Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters

Evangelia Kalyvianaki
Computer Laboratory
University of Cambridge
15 JJ Thomson Avenue
CB3 0FD, UK
ek264@cl.cam.ac.uk

Themistoklis Charalambous
Department of Engineering
University of Cambridge
Trumpington Street
CB2 1PZ, UK
tc257@eng.cam.ac.uk

Steven Hand
Computer Laboratory
University of Cambridge
15 JJ Thomson Avenue
CB3 0FD, UK
smh22@cl.cam.ac.uk

ABSTRACT

Data center virtualization allows cost-effective server consolidation which can increase system throughput and reduce power consumption. Resource management of virtualized servers is an important and challenging task, especially when dealing with fluctuating workloads and complex multi-tier server applications. Recent results in control theory-based resource management have shown the potential benefits of adjusting allocations to match changing workloads.

This paper presents a new resource management scheme that integrates the Kalman filter into feedback controllers to dynamically allocate CPU resources to virtual machines hosting server applications. The novelty of our approach is the use of the Kalman filter—the optimal filtering technique for state estimation in the sum of squares sense—to *track* the CPU utilizations and update the allocations accordingly. Our basic controllers continuously detect and self-adapt to unforeseen workload intensity changes.

Our more advanced controller self-configures itself to any workload condition without any *a priori* information. Indicatively, it results in within 4.8% of the performance of workload-aware controllers under high intensity workload changes, and performs equally well under medium intensity traffic. In addition, our controllers are enhanced to deal with multi-tier server applications: by using the pair-wise resource coupling between application components, they provide a 3% on average server performance improvement when facing large unexpected workload increases when compared to controllers with no such resource-coupling mechanism. We evaluate our techniques by controlling a 3-tier Rubis benchmark web site deployed on a prototype Xen-virtualized cluster.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques, Modeling techniques.

ICAC'09, June 15–19, 2009, Barcelona, Spain.

General Terms

Management, Measurement, Performance.

1. INTRODUCTION

Recent advances in virtualizing commodity hardware (e.g. [4]) are changing the structure of the data center. A physical server is transformed into one or more virtual machines (VMs) that dynamically share the underlying hardware resources, and applications run within these isolated environments. Each VM is subject to management operations such as creation, deletion, and migration between physical machines, as well as run-time resource allocation. These features enable resource sharing in arbitrary combinations between applications and physical servers and provide the means for efficient server consolidation. However, to capitalize on this technology, it is *essential* to adaptively provision virtualized applications with resources commensurate with their workload demands. These demands are difficult to estimate since their characteristics typically change over time.

Fluctuating workloads [2] cause diverse and changing resource demands on the system components. Adjustable resource allocations that follow the workload fluctuations for virtualized applications are thus important to create a high performance server consolidation environment. Indeed, if each application is properly provisioned, additional resources can be used otherwise, e.g., to run additional applications, or to increase the throughput of existing ones.

In this paper we present an innovative control-based allocation approach that integrates the Kalman filtering technique [7] into a set of feedback controllers to dynamically provision CPU resources to multi-tier virtualized applications. Kalman filters have been used previously to estimate the parameters of a queueing model in a simulation environment [17]; however, to our knowledge, this is the first time that Kalman filters have been used directly to both track the CPU utilization of virtualized servers and to guide their resource allocations.

We formulate the allocation problem as a CPU utilization *tracking problem* where a controller aims to track and maintain the CPU allocation above the utilization within a certain safety margin. This is an intuitive approach to resource provisioning in which each VM is allocated resources as needed. Maintaining the CPU allocation to a reference input has been adopted by commercial products (e.g. HP Workload Manager), and other research prototypes [15, 11].

However our controllers uniquely integrate a very powerful filtering technique into linear feedback controllers.

The rest of this paper is organized as follows. Section 2 further motivates our approach to resource provisioning and the use of Kalman filters. The application performance model and the controllers are introduced in Section 3. Section 4 describes the Rubis benchmark and the evaluation platform as well as presenting our experimental results. Related work is discussed in Section 5. Finally, Section 6 concludes and looks to future work.

2. MOTIVATION

Resource provisioning techniques for virtualized applications broadly belong to two main categories: (a) *constraint-free* and (b) *constraint-based*. In constraint-free provisioning, each application is able to use up to the maximum physical capacity of the hosted server. This approach is administratively simple and therefore easy to implement. However, it does not provide any application performance guarantees, especially under conditions of contention [11]. Any of the applications could dominate resource use leaving the rest starving. Furthermore, due to the constant changes of the CPU utilizations by the hosted applications, it is very difficult to estimate the available free resources per physical machine, thus making any consolidation planning that respects application performance guarantees hard to implement.

In constraint-based resource configuration, each application is constrained to use a subset of the physical resources; popular approaches include *limit-based* and *adaptive upper-bound thresholds* configurations. For instance, the VMware DRS resource management tool [1] bounds CPU resource utilization between user-configured lower and upper limits. The advantages of this approach are two-fold. Firstly, a certain minimum application performance is guaranteed due the use of the lower limit. Secondly, since the utilization of all co-located VMs cannot pass beyond their aggregate upper limit, it is possible to estimate the free available resources for hosting additional applications. Although limit-based configuration is a simple and effective mechanism for performance-assured consolidation, it can fail when applications host diverse and frequently changing workloads. Performance violations occur when an application demands more resources than its upper limit, and resources are wasted when an application requires even less than its lower limit.

Resource configuration based on adaptive upper-bound thresholds addresses these shortcomings. In this case, a VM is continuously updated with the maximum resources it can use. The allocation for each application dynamically adapts to workload demands so as to always meet application performance guarantees. This mechanism enables other applications to be consolidated based on the available free resources. There have been several recent systems which use this basic approach via control theory (e.g. [11]).

In this paper we present a new control theory-based resource allocation management system for VMs that uses a simple performance model and the Kalman filter to track noisy resource utilizations and update the allocations.[1] The key novelty of our approach is the integration of a filtering technique into feedback controllers. We have chosen the Kalman filter since it is the optimal linear filtering technique when certain conditions hold and has good performance even when those conditions are relaxed. Using a filtering approach makes our controllers operate smoothly across different workloads. We have also extended our work to use the pair-wise resource coupling between components in multi-tier applications to adjust more rapidly to workload changes. Finally, and most importantly, we present a zero-configuration mechanism to detect and adapt to workload conditions without any advance information.

