Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters

EVANGELIA KALYVIANAKI, City University London
THEMISTOKLIS CHARALAMBOUS, Royal Institute of Technology
STEVEN HAND, Microsoft Research

Resource management of virtualized servers in data centers has become a critical task, since it enables cost-effective consolidation of server applications. Resource management is an important and challenging task, especially for multitier applications with unpredictable time-varying workloads. Work in resource management using control theory has shown clear benefits of dynamically adjusting resource allocations to match fluctuating workloads. However, little work has been done toward adaptive controllers for unknown workload types. This work presents a new resource management scheme that incorporates the Kalman filter into feedback controllers to dynamically allocate CPU resources to virtual machines hosting server applications. We present a set of controllers that continuously detect and self-adapt to unforeseen workload changes. Furthermore, our most advanced controller also self-configures itself without any *a priori* information and with a small 4.8% performance penalty in the case of high-intensity workload changes. In addition, our controllers are enhanced to deal with multitier server applications: by using the pair-wise resource coupling between tiers, they improve server response to large workload increases as compared to controllers with no such resource-coupling mechanism. Our approaches are evaluated and their performance is illustrated on a 3-tier Rubis benchmark website deployed on a prototype Xen-virtualized cluster.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Measurement Techniques, Modeling Techniques

General Terms: Management, Measurement, Performance

Additional Key Words and Phrases: Kalman filter, feedback control, multitier server applications, resource management, virtual machines

ACM Reference Format:
Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2014. Adaptive resource provisioning for virtualized servers using Kalman filters. ACM Trans. Autonom. Adapt. Syst. 9, 2, Article 10 (June 2014), 35 pages.
DOI: http://dx.doi.org/10.1145/2626290

1. INTRODUCTION

Virtualization technologies (e.g., [Barham et al. 2003]) have transformed 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

---

This work was conducted when E. Kalyvianaki, T. Charalambous, and S. Hand were at the University of Cambridge.

Primary author's address: E. Kalyvianaki, Department of Computer Science, City University London, Northampton Square, London EC1V 0HB, UK; email: evangelia.kalyvianaki.1@city.ac.uk.uk; Themistoklis Charalambous, Automatic Control Lab, School of Engineering, Royal Institute of Technology (KTH), Osquldas vag 10, 100-44 Stockholm, Sweden; email: themisc@kth.se; Steven Hand, Microsoft Research Silicon Valley, 1288 Pear Avenue, Mountain View, CA 94043, United States; email: steven.hand@microsoft.com.

DOI: http://dx.doi.org/10.1145/2626290

run within these isolated environments. Each VM is subject to basic management operations such as creation, deletion, and runtime 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.

Existing resource provisioning approaches may be broadly categorized as either *constraint free* or *constraint based*. In constraint-free provisioning, each application can use up to the maximum physical capacity of the hosted server. Although this approach is administratively simple, it does not provide any application performance guarantees. Under conditions of contention, any of the applications could dominate resource use, leaving the rest starving [Padala et al. 2007].

In a constraint-based approach, each application is configured with an upper-bound threshold to cap its maximum resource allocation. When applied in the context of virtualized server applications, a challenge is how to dynamically update the upper bound to match the diverse and time-varying workloads [Almeida et al. 2002] with changing resource demands. Related work in the area [Padala et al. 2007] has built nonlinear feedback controllers to continuously update the maximum CPU allocation based on CPU utilization measurements.

In this article, we incorporate the Kalman filter technique [Kalman 1960] to track noisy CPU utilizations into feedback controllers for multitier virtualized applications. Rather than using Kalman filters to estimate the parameters of a queueing model in a simulation environment [Zheng et al. 2005], we use the Kalman filters as linear 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 ensures our controllers operate smoothly across different workloads. We have also extended our work to use the pair-wise resource coupling between components in multitier 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.

This article extends our previous work [Kalyvianaki et al. 2009, 2010] and it contains:

(1) the formulation of the allocation problem as a CPU utilization *tracking* one;
(2) the integration of the Kalman filter into the Kalman Basic feedback Controller (KBC) to maintain the CPU allocation above the tracking utilization within a certain safety margin;
(3) the modeling of the resource coupling of multitier applications with the CPU covariance utilization matrix that seamlessly integrates with the Kalman filtering technique to the Process Noise Covariance Controller (PNCC);
(4) an additional adaptive mechanism to self-configure the parameters of the PNCC controller leading to the Adaptive-PNCC controller (APNCC);
(5) a theoretical and experimental demonstration of the ability of the Kalman gain values to tune the responsiveness of the controllers; and
(6) an extensive evaluation on a prototype virtualized cluster deploying the three-tier Rubis benchmark.

The rest of this article is organized as follows. In Section 2, we present the resource management architecture to control the CPU allocations of a prototype virtualized cluster. Section 3 builds upon the system identification process and derives the application performance model. The Kalman controllers are introduced in Section 4, while

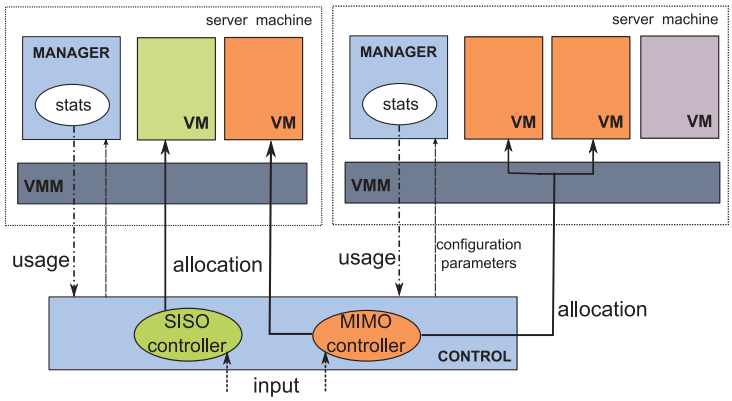


Fig. 1. Resource management architecture.

experimental results are presented in Section 5. Related work is discussed in Section 6. Finally, Section 7 concludes the article and discusses future work.

2. ARCHITECTURE AND TOOLS

This section presents the resource provisioning architecture and its implementation in the prototype virtualized cluster built for the purposes of system identification and performance evaluation.

In this work, we assume that application division to components, their placement in VMs, and the placement of VMs to physical servers are known a priori and remain static throughout the application execution. The maximum possible entitlement of CPU resources (i.e., the number of cores) to a VM is known and assigned at VM initialization. Our work focuses on the runtime CPU allocation of virtualized applications in server consolidation cases. For example, our approach can be used to control the CPU allocation of virtualized applications with strict quality-of-service guarantees. By capping the allocation of certain VMs, the remaining, if any, physical resources can be used by other virtualized applications with no strict guarantees such as in batch processing.

2.1. Management Architecture

The *target performance goal* of the management architecture is to continuously provision the virtualized application with enough CPU resources (subject to physical capacity) to *adequately* serve its incoming requests from a variable workload. We formally define this goal in Section 3.1.

In this section, we present the building blocks of the management architecture as shown in Figure 1. It implements a feedback control loop where management data flow at regular intervals between the `control` and the `manager` components through the *control input(s)* and *control output(s)* signals.

The `manager` component monitors the performance of applications running in VMs and submits performance measurements to the `control` component in near real time. The `manager` connects to the *virtual machine monitor* (VMM) that controls all VMs running on a particular physical server. The `controller` block in the `control` component executes the most prominent operation of the provisioning process. It implements the *control law* function to match the control output with the target performance goal. The control law relies on the application performance model to map control input(s) to control output(s).

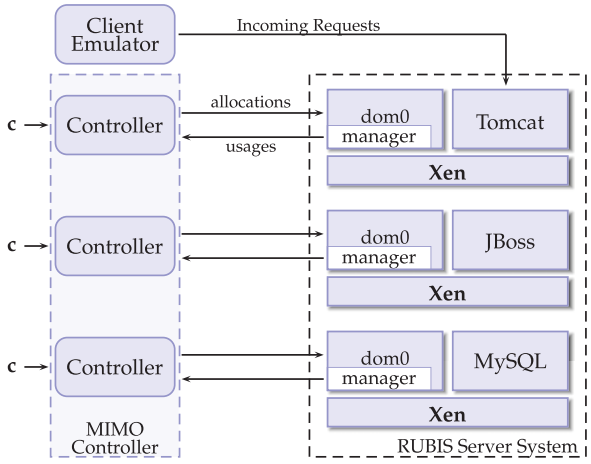


Fig. 2. Virtualized prototype and resource control system.

We derive the application model offline during the *system identification process* in Section 3. The model associates resource measurements to CPU allocations and the way they affect the application performance. Any deviation from the target performance, called *control error*, is corrected by the `controller`. The updated CPU allocations are remotely applied to the VM(s) by the `control` after performing any necessary transformations (e.g., checking that the new allocation does not exceed the total physical machine capacity).

Finally, the architecture supports remote resource allocation of arbitrary combinations of deployed components to physical machines. This is achieved by the clear separation of functions between the `manager` and `control` blocks. The `manager` operates as a server that monitors the usage of all VMs on the same physical machine as requested by its client (i.e., the `control` block). There can be one or more `control` blocks that manage the allocations of each, a subset of, or all of the application components. In this article, we present one *Single-Input-Single-Output (SISO)* controller implemented by one `control` block per application component and two *Multiple-Input-Multiple-Output (MIMO)* controllers realized by one `control` for all application components. Figure 1 illustrates the use of the `manager` and `control` blocks in the cases of a SISO and a MIMO controller for an application with three VMs. For the rest of this article and without loss of generality, we assume that a VM hosts a single application component. In the remaining, we use the terms tier, component, and VM interchangeably.

2.2. Prototype Virtualized Cluster

Figure 2 shows the virtualized prototype cluster and the three-component Rubis server application benchmark [Amza et al. 2002] deployed on three machines. Each machine runs the Xen hypervisor [Barham et al. 2003]. Each of the three Rubis server tiers—these are the Tomcat web server, the JBoss application server, and the MySQL database server—is deployed on a single VM running on a separate physical machine. A fourth machine hosts the Rubis Client Emulator that generates the requests to the server. There are three `manager` components and each runs within the Xen `dom0` VM. The `dom0` control VM is created at boot time by the Xen hypervisor to manage VMs and handle their resource multiplexing. We have implemented the `manager` and the `control` blocks using the Python programming language. The `manager` records new CPU usage statistics every 1 second using the `xentop` program provided by Xen to account for resource utilizations of running VMs.

The prototype cluster is deployed on typical x86 server machines used for commercial applications. All machines are identical, with two AMD Opteron Processors running at 2.4GHz, 4GB of main memory, 70GB of SCSI disk space, and a NetXtreme Gigabit Ethernet Card. All machines are connected over a Gigabit Ethernet network. Finally, all VMs run the commercial SUSE Linux Enterprise Server (SLES) 10, popular for server application deployment, with Linux-xen 2.6.16 and Xen 3.0.2. Xen is configured to use the Simple Earliest Deadline First (SEDF) CPU scheduler. The hardware and software setup of the server machines makes the cluster a realistic small-scale implementation of a virtualized data center.

In order to ensure that the application performance depends solely on the controller(s) allocations, we take the following actions. First, all machines have two CPUs and each one of the two VMs per physical machine is pinned on a separate CPU. This setup enables us to study the impact of the controller(s) on server performance without worrying about scheduling artifacts among VMs sharing the same CPU. Second, we configure the SEDF scheduler to operate on the *non–work-conserving* mode, where the maximum CPU a VM can use is capped by its allocation regardless of any free available resources by the machine. 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 never becomes a bottleneck.

2.3. Rubis Benchmark

The Rice University Bidding System (Rubis) [Amza et al. 2002] is a prototype auction website server application that follows the eBay.com model. Rubis was originally designed for testing web sites [Amza et al. 2002; Cecchet et al. 2002, 2003] and it has since been used for fault detection [Chen et al. 2004] and VM resource provisioning [Padala et al. 2007]. Rubis implements the basic operations of an auction site (i.e., selling, browsing, and bidding). A Rubis client can perform 27 different types of requests including browsing items from a category or a region, bidding on an item, and buying and selling an item. The Rubis Client Emulator provides two workload mixes: (a) the *browsing* mix with read-only requests and (b) the bidding mix with 15% read-write requests. For the rest of the article, we use the browsing mix, unless stated otherwise, since initial evaluation showed similar performance from the two mixes.

Finally, we have modified the Rubis Emulator to record the response time of each request (i.e., the time duration between the initiation of a request at the Emulator and the time the request response returns to the Emulator).

