the optimal controller to coordinate resource allocation to all the individual components. In addition, instead of penalizing on large control actions, we added a term to the quadratic cost function that penalizes large variations in the control actions. This helps improve stability in the closed-loop system and reduces oscillations in the output metric.

To test our controller design, we have built a testbed for a small shared hosting platform using Xen virtual machines [4], referred to as *virtual containers*. We encapsulate each tier of an application in a virtual container and attempt to control the resource allocation at the container level. We experimented with a multi-tier application in our testbed: a two-tier implementation of RUBiS [5], an online auction web site. We than created workload conditions that put the system in a state where multiple nodes hosting multiple virtual containers are saturated, and tested the performance of our controller under a changing control target as well as time-varying workloads. Experimental evaluation of the self-tuning optimal controller validates the following two key results. First, the closed-loop control system can track the specified level of service differentiation between the co-hosted applications when both tiers of the applications become a bottleneck. Second, our control system maintains the desired level of service differentiation in spite of workload variations in the applications.

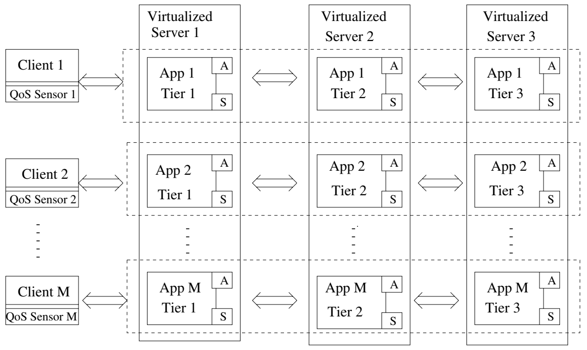

Fig. 1. A shared hosting platform for $M$ multi-tier applications

The remainder of this paper is organized as follows. Section II presents the architecture of a shared hosting platform and explicitly defines the service differentiation problem. An online system modeling approach as well as a self-tuning optimal controller design are described in Section III. Section IV presents the results from our experimental evaluation. Section V discusses the related work. Finally, conclusions and future work are discussed in Section VI.

## II. Problem Statement

In this paper, we consider a shared hosting platform as depicted in Figure 1. On such a platform, multiple multi-tier applications share a common pool of server resources, and each tier of each application is hosted inside a virtual container on a shared physical server. A virtual container can be a virtual machine (VM) provided by hypervisor technologies including Xen [4] and VMWare or OS-level virtualization like OpenVZ [6] and Linux VServer [7]. This shared infrastructure paradigm has gained interest in many enterprises due to the reduction of infrastructure and operational costs in data centers. Although the grouping of application tiers on each physical server can be arbitrary in principle, we specifically chose the design where the same tiers from different applications are hosted on the same physical server, as indicated in Figure 1. This is a natural choice for many shared hosting platforms for potential savings in software licensing costs.

Resource demands of enterprise applications vary depending on the number of concurrent users and workload mix. Shared servers will become saturated, when the aggregate demand from all the application components sharing the server exceeds the total capacity of that server. This resource contention causes degraded applications performance without explicit control of resources. However, on a shared hosting platform, it is often the case that different applications be serviced with different levels of priority, depending on their respective service level agreements (SLA). Therefore, it is desirable to provide performance isolation and differentiation among co-hosted applications in order to maximize the overall business value generated by these applications. This paper addresses this issue by designing a closed-loop resource controller that dynamically adjusts resource allocations for individual application components

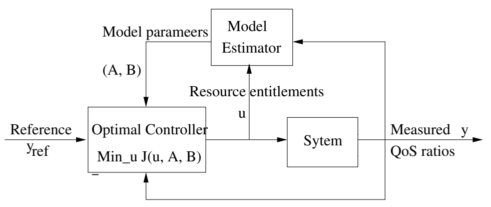

Fig. 2. A self-tuning optimal resource control system

in a shared hosting environment such that a desired level of QoS differentiation can be achieved.

Let the number of co-hosted applications be $M$, and each application has $N$ tiers. For example, $N = 3$ in Figure 1. We focus on a specific setting where the " $N$ is also the number of virtualized servers for hosting the applications. In future work, we will also look at scenarios where the number of servers is larger than the number of tiers, in which case distributed algorithms may be needed. In addition, we focus on CPU capacity as the shared resource in this paper, because explicit control over CPU allocation is most widely available across different virtualization technologies.