[1]We have introduced this approach in [8]. This is an extended version with an extensive experimental evaluation of the controllers. In particular, it emphasizes the evaluation of the self-configuring controller.

3. SYSTEM

This paper presents 3 Kalman-based feedback controllers:

1. The Single Input Single Output (SISO) Kalman Basic Controller, hereafter denoted as the **KBC**. This controller dynamically allocates CPU resources to individual VMs which can either host independent server applications, or be part of a multi-tier application spanning several VMs.

2. The Multiple Input Multiple Output (MIMO) Process Noise Covariance Controller, referred to as the **PNCC**. This controller adjusts the allocations of all the VMs of a multi-tier application, making use of the pair-wise covariances between VM resource utilization to capture the correlations between components. The goal of the PNCC is to allocate resources rapidly to multi-tier applications when compared to the KBC controller.

3. The Adaptive MIMO PNCC, or **APNNC**. The APNCC, like the PNCC, collectively allocates CPU resources to all VMs of an application. However in addition it self-configures its parameters and self-adapts to diverse workload conditions.

Each controller allocates CPU resources to VMs based solely on resource utilization observations and the application performance model. Each VM is regarded as a black-box which can host an application tier or a whole application. The terms tier, component, and VM are used interchangeably throughout. The rest of this section presents the application performance model and the controller designs.

3.1 Application Performance Model

The controllers use a simple and intuitive application performance model enhanced with Kalman filtering to track the CPU resource utilization of VMs.

Our performance model uses the well-known observation that when a server application reaches its saturation point, its performance—e.g., request-response times—degrades substantially. To maintain good performance the application should be allocated more resources than its current utilization; this excess of resources is usually referred to as *headroom*. However, maintaining the resource capacity above the demands is not trivial due to diverse workload conditions that usually cause substantial variance in utilization.

We start by modelling the time-varying CPU usage as a one-dimensional random walk. The system is therefore governed by the following linear stochastic difference equation:

$$v_{k+1} = v_k + t_k, \quad (1)$$

where v_k is the percentage of the total CPU capacity of a physical machine actually used by a component during the

interval indexed by k; and the independent random variable t_k represents the process noise and is assumed to be normally distributed. Intuitively, in a server system the CPU usage in interval v_{k+1} will generally depend on the usage of the previous interval v_k as modified by changes, t_k, caused by request processing, e.g. processes being added to or leaving the system, additional computation by existing clients, lack of computation due to I/O waiting, and so on.

If v_k is known, the controllers should maintain the allocation at a certain level of the usage in order to sustain good application performance; this can be customized for each server application or VM. Therefore, the purpose of the controllers is to track the utilization v_k and maintain the allocation above the utilization by a certain threshold. In what follows we present an integrated approach to application performance modelling and controller design.

3.2 Controller Design

As mentioned previously, our controllers use the Kalman filter to track the utilization and update the allocation accordingly. This approach essentially uses a filtering technique to eliminate the noise of the CPU utilization signal coming from transient workload changes while still discovering its main fluctuations.

The Kalman Filter [7] is a data processing method that estimates the state of a linear stochastic system in a recursive manner based on noisy measurements. The Kalman filter is optimal in the sum square error sense under the following assumptions: (a) the system is described by a *linear* model and (b) the process and measurement noise are *white* and *Gaussian*. It is also computationally attractive, due to its recursive computation, since the production of the next estimate only requires the updated measurements and the previous predictions.

3.2.1 KBC

The SISO KBC controller is a utilization tracking controller for individual VMs. All metrics presented in this subsection are scalar and refer to a single component. We define a component's *CPU allocation* a to be the percentage of the total CPU capacity of a physical machine allocated to a running VM; and u to be the measured/observed value of the utilization v.

The purpose of the controller is to compute the allocation of each VM using the application performance model that is based on tracking the utilization. The allocation signal is defined by:

$$a_{k+1} = a_k + z_k, \tag{2}$$

and is related to the utilization measurement u_k by:

$$u_k = ca_k + w_k, \tag{3}$$

where c denotes the headroom resources and is customized for each server application or VM. The independent random variables z_k and w_k represent the process and measurement noise respectively, and are assumed to be normally distributed:

$$p(z) \sim N(0, Q),$$
$$p(w) \sim N(0, R).$$

The measurement noise variance R might change with each time step or measurement, and the process noise variance Q will almost certainly change, reflecting different system dynamics. However for the purposes of the KBC both are assumed to be stationary during the filter operation. Another approach which considers *non-stationary* noise is presented later.

Equations (2) and (3) describe the system dynamics, and hence the required allocation for the next interval is a direct application of Kalman filter theory, used here to track the utilization v_k through the measurements u_k and subsequently the allocation a_{k+1}. This process proceeds as follows:

$\widetilde{a}_k$ is defined as the *a priori* estimation of the CPU allocation, that is the predicted estimation of the allocation for the interval k based on previous measurements. $\widehat{a}_k$ is the *a posteriori* estimation of the CPU allocation, that is the corrected estimation of the allocation based on new measurements. Similarly, the *a priori* estimation error variance is $\widetilde{P}_k$ and the *a posteriori* estimation is $\widehat{P}_k$. The predicted *a priori* allocation for the next interval $k+1$ is given by:

$$\widetilde{a}_{k+1} = \widehat{a}_k, \tag{4}$$

where the corrected *a posteriori* estimation over the previous interval is:

$$\widehat{a}_k = \widetilde{a}_k + K_k(u_k - c\widetilde{a}_k). \tag{5}$$

At the beginning of interval $k+1$, the controller applies the *a priori* $\widetilde{a}_{k+1}$ allocation. If the $\widetilde{a}_{k+1}$ estimation exceeds the available physical resources, the controller allocates the maximum available; hence the filter is active only in the under-loaded situation where the dynamics of the system are linear. The correction Kalman gain between the actual and the predicted measurements is:

$$K_k = c\widetilde{P}_k(c^2\widetilde{P}_k + R)^{-1}. \tag{6}$$

The Kalman gain K_k stabilizes after several k iterations. The *a posteriori* and *a priori* estimations of the error variance are respectively:

$$\widehat{P}_k = (1 - cK_k)\widetilde{P}_k, \tag{7}$$

$$\widetilde{P}_{k+1} = \widehat{P}_k + Q. \tag{8}$$

Kalman Gain:
The Kalman gain is important when computing the allocation for the next interval $\widetilde{a}_{k+1}$. It is a function of the variables Q and R which describe the dynamics of the system. In general, K_k monotonically increases with Q and decreases with R. This is also explained intuitively: Consider a system with large process noise Q. Its states experience large variation, and this is observed in any measurements taken also. The filter should then increase its confidence in the new error (the difference between the predicted state and the measurement), rather than the current prediction, in order to keep up with the highly variable measurements. Therefore the Kalman gain is relatively large. On the other hand, when the
measurement noise variation R increases, the new measurements are biased by the included measurement error. The filter should then decrease its confidence in the new error and in this case the Kalman gain values are relatively smaller. In addition, by scaling the Q and R values, the Kalman gain values can be tuned to make the controllers operate in different filtering modes — this is verified by our results in Section 4.