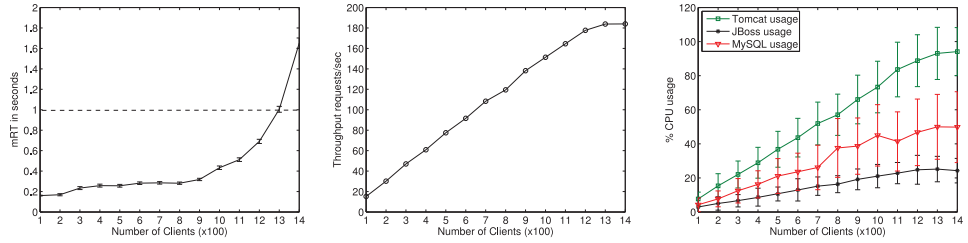
3. SYSTEM IDENTIFICATION

When dealing with multicomponent applications with unknown workloads, it is difficult to know *a priori* their precise performance model. To this end, during the system identification process, we employ a *black-box* approach to system modeling where we subject the application to variable workloads and measure its performance to model single-component applications in Section 3.2 and multicomponent coupling in Section 3.3. First, we start by deriving the target performance goal in Section 3.1.

3.1. Target Performance Goal

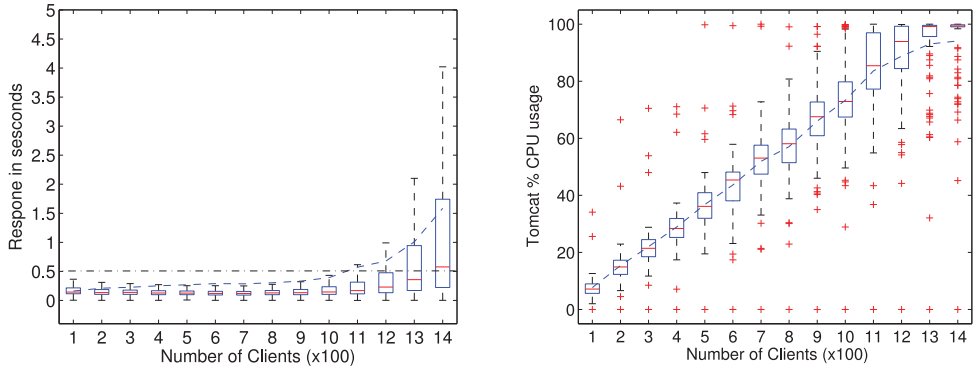
We define the target performance goal with respect to the number of clients the Rubis application can sustain effectively as measured by their request response times. We study the performance with respect to a varying number of clients issuing requests and we report results of the browsing workload-type mix. Preliminary results on the bid mix showed similar performance characteristics.

First, we measure the application performance when each component is allocated 100% of the CPU capacity and the number of clients varies from 100 to 1,400 for



(a) Response (b) Throughput (c) CPU utilisation

Fig. 3. System identification results. Error bars in Figure 3(a) correspond to a 95% confidence interval (CI) around the mean and in Figure 3(c) they show ±1 standard deviation (σ) around the mean.



(a) Response distributions (b) Tomcat CPU distributions

Fig. 4. Dataset statistics. For each of the 14 datasets from Figure 3, we show the first Q_1 and third quartiles Q_3 (box) and the median (line within each box). In each boxplot, whiskers (single lines above and below each box) are extended to 1.5 the interquartile range (IQR) above the Q_3 and below the Q_1. The dashed blue line indicates the mean in each dataset. Crosses show outliers. The JBoss and MySQL utilizations (not shown) are (or are very close to being) normally distributed since these components do not saturate.

200s in total each time. Figure 3(a) shows the mean client response time (`mRT`) and Figure 3(b) illustrates the corresponding throughput (`throughput`). As the number of clients increases up to 1,200, the `mRT` stays well below 1s and the `throughput` increases linearly with the number of clients. When the number of clients rises beyond 1,200, the `mRT` grows beyond 1 second, while the `throughput` remains constant. As expected, the figures show that there is a point of saturation below which the server operates effectively and above which its performance is unpredictable. Here, the server saturates at around 1,200 clients.

The CPU utilizations across tiers are shown in Figure 3(c). Each component uses more CPU resources as more clients issue requests. When the number of clients exceeds 1,200, the Tomcat VM reaches almost 100% of its allocation and it cannot serve more clients. It becomes the bottleneck component and as a result the `mRT` increases above 1s.

Therefore, the target performance goal for the current cluster is as follows: *The Rubis application can serve up to 1,200 clients with a performance of* $\texttt{mRT} \leq 1s$. Note that this denotes the level of performance the server is expected to achieve, even when our controllers dynamically allocate CPU resources.

In the aforementioned analysis, we use the mean statistic to summarize the response time and CPU utilization distributions. Figures 4(a) and 4(b) present a summary of the main statistics of the datasets' distributions as explained in the caption. The `mRT` for each dataset is also shown by the dashed line across boxplots. We observe that the

median statistic also captures a similar behavior: *The Rubis application can serve up to 1,300 clients with a performance of median response time* $< 0.5s$. Without loss of generality, for the rest of this article we use the mean as the centrality index.

3.2. CPU Resource Provisioning Model

The application performance model provides the relationship between the resource measurement(s) in the control output(s) and the CPU allocation to match the target goal. In this section, we identify the metric that best captures this relationship.

In Figure 3, we observe that the CPU utilization provides a very good indication of the allocation itself and it also relates to server performance; if a server is CPU saturated, it is very likely that its performance is degraded. A component utilization indicates its required allocation, and to maintain the reference performance, the controller should *follow* components' utilizations. A change in the usage observed over one period of time can be used to set the allocation for the next one. There are several advantages for using the utilization as the control output: (a) it is easily measured at the server side with a negligible overhead and (b) it is widely applicable across applications.

A simple way to model this observation is to always assign the allocations to the latest measured mean utilization. However, the mean statistic does not capture the utilization variability, which further affects server performance. To capture such implications, we perform the following experiments. For a stable workload (i.e., 800 clients of the browsing mix), the allocation of only one component at a time is the sum of two quantities: (a) the component mean utilization as it was measured offline for the particular workload and (b) an `extra` allocation, which is gradually increased from 0 up to 40 in steps of 5. The allocation for the other two components is set to 100% of their CPU capacity.

Figures 5(a) and 5(b) illustrate the `mRT` and the `throughput` respectively when the Tomcat component is subject to varying allocation. As the `extra` allocation increases, the `mRT` decreases and the `throughput` increases. Both the `mRT` and `throughput` stabilize when the `extra` allocation is 15. Increasing the allocation beyond this value does not improve the performance significantly. Similar experiments are performed for the JBoss and MySQL components in Figures 5(c), 5(d), 5(e), and 5(f), and in both cases the performance stabilizes when the `extra` allocation reaches 10. Note that the current experiments provide only an approximation of the `extra` allocation values.

Results indicate that to maintain the target performance, the allocation can be assigned to the mean utilization augmented by an additional `extra` value to capture its variability. Therefore, for a single-tier application, the relationship between the CPU utilization u and the allocation a can be modeled as:

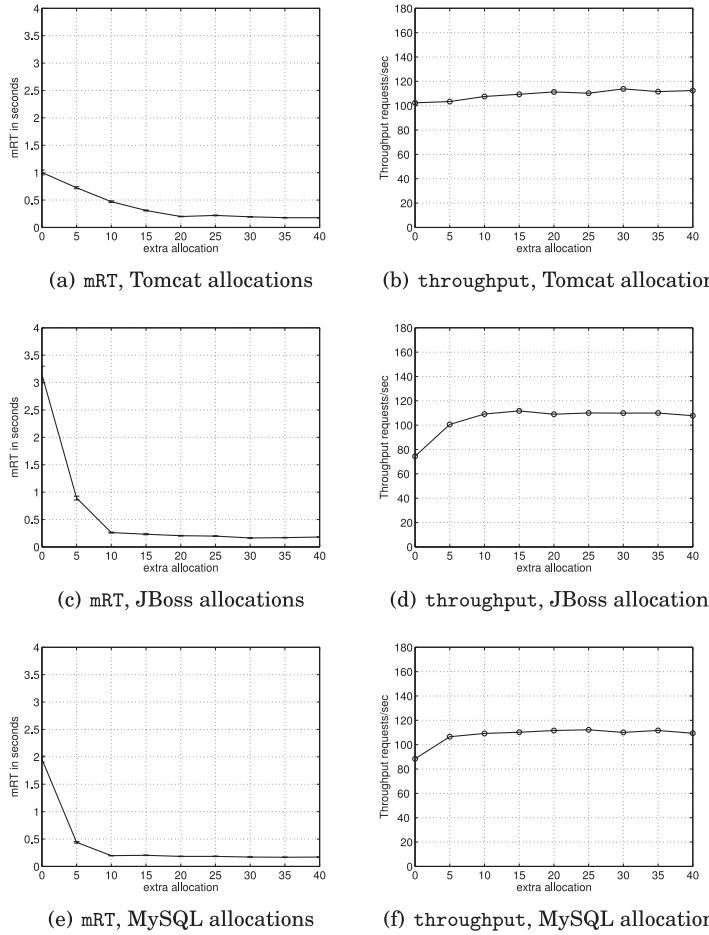
$$a = r \times u, \tag{1}$$

where $r \geq 1$ shows the `extra` allocation.[1]

Others have also identified such a relationship; for example, a common practice in data centers is to maintain a *headroom* of CPU resources above the utilization to allow applications to cope with workload fluctuations. In Padala et al. [2007], a two-tier virtualized Rubis server achieves good performance as long as the allocation is at least equal to a utilization proportion set well above 1.

We differ from these works in the following ways: (a) we provide a generic approach to modeling the CPU utilization changes of server applications using random walks (cf. Section 4.1), (b) we integrate the Kalman filtering technique with the performance

[1]Note that the target server performance can be achieved for various r values below a certain threshold. To estimate the minimum such value, we would require extensive offline analysis that would be tailored for a specific application. The scope of the current work is to derive a general application-independent model based on generic application characteristics.



(a) mRT, Tomcat allocations (b) throughput, Tomcat allocations

(c) mRT, JBoss allocations (d) throughput, JBoss allocations

(e) mRT, MySQL allocations (f) throughput, MySQL allocations

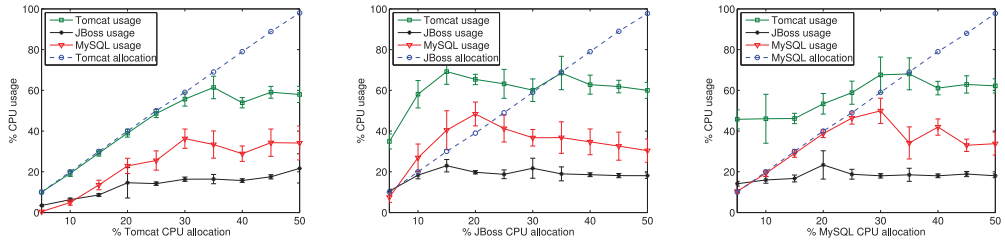
Fig. 5. CPU resource variability. Error bars in the mRT figures show a 95% CI around the mean.

model, and (c) we incorporate the resource coupling of multitier applications within our feedback controllers.

3.3. Intercomponent Resource Coupling

There is a resource coupling between the components of multitier applications (e.g., [Zhang et al. 2007]). If a component does not have enough resources to process incoming requests, it becomes a bottleneck component and the rest of the components cannot process additional requests.

This is further illustrated by the following experiment. The CPU allocation of one of the three components is varied from 10 to 100 in increments of 10 while the number of clients is kept constant at 800, and each of the other two components is allocated 100% of its CPU capacity. First, the allocation of the Tomcat component is varied. As shown in Figure 6(a), its usage follows the allocation until the allocation exceeds the required one for the current workload. The usage for the other two components increases slowly, despite their having the necessary resources to serve 800 clients. In this case, the bottleneck component is Tomcat, and since it does not have adequate resources to cope with the current workload, the other components' usages are affected



(a) Tomcat (b) JBoss (c) MySQL

Fig. 6. Intercomponent resource coupling. Error bars show a 95% CI around the mean usage.

as well. Similar behavior is observed when either the JBoss or the MySQL components are the bottlenecks, as shown in Figures 6(b) and 6(c).

Overall, in the case of a bottleneck, an increase of its allocation eventually leads to the increase in the CPU usage of the other components, suggesting that their allocations should be increased as well. A controller that takes into account the CPU usage of all the components and assigns the CPU allocation to each of them will clearly do better than one that does not.

4. KALMAN CONTROLLERS

This section presents three novel controllers based on the Kalman filtering technique to encapsulate the observation from the system identification analysis that VM allocations should *follow* the CPU utilizations of virtualized applications. The controllers include (a) the KBC controller (Section 4.1) that adjusts the CPU allocations of individual tiers, (b) the PNCC controller (Section 4.2) that expands the KBC design to use the resource coupling of multitier applications, and (c) the APNCC controller (Section 4.3) that further extends the PNCC design to estimate online the parameters of the performance model. Table I summarizes the notation we use for the controllers.