Let $C_j$ be the total CPU capacity of server $j$, which is usually normalized to $100\%$. We refer to the CPU capacity allocated to a virtual container as "resource entitlement", specified as a percentage of the server capacity. Let $u_{i,j}$ denote the resource entitlement for tier $j$ of application $i$. Since $\sum_{i=1}^{M} u_{i,j} = C_j$, $1 \leq j \leq N$, there are a total of $(M-1) \times N$ such independent variables. Hence, the inputs to the controlled system are $u = [u_{1,1}, u_{1,2}, \cdots, u_{1,N}, \cdots, u_{M-1,1}, u_{M-1,2}, \cdots, u_{M-1,N}]^T$.

Let $Q_i$ denote application $i$'s end-to-end QoS metric. Examples of this metric include mean response time or throughput over a time interval. We then define by $y_i = \frac{Q_i}{\sum_{m=1}^{M} Q_m}$, $1 \leq i \leq M$, which is the normalized QoS ratio for application $i$. Since $\sum_{i=1}^{M} y_i = 1$, only $M-1$ of all the $y_i$'s are independent. As a result, the outputs of the controlled system are $y = [y_1, \cdots, y_{M-1}]^T$. The goal of our resource controller is to find appropriate values for $u_{i,j}$, for all $i$'s and $j$'s, such that $y_i = y_{ref,i}$, where $y_{ref,i}$ is the desired QoS ratio for application $i$, $1 \leq i \leq M-1$.

## III. A Self-Tuning Optimal Controller

In this section, we describe the design of a self-tuning optimal resource controller for ensuring service differentiation during system overload. A block diagram of the closed-loop control system is shown in Figure 2. The controller consists of two key modules: a *model estimator* that learns and periodically updates a linear model between the resource entitlements for individual application components and the QoS ratios, and an *optimal controller* that computes the optimal resource entitlements based on estimated model parameters and a quadratic cost function.

### A. Online Model Estimation

Let us use $I = (M-1) \times N$ to denote the input dimension, and use $O = (M-1)$ to denote the output dimension, where $M$ is the number of applications, and $N$ is the number of tiers in the applications. We use the following linear, auto-regressive MIMO model to represent the input-output relationship in the controlled system.

$$A(q^{-1})y(k) = B(q^{-1})u(k) + e(k), \tag{1}$$

where $A(q^{-1})$ and $B(q^{-1})$ are matrix polynomials in the backward-shift operator:

$$\begin{aligned} A(q^{-1}) &= I - A_1 q^{-1} - \ldots - A_n q^{-n}, \\ B(q^{-1}) &= B_0 q^{-1} + \ldots + B_{n-1} q^{-n}. \end{aligned} \tag{2}$$

Note that $A_l \in \Re^{O \times O}$, $B_m \in \Re^{O \times I}$, $0 < l, m \leq n$, where $n$ is the order of the system. $\{e(k)\}$ is a sequence of independent, identically distributed $O$-dimensional random vectors with zero means. It is further assumed that $e(k)$ is independent of $y(k-j)$ and $u(k-j)$ for $j > 0$. We use $e(k)$ to represent disturbances in the system that are not accounted for by the model. The linear model is a local approximation of the real system dynamics that is typically nonlinear.

The use of a MIMO model allows us to capture interactions and dependencies among resource entitlements for different application components. For example, reducing resource entitlement for one application tier will increase resource entitlements for other application components on the same node, and may reduce the load coming into the next tier of the same application. Such dependencies cannot be captured by individual SISO models. In addition, a MIMO model enables the controller to make tradeoffs between different applications and their components when the system does not have enough capacity to meet all individual needs.

We use offline system identification to estimate the order of the system ($n$), which is usually low in computer systems [8]. The coefficient matrices $A_l$ and $B_m$, where $0 < l, m \leq n$, are estimated online for the following reason. The values of these parameters may or may not change as system operating conditions and workload dynamics change, depending on the specific situation. (See [9] for an example of this behavior in a much simpler system.) If offline experiments are to be used to identify a single model for controller design, there is no simple way for us to test in advance all combinations of different operating conditions to be sure that our model is sufficiently representative of all cases. Therefore, a self-learning approach is preferred where model parameters are estimated online and updated whenever new data has become available.