3.2.2 PNCC

The MIMO PNCC controller further extends the KBC controller to consider *resource coupling* between the components utilizations in multi-tier applications. (The utilization of the different components in multi-tier servers are correlated since the components work together to serve incoming requests).

Like the KBC, the PNCC adjusts the allocation of each component based on its own error. However in addition, using the covariance process noise, it also adjusts based on the errors caused by the other components. If n is the number of application components, then the PNCC Kalman filter equations for stationary process and measurement noise are:

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \mathbf{W}_k, \tag{9}$$

$$\mathbf{u}_k = \mathbf{C}\mathbf{a}_k + \mathbf{V}_k, \tag{10}$$

$$\widehat{\mathbf{a}}_k = \widetilde{\mathbf{a}}_k + \mathbf{K}_k(\mathbf{u}_k - \mathbf{C}\widetilde{\mathbf{a}}_k), \tag{11}$$

$$\mathbf{K}_k = \mathbf{C}\widetilde{\mathbf{P}}_k(\mathbf{C}\widetilde{\mathbf{P}}_k\mathbf{C}^T + \mathbf{R})^{-1}, \tag{12}$$

$$\widehat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{C}\mathbf{K}_k)\widetilde{\mathbf{P}}_k, \tag{13}$$

$$\widetilde{\mathbf{a}}_{k+1} = \widehat{\mathbf{a}}_k, \tag{14}$$

$$\widetilde{\mathbf{P}}_{k+1} = \widehat{\mathbf{P}}_k + \mathbf{Q}, \tag{15}$$

where $\mathbf{a}_k \in \mathbb{R}^{n\times 1}$ and $\mathbf{u}_k \in \mathbb{R}^{n\times 1}$ are the allocation and usage vectors respectively and each row corresponds to a component; $\mathbf{W}_k \in \mathbb{R}^{n\times 1}$ is the process noise matrix; $\mathbf{V}_k \in \mathbb{R}^{n\times 1}$ is the measurement noise matrix; $\mathbf{C} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with the target value c for each component along the diagonal; $\widetilde{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ and $\widehat{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ are the *a priori* and *a posteriori* error covariance matrices; $\mathbf{K}_k \in \mathbb{R}^{n\times n}$ is the Kalman gain matrix; and $\mathbf{R} \in \mathbb{R}^{n\times n}$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are the covariance matrices of the stationary measurement and process noises respectively. For matrices $\mathbf{Q}$ and $\mathbf{R}$, the diagonal elements correspond to the process and measurement noise for each component. The non-diagonal elements of the matrix $\mathbf{Q}$ correspond to the process noise covariance between different components. Similarly, the non-diagonal elements of the $\mathbf{K}_k$ matrix correspond to the gain between different components. For a 3-tier application, for example, the *a posteriori* $\widehat{\mathbf{a}}_k(1)$ estimation of the allocation of the first component at interval k is the result of the *a priori* estimation $\widetilde{\mathbf{a}}_k(1)$ of the allocation plus the corrections from all components' innovations, given by:

$$\begin{aligned}\widehat{\mathbf{a}}_k(1) = \widetilde{\mathbf{a}}_k(1) &+ \mathbf{K}_k(1,1)(\mathbf{u}_k(1) - \mathbf{C}(1,1)\widetilde{\mathbf{a}}_k(1)) \\ &+ \mathbf{K}_k(1,2)(\mathbf{u}_k(2) - \mathbf{C}(2,2)\widetilde{\mathbf{a}}_k(2)) \\ &+ \mathbf{K}_k(1,3)(\mathbf{u}_k(3) - \mathbf{C}(3,3)\widetilde{\mathbf{a}}_k(3)).\end{aligned}$$

The covariances in this case indicate the coupling of the utilization changes between components.

3.2.3 Adaptive-PNCC (APNCC)

So far only stationary process and measurement noises have been considered. Both controllers can be easily extended to adapt to operating conditions by considering non-stationary noises. For example in the case of the PNCC controller, all formulae are as before but instead of the stationary $\mathbf{Q}$, the dynamic $\mathbf{Q}_k$ is now used. In this case, $\mathbf{Q}_k$ is self-configured every several intervals with the latest computations of variances and covariances. Since our measurement sensors are simple, the measurement noise variance is considered to always be stationary, i.e. $\mathbf{R}_k = \mathbf{R}$.

3.2.4 Modeling Variances and Covariances

The allocation is considered to be proportional to the usage. Hence we can estimate its process noise variance Q by estimating the usage variance and then using the following formula (*var* denotes variance):

$$var(a) \simeq var(\frac{u}{c}) = \frac{1}{c^2}var(u). \tag{16}$$

The usage process noise corresponds to the evolution of the usage signal in successive time frames. Estimating its variance directly is difficult, since the usage signal itself is an unknown signal, which does not correspond to any physical process well described by a mathematical law. The usage variance is calculated from measurements of the CPU utilization.

Finally, the measurement noise variance R corresponds to the confidence that the measured value is very close or not to the real one. Once more it is difficult to compute the *exact* amount of CPU usage. However, given the existence of relatively accurate measurement tools, a small value (such as $R = 1.0$) acts as a good approximation of possible measurement errors.