4.1. Kalman Basic Controller

The SISO Kalman Basic Controller (KBC) uses the Kalman filter to *track* the utilization and update the allocation accordingly. Since first presented by R.E. Kalman in his seminal 1960s paper [Kalman 1960], the Kalman filter has been used in a large number of areas including autonomous or assisted navigation, interactive computer graphics, and motion prediction. It is a data filtering 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 squared error sense under the assumptions that the system is described by a *linear* model, and 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; for a more comprehensive analysis of the Kalman filter refer to, for example, Simon [2006], Maybeck [1979], and Welch and Bishop [1995].

The key contribution of the KBC controller is the integration of a very powerful filtering technique into a linear feedback allocation controller. Rather than using Kalman filters to estimate the parameters of a performance model [Zheng et al. 2005], here, Kalman filters are used both as a tracking method *and* to build a feedback controller. The Kalman filter is particularly attractive since it is the optimal linear filtering technique when certain conditions hold, has good performance even when the conditions are relaxed, and uses the dynamics of the tracking signal to better track the signal itself. The latter is particularly beneficial for the noisy CPU utilization signals.

Table I. Kalman Controllers Notation

Symbol	Description
SISO controller	
n	number of application components
i	component index
a_k, a^i	CPU allocation at interval k, CPU allocation of component i
u_k, u^i	measured CPU usage at interval k, measured CPU usage of component i
t_k	process noise of real utilization v at interval k
z_k	process noise of allocation a at interval k
w_k	measurement noise at interval k
c	fraction of the utilization that accounts for the final allocation
Q	process noise variance
S	measurement noise variance
K_∞	steady-state Kalman gain
K_k	Kalman gain at interval k
$\widetilde{a}_k$	*a priori* allocation estimation at interval k
$\widehat{a}_k$	*a posteriori* allocation estimation at interval k
$\widetilde{P}_k$	*a priori* estimation error variance at interval k
$\widehat{P}_k$	*a posteriori* estimation error variance at interval k
MIMO controller	
$\mathbf{a}_k$	allocation for all components at interval k, $\mathbf{a}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{u}_k$	measured utilization for all components at interval k, $\mathbf{u}_k \in \mathbb{R}^{n\times 1}$
$\widetilde{\mathbf{a}}_k$	*a priori* allocation estimations for all components at interval k, $\widetilde{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$
$\widehat{\mathbf{a}}_k$	*a posteriori* allocation estimations for all components at interval k, $\widetilde{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{W}_k$	process noise for all components at interval k, $\mathbf{W}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{V}_k$	measurement noise for all components at interval k, $\mathbf{V}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{C}$	array with the c values for all components along its diagonal, $\mathbf{C} \in \mathbb{R}^{n\times n}$
$\mathbf{Q}$	process noise covariance matrix, $\mathbf{Q} \in \mathbb{R}^{n\times n}$
$\mathbf{S}$	measurement noise covariance matrix, $\mathbf{S} \in \mathbb{R}^{n\times n}$
$\mathbf{K}_k$	Kalman gains for all components, $\mathbf{K} \in \mathbb{R}^{n\times n}$
$\mathbf{Q}_k$	process noise covariance matrix at interval k, $\mathbf{Q}_k \in \mathbb{R}^{n\times n}$
$\mathbf{R}_k$	measurement noise covariance matrix at interval k, $\mathbf{R} \in \mathbb{R}^{n\times n}$

All metrics presented for the KBC controller are scalar and refer to a single component. We model the time-varying CPU usage as a one-dimensional random walk. The system is thus governed by the following linear stochastic difference equation:

$$v_{k+1} = v_k + t_k, \tag{2}$$

where v_k is the percentage of the total CPU capacity of a physical machine actually used by a component 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, etc.). Knowing the process noise and the usage over the previous interval, one can predict the usage for the next interval.

To achieve reference performance, the KBC controller uses the performance model from Equation (1). To this end, the allocation should be maintained at a certain level $\frac{1}{c}$ of the usage, where $c \leq 1$ is customized for each server application or VM.[2] The

[2]Note that $c = \frac{1}{r}$, where the `extra` allocation r was used to model the relationship between the CPU utilization and allocation in Equation (1).

allocation signal is described by:

$$a_{k+1} = a_k + z_k, \tag{3}$$

and the utilization measurement u_k relates to the allocation a_k, as:

$$u_k = ca_k + w_k. \tag{4}$$

The independent random variables z_k and w_k represent the process and measurement noise, respectively, and are assumed to be normally distributed:

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

Both the measurement and process noise variances (i.e., S and Q, respectively) might change with each time step or measurement. However, for the rest of this section, they are assumed to be stationary during the filter operation. Later, another approach, which considers nonstationary noise, is presented. Given that Equations (3) and (4) describe the system dynamics, the required allocation for the next interval is a direct application of the Kalman filter theory and is presented later.

We denote $\widetilde{a}_k$ to be the *a priori* estimation of the CPU allocation (i.e., the predicted estimation of the allocation for the interval k based on previous measurements). We denote $\widehat{a}_k$ to be the *a posteriori* estimation of the CPU allocation (i.e., the corrected estimation of the allocation based on 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:

$$\widetilde{a}_{k+1} = \widehat{a}_k, \tag{5}$$

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

$$\widehat{a}_k = \widetilde{a}_k + K_k(u_k - c\widetilde{a}_k). \tag{6}$$

At the beginning of the $(k+1)^{\text{th}}$ interval, 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. In the region where the allocation is saturated, the Kalman filter is basically inactive. Thus, the filter is active only in the underloaded situation where the dynamics of the system are linear. The correction Kalman gain between the actual and the predicted measurements is:

$$K_k = c\widetilde{P}_k\left(c^2\widetilde{P}_k + S\right)^{-1}. \tag{7}$$

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

$$\widehat{P}_k = (1 - cK_k)\widetilde{P}_k, \tag{8}$$

$$\widetilde{P}_{k+1} = \widehat{P}_k + Q. \tag{9}$$

Steady-State Kalman Gain. The Kalman gain is important when computing the allocation $\widetilde{a}_{k+1}$ for the next interval. It is a function of Q and S, which describe the dynamics of the system. In general, K_k monotonically increases with Q and decreases with S. We first explain this correlation intuitively. Consider a system with large process noise Q. Its states experience large variation, and this is shown by the measurements as well. The filter should then increase its confidence in the new error, rather than the current prediction, to keep up with the highly variable measurements. Therefore, the Kalman gain is relatively large. On the other hand, when the measurement noise variation S increases, the new measurements are biased by the included measurement

error. The filter should then decrease its confidence in the new error as indicated by the smaller values of the Kalman gain.

In fact, the Kalman gain depends on the ratio $\frac{S}{Q}$ as shown below. According to Simon [2006, Section 5.4.2], the steady-state Kalman gain, K_∞, is given by:

$$K_\infty = \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + S_k}, \tag{10}$$

where P_k^- is the steady-state covariance and for a scalar, the time-invariant Kalman filter is given by:

$$\lim_{k\to\infty} P_k = \frac{\tau_1}{2H^2}, \tag{11}$$

where τ_1 is:

$$\tau_1 = \sqrt{H^2Q + S(F+1)^2}\sqrt{H^2Q + S(F-1)^2}, \tag{12}$$

and H represents the transition between the states and the measurements; F represents the transition between states in the absence of noise.

In the case of a KBC controller $H = c$ and $F = 1$ and by using Equations (12), (11), and (10), the steady-state Kalman gain for the KBC controller is:

$$K_\infty = \frac{c + \sqrt{c^2 + 4\frac{S}{Q}}}{c^2 + c\sqrt{c^2 + 4\frac{S}{Q}} + 2\frac{S}{Q}}. \tag{13}$$

Therefore, the Kalman gain values depend on the ratio $\frac{S}{Q}$. By tuning the values of Q and S, the filter can be more or less reactive to workload changes. We will demonstrate this behavior in the experimental evaluation.

KBC Controller Stability.

PROPOSITION 4.1. *The KBC controller is stable when* $0 < K < \frac{2}{c}$.

PROOF. The KBC control law is:

$$a_{k+1} = a_k + K_k(u_k - ca_k). \tag{14}$$

The Z-transform of the allocation signal is:

$$zA(z) - za(0) = A(z) + KU(z) - cKA(z) \Leftrightarrow \tag{15}$$

$$(z - 1 + cK)A(z) = KU(z) + za(0). \tag{16}$$

The transfer function is thus given by:

$$T(z) = \frac{A(z)}{U(z)} = \frac{K}{z - 1 + cK}, \tag{17}$$

where the denominator is the characteristic function. The pole of the characteristic equation is given by:

$$z - 1 + cK = 0 \Leftrightarrow z = 1 - cK \tag{18}$$

and for stability it suffices for the pole to be within the unit circle, that is:

$$|z| < 1 \Leftrightarrow |1 - cK| < 1 \Leftrightarrow 0 < K < \frac{2}{c}. \tag{19}$$

Inequality Equation (19) is valid for all gain values; that is, from Equations (19) and (13) we have that:

$$\frac{c+\sqrt{c^2+4\frac{S}{Q}}}{c^2+c\sqrt{c^2+4\frac{S}{Q}}+2\frac{S}{Q}} < \frac{2}{c} \Leftrightarrow$$

$$c+\sqrt{c^2+4\frac{S}{Q}} < 2c+2\sqrt{c^2+4\frac{S}{Q}}+\frac{4\frac{S}{Q}}{c} \Leftrightarrow$$

$$0 < \sqrt{c^2+4\frac{S}{Q}}+c+\frac{4\frac{S}{Q}}{c},$$

which is always valid. Therefore, the KBC controller is stable for all values of the Kalman gain given by Equation (13). □

Modeling Variances. To obtain a good estimation of the variance Q, since it is considered to be proportional to the usage, it is enough to estimate the usage variance and then evaluate it via the following formula (var denotes variance):

$$var(a) \simeq var\left(\frac{u}{c}\right) = \frac{1}{c^2} var(u). \tag{20}$$

The usage process noise corresponds to the evolution of the usage signal in successive time intervals. Estimating its variance is difficult, since the usage signal itself is an unknown signal and does not correspond to any physical process well described by a mathematical law. We estimate the usage variance from measurements of the CPU utilization. In the KBC controller case, we compute the process variance Q offline.

Finally, the measurement noise variance S corresponds to the confidence that the measured value is very close to the real one. As before, it is difficult to compute the exact amount of CPU usage. However, given the existence of relatively accurate measurement tools, a small value ($S = 1.0$ is used for this article) acts as a good approximation of possible measurement errors.

4.2. Process Noise Covariance Controller

This section presents the MIMO Process Noise Covariance Controller (PNCC), which extends the KBC controller to consider the resource coupling between components in multitier applications. In this case, the allocation for each component is adjusted based on the errors of the current component in addition to the errors caused by the other components, through the covariance process noise. If n is the number of application components, then the PNCC Kalman filter equations for stationary process and measurement noise take the form:

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \mathbf{W}_k, \tag{21}$$

$$\mathbf{u}_k = \mathbf{C}\mathbf{a}_k + \mathbf{V}_k, \tag{22}$$

$$\widehat{\mathbf{a}}_k = \widetilde{\mathbf{a}}_k + \mathbf{K}_k(\mathbf{u}_k - \mathbf{C}\widetilde{\mathbf{a}}_k), \tag{23}$$

$$\mathbf{K}_k = \mathbf{C}\widetilde{\mathbf{P}}_k(\mathbf{C}\widetilde{\mathbf{P}}_k\mathbf{C}^T + \mathbf{S})^{-1}, \tag{24}$$

$$\widehat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{C}\mathbf{K}_k)\widetilde{\mathbf{P}}_k, \tag{25}$$

$$\widetilde{\mathbf{a}}_{k+1} = \widehat{\mathbf{a}}_k, \tag{26}$$

$$\widetilde{\mathbf{P}}_{k+1} = \widehat{\mathbf{P}}_k + \mathbf{Q}, \tag{27}$$

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, respectively; $\mathbf{K}_k \in \mathbb{R}^{n\times n}$ is the Kalman gain matrix and $\mathbf{S} \in \mathbb{R}^{n\times n}$; $\widetilde{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$ and $\widehat{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$ are the *a priori* and *a posteriori* allocation vectors, respectively, and each row corresponds to a component; and $\mathbf{Q}, \mathbf{S} \in \mathbb{R}^{n\times n}$ are the measurement and process noise matrices, respectively. For matrices $\mathbf{Q}$ and $\mathbf{S}$, the diagonal elements correspond to the process and measurement noise for each component. The nondiagonal elements of the matrix $\mathbf{Q}$ correspond to the process noise covariance between different components. Similarly, the nondiagonal elements of the $\mathbf{K}_k$ matrix correspond to the gain between different components. For a three-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, 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 between variables show how much each variable is changing if the other one is changing as well. In this case, the covariances indicate the coupling of the utilization changes between components.