For convenience of notation, we rewrite the system model in the following form, which we use in the rest of the paper:

$$y(k+1) = X\phi(k) + e(k+1), \tag{3}$$

where

$$\begin{aligned} &X = [B_0 \ \ldots \ B_{n-1} \ A_1 \ \ldots \ A_n], \\ &\phi(k) = [u^T(k) \ \ldots \ u^T(k-n+1) \ \ y^T(k) \ \ldots \\ &\qquad y^T(k-n+1)]^T. \end{aligned}$$

We use a recursive least squares (RLS) estimator with *directional forgetting* [10] to estimate the parameter matrix

$X$, as defined by the following equations:

$$\begin{aligned}\hat{X}(k+1) &= \hat{X}(k) + \frac{\epsilon(k+1)\phi^T(k)P(k-1)}{\lambda + \phi^T(k)P(k-1)\phi(k)}\ , \\ \epsilon(k+1) &= y(k+1) - \hat{X}(k)\phi(k)\ , \qquad (4) \\ P^{-1}(k) &= P^{-1}(k-1) + (1 + (\lambda - 1)\cdot \\ &\quad \frac{\phi^T(k)P(k-1)\phi(k)}{(\phi^T(k)\phi(k))^2})\phi(k)\phi^T(k)\ ,\end{aligned}$$

where $\hat{X}(k)$ is the estimate of the true value of $X$, $\epsilon(k)$ is the estimation error vector, $P(k)$ is the covariance matrix, and $\lambda$ is the *forgetting factor* $(0 < \lambda \leq 1)$.

## B. A Linear Quadratic Optimal Controller

For the controller design, we aim at minimizing the following quadratic cost function:

$$J = E\{\|W(y(k+1)-y_{ref}(k+1))\|^2+\|Q(u(k)-u(k-1))\|^2\}, \qquad (5)$$

where $W \in \Re^{O\times O}$ is a positive-semidefinite weighting matrix on the tracking errors and $Q \in \Re^{I\times I}$ is a positive-definite weighting matrix on the control actions.

The goal of the controller is to steer the system into a state of optimum reference tracking, while penalizing large changes in the control variables. The $W$ and $Q$ weighting matrices are commonly chosen as diagonal matrices. Their relative magnitude provides a way to tradeoff tracking accuracy for better stability in the control actions.

In the following, we derive the optimal controller by explicitly capturing the dependency of the cost function $J$ on $u(k)$. We first define

$$\begin{aligned}\tilde{\phi}(k) &= [0\ u^T(k-1)\ \ldots\ u^T(k-n+1)\ y^T(k)\cdots \\ &\quad \cdots\ y^T(k-n+1)]^T\ . \qquad (6)\end{aligned}$$

Then we have,

$$\begin{aligned}J &= E\{\|W(y(k+1) - y_{ref}(k+1))\|^2 \\ &\quad +\|Q(u(k) - u(k-1))\|^2\} \\ &= E\{\|W(\hat{X}(k)\tilde{\phi}(k) + \hat{B}_0 u(k) + \epsilon(k+1) \\ &\quad -y_{ref}(k+1))\|^2\} + \|Q(u(k) - u(k-1))\|^2 \\ &= \|W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1))\|^2 + \|W\hat{B}_0 u(k)\|^2 \\ &\quad +2u^T(k)\hat{B}_0{}^T W^T W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1)) + \\ &\quad +\|Q(u(k)\|^2 + \|Q(u(k-1)\|^2 \\ &\quad -2u(k-1)^T Q^T Q u(k) + E\{\|W\epsilon(k+1)\|^2\}\ .\end{aligned}$$

The cost function $J$ is at its minimum where the following derivative is zero.

$$\begin{aligned}\frac{\partial J}{\partial u(k)} = &\ 2(W\hat{B}_0)^T W(\hat{X}(k)\tilde{\phi}(k) - y_{ref}(k+1)) \\ &+2(W\hat{B}_0)^T W\hat{B}_0 u(k) + 2Q^T Q u(k) \\ &-2Q^T Q u(k-1) = 0. \qquad (7)\end{aligned}$$

Solving for $u(k)$ gives us the following optimal control law:

$$\begin{aligned}u^*(k) = &\ ((W\hat{B}_0)^T W\hat{B}_0 + Q^T Q)^{-1}[(W\hat{B}_0)^T W \\ &(y_{ref}(k+1) - \hat{X}(k)\tilde{\phi}(k)) + Q^T Q u(k-1)]. \qquad (8)\end{aligned}$$

Note that $\hat{X}(k)$ and $\hat{B}_0$ are estimates of the model parameters obtained using the RLS estimator (4). The derivation of the control law here is adapted from the controller synthesis in [3] and [11]

## IV. Experimental Evaluation

In this section, we present the experimental evaluation results of our controller design on a two-tier testbed, as an example of the shared hosting platform presented in Section II.