Like the computation of the allocation variances, the covariances between the components' allocations are computed based on the usage covariances. If u^i and u^j are the measured usages between components i and j, then the covariance between their allocations a^i and a^j is computed as (*cov* denotes the covariance):

$$cov(a^i, a^j) \simeq cov(\frac{u^i}{c}, \frac{u^j}{c}) = \frac{1}{c^2}cov(u^i, u^j). \tag{17}$$

When the KBC or the PNCC controllers are used, the stationary process variances or covariances are computed offline in advance and remain constant at run-time. In the case of APNCC, they are computed on-line and their values are updated every few controller intervals. Different approaches will be compared in the evaluation.

4. EXPERIMENT EVALUATION

In this section we evaluate the controllers using a prototype virtualized cluster hosting a 3-tier Rubis [3] benchmark auction web server subject to varying workload conditions.

4.1 Prototype Virtualized Cluster

Figure 1 illustrates the prototype virtualized cluster which consists of three machines connected by Gigabit Ethernet and each running the Xen 3.0.2 hypervisor [4] and hosting the Rubis server application. Rubis is a prototype auction web server which models eBay.com. Each one of the three server components—Tomcat web server, JBoss application server and MySQL DB server—is deployed in a separate VM running on a separate physical machine. A fourth machine hosts the Rubis Client Emulator used to generate requests. In this paper we use two workload mixes available by the Rubis distribution: the *browsing mix* (BR) contains read-only requests and the *bidding mix* (BD) that includes 15% read-write requests. BR is mostly used unless stated otherwise. The Client Emulator also records the response times of requests and is used to evaluate the performance of the controllers.

We have built two modules to control the resource management process: the *manager* and the *controller* modules. Periodically, the `manager` module submits the mean CPU

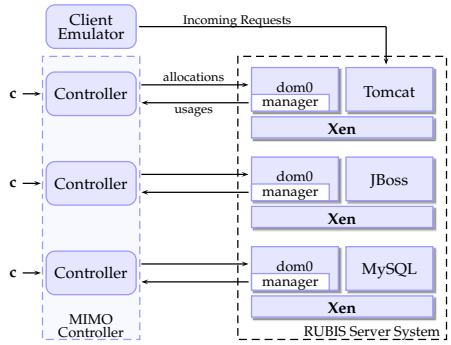


Figure 1: Virtualized prototype and control system. Solid lines between the controller modules and the Rubis Server System depict the three KBC SISO controllers. The MIMO controllers are shown by the dashed rectangle.

usage for the VM under control over the last interval to the `controller` module(s). The `controller(s)` compute the allocations for the next interval and enforce the new allocations for the specified VMs by using the CPU scheduler interface exported by Xen. Our prototype uses the "simple EDF" (SEDF) scheduler configured with the capped option so that no VM can use any more CPU time that it has been allocated. The `controller` modules run on the same machine as the Client Emulator.

This paper focuses on evaluating the performance of the Kalman filters for controlling the CPU allocations for virtualized server applications. To ensure that the server performance depends solely on the controller(s) CPU allocations, certain actions are taken. All machines have two CPUs, and each one of the two VMs per physical machine is pinned on a separate CPU. This simple setup enables us to study the impact of the controller(s) allocations on server performance without worrying about scheduling artifacts among running VMs sharing the same CPU. Finally, for all the experiments each VM is allocated memory as required when first created and this allocation is kept constant throughout. The network bandwidth is also measured and is never a bottleneck.

4.2 Preliminaries

This section describes the experimental methodology and sets the values of those parameters (i.e. controller interval, c, variances/covariances) that are kept the same in several experiments. The evaluation mainly uses two workload experiments:

Workload W0(`t1,t2,t3`): During this experiment, 300 clients issue requests for `t3` intervals in total. At the $\texttt{t1}^{th}$ interval, another 300 clients are added until the $\texttt{t2}^{th}$ interval. All requests belong to the BR mix. This experiment simulates a workload of *medium intensity* since there are two workload changes and the number of clients remains constant for the rest of the experiment.

Results of each experiment are illustrated by two graphs (see, e.g., Figure 2 presented later). The first graph shows the average Tomcat component utilization and the average corresponding allocation for each interval (denoted as *sample point* in the graphs). Due to space limitations, graphs of the JBoss and MySQL components are omitted, but they give qualitatively similar results. The second graph presents the server performance: mean response time (hereafter denoted as `mRT`) (in seconds) for each controller interval. The `mRT` data are *not* used to control the allocations; rather they are captured to provide a graphical representation of server performance.

Each controller is evaluated for its ability to: (a) follow the utilizations, and (b) maintain good server performance under workload changes. When the server components are adequately provisioned for any workload type the server maintains the `mRT` of requests ≤ 1 second (s).[2] This is the performance level the server is expected to achieve even when its resources are dynamically provisioned by the controllers and is used hereafter to evaluate the controllers performance.

Workload W1: During this experiment 200 clients issue requests for 60 intervals in total. At the 30^{th} interval, another 600 are added for the next 30 intervals. All requests belong to the BR mix. This experiment simulates an unexpected *large* workload increase and is used to assess the controller performance where components are most likely to saturate. The performance of the controllers is examined only for the duration of the change between the intervals 30 up to 50, until the server is settled down to the increased workload. In this way, emphasis is given only to the actual workload change.

We use three evaluation metrics, all of which are computed over a period of several intervals.

1. **CR** is the total number of completed requests.
2. **NR** denotes the percentage of requests with response time $\leq$ 1s over CR.
3. **RMSE** is the root mean squared error of the utilization prediction given by $\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{l-predicted(l)}{predicted(l)}\right)^2}$, where l denotes a request response time and N is the total number of requests.

The CR and NR metrics give aggregate numbers over request characteristics for some time duration. When the values of either NR or CR increase among experiments of the same type and duration, the performance of the server has improved. The RMSE metric provides a more detailed view of the request response times. For its calculation the predicted value is the `mRT` for a specific number of clients. Since the current system does not predict individual request response times, the RMSE uses the `mRT`. Therefore, an error will always be present between the model predictions (`mRT`) and the measured response times.[3] In general, the smaller the RMSE values the closer the response times to the `mRT` and the better the server performance. A combination of the three metrics CR, NR, and RMSE provide enough information to compare the different controllers.

4.2.1 Controller Interval

Intervals of 5s and 10s were examined. These intervals are chosen because if one takes the mean of the utilizations sampled with these resolutions, then this is very close to the actual long-term mean (e.g. 100s) of the system utilization.

[2]This was extensively measured under changing workloads. Due to space limitations these results are not presented here.
[3]For example, the RMSE in the case of a stationary workload mix of 600 clients for 20 intervals was 2.17.