PNCC Stability. For ensuring the stability of the PNCC, we use the following theorem.

THEOREM 4.2 [COSTA AND ASTOLFI 2008, THEOREM 2]. *Let the following linear time-invariant system:*

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{w}_k, \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{v}_k,\end{aligned} \tag{28}$$

where $\mathbf{x} \in \mathbb{R}^{n\times 1}$ *is the state;* $\mathbf{y} \in \mathbb{R}^{r\times 1}$ *is the observed variable;* $\mathbf{w} \in \mathbb{R}^{p\times 1}$ *and* $\mathbf{v} \in \mathbb{R}^{q\times 1}$ *form stationary zero-mean independent white noise processes satisfying* $\mathbb{E}\{\mathbf{w}_k\mathbf{w}_k^T\} = \mathbf{I}$ *and* $\mathbb{E}\{\mathbf{v}_k\mathbf{v}_k^T\} = \mathbf{I}$*, respectively; and the independent random variable* $\mathbf{x}_0$ *is such that* $\mathbb{E}\{\mathbf{x}_0\} = \bar{\mathbf{x}}_0$ *and* $\mathbb{E}\{\mathbf{x}_0\mathbf{x}_0^T\} = \Psi$*. The Kalman filter corresponding to Equation* (28) *is stable if, and only if, the following conditions hold:*

(H_1) $(\mathbf{A}, \mathbf{C})$ *is detectable.*
(H_2) *Unreachable modes of* $(\mathbf{A}, \mathbf{E})$ *do not lie in the unit circle (*$\mathbf{E}$ *is the available data regarding* $\mathbf{B}$*).*
(H_3) $(\mathbf{A}, \mathbf{E})$ *is semistabilizable or* $\Sigma > 0$ *(*Σ *is the available data regarding* Ψ*).*

These necessary and sufficient conditions for stability of the Kalman filter concern even cases in which the covariance error is bounded. Regarding (H_1), we can easily show that since $\mathbf{A} = \mathbf{I}$ and $\mathbf{C}$ is a diagonal matrix with entries smaller than 1, our system, which represents a random walk, is observable, and as a result detectable. Regarding (H_2), $\mathbf{E}$ is such that $\mathbf{E}\mathbf{E}^T = \mathbf{Q}$ and $(\mathbf{I}, \mathbf{Q}^{1/2})$ is controllable and as a result unreachable modes of $(\mathbf{I}, \mathbf{Q}^{1/2})$ do not lie in the unit circle. Finally, regarding (H_3), we choose $\mathbf{P}_0 = \mathbf{S}$ and hence, $\Sigma > 0$.

Table II. Roadmap of the Experimental Evaluation Section

Section	Description
5.1	Experiments description, evaluation metrics, parameters setting
5.2	KBC performance for stationary and varying workloads, KBC responsiveness with Kalman gain values, SASO properties
5.3	PNCC performance for varying workloads, SASO properties
5.4	KBC and PNCC comparison during workload changes
5.5	APNCC performance across number of clients and workloads
5.6	PNCC and APNCC comparison during workload changes
5.7	AKBC comparison with a state-of-the-art feedback controller

Modeling Covariances. Similarly to the computation of the allocation variances, the covariances between the component allocations are computed offline 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):

$$cov(a^i, a^j) \simeq cov\left(\frac{u^i}{c}, \frac{u^j}{c}\right) = \frac{1}{c^2} cov(u^i, u^j). \tag{29}$$

4.3. Adaptive Process Noise Covariance Controller

So far only stationary process and measurement noises have been considered. We now extend the PNCC controller to adapt to varying operating conditions by considering nonstationary noise. We present the Adaptive Process Noise Covariance Controller (APNCC), which has the same formulae as the PNCC controller, but instead of the stationary $\mathbf{Q}$, the dynamic $\mathbf{Q}_k$ is now used. In this case, $\mathbf{Q}_k$ is updated every several intervals with the latest computations of variances and covariances from CPU utilization measurements. For simplicity, the measurement noise variance is considered to always be stationary; that is, $\mathbf{S}_k = \mathbf{S}$. In the same way, we can extend the KBC controller to the Adaptive Kalman Basic Controller (AKBC) to adapt to varying workloads by dynamically updating Q_k every few intervals.

5. EXPERIMENTAL EVALUATION

This section evaluates the performance of each Kalman-based controller separately and also compares the SISO against the MIMO controllers. Finally, it evaluates the adaptive APNCC against the offline approach used by the PNCC. The detailed roadmap of this section is shown in Table II. We start by discussing the types of experiments and evaluation metrics used.

5.1. Preliminaries

Experiments Description. We use three types of experiments, summarized in Table III. A `Stable(t1, t2, t3)` experiment evaluates the basic functions of a controller to adjust its allocations to follow the utilizations while it maintains the target performance, that is, `mRT` $\leq$ 1s under workload changes. We use different graphs to illustrate the results. We show the average component CPU utilization and allocation for each Rubis tier for each controller interval, which is indicated as *sample point* in the graphs. Additional graphs illustrate the performance of Rubis for each interval, that is, (a) the `mRT` measured in seconds, (b) the `throughput` measured in requests/second, and (c) the cumulative distribution function (CDF) of response times for the duration of the experiment. Recall that the `mRT` data are *not* used to control the allocations but rather are captured to provide a graphical representation of server performance. Also, the Kalman gains for the controllers may be depicted. A `Varying(n,d)` experiment

Table III. Summary of the Performance Evaluation Experiments

Symbol	Description
`Stable(t1, t2, t3)`	**Varying number of clients:** 300 clients issue requests for `t3` intervals in total. At the $t1^{th}$ interval, another 300 clients are added until the $t2^{th}$ interval.
`Varying(n,d)`	**Stationary workload:** `n` number of clients issue requests for `d` intervals in total.
`Spike`	**Large workload change in the number of clients:** 200 clients issue requests for 60 intervals in total. At the 30^{th} interval, another 600 are added for the next 30 intervals.

Table IV. Performance Evaluation Metrics

Symbol	Description
CR	number of completed requests
NR	percentage of completed requests with response time $\leq$ 1s
RMS	root mean squared (RMS) error of request response times $$RMS = \sqrt{\frac{1}{N}\sum_{i=1} N(\frac{l(i) - \hat{l}(i)}{\hat{l}(i)})^2},$$ where N is the number of requests and $l(i)$ is the measured and $\hat{l}(i)$ is the predicted response time for the i^{th} request. The predicted response time for every request is the `mRT` for a certain number of clients calculated during the system identification process. We use the data from Figure 3(a).
`additional` allocation	sum of the differences of CPU resources between the allocations and the utilizations of a component
COV	coefficient of variation (COV), $$\text{COV} = \frac{s}{\bar{x}},$$ where s is the sample standard deviation and $\bar{x}$ is the sample mean of a statistic

explores the implications from different configuration values on controller performance. Finally, a `Spike` experiment is used to compare the SISO against the MIMO controllers. These experiments are designed to stress the server under sudden and large workload increases and evaluate the potential of the MIMO controllers to react faster due to the incorporation of resource coupling. In this case, the controllers' performance is evaluated only for the duration of the workload change until the system is stabilized to a fixed number of clients.

Evaluation Metrics. We evaluate the controllers using five different performance metrics shown in Table IV. All metrics are calculated over a duration of several intervals and given in the text when used. The CR and NR metrics capture overall performance and server performance improves when the values of either the CR or the NR metric increases. The RMS metric provides a detailed evaluation of request response times. Note that our approach does not predict response times for individual requests and hence, a small negligible error will always be present between the predicted and the measured response time values. For example, the `mRT` for 600 clients is 0.282s (Figure 3(a)) and the RMS during a `Varying(600,20)` experiment is measured and is 2.1703. The smaller the RMS values, the closer the response times are to the `mRT`. Together, the three metrics CR, NR, and RMS provide enough information to evaluate and compare the controllers.

The `additional` allocation metric evaluates the controllers' resource allocations. If similar performance, as measured with the CR, NR, and RMS metrics, is achieved for

Table V. Offline Measured Utilization Variances, Q_0, and Covariances, $\mathbf{Q}_0$

Component	Usage variance
Tomcat	28.44
JBoss	4.75
MySQL	47.43

Component pair	Usage covariance
(Tomcat, JBoss)	2.36
(Tomcat, MySQL)	5.06
(JBoss, MySQL)	1.80

different values of `additional` allocation, then the smallest values are preferred. In this way, more resources are available for other applications to run.

The COV metric measures the variability of the allocation and the utilization signals. Different degrees of allocation variability might be appropriate depending on the applications sharing a virtualized cluster. For example, consider a server application colocated with a batch-processing workload (e.g., MapReduce [Dean and Ghemawat 2004]), which does not have any real-time performance guarantees. The performance of the batch-processing workload is not highly affected by small fluctuations in its share of allocated CPU resources coming from noisy CPU allocations to the server application. At the same time, a server application might be very sensitive even to transient saturations, which can cause unpredictably high response times. Depending on the types of virtualized applications, different allocation variability might be desired.

Parameter Configuration. The controller interval is important since it controls the frequency of the new allocations and the time period over which the usage is averaged and used by the controllers to make new allocations. With a small interval, a controller reacts quickly to workload changes but is prone to transient workload fluctuations. A better approximation is achieved with a larger interval, as the number of sample values increases, but controller responses can be slower. Depending on the workload characteristics, the interval can be set to smaller values for frequently changing workloads and larger ones for more stable workloads. We examined intervals of 5s and 10s. Average utilizations over both durations were close to the mean utilization over long runs (e.g., 100s), and hence, both intervals are suitable to summarize usages. To achieve faster responses to workload changes, hereafter, we use intervals of 5s.

The parameter c, which controls the `extra` allocation, is set to 60%. Although 60% might seem low and might "waste" resources, this value enables the evaluation of the controllers with few implications from transient component saturation.

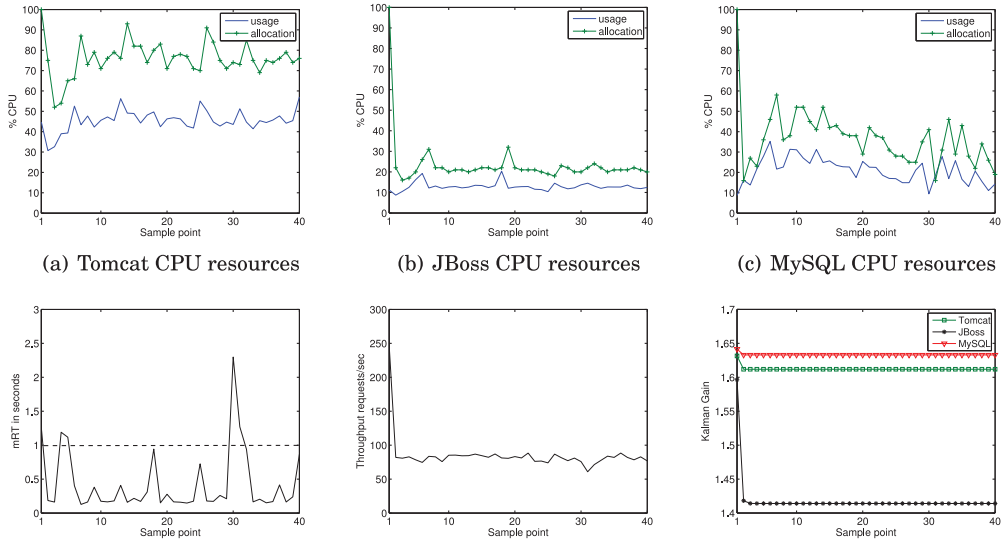
The KBC and the PNCC controllers use offline computed process variances and covariances, which are essential to the computation of the Kalman gains. Their values are computed based on Equations (20) and (29) using utilization measurements from a `Varying(600,40)` experiment repeated 10 times for statistically confident results and where each component is allocated 100% of its CPU. The left-hand-side table of Table V shows the utilization variances of the three components, referred to as Q_0. The right-hand-side table of Table V depicts the utilization covariances between components. The covariance matrix with the offline computed variances and covariances is referred to as $\mathbf{Q}_0$.

Finally, the measurement noise variance is set to a small value, that is, $S = 1.0$, given the existence of relatively accurate VM CPU measurement tools. This value acts as a plausible approximation of possible measurement errors.

5.2. KBC

This section evaluates the KBC controller against stationary and varying workloads. It also explores the effects of the Kalman gain on the responsiveness of the controller to workload changes and its allocation variability.

Stationary Workload. Figure 7 illustrates a `Varying(600,40)` experiment for a stationary workload. The CPU allocations correctly track the utilizations for each



(a) Tomcat CPU resources (b) JBoss CPU resources (c) MySQL CPU resources

(d) Requests mRT (e) throughput (f) Kalman gains

Fig. 7. KBC performance for stationary workload and Q_0 values.