### A. A Two-Tier Testbed

The testbed consists of five HP Proliant servers, each with two processors, 4 GB of RAM, one Gigabit Ethernet interface, and two local SCSI disks. Two servers are used to host two instances of the RUBiS application [5], an online auction prototype. We use a two-tier implementation consisting of an Apache web server and a MySQL database (DB) server. Each tier of an application is hosted inside a Xen virtual machine (or *domain* in Xen's terminology). One server node is used to host two "web domains" (referred to as the web node), and the other node is used to host two "DB domains" (referred to as the DB node). The hardware resources on each node are shared between the two domains that host the application components and the host domain (`dom0`). In our experiments, we restrict the two application component domains to share a designated CPU and direct `dom0` to use the other CPU to prevent interference.

Two other nodes are used to generate client requests to the two applications. The RUBiS clients are configured to submit workloads of different mixes as well as workloads of time-varying intensity. We use a workload mix called the *browsing mix* that consists primarily of static HTML page requests that are served by the web server (see [5] for more details). Each RUBiS client also provides a sensor that measures the client-perceived QoS metrics such as average response time and throughput over a period of time.

We use the credit-based CPU scheduler in the hypervisor of Xen 3.0.3 unstable branch [12] to realize resource entitlements to individual domains. It implements weighted fair sharing of the CPU capacity among multiple domains, by dividing CPU time into fixed-length intervals and allocating each domain a certain share of the time in each interval. Since these shares can be changed at run time, the scheduler serves as an actuator in our control loop. We use the capped mode of the credit scheduler, where a domain cannot use more than its share of the CPU time, even if there are CPU cycles available. This mode allows us to explicitly control how much resource a domain has access to, thus providing better performance isolation between multiple application components sharing the same node.

The Xen hypervisor also provides a sensor to measure how many of the entitled CPU cycles are actually consumed by each domain in a given period of time. This data will be included in all of our experimental results to reveal resource utilization levels of individual domians hosting different application components.

Finally, the last node runs our feedback-driven resource controller that communicates with the sensors and actuators

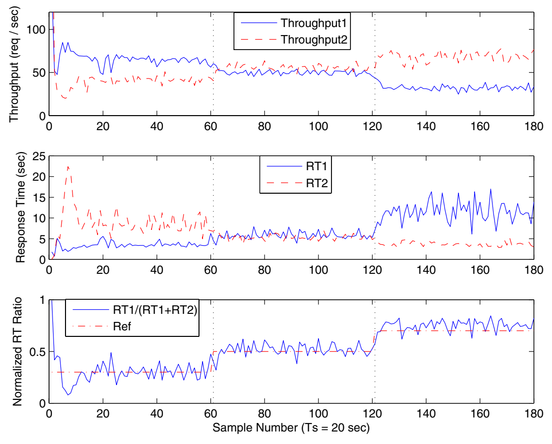

Fig. 3. Application QoS metrics as the reference for QoS ratio changes

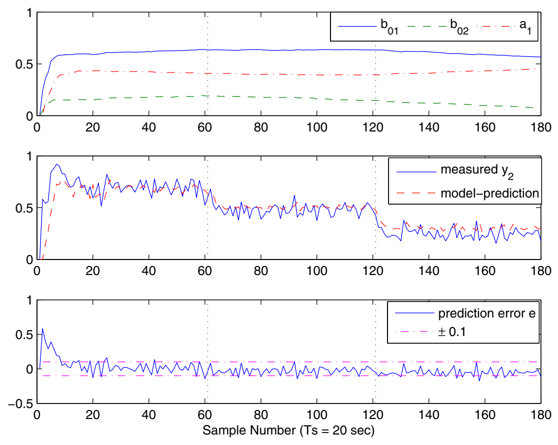

Fig. 4. Model parameter estimation as the reference for QoS ratio changes

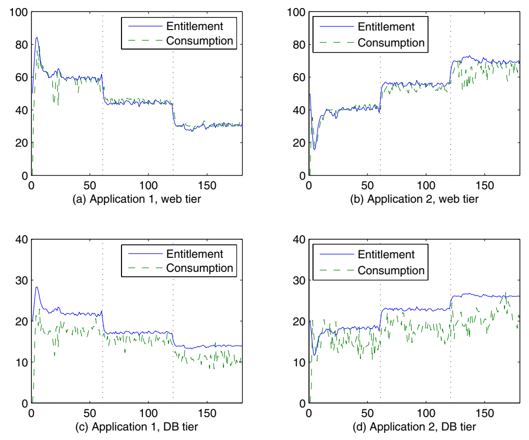

Fig. 5. CPU entitlement and consumption for individual application components as the reference for QoS ratio changes

and makes appropriate resource allocation decisions on a periodic basis.