Thus, these resolutions are not too coarse to describe the mean system performance and still do not impose a significant overhead due to system measurement. The evaluation in this paper uses a 5s interval throughout.

4.2.2 Parameter c

The parameter c, which denotes the level of the CPU utilization rate to the allocation, is set to 60%. Although 60% might seem low and that resources are "wasted" and hence less resources are left for another application to run on the same host, this value enables us to study the controllers' performance with few implications from saturated components. We have found that when the allocation approaches the utilization, the server performance degrades. A low utilization rate is chosen, so as to avoid transient component saturation as much as possible.

4.2.3 Variances and Covariances

The KBC and the PNCC controllers use off-line computed process variances and covariances which are essential to the computation of the Kalman gains.

Their values are computed based on Eq. (16) and (17) using utilization measurements coming from an experiment of a stationary workload mix of 600 clients for 40 intervals, repeated ten times for statistically confident results, and where each component is allocated 100% of its CPU. The allocation variances are: var(Tomcat, JBoss, MySQL)=(79, 13.19, 132). These values are hereafter referred to as Q_0. The allocation covariances are: cov(Tomcat, JBoss)= 6.5, cov(Tomcat, MySQL)=14.05, and cov(JBoss, MySQL)=5. The allocation covariance matrix that contains the above values is denoted as $\mathbf{Q}_0$. Since it is very hard if not impossible to compute all the Q_0 and $\mathbf{Q}_0$ values for all possible combinations of workload mixes, these numbers are used as an approximation in the case of the KBC and PNCC controllers for the different workloads.

Finally, the measurement noise variance is set to a small value (e.g. $R = 1.0$) given the existence of relatively accurate VM CPU measurement tools. This value acts as a good approximation of possible measurement errors.

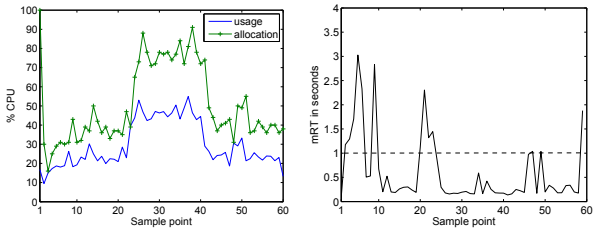
4.3 KBC

Figure 2 illustrates the KBC allocations and server performance during a W0(20,40,60) experiment where the initial allocations are set to 100%. Each controller tracks the usage fluctuations and adjusts the allocations accordingly, therefore the first goal of the control system is achieved. Figure 2(b) depicts the `mRT` for each interval which remains well bellow 1s for the majority of the intervals and 87.92% of requests have response times $\leq$ 1s. The second goal of the system, that is to maintain good server performance, is also achieved. The `mRT` exceeds 1s during the intervals where one or more components are CPU saturated; this is standard Rubis behavior.

Every KBC controller adjusts the allocations to match any transient utilization changes. This is due to the high values of the KBC Kalman gains (second column in Table 1) and cause each controller to have an increased confidence in the new observations and follow them rather than the old predictions. Although in this way the controllers react more quickly to any workload change, they also make unnecessary allocation adjustments, which could cause temporary component saturation.

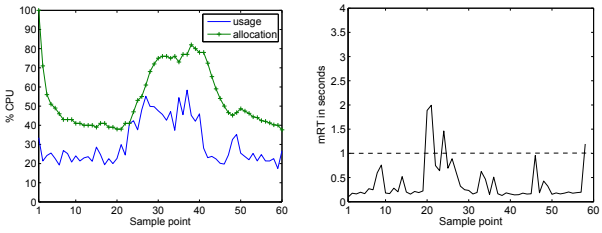
tier	Gains for KBC Q_0	Gains for KBC $Q_0/400$
Tomcat	1.61	0.38
JBoss	1.44	0.17
MySQL	1.63	0.48

Table 1: KBC Kalman gains.



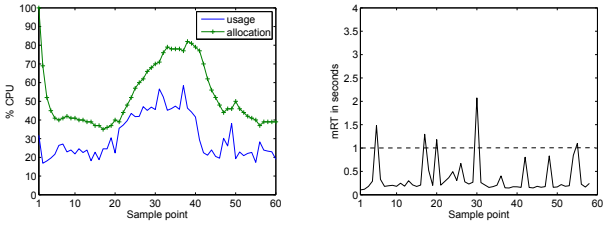
(a) Tomcat CPU (b) Requests `mRT`

Figure 2: KBC Performance, Q_0 Values.



(a) Tomcat CPU (b) Requests `mRT`

Figure 3: KBC Performance, $Q_0/400$ Values.

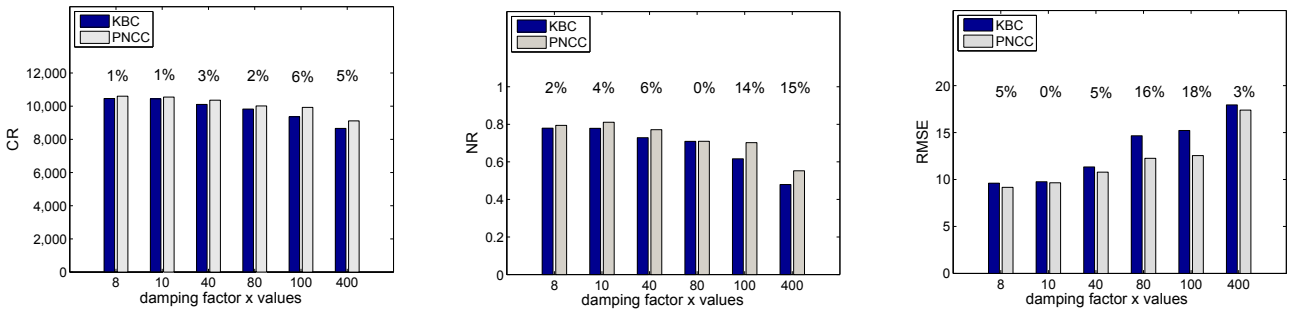


(a) Tomcat CPU (b) Requests `mRT`

Figure 4: PNCC Performance, $\mathbf{Q_0}/400$ Values.