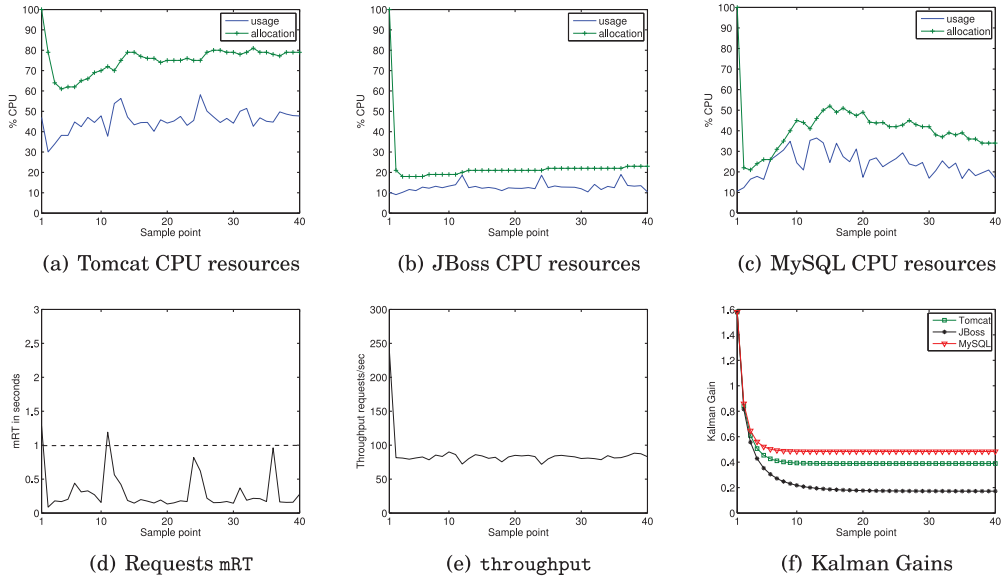
component and server performance is mostly sustained below its target value. The mRT spikes are caused when a component is very close to saturation as each KBC controller corrects its allocation to match even the most subtle utilization changes.

Kalman Gain. In Figure 7, the allocations correctly "follow" the usage. However, it might be more useful for a controller to adjust the underlying trends of the CPU signal while not being strongly affected by its variance. In the Kalman filter, this is achieved by tuning Q and S, which describe the dynamics of the system and affect the gain K (Section 4.1). In general, K monotonically increases with Q and decreases with S. When Q is large, the system experiences large variations and so the filter uses a higher K to correct its predictions according to the highly variable measurements. On the other hand, an increased S indicates measurement errors and so the filter with a lower gain K becomes more conservative in updating its state according to the new measurements.

To study the effects of the relative values of the Q and S on the controller performance, we perform a Varying(600,40) experiment where we divide Q by 400 while keeping S the same. Results are shown in Figure 8. The allocations are smoother than the utilizations for all three components and also smoother than shown previously in Figure 7. In the latter case, the values of the Kalman gains (Figure 8(f)) are smaller than previously (Figure 7(f)) and therefore the filter has more confidence in the predicted values than in the measured ones. The server performance is again maintained below its target performance, although with fewer mRT spikes than before.

To systematically evaluate the effects of the Kalman gain, we divide the Q_0 variances by different values of the *damping factor* x drawn from $Y = \{1, 4, 8, 10, 40, 80, 100, 400, 800, 1000\}$. The Y set covers a wide range of values, and for each x value we perform a Varying(600,200) experiment. Results are shown in Figure 9 and Table VI.

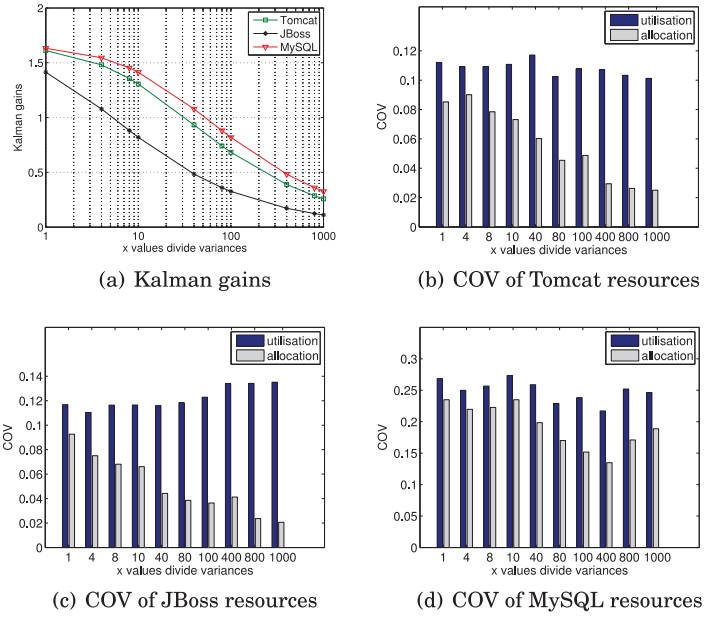
First, we discuss the effects of the Kalman gain on the allocation variability using Figure 9. As the damping factor x increases, Q decreases and the Kalman gain for each component drops (Figure 9(a)), indicating that the predicted value of the allocation becomes more important than the new measurement. This further affects the



(a) Tomcat CPU resources (b) JBoss CPU resources (c) MySQL CPU resources

(d) Requests `mRT` (e) `throughput` (f) Kalman Gains

Fig. 8. KBC performance for stationary workload and $Q_0/400$ values.



(a) Kalman gains (b) COV of Tomcat resources

(c) COV of JBoss resources (d) COV of MySQL resources

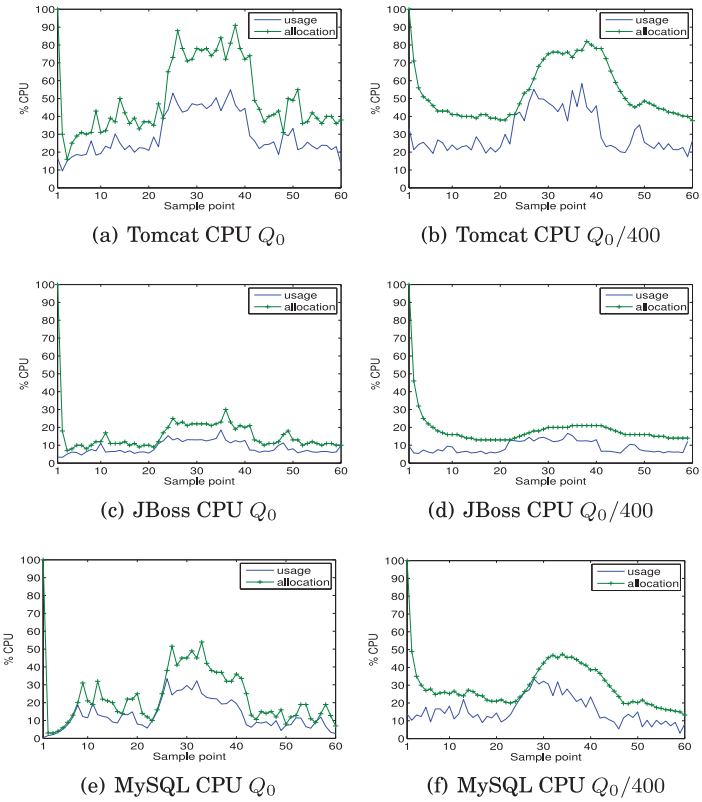
Fig. 9. KBC performance for stationary workload and different x values.

variability of the allocation signal. As Q decreases, the allocation COV decreases too, while the utilization COV remains almost the same for all three components; the allocations are smoother due to the small Kalman gains. In the case of MySQL, the COV increases for the last two x values. This is because there is a small utilization fluctuation that changes the mean utilization depicted in the allocation as well.

Second, we examine server performance using Table VI that shows the CR, NR, and RMS metrics for the duration of the experiment. As x increases, the performance of the

Table VI. KBC Performance for Stationary Workload and Different x Values

x values	CR	NR	RMS	`additional`
1	82840	0.9164	2.3433	10145
4	82949	0.9304	2.0628	10477
8	82940	0.9324	1.8298	10350
10	83512	0.9365	1.7725	10374
40	83554	0.9350	1.8826	10530
80	83700	0.9429	1.5800	10530
100	83564	0.9425	1.7291	10587
400	83860	0.9462	1.5017	10972
800	83639	0.9470	1.3871	11091
1000	83962	0.9504	1.4417	11085



(a) Tomcat CPU Q_0 (b) Tomcat CPU $Q_0/400$

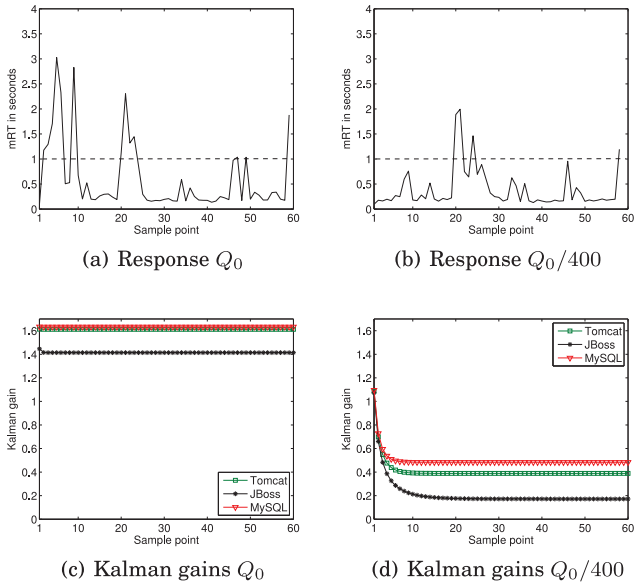
(c) JBoss CPU Q_0 (d) JBoss CPU $Q_0/400$

(e) MySQL CPU Q_0 (f) MySQL CPU $Q_0/400$

Fig. 10. KBC allocations for variable workload and Q_0 and $Q_0/400$ values.

server improves: it serves more clients (increasing CR), the percentage of requests with response times $\leq$ 1s increases too (increasing NR), and all requests are closer to the `mRT` (decreasing RMS). However, note that server performance improves with increasing x because the workload is stable with few negligible transient fluctuations, which do not affect the controllers. As the allocations are slower to act to resource changes, this might cause slower reactions to workload changes. This is also shown by the increasing `additional` resources allocated to components. This issue is considered next.

Workload Changes. Figures 10 and 11 illustrate the KBC allocations and server performance during two `Stable(20,40,60)` experiments. The controllers on the left are



(a) Response Q_0 (b) Response $Q_0/400$

(c) Kalman gains Q_0 (d) Kalman gains $Q_0/400$

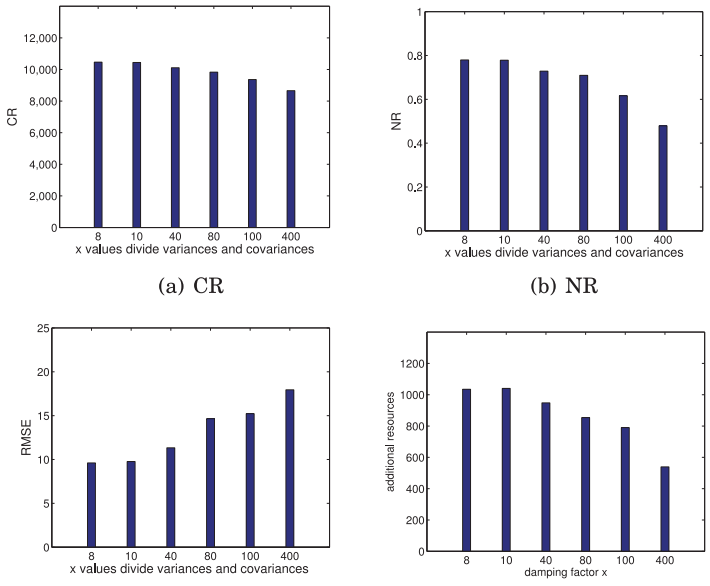
Fig. 11. KBC server performance for variable workload and Q_0 and $Q_0/400$ values.

configured with Q_0 values and the right-hand-side ones are set with $Q_0/400$ values. In both cases, the controllers track the usage fluctuations and the allocations are adjusted accordingly (Figure 10). However, it is apparent that the allocations of the right-hand figures are slower to follow the resource changes. This is due to their smaller gains (Figures 11(d) and 11(c)), which makes the controllers have more confidence in their predicted values than in the new measurements.