## *B. Parameter Settings*

With this testbed, $M = 2$, $N = 2$, and the controlled system is a two-input-one-output system. The input variables are $u = [u_{11}\ u_{12}]^T$, where $u_{1j}$ denotes the resource entitlement for tier $j$ of application 1, $j = 1, 2$. In particular, tier 1 is the web tier and tier 2 is the DB tier. The resource entitlement for tier $j$ of application 2 can be determined by $u_{2j} = C_j - u_{1j}$, where $C_j$ is the total CPU capacity of server node $j$. By experimenting with the RUBiS workload, we notice that having 500 concurrent users for each application would creat a total CPU demand of more than 100% on the web node and more than 40% on the DB node. Therefore, by making $C_1 = 100\%$ and $C_2 = 40\%$, we create a scenario where both the web node and the DB node are saturated, in which case our resource controller is needed to provide service differentiation between the two RUBiS applications.

We choose a control interval (or sampling interval) of $T_s = 20$ seconds, which offers a good balance between responsiveness of the controller and predictability of the measurements. For the RUBiS applications, we choose mean response time per interval as the QoS metric. Let $RT_i$ be the measured mean response time for application $i$, and $y_i$ be the normalized RT ratio for application $i$, i.e., $y_i = RT_i/(RT_1+RT_2)$. We let $y = y_1$ be the output of our control system, and $y_{ref}$ be the desired level of QoS differentiation between the two applications.

In the next two subsections, we evaluate the performance of our resource controller using two different experiments. For the results shown, a first order ($n = 1$) ARX model is used in the model estimator, along with a forgetting factor of $\lambda = 0.95$. $W = 1$ and $Q = diag(1, 1)$ are used in the optimal controller. Different values for the controller parameters have been tested. Interestingly, the resulting performance is either worse or comparable to that from using the default settings, therefore is not shown here.

## *C. Performance in Reference Tracking*

First, we fix all the other parameters, and let the control target, $y_{ref}$, vary from 0.3 to 0.5 then to 0.7. Each reference value is used for a period of 60 control intervals.

Figure 3 shows the measured per-interval throughput in requests per second (top) and the mean response time (middle) for the two applications, as well as the normalized RT ratio $y_1$ against its reference value (bottom) over a period of 180 control intervals (one hour). The vertical dashed lines indicate when there is a step change in the reference value. As we can see, the controlled output is able to track the changes in its reference pretty closely. The rising time is always within two intervals from the time when a step change occurs. The respective performance metrics for both applications are also behaving as we expected. For example, a $y_{ref}$ value of 0.3 gives preferential treatment to application 1, where application 1 achieves higher throughput and lower average response time than application 2 does. When $y_{ref}$ is set at 50%, both applications achieve comparable perfor-

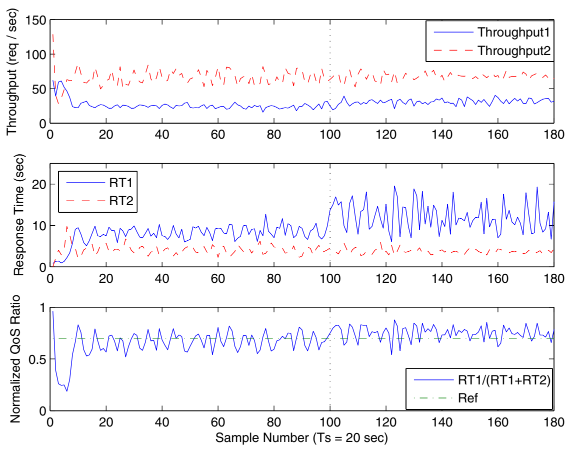

Fig. 6. Application QoS metrics with workload variation

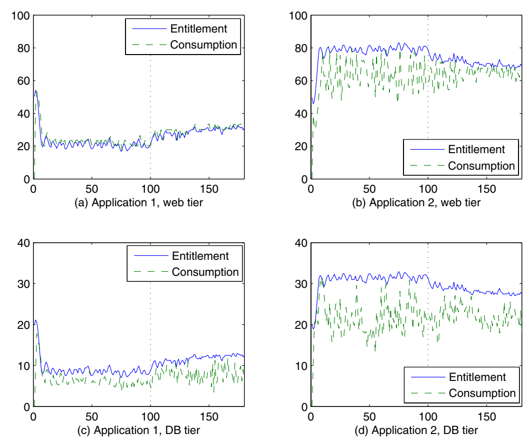

Fig. 7. CPU entitlement and consumption for individual application components with workload variation

mance. Finally, as $y_{ref}$ is increased to $70\%$, application 2 is able to achieve a higher level of performance than application 1 does, which is consistent with our expectation.