To alleviate this situation the KBC can be configured to operate on different modes of workload noise filtering. For example, Figure 3 illustrates a W0(20,40,60) experiment where the Kalman gains (third column in Table 1) are configured to smaller values (as explained later) than before. In the latter experiment, the controllers have less confidence in the new measurements and they follow the utilization patterns in a less aggressive manner. The performance of the server is now more stable with less `mRT` spikes. Overall, the percentage of requests with `mRT` $\leq$ 1s has now increased to 89.6%.

The advantage of integrating the Kalman filtering into a feedback controller is not only it operates under different



(a) CR (b) NR (c) RMSE

Figure 5: PNCC and KBC Comparison for W1 Experiments. Percentages shown in each case represent the absolute metric difference of the PNCC over the KBC with a 90% CI after a t-test is applied. For each x value experiments are repeated 20 times.

filtering modes but in addition this is based on information coming from the application resource utilization itself rather than relying on ad-hoc methods or system identification analysis. The KBC controller operates on different filtering modes based on the gain which depends on the process Q and measurement noise variance R. In Figure 3 the process noise variances are set to $Q_0/400$ and are smaller than the R values. The controller considers the workload fluctuations/noise to come from the measurements themselves, is less confident in following the utilization changes and hence operates in a smooth filtering mode. Therefore, by simply scaling the initial Q_0 values in respect to R, both of which embody information regarding workload utilization, different modes of filtering are achieved.

However, when configured to filter out the noise, the controllers respond slowly to substantial workload increases; there are more `mRT` spikes around the 20^{th} interval in the case of the KBC in Figure 3(b) than in case of the KBC in Figure 2(b). A controller cannot distinguish between a small workload fluctuation and a substantial workload change. The PNCC controller evaluated next addresses this issue.

4.4 PNCC

The PNCC controller is designed to react fast to workload changes for multi-tier servers by considering the resource coupling from all components, even when it operates under a smooth filtering mode.

Figure 4 shows the PNCC controller allocations for a W0(20,40,60) experiment with workload fluctuations; **Q** matrix is set to the off-line computed values $\mathbf{Q_0}$ again divided by 400. The figures show that the PNCC not only tracks main utilization fluctuations, but is also able to adapt to the workload increase faster than the previous KBC controllers. Figure 4(b) illustrates that the `mRT` stays below 1s for most of the intervals of the workload increase.

The PNCC ability to adapt to workload changes faster than the KBC controllers comes from the utilization resource covariance between components by using the matrix **Q**. The final allocation of each component is the result of its own error plus the errors from the other components. For instance, the Tomcat allocations are the result of its own error (multiplied by the Tomcat gain) in addition to the control errors from JBoss (multiplied by the Tomcat-JBoss gain) and MySQL (multiplied by the Tomcat-MySQL). The Kalman gain values used by the Tomcat component in Figure 4 are: (Tomcat, Tomcat-JBoss, Tomcat-MySQL)= $(0.38, 0.0012, 0.0025)$.

4.4.1 W1 Comparison

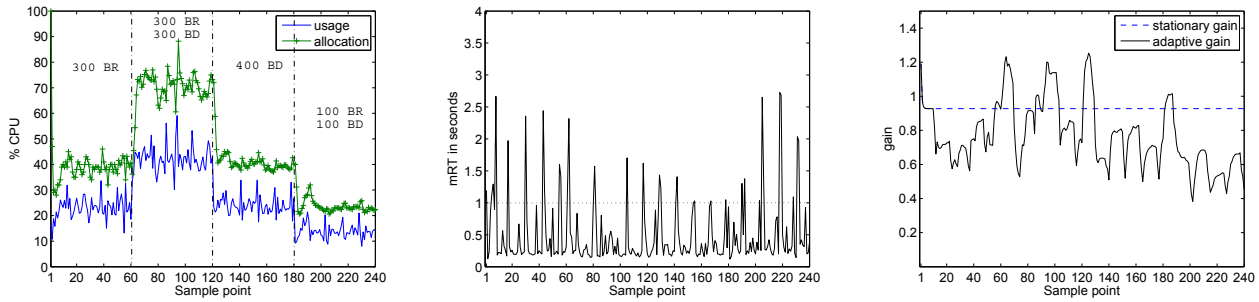
The performance of the PNCC controller in adjusting to substantial workload changes is now examined for different filtering modes. The covariance matrix **Q** is divided by x, hereafter denoted as the *damping factor*, drawn from $X = (x \in \{8, 10, 40, 80, 100, 400\})$. These values are selected to exercise a wide range of Kalman gains from relatively large to small values. Larger x values than those in X do not significantly change the Kalman gain values, and therefore are not used in the evaluation. Figure 5 shows the performance differences between the PNCC and the KBC as measured using all three metrics in an W1 experiment. The performance of both controllers decrease (CR and NR drop and the error RMSE increases) as the x damping factor increases. This is because the Kalman gains decrease and all controllers become less confident in their predictions than the measured values and hence they are slower to adapt to the increasing resource demands.

However, the PNCC controller equals or improves on the performance of the KBC controllers when considering all three metrics and all values of x. The average improvement across configurations according to the CR metric is 3%. In fact, the performance improvement of the PNCC over the KBC increases as the x values increase. In these cases, where the small Kalman gains make the controllers slow to react to workload changes, the improvement of the PNCC over the KBC is more apparent. In all cases, the PNCC reacts faster to workload changes than the KBC controllers because it incorporates all components errors.

4.5 APNCC

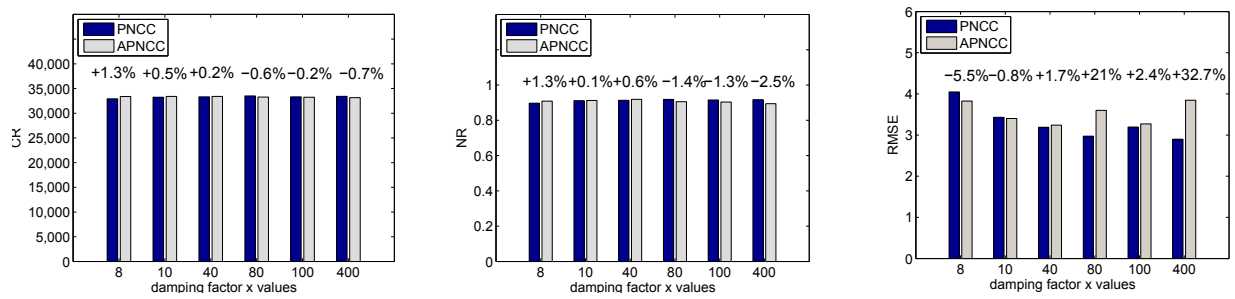
Both the KBC and the PNCC controllers use off-line computed variances and covariances. This section evaluates the Adaptive PNCC controller (APNCC) which estimates the covariance matrix **Q** online from utilization measurements. The advantage of this controller is that it self-configures its parameters and adapts to workload dynamics as they happen.