We now evaluate the effects of the Kalman gains under workload increases. We perform `Spike` experiments repeated 20 to 40 times for each $x \in X = \{8, 10, 40, 80, 100, 400\}$ and results are shown in Figure 12. Note that X is a subset of the initial set Y of damping factors. Additional experiments, not presented here, showed that results with values $x \in \{1, 4, 800, 1000\}$ do not significantly contribute to the evaluation. Results show that as Q decreases, when the x damping factor increases, both CR (Figure 12(a)) and NR (Figure 12(b)) decrease too. When Q decreases, the Kalman gains decrease too and the KBC controllers become less confident in their predictions than the measured values, and hence they are slower to adapt to the increasing resource demands, also shown by the decreasing `additional` resources in Figure 12(d). For the same reasons, the RMS (Figure 12(c)) increases while Q decreases.

SASO Properties. We examine three of the SASO properties: settling times, maximum overshoot, and zero steady-state error for the KBC controllers. The analysis uses Figures 13(a) and 13(b), which are produced using the data from Figures 10(a) and 10(b), to illustrate allocations based on two Kalman gains, which cause diverse controller behavior. Each figure shows the measured allocation (line labeled "allocation") and a hypothetical allocation as calculated from the utilization signal divided by the c parameter (line labeled "utilization/c"). The latter corresponds to the steady-state output of the controller according to the reference input. In this case, there is a reference output and every KBC controller should converge to it because they are integral controllers and have zero steady-state error.

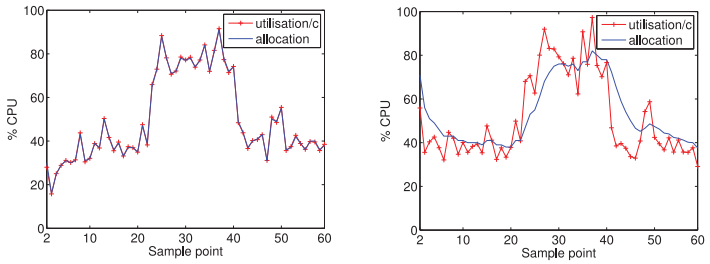
The settling times depend on the values of the Kalman gains. In Figure 13(a), where the Kalman gain is relatively large, the "allocation" signal is identical to the signal



(a) CR (b) NR

(c) RMS (d) `additional`

Fig. 12. KBC performance for workload increases and different x, Q values.



(a) Tomcat CPU for Q_0 (b) Tomcat CPU for $Q_0/400$

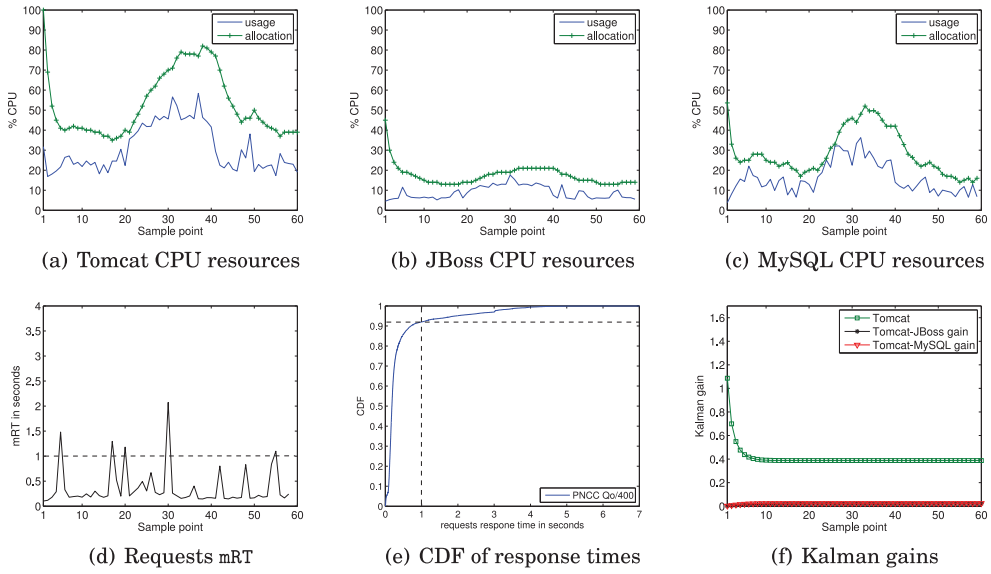
Fig. 13. Settling times and overshoot for KBC controllers.

that corresponds to the steady-state values; therefore, the controller converges in just one interval. In Figure 13(a), where the Kalman gain is smaller, the controller takes a few intervals to converge to values very close to the steady state, despite the noisy utilizations. Additionally, the KBC controllers do not overshoot. In both cases, they approach the steady state without exceeding its value (e.g., intervals $20 - 40$).

Summary. To summarize, evaluation showed that the KBC controllers achieve their goals so that the allocations follow the utilizations and server performance remains close to its target value during both stationary and variable workload conditions. Depending on the Kalman gain values, the controllers achieve different variability in the allocation and subsequently exhibit different server performance.

5.3. PNCC

The PNCC controller merges the KBC controllers into one MIMO design to incorporate the resource correlation of the components. To evaluate its performance, we perform a `Stable(20,40,60)` experiment with workload fluctuations. Results in Figure 14 show that the PNCC allocations adapt to the utilization fluctuations (Figures 14(a), 14(b),



(a) Tomcat CPU resources (b) JBoss CPU resources (c) MySQL CPU resources

(d) Requests mRT (e) CDF of response times (f) Kalman gains

Fig. 14. PNCC performance for variable workload and $\mathbf{Q_0}/400$ values.

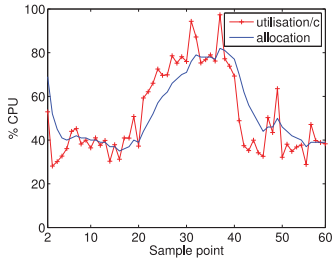


Fig. 15. Settling times and overshoot for the PNCC controller.

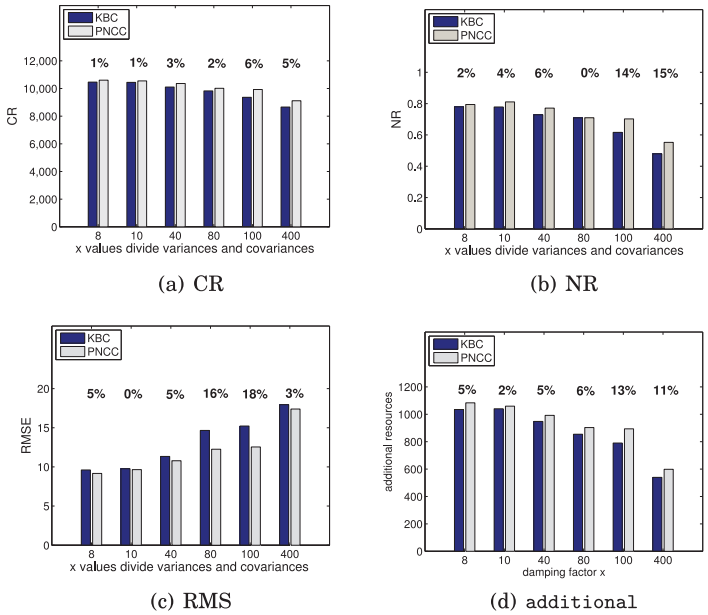
and 14(c)). Also, the performance of the server is very close to the target value since the mRT (Figure 14(d)) stays below 1s for most of the experiment duration.

Properties. We now examine the PNCC SASO properties using Figure 15 where we use the data from Figure 14(a). We apply the same methodology as in the KBC case using a hypothetical allocation line. The PNCC allocations should converge to the hypothetical line because the PNCC is an integral controller and has zero steady-state error. Figure 15 shows that it has short settling times despite being configured for smooth allocations (i.e., $\mathbf{Q_o}/400$) as it approaches steady-state values in just the first few intervals (i.e., 15). Finally, the PNCC does not overshoot as its allocations do not exceed the steady-state values (e.g., intervals $20 - 40$).

5.4. KBC and PNCC Comparison

This section compares the KBC and the PNCC controllers against sudden workload increases to stress controller allocations.

Figure 16 shows the performance differences between the two controllers for different Spike experiments and damping factor $x \in X$ values. The PNCC controller has equal or better performance over the KBC controllers when looking at all three metrics combined for each x value. As the x values increase, both controllers are slower to



(a) CR (b) NR

(c) RMS (d) `additional`

Fig. 16. KBC and PNCC comparison for `Spike` experiments and different x. Percentages represent the absolute metric difference of the PNCC controller over the KBC with a 90% CI after a t-test is performed. We report a 0% difference when, after applying the t-test over the datasets for the two controllers, the corresponding means show no difference at the 90% CI.

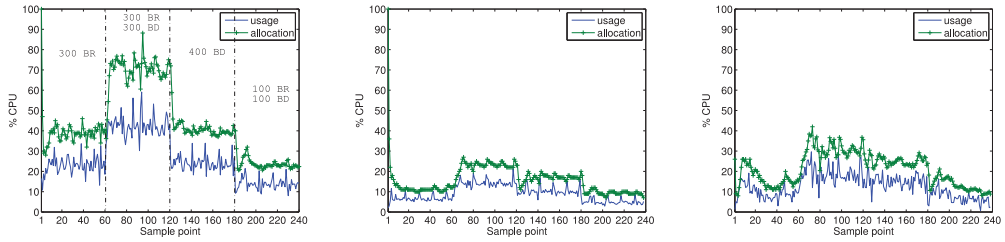
adjust their allocations. However, the performance improvement from the PNCC is increased as the x values increase. In these cases, where the small Kalman gains make the allocations slow to react to workload changes, the improvement of the PNCC over the KBC is more apparent. The PNCC is able to react faster to workload changes than the KBC controllers because it incorporates a combined error from all components.

5.5. APNCC

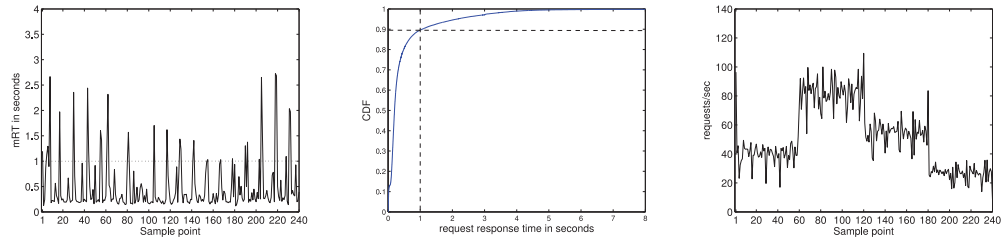
The APNCC controller estimates online the covariance matrix $\mathbf{Q}$ from utilization measurements and updates its values every few intervals to capture the workload dynamics as they happen.

We evaluate the APNCC with varying clients coming from either the browsing (BR) or the bidding (BD) mix, as shown in Figure 17(a). The covariance matrix $\mathbf{Q}$ is updated using a sliding window mechanism, which calculates the gains for every controller interval from measurements over the last 10 intervals with a slide of one interval, and the controller always uses $\mathbf{Q}/40$ values. Results are shown in Figure 17. The controller adjusts the allocations to track the utilizations of unknown and diverse workloads for the duration of the experiment and the performance of the server is very close to its target value. There are a few prolonged spikes when the workload increases, but 89.58% of the requests have response times $\leq$1s (Figure 17(e)). Finally, the `throughput` changes according to the number of clients with some fluctuations (Figure 17(f)).

Figure 18 illustrates the Kalman gain values as computed throughout the experiment and adapted to different utilization fluctuations. The adaptation mechanism captures the workload changes and the controller parameters are updated accordingly (e.g., the Tomcat gain increases around the interval 60 when the workload changes). While it is difficult for the offline parameter estimation to capture all the dynamics of the system,

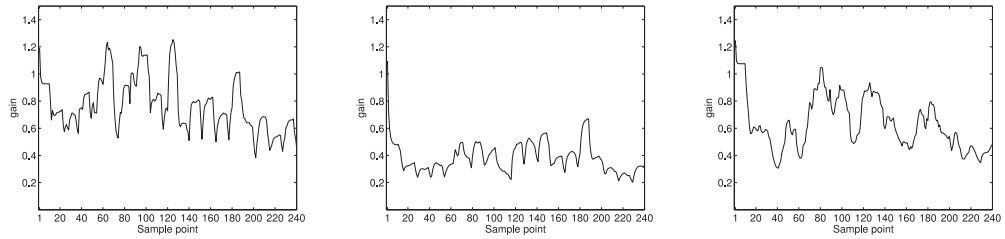


(a) Tomcat CPU resources (b) JBoss CPU resources (c) MySQL CPU resources



(d) Requests `mRT` (e) CDF of requests `mRT` (f) `throughput`

Fig. 17. APNCC performance for varying clients, two workload mixes, and $\mathbf{Q}/40$ values.



(a) Tomcat (b) JBoss (c) MySQL

Fig. 18. APNCC gains for varying clients, two workload mixes, and $\mathbf{Q}/40$ values.

the APNCC controller follows all utilization changes without *a priori* knowledge of the workload mixes or the number of clients.