Figure 4 demonstrates effectiveness of our model estimation. The top figure shows the estimated values of the three parameters in the first order ARX model. The values converged fairly quickly (in less than 10 intervals) at the beginning, and stayed relatively stable afterwards. In steady states, the response time of a given application is a monotonic non-increasing function of the CPU entitlements to the containers hosting the application. That's why we feed $y_2 = 1 - y_1$ into the RLS algorithm such that the estimated values for $B_0 = [b_{01},\ b_{02}]$ are positive. The middle figure compares the measured values of $y_2$ to the model-predicted values. Excluding the first 10 samples where the model was converging, the calculated $r^2$ value between the two time series is $86\%$. This means the model is able to capture most of the variation in the data in spite of its simplicity. The bottom figure shows the prediction error, which stays within $\pm 0.1$ most of the time. These results suggest that a first-order linear model is sufficient for characterizing the input-output relationship in our system.

To give more insights to the behavior of our control system, we show in Figure 5 the corresponding CPU entitlements and resulting CPU consumptions of individual application components. As we can see, as $y_{ref}$ goes from 0.3 to 0.5 to 0.7, our controller allocates less and less CPU resource to both tiers in application 1, and more resource to application 2.

### D. Performance with Varying Workloads

In the second experiment, we fix the target RT ratio at $y_{ref} = 0.7$. We vary the intensity of the workload for application 1 from 300 to 500 concurrent users, while application 2's workload stays at 500 users throughout. This effectively creates varying resource demand in both tiers of application 1, and the controller needs to react accordingly to maintain the normalized RT ratio at a fixed value.

Figure 6 shows the measured throughput (top) and mean response time (middle) for the two applications, as well as the normalized RT ratio $y_1$ against its reference value (bottom) over a period of 180 control intervals. Again the vertical dashed line indicates when there is a step change in the workload. We can see that the controller is able to maintain the specified level of QoS differentiation in spite of workload changes in the applications. The reason why the throughput for application 1 did not increase much after the number of users increased from 300 to 500 is because of the closed-loop nature of the RUBiS client. It always waits for the reply to the previous request to return before sending a new request. This means, when the response time goes up, the offered load from the client side drops accordingly. Since a $70\%$ control target gives application 2 higher priority over application 1, even as application 1's workload increases, the controller forces it to back off so that the additional load on application 1 does not affect the performance of application 2, which is a validation of the performance isolation and differentiation provided by our controller.

Figure 7 shows the CPU entitlements and resulting CPU consumptions of all four application components. We see that application 2 gets allocated much higher CPU entitlements in both the web and the DB tiers. As the workload changes from 300 users to 500 users in application 1, the level of differentiation between the two applications at the resource level changes accordingly, in order to maintain the QoS ratio at a fixed level.

## V. Related Work

Control theory has recently been applied to computer systems for resource management and performance control [13], [14], [15], [16]. The application areas include web server performance guaranteees[17], dynamic adjustment of the cache size for multiple request classes [18], guaranteed relative delays in web servers by connection scheduling [19], CPU and memory utilization control in web servers [20], and to adjust the resource demands of virtual machines based on resource availability [21]. In this paper we focus on coordinated resource allocation across different tiers of multiple multi-tier applications, whereas most of previous works study either front-end admission control (i.e. lack of coordination across different tiers) or resource allocation to single-tier applications. Our controller can adjust the the allocation of system resources to application components in response to system and workload changes. Our approach also has the advantage that it does not require any application modifications as we only use sensors and actuators provided in the virtualization layer along with external application QoS sensors.

Dynamic resource management in a cluster environment has been studied with goals such as QoS awareness, performance isolation and higher resource utilization [22],[23],[24],[25]. It is formulated as an online optimization problem in [22] using periodic utilization measurements and resource allocation is implemented via request distribution. In [23], resource provisioning for large clusters hosting multiple services is modeled as a "bidding" process in order to save energy. The active server set of each service is dynamically resized adapting to the offered load. In [24], an integrated framework is proposed combining a cluster-level load balancer and node-level class-aware scheduler to achieve both overall system efficiency and individual response time goals. In [25], resource allocation is formulated as a two-dimensional packing problem, enforced through dynamic application instance placement in response to varying resource demands. In this paper, we study more fine-grained dynamic resource allocation in a virtualized server environment where application components are hosted inside individual virtual machines as opposed to individual nodes in a server cluster, and resource allocation is implemented through a fair share scheduler at the hypervisor level.