The APNCC is initially evaluated using a similar experi-



(a) Tomcat CPU allocation, usage (b) Requests `mRT` (c) Tomcat Kalman gain

Figure 6: APNCC Performance, Q/40 Values.



(a) CR (b) NR (c) RMSE

Figure 7: PNCC and APNCC Comparison for W0(40,80,120) Experiments. Percentages in each case show the metric difference of the APNCC controller over the PNCC with a 95% CI after a t-test is applied. A positive sign in the percentage indicates that the metric in the APNCC case has higher value than in the case of the PNCC. A negative sign shows the opposite.

ment to W0 where the workload intensity and mix are varied as shown in Figure 6. This experiment runs for longer than before and the Kalman gains are updated based on a sliding window mechanism that uses the utilization measurements of the last 10 intervals. In all cases the controller uses $\mathbf{Q}/40$ values. The controller tracks the utilization changes across diverse workload conditions, i.e. different workload mixes and number of clients; the `mRT` is $\leq$ 1s for the majority of the intervals and 89.58% of the requests have response times $\leq$ 1s. The spikes in the response times (Figure 6(b)) are due to transient JBoss CPU increases, caused by the Rubis benchmark, which cause the JBoss component to saturate.[4]

The online computed Kalman gain values are shown in Figure 6(c). The same figure also depicts the Kalman gains when computed off-line for 600 clients (dashed line) for reference purposes. The figure shows an increase of the gain values for the intervals $60 - 120$ during which the number of clients increases to 600. The adaptation mechanism captures the workload changes and the controller parameters are updated accordingly.

4.5.1 W0 Comparison

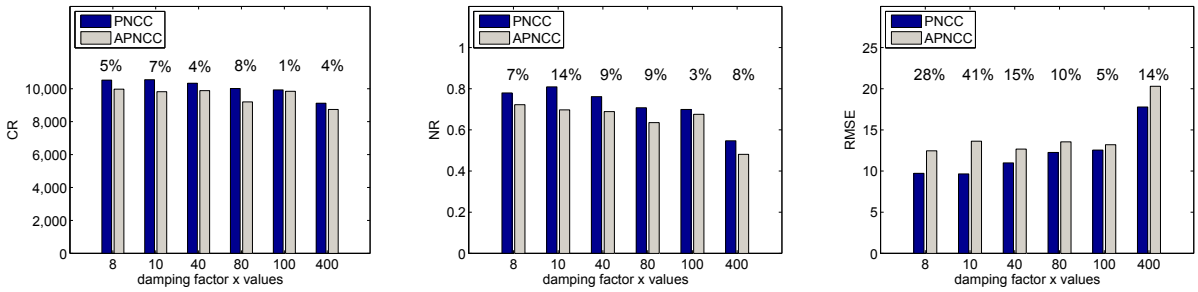
We also compare the APNCC with the PNCC using W0 experiments for various filtering modes. For different values of the damping factor $x \in X$ and for each different controller, a W0(40,80,120) experiment is repeated five times. In the case of the APNCC, the covariance matrix $\mathbf{Q}$ is estimated every 10 controller intervals from the utilization measurements. Results are shown in Figure 7. According to the CR and NR metrics, the self-adaptive APNCC performs equally well as the workload-aware PNCC. Finally, in the RMSE case, neither controller seems to perform better than the other according to all x values.

4.5.2 W1 Comparison

Here we examine the way the APNCC adapts to a substantial workload increase. Figure 8 depicts the APNCC Kalman gains for all three application components in an W1 experiment with $x = 8$. For the first ten intervals the APNCC gains are set to the PNCC off-line computed values. For every subsequent interval the gains are updated based on a sliding window mechanism that uses the utilization measurements of the last 10 intervals. Figure 8 shows that the APNCC adapts the gains to both workload conditions (viz. 200 clients during the first half of the experiment and 800 clients for the second half). In addition, during the workload transition, the gains increase because of the changes in the utilization variances. During the transition, the APNCC gains adapt to the workload increase without any *a priori* knowledge, but simply using the utilization variations as they happen.

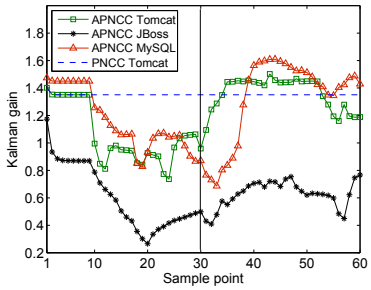
We further evaluate the APNCC against different filtering modes for $x \in X$. The results are shown in Figure 9

[4]Due to space limitation the JBoss graph is not shown here.



(a) CR (b) NR (c) RMSE

Figure 9: PNCC and APNCC Comparison for W1 Experiments. Percentages in each case show the absolute metric difference of the APNCC over the PNCC with a 95% confidence interval after a t-test is applied. For each x value experiments are repeated 20 times.



(a) Kalman gains, x=8

Figure 8: APNCC Kalman Gains for $x = 8$.

along with the PNCC performance for an equivalent experiment for comparison purposes. As shown by all evaluation metrics, the APNCC performance decreases with increasing x. This is aligned with previous observations since, as x increases, the APNCC operates in a smoother filtering mode and therefore is slower to detect any changes.

It is interesting to observe that when comparing the PNCC and the APNCC controllers for the same x values the APNCC performance slightly decreases. This is because at the time of the workload change, the PNCC off-line gain values are larger than the APNCC online estimated ones, as shown in Figure 8. This makes the PNCC react faster to the workload change. However, although the APNCC gains are smaller at the time the workload increase starts, the APNCC detects the change and adapts accordingly. Indicatively, its performance in terms of CR is on average only 4.8% worse than the PNCC across all configurations.

The PNCC off-line gains are computed for only a single workload type and cannot capture the range of utilization variability in a server application. The APNCC adapts to substantial workload increases resulting in a consistent good behavior, while the PNCC performance depends on how close its off-line gains capture the workload variability.