5.6. PNCC and APNCC Comparison

To better evaluate the adaptive APNCC controller, this section compares this controller against the PNCC using both `Stable` and `Spike` experiments.

First, for each $x \in X$ and for each controller, we perform a `Stable(40,80,120)` experiment repeated five times. The covariance matrix $\mathbf{Q}$ is estimated every 10 intervals from the utilization measurements. Results are shown in Figure 19. According to all three metrics calculated for the duration of the experiment, the two controllers perform comparably. The APNCC controller performs equally well to the PNCC and it eliminates the need for any offline computations.

Second, we compare the two controllers using `Spike` experiments to simulate a large workload increase. $\mathbf{Q}$ is updated using a sliding window of 10 intervals and one interval slide. For every value $x \in X$, each experiment is performed 20 times. Results in Figure 20 show that the APNCC controller performs almost as well as the PNCC.

To shed light on the way the APNCC online mechanism works, Figure 21 depicts the APNCC Kalman gain values in the case of two different x values. Both figures also

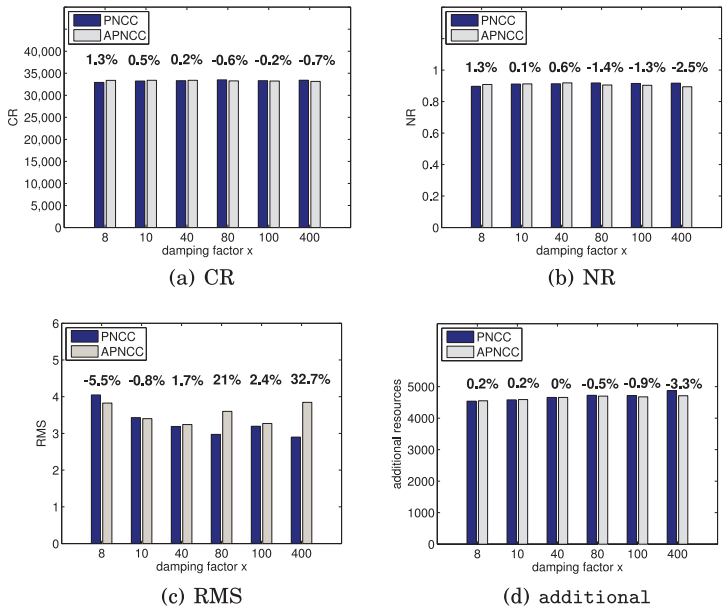


Fig. 19. PNCC and APNCC comparison for `Stable(40,80,120)` experiments. Percentages in each case show the metric difference of the APNCC controller over the PNCC with a 95% CI after a t-test is applied. A negative sign in the percentage indicates that the metric in the APNCC case has a lower value than in the case of the PNCC.

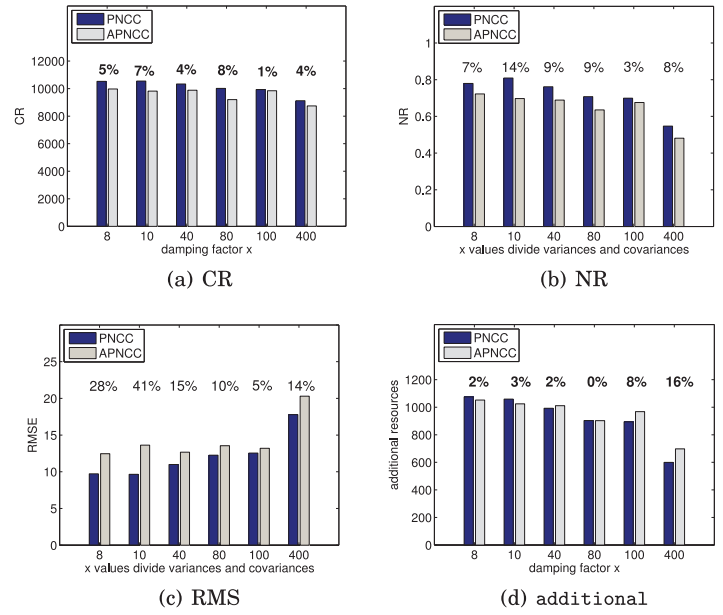
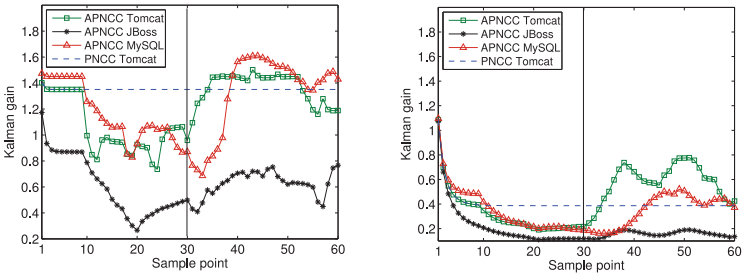


Fig. 20. PNCC and APNCC comparison for `Spike` experiments. Percentages in each case show the absolute metric difference of the APNCC controller over the PNCC with a 95% CI after a t-test is applied.



(a) Kalman gains, x=8 (b) Kalman gains, x=400

Fig. 21. APNCC and PNCC Kalman gains for $x = 8$ and $x = 400$.

show the offline computed gains of the PNCC controller for the Tomcat component only (dashed line).[3] Initially, the APNCC gain values are assigned to the offline computed ones, and after the first 10 intervals they are adapted for every interval. The figures clearly show that the absolute gain values are affected by the damping factor x. For example, for small x values, the gains (Figure 21(a)) are larger than for large x values (Figure 21(b)). However, their behavior with respect to detecting the workload increase is similar and explained later.

Initially, the APNCC gain values decrease from the offline ones computed for 600 clients to reflect the current 200 clients. When the workload increase at the 30^{th} interval, the adaptation mechanism detects the utilization change and all gain values are adapted from there on. The absolute gain values increase beyond the values of the offline computed values as the clients have now increased to 800.

The PNCC controller in the `Spike` experiments performs slightly better than the APNCC when comparing same x values (Figure 20). This is because when the workload change happens, the PNCC offline computed gain values are larger than the online APNCC estimated ones. This causes the PNCC to operate with a larger gain than the APNCC when the workload increases, at the 30^{th} interval. In this case, the PNCC controller reacts faster to the workload change for the next few intervals. However, although the APNCC gain is small when the workload increase starts, the controller detects the change and adapts its gain according to the new utilizations. At the end, its performance is only slightly worse than the PNCC.

This situation is one where the offline estimated gain values overestimate the real values as being captured by the online mechanism. For example, consider the opposite case where the offline gain values are smaller than the online ones before the workload increase occurs. This occurs when the APNCC is configured for $x = 8$ and the PNCC for $x = 400$ (APNCC gains are shown in Figure 21(a) and the Tomcat PNCC gain is shown in Figure 21(b) with the dashed line). In this case, the APNCC performs better than the PNCC as illustrated in Figure 20 (APNCC performance for $x = 8$ and PNCC for $x = 400$).

The APNCC does not need any *special* mechanism to detect or predict a large workload increase. This controller, which treats all workload changes the same, automatically adapts its gain to values that depict the importance or not of the workload change. Finally, when tuning the APNCC controller, the x damping factor values do not seem to be that important when it comes to workload increases. One can essentially choose any x value and the controller would still adjust its gain to capture a substantial workload increase.

[3]Similar observations hold for the other two components.

5.7. AKBC Comparison with a State-of-the-Art Feedback Controller

This section compares the AKBC controller against an existing approach that also proposes control theory to dynamically adjust the CPU allocation of virtualized applications. In particular, we compare against the single-component utilization nonlinear controller from Padala et al. [2007].

CPU Demand Simulator. For the comparison, we use a purpose-built simulator to vary the CPU utilization of a single virtualized application component and observe the CPU allocation enforced by a controller. The simulator enables us to experiment with different distributions of the CPU utilization signal—in the Rubis prototype deployment we observed that the CPU utilization signal follows only a normal distribution as shown in Section 3.1. In particular, we allow the CPU utilization to follow either a normal or a uniform distribution.

Furthermore, we vary the CPU utilization to follow a *s*awtooth pattern, where we change the utilization rapidly from large to small values in short intervals. In particular, every 20 intervals the CPU utilization doubles from 30% to 60% and then gradually drops to 30% in the next 10 intervals. The simulator enables us to produce a demanding workload with different utilization distributions.

The simulator model requests response times as captured in Section 3. In our model, response times depend on the relative CPU utilization of the allocation. In particular, we set three regions of operation: (a) when the CPU utilization is below 70%, the `mRT` stays well below 1s; (b) when the CPU utilization is above 70% and below 80%, the `mRT` increases but still remains below the 1s threshold; and finally, (c) when the utilization is above 80%, the `mRT` grows well above the 1s threshold and increases linearly to the demand.

Throughout the comparison, the AKBC controller parameters are set to $S = 1.0$ and $Q_0 = 209$. The initial variance Q_0 is measured offline when all resources are allocated to the virtualized single-tier application. Also, the reference c value is set to 60% as in the evaluation of the prototype Rubis deployment. The AKBC controller updates its Kalman gain online from CPU utilization measurements using a sliding window of 10 intervals with a slide of a single interval. All simulations run for 100 intervals in total and different executions showed similar results. Similar to the prototype evaluation, we use the `additional` and COV metrics to compare the CPU allocation of the different approaches. In addition, we report the `mRT` for the duration of the simulation.

Simulation Results. In their work, Padala et al. [2007] introduce an adaptive integral nonlinear utilization controller to assign the CPU allocation of virtualized applications based on a reference CPU utilization. The controller uses a self-tuning gain that depends on the resource utilization of the previous interval and the reference CPU utilization. The gain also has a tunable parameter λ to adjust its aggressiveness. The authors prove that their controller is stable when $\lambda < \frac{1}{c}$, where c is the reference relative CPU utilization and is defined as in our work. We compare against the AKBC controller since Padala et al. [2007] do not consider the resource coupling among components in multitier applications.

Table VII shows the comparison results between the AKBC and the Padala et al. [2007] controller when the CPU utilization noise variance is subject to a normal distribution and for different Kalman gain and λ values. We vary the tunable parameter λ in the region $[0.99\frac{1}{c}, \ldots, 0.01\frac{1}{c}]$ for a stable controller. Results show that although both controllers are set to keep the utilization at 60%, the AKBC controller on average maintains a lower `mRT` with fewer additional resources and a much lower allocation COV.

Depending on the parameter values for each controller, we observe different performance. In the AKBC controller, the higher the damping factor, the smaller the gain is

Table VII. KBC Comparison against Existing Feedback Controllers, CPU Utilization Variance Normally Distributed

AKBC				Padala et al. [2007]			
Damping factor	mRT	additional	COV	λ	mRT	additional	COV
1	0.45	2143	25	0.99	0.69	3391	789
4	0.46	2212	45	0.9	0.62	3362	668
8	0.44	2285	48	0.7	0.53	3309	466
10	0.41	2348	48	0.5	0.43	3271	309
40	0.16	3194	57	0.3	0.40	3290	178
80	0.08	3780	35	0.1	0.30	3465	41
100	0.07	3948	27	0.01	0.11	4584	7

Table VIII. KBC Comparison against Existing Feedback Controllers, CPU Utilization Variance Uniformly Distributed

AKBC				Padala et al. [2007]			
Damping factor	mRT	additional	COV	λ	mRT	additional	COV
1	0.47	2060	12	0.99	0.36	2593	346
4	0.46	2202	46	0.9	0.34	2593	346
8	0.44	2286	51	0.7	0.32	2567	274
10	0.41	2353	52	0.5	0.27	2574	215
40	0.16	3202	60	0.3	0.27	2638	147
80	0.08	3786	36	0.1	0.24	2891	38
100	0.07	3953	28	0.01	0.06	4257	11

and the controller is less aggressive with smoother and higher allocations and so there is better mRT. We observed that for high values of the damping factor ($\geq$100), there is a negligible improvement in the mRT and so we report results only up to $x = 80$. In the Padala et al. [2007] case, a lower λ significantly reduces the aggressiveness of the controller to react to CPU utilization changes and so additional resources increase. However, the AKBC self-adaptive controller is able to maintain a good performance of low mRT and additional resources and smooth allocation across the damping factor values.

Table VIII shows the comparison results when the process noise is uniformly distributed. Note that the Kalman filter is not optimal in this case. We note that the Padala et al. [2007] controller on average achieves a lower mRT at the expense of a much higher additional allocation and a very noisy CPU allocation as shown by the high COV values. In this case, the self-adaptive AKBC controller performs very well with lower additional allocations and COV values. This is achieved even though the utilization variance is uniform and the Kalman filter is not optimal. Overall, the AKBC controller achieves a stable performance despite the distribution of the utilization variance and the configuration of its parameters. Although both controllers are self-adaptive, the linear AKBC is more robust to its parameter values and the distribution utilization variances.