There are other efforts on dynamic resource allocation in shared data centers. In [26], time series analysis techniques are applied to predict workload parameters, and allocation involves solving a constrained nonlinear optimization problem based on estimation of resource requirements. A recent study is described in [27] for dynamic provisioning of multi-tier web applications. With the estimation of the demand in each tier, the number of servers are dynamically adjusted using a combination of predictive and reactive algorithms. In this paper, the kernel scheduler in the virtual machine monitor is used as the actuator for resource control. The resource demand of a workload is assumed to be time-varying which may or may not be predictable. Dynamic resource allocation is done with tunable time granularity based on the measured VM utilization and application-level QoS metrics. No estimation is required for the workload demand, and the controller adapts to the changing demand from the workload automatically. Moreover, our controller can deal with resource contention between multiple applications and achieve a desired level of performance differentiation.

In our prior work, we have developed a suite of dynamic allocation techniques for virtualized servers, including adaptive control of resource allocation under overload conditions [28], nonlinear adaptive control for dealing with nonlinearity and bimodal behavior of the system [9], and nested control for a better tradeoff between resource utilization and application-level performance [29]. These approaches are suitable for applications that are hosted inside a single virtual machine. In this paper, we present a dynamic resource allocation system for multi-tier applications with individual components hosted in different virtual machines.

Traditional work on admission control to prevent computing systems from being overloaded has focused mostly on web servers. Recent work has focused on multi-tier web applications. A "gatekeeper" proxy developed in [30] accepts requests based on both online measurements of service times and offline capacity estimation for web sites with dynamic content. Control theory is applied in [31] for the design of a self-tuning admission controller for 3-tier web sites. In [32], a self-tuning adaptive controller is developed for admission control in storage systems based on online estimation of the relationship between the admitted load and the achieved performance. These admission control schemes are complementary to the dynamic allocation approach we describe in this paper, because the former shapes the incoming resource demand into the system whereas the latter adjusts the supply of resources for handling the demand.

Proportional share schedulers allow reserving CPU capacity for applications [33], [34], [35]. In additional to enforcing the desired CPU shares, our controller can also dynamically adjust these share values based on application-level QoS metrics. It is similar to the feedback controller in [36] that allocates CPU to threads based on an estimate of thread's progress, but our controller operates at a much higher layer based on end-to-end QoS metrics that span multiple tiers in a given application. Other work includes resource overbooking in shared cluster environments which leverages application profiles [37] and calendar patterns (e.g., time of day, day of week) [38] to provide weak, statistical performance guarantees. However, these approaches require application demand profiles to be relatively stable and do not provide performance differentiation under overload situations. In contrast, our controller can cope with workload variations, even short term unanticipated changes and provides performance differentiation.

## VI. Conclusions and Further Work

In this paper, we address the problem of providing service differentiation between co-hosted applications when multiple shared servers are being overloaded on a shared hosting platform. Experimental evaluation of the self-tuning optimal control design validates that the closed-loop control system can track a specified level of service differentiation between the two co-hosted applications when both tiers of the applications become a bottleneck. And this can be achieved in spite of workload variations.

We identified the following topics for future research. First, in this study, the multiple applications are homogeneous in terms of running on the same set of stages. We would like to extend our approach to handle heterogeneous applications where different applications can run on different set of stages. Second, we have focused on CPU as the only resource shared among different applications. In practice, there are other resources, such as memory, network bandwidth and disk I/O, that may also be shared and may become bottlenecks and degrade end-to-end application performance. As ongoing work, we are experimenting with actuators provided by various virtualization technologies and will design more comprehensive feedback controllers that can coordinate the scheduling and allocation of multiple types of resources on a shared hosting platform. Finally, we are also interested in building a larger testbed with more than two applications to demonstrate the scalability of this approach, and study the general optimization problem where more resource and performance constraints could be considered.

## REFERENCES

[1] S. Graupner, J. Pruyne, and S. Singhal, "Making the utility data center a power station for the enterprise grid," Hewlett Packard Laboratories, Tech. Rep. HPL-2003-53, March 2003. [Online]. Available: http://www.hpl.hp.com/techreports/2003/HPL-2003-53.pdf

[2] P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, and K. Shin, "Adaptive control of virtualized resources in utility computing environments," in *Proceedings of EuroSys2007*, March 2007.

[3] M. Karlsson, X. Zhu, and C. Karamanolis, "An adaptive optimal controller for non-intrusive performance differentiation in computing services," in *Proceedings of the 5th International Conference on Control & Automation*, June 2005.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2003, pp. 164–177.

[5] C. Amza, A. Ch, A. Cox, S. Elnikety, R. Gil, K. Rajamani, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks," in *Proceedings of WWC-5: IEEE 5th Annual Workshop on Workload Characterization*, Oct. 2002.