The APNCC does not need any special mechanism to detect a large workload increase. This controller, which treats all workload changes the same, automatically adapts its gains to depict the importance or not of the workload variation. Finally, when tuning the APNCC controller, one can essentially choose an x value and the controller would still adjust its gain to capture any subsequent workload changes.

5. RELATED WORK

5.1 Control-Centric Resource Provisioning

Single-Tier Applications: [15, 18] present feedback controllers to allocate CPU resources of a single-tier Apache server running on the HP-UX PRM resource container. These include (a) a proportional-integral controller that regulates the inverse `mRT` based on its linear relationship to the CPU allocation identified by system analysis; (b) a non-linear controller which regulates the relative utilization; and (c) combinations of the above. Compared to these methods, our Kalman-based controllers are linear and are based on a simple yet widely applicable model of the CPU utilizations.

Multi-Tier Applications: The control of multi-tier server applications has also gained attention. In [11] a two-layered controller that regulates the relative utilization of two instances of two-tier virtualized Rubis servers co-hosted on two physical servers is presented. The authors use the first layer controller from [15] to regulate the relative utilization for each tier and a second layer controller to further adjust the allocations using a QoS differentiation metric in cases of CPU contention. Wang *et al.* [14] present a 3-layer nested control design to control the CPU of a 3-tier Rubis application. The two inner loops are similar to [18]. The outer loop provides a better approximation of the corresponding utilization per tier in respect to the reference `mRT` as computed by a transaction mix performance model. Our MIMO controllers track the utilization using a simple CPU resource coupling performance model updated online rather than relying on application-specific transaction mixes.

Liu *et al.* [10] address the problem of resource sharing in cases of contention in shared virtualized clusters. Their controller allocates CPU resources based on a QoS response time ratio that exists in the overload region. Our MIMO Kalman controllers use the online resource coupling model of the (easily derived) metric of CPU utilization.

5.2 Other Approaches

In addition to control theory, other techniques have been used to manage resources in virtualized environments. For

instance, Xu *et al.* [16] present a 2-layer resource management system to minimize the resources consumed for single-tier applications to meet their SLAs, while maximizing the profit of a utility function over the shared resources' revenue using fuzzy modelling. In [6], the authors use a hybrid approach with queueing models and optimization techniques to decide component placement for consolidated virtualized applications.

5.3 Non-Virtualized Clusters

Before virtualization became widely adopted, other systems were developed to manage resource multiplexing. Here we briefly discuss some previous approaches to CPU resource allocation.

In Sharc [12], an exponentially-weighted moving average (EWMA) filter is used to estimate future CPU and network bandwidth resources based on past observations. The filter uses statically assigned parameters and can operate in a range of modes from being aggressively adaptive to changes of the observed signal (agile filter), or to being smooth on transient fluctuations (stable filter). However, this filter works only in one mode at a time and therefore it is not adaptive to different operating conditions.

To address the above limitations, Chase *et al.* [5] use a *flop-flip* filter based on similar filters from [9]. The flop-flip filter uses the moving average of the estimations over a 30s window and, if that estimation fails outside one standard deviation, it switches to the new moving average. The authors use this filter to smooth particularly bursty signals. The Kalman controllers we propose adapt dynamically to operating conditions without using pre-set values. Thus, they are much easier to deploy and require minimal configuration.

Urgaonkar *et al.* [13] use a profiling phase during which the application is run under realistic workloads to derive its resource utilization distribution. The authors also propose online updating of the resource distributions periodically. Our adaptive controller does not use off-line measurements and instead operates solely on short-term observations to make predictions of the workload utilizations.

6. CONCLUSIONS AND FUTURE WORK

High consolidation in virtualized clusters requires adaptive resource management of server applications. Control theory has been used to adjust the CPU allocations based on past utilization observations. This paper has presented the integration of the Kalman filter into feedback controllers for dynamically allocating the CPU resources of multi-tier virtualized servers. Experimental evaluation shows that: (a) filtering the utilization signal enables us to follow the workload changes without being strongly affected by transient fluctuations, and (b) our adaptive controller configures its parameters online using past utilization observations and self-adapts to workload conditions. We plan to further evaluate our controllers for high performance consolidation.

7. REFERENCES

[1] VMware Distributed Resource Scheduler (DRS). `http://www.vmware.com/pdf/vmware_drs_wp.pdf`, 2008.

[2] V. Almeida, M. Arlitt, and J. Rolia. Analyzing a Web-Based System's Performance Measures at Multiple Time Scales. *SIGMETRICS Performance Evaluation Review*, 30(2):3–9, 2002.

[3] C. Amza, A. Chandra, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and Implementation of Dynamic Web Site Benchmarks. In *Proc. of WWC-5*, pages 3–13, 2002.

[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 *Proc. of SOSP*, pages 164–177, 2003.

[5] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centres. In *Proc. of SOSP*, pages 103–116, 2001.

[6] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments. In *Proc. of ICAC*, pages 23–32, 2008.

[7] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] E. Kalyvianaki, T. Charalambous, and S. Hand. Applying Kalman Filter to Dynamically Resource Provisioning of Virtualized Server Applications. In *Proc. of FeBID*, pages 39–44, 2008.

[9] M. Kim and B. Noble. Mobile Network Estimation. In *Proc. of MobiCom*, pages 298–309, 2001.

[10] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In *Proc. of the IEEE CDC*, pages 3792–3799, 2007.

[11] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *Proc. of EuroSys*, pages 289–302, 2007.

[12] B. Urgaonkar and P. Shenoy. Sharc: Managing CPU and Network Bandwidth in Shared Clusters. *IEEE TPDS*, 15(1):2–17, 2004.

[13] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proc. of OSDI*, 2002.

[14] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal. AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments. In *Proc. of FeBID*, 2007.

[15] Z. Wang, X. Zhu, and S. Singhal. Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions. In *Proc. of DSOM*, pages 133–144, 2005.

[16] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *Proc. of ICAC*, 2007.

[17] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai. Tracking Time-Varying Parameters in Software Systems with Extended Kalman Filters. In *Proc. of CASCON*, 2005.

[18] X. Zhu, Z. Wang, and S. Singhal. Utility-Driven Workload Management using Nested Control Design. In *Proc. of the American Control Conference*, 2006.