6. RELATED WORK

This section discusses related approaches for dynamic CPU resource allocation for single- and multitier virtualized applications. Our discussion is focussed on methods that use feedback control, profiling, and predictive control. We then review related work on filters developed for early, nonvirtualized shared clusters. We finish by briefly discussing the most relevant related work from the areas of machine learning and queueing theory and present approaches to capture virtualization-induced resource overheads.

Feedback Control. Early work in the area [Wang et al. 2005; Zhu et al. 2006] present feedback controllers to allocate CPU resources for the HP-UX PRM resource containers. These works propose (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 nonlinear controller that regulates the relative utilization, and (c) combinations of the two. Compared to these methods, which rely on application-specific models obtained via system identification, our Kalman-based controllers are linear and are based on a simple yet widely applicable model of the CPU utilizations.

With the virtualization of the data center, there has been a resurgence in control-based techniques for resource allocation. Padala et al. [2007] present a two-layered controller to regulate the relative utilization of two instances of two-tier virtualized Rubis servers colocated on two physical servers. The authors use the first layer controller from Wang et al. [2005] to regulate the relative utilization for each tier and a second layer controller to further adjust the allocations using a performance differentiation metric in cases of CPU contention.

Wang et al. [2007] present a three-layer nested control design to control the CPU of a three-tier Rubis application. The two inner loops are similar to Zhu et al. [2006]. 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 Kalman 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. [2007] address the problem of resource sharing in cases of contention in shared virtualized clusters. Their controller allocates CPU resources based on a quality-of-service 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. Padala et al. [2009] use a second-order ARMA model to capture online the relationship between multiple resource utilizations and application performance metrics. Experimental evaluation on two workload types shows that their model captures this relationship with high accuracy. The authors use this model to build resource allocation controllers based on past allocations and performance values. Our controllers emphasize the CPU utilization tracking approach and apply a widely used general model between application performance and allocation, which we validate experimentally.

Kalyvianaki et al. [2010] present a MIMO controller to control the CPU allocations of multitier web server applications that relies on a predefined linear-only resource coupling model, built *a priori* from offline measurements.

Profiling. Gong et al. [2010] propose a combined approach that uses either a state- or a signature-based pattern predictor for resource allocation using previous utilization measurements. The Kalman-based controllers rely on a few recent resource measurements to predict unseen workload patterns and are also optimized for multitier applications.

Nguyen et al. [2013] propose a wavelet-based approach for online demand prediction for resource utilizations. The advantage of this approach is the decomposition of the original signal into multiple detailed signals that capture different patterns and finally are synthesized to an approximation signal for predictions. The Kalman-based approach is also able to filter out temporarily variations and track the main fluctuations of the observed signal. To make predictions, the Kalman controllers simply require the previous measurement when compared to wavelet transforms that analyze a longer history of past measurements. Nguyen et al. [2013] also propose an online adaptive model that *learns* the performance violations as a function of the relative resource

utilization. However, this approach requires a long history of 10 to 20 minutes to generate a new model.

Shen et al. [2011] focus on methods for fast corrections against underprovisioning errors. They present an approach that combines online adaptive padding based on burst detection with additional allocation corrections using feedback from SLO violations and relative utilization. Our online Kalman gain adaptation captures the large utilization variance occurring during workload bursts and thus enables the controllers to allocate resources commensurate to utilization. Charalambous and Kalyvianaki [2010] propose the use of the min-max $\mathcal{H}_\infty$ filters to minimize the maximum error during underprovisioning. Their simulation-based evaluation uses a theoretical function to account for the request response degradation in saturation and shows improved performance over the Kalman filters. Our controllers can work in conjunction with the $\mathcal{H}_\infty$ filters to improve the performance during periods of saturation.

Predictive Control. Xu et al. [2006] present a predictive controller that regulates the relative utilization of a single-tier virtualized server based on three time-series prediction algorithms, namely, the AR auto-regressive model, the ANOVA decomposition, and the MP multipulse model. Results show that the predictive controllers outperform the relative utilization feedback controller of Wang et al. [2005] when the utilization exhibits regular patterns. However, unseen utilization patterns cause the predictive controller to fail to provide adequate resources for the server application. The Kalman filter is a very powerful method that predicts future demands based on past measurements and directly relates measurements to system states.

Filtering Methods. Simple time-series analysis techniques have also been used to predict future resource demands in modern virtualized shared clusters. A first-order AR predictor is used by the Sandpiper system to estimate future resource demands in virtualized servers [Wood et al. 2007].

Before virtualization became widely adopted, several systems were developed to manage resource multiplexing in shared clusters. We briefly discuss the approaches to CPU resource allocation related to filters. In Sharc [Urgaonkar and Shenoy 2004], 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 is not adaptive to different operating conditions.

To address these limitations, Chase et al. [2001] use a *flop-flip* filter based on similar filters from Kim and Noble [2001]. The flop-flip filter uses the moving average of the estimations over a 30s window and, if that estimation fails outside 1 standard deviation, it switches to the new moving average. The authors use this filter to smooth particularly bursty signals. The Kalman controllers adapt dynamically to operating conditions without using predefined values. Thus, they are easier to deploy and require minimal configuration.

Urgaonkar et al. [2002] 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 offline measurements and instead operates solely on short-term observations to make predictions of the workload utilizations.

Machine Learning. There exist machine-learning approaches to performance modeling in virtualized environments. Xu et al. [2007] use fuzzy modeling to minimize resource consumption for single-tier applications while meeting SLAs and maximizing a utility function modeled as profit over the shared resource revenue. Tesauro et al. [2007] apply reinforcement learning to data center server allocation. The authors

employ a two-layer resource management scheme. For each application at the first layer, an application manager provides a utility curve of its expected value based on the number of allocated servers. At the second layer, a resource arbiter decides how to allocate servers among all applications so as to maximize a global utility function. Kundu et al. [2012] apply different models across the input parameter space using neural networks and support vector machines.

Queueing Models. Another approach to performance modeling of virtualized applications involves the adoption of queueing models. Urgaonkar et al. [2008] combine proactive and reactive mechanisms to decide when to allocate resources and then use a queueing model to determine the resource demands of multitier applications. Menasce and Bennani [2006] study in a simulation environment a dynamic provisioning approach that combines a controller to allocate resources with analytical models of the CPU resources assigned to workloads with different priorities. Jung et al. [2008] use a hybrid approach with queueing models and optimization techniques for component placement in consolidated virtualized applications.

Virtualization-Induced Resource Overheads. There are several previous works that measure the performance overheads experienced by applications running on a virtualized environment [Cherkasova and Gardner 2005; Menon et al. 2005]. In the same scope, Wood et al. [2008] devise a modeling approach to capture the performance implications when migrating an application from a nonvirtualized environment to a virtualized one.

7. 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 observations of past utilization. This article has described how to transform the CPU allocation problem into a tracking one and how to incorporate the Kalman filter into feedback controllers for dynamically allocating the CPU resources of multitier virtualized servers. Our main experimental evaluation findings showed 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 can effectively configure its parameters online using past utilization observations to self-adapt to workload conditions without requiring online analysis.

In future work, we plan to evaluate our controllers against complex consolidation scenarios in modern multicore systems with the latest resource schedulers and capture VM interference with our system model. In addition, our goal is to extend our work to address the allocation problem of additional resources such as memory and network bandwidth. We plan to test our system model against other types of resources and workloads. Finally, a very challenging problem is to dynamically find the best value of the c parameter across applications and their components. As part of our current ongoing work, we are looking into optimization techniques to minimize the required additional headroom allocation and integrate this approach with the current Kalman controllers.

REFERENCES

Virgilio Almeida, Martin Arlitt, and Jerry Rolia. 2002. Analyzing a web-based system's performance measures at multiple time scales. *SIGMETRICS Performance Evaluation Review* 30, 2 (2002), 3–9.

Cristiana Amza, Anupam Chandra, Alan L. Cox, Sameh Elnikety, Romer Gil, Karthick Rajamani, Willy Zwaenepoel, Emmanuel Cecchet, and Julie Marguerite. 2002. Specification and implementation of dynamic web site benchmarks. In *Proceedings of the 5th Annual IEEE Workshop on Workload Characterization (WWW-5)*. 3–13.

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, New York, NY, 164–177.

Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. 2003. Performance comparison of middleware architectures for generating dynamic web content. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware'03)*. Springer-Verlag, New York, NY, 242–261.

Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. 2002. Performance and scalability of EJB applications. In *Proceedings of the 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA'02)*. ACM, New York, NY, 246–261. DOI:http://doi.acm.org/10.1145/582419.582443

Themistoklis Charalambous and Evangelia Kalyvianaki. 2010. A min-max framework for CPU resource provisioning in virtualized servers using $\mathcal{H}_\infty$ filters. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC'10)*. 3778–3783.

Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*. ACM, New York, NY, 103–116. DOI:http://dx.doi.org/10.1145/502034.502045

Mike Y. Chen, Anthony Accardi, Emre Kiciman, Jim Lloyd, Dave Patterson, Armando Fox, and Eric Brewer. 2004. Path-based faliure and evolution management. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*. USENIX Association, Berkeley, CA, 309–322.

Ludmila Cherkasova and Rob Gardner. 2005. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In *Proceedings of the Annual USENIX Technical Conference (USENIX'05)*. USENIX Association, Berkeley, CA, 387–390.

Eduardo F. Costa and Alessandro Astolfi. 2008. On the stability of the recursive Kalman filter for linear time-invariant systems. In *Proceedings of the American Control Conference (CDC'08)*. 1286–1291.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI'04)*. ACM, New York, NY, 137–150.

Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (CNSM'10)*. IEEE Computer Society, Washington, DC, 9–16. DOI:http://dx.doi.org/10.1109/CNSM.2010.5691343

Gueyoung Jung, Kaustubh R. Joshi, Matti A. Hiltunen, Richard D. Schlichting, and Calton Pu. 2008. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proceedings of the 2008 International Conference on Autonomic Computing (ICAC'08)*. IEEE Computer Society, Washington, DC, 23–32. DOI:http://dx.doi.org/10.1109/ICAC.2008.21

Rudolph E. Kalman. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering* 82, Series D (1960), 35–45.

Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2009. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing (ICAC'09)*. ACM, New York, NY, 117–126.

Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2010. Resource provisioning for multi-tier virtualized server applications. *Computer Measurement Group Journal (CMG), Spring Issue* 126 (2010), 6–17.

Minkyong Kim and Brian Noble. 2001. Mobile network estimation. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom'01)*. ACM, New York, NY, 298–309. DOI:http://dx.doi.org/10.1145/381677.381705

Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. 2012. Modeling virtualized applications using machine learning techniques. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE'12)*. ACM Press, New York, 3–12. DOI:http://dx.doi.org/10.1145/2151024.2151028

Xue Liu, Xiaoyun Zhu, P. Padala, Zhikui Wang, and S. Singhal. 2007. Optimal multivariate control for differentiated services on a shared hosting platform. In *Proceedings of the 46th IEEE Conference on Decision and Control*. IEEE Computer Society, Washington, DC, 3792–3799.

Peter S. Maybeck. 1979. *Stochastic Models, Estimation, and Control*. Mathematics in Science and Engineering, Vol. 141. Academic Press, New York.

Daniel A. Menasce and Mohamed N. Bennani. 2006. Autonomic virtualized environments. In *Proceedings of the International Conference on Autonomic and Autonomous Systems (ICAS'06)*. IEEE Computer Society, Washington, DC, 28. DOI:http://dx.doi.org/10.1109/ICAS.2006.13

Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel. 2005. Diagnosing performance overheads in the xen virtual machine environment. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments (VEE'05)*. ACM Press, New York, NY, 13–23. DOI:http://dx.doi.org/10.1145/1064979.1064984

Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. 2013. AGILE: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC'13)*. 69–82.

Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. 2009. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European Conference on Computer systems (EuroSys'09)*. ACM, New York, NY, 13–26. DOI:http://dx.doi.org/10.1145/1519065.1519068

Pradeep Padala, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. 2007. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys'07)*. ACM, New York, NY, 289–302. DOI:http://dx.doi.org/10.1145/1272996.1273026

Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. CloudScale: Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*. ACM, New York, NY. DOI:http://dx.doi.org/10.1145/2038916.2038921

Dan Simon. 2006. *Optimal State Estimation*. John Wiley & Sons, Inc.

Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. 2007. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing* 10, 3 (2007), 287–299.

Bhuvan Urgaonkar and Prashant Shenoy. 2004. Sharc: Managing CPU and network bandwidth in shared clusters. *IEEE Transactions Parallel Distributed Systems (TPDS'04)* 15, 1 (Jan. 2004), 2–17. DOI:http://dx.doi.org/10.1109/TPDS.2004.1264781

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. 2008. Agile dynamic provisioning of multi-tier internet applications. *