[6] "OpenVZ." [Online]. Available: http://en.wikipedia.org/wiki/OpenVZ

[7] "Linux VServer." [Online]. Available: http://linux-vserver.org/

[8] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*, ser. ISBN: 0-471266-37-X. Wiley-IEEE Press, August 2004.

[9] Z. Wang, X. Zhu, and S. Singhal, "Utilization and slo-based control for dynamic sizing of resource partitions," in *Proceedings of the 16th IFIP/IEEE Distributed Systems: Operations and Management, Barcelona, Spain*, Oct. 2005.

[10] R. Kulhavý, "Restricted exponential forgetting in real-time identification," in *Automatica*, vol. 23(5), September 1987, pp. 589–600.

[11] L. Sun, J. Krodkiewski, and Y. Cen, "Control Law Synthesis for Self-Tuning Adaptive Control of Forced Vibration in Rotor Systems," in *2nd International Symposium MV2 on Active Control in Mechanical Engineering*, October 1997, pp. S9–25–37.

[12] "Xensource." [Online]. Available: http://www.xensource.com/

[13] Y. Diao, J. Hellerstein, S. Parekh, R . Griffith, G. Kaiser, and D. Phung, "A control theory foundation for self-managing computing systems," *IEEE journal on selected areas in communications*, vol. 23, no. 12, pp. 2213–2222, Dec. 2005.

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. IEEE Press/Wiley Interscience, 2004.

[15] J. L. Hellerstein, "Designing in control engineering of computing systems," in *Proceedings of American Control Conference*, 2004.

[16] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing controllable computer systems," in *Proceedings of the USENIX Workshop on Hot Topics in Operating Systems*, Jun. 2005, pp. 49–54.

[17] T. Abdelzaher, K. 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] Y. Lu, T. Abdelzaher, and A. Saxena, "Design, implementation, and evaluation of differentiated caching serives," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, May 2004.

[19] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "A feedback control approach for guaranteeing relative delays in web servers," in *Proc. of the IEEE Real-Time Technology and Applications Symposium*, 2001.

[20] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury, "Mimo control of an apache web server: Modeling and controller design," in *Proceedings of American Control Conference (ACC)*, 2002.

[21] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly virtual machines: leveraging a feedback-control model for application adaptation," in *Proceedings of the 1st International Conference on Virtual Execution Environments (VEE)*, M. Hind and J. Vitek, Eds. ACM, june 2005, pp. 2–12.

[22] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: a mechanism for resource management in cluster-based network servers," in *Proceedings of the international conference on Measurement and modeling of computer systems(ACM SIGMETRICS)*, 2000, pp. 90–101.

[23] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles(SOSP)*, October 2001.

[24] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated resource management for cluster-based internet services," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 225 – 238, 2002.

[25] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *Proceedings of the 15th International Conference on World Wide Web*, May 2006, pp. 595–604.

[26] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proceedings of the Eleventh IEEE/ACM International Workshop on Quality of Service (IWQoS)*, June 2003.

[27] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Proceedings of International Conference on Autonomic Computing (ICAC)*, 2005, pp. 217–228.

[28] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," in *Proceedings of the 9th International Symposium on Integrated Network Management*, May 2005.

[29] X. Zhu, Z. Wang, and S. Singhal, "Utility driven workload management using nested control design," in *Proceedings of American Control Conference (ACC)*, Jun. 2006.

[30] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel, "A method for transparent admission control and request scheduling in e-commerce web sites," in *Proceedings of the 13th international conference on World Wide Web*, 2004.

[31] A. Kamra, V. Misra, and E. Nahum, "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites," in *Proc. of the International Workshop on Quality of Service (IWQoS)*, Jun. 2004.

[32] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance isolation and differentiation for storage systems," in *Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS)*, 2004.

[33] M. B. Jones, D. Rosu, and M.-C. Rosu, "Cpu reservations and time constraints: efficient, predictable scheduling of independent activities," in *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, October 1997.

[34] J. Nieh and M. Lam, "The design, implementation, and evaluation of smart: A scheduler for multimedia applications," in *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, October 1997.

[35] C. A. Waldspurger and W. Weihl, "Lottery scheduling: flexible proprotional-share aresource management," in *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994.

[36] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, February 1999.

[37] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," in *Proceedings of the Fifth symposium on operating systems design and implementation (OSDI)*, Dec. 2002, pp. 239 – 254.

[38] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, "Statistical service assurances for applications in utility grid environments," *Performance Evaluation Journal*, vol. 58, no. 2-3, November 2004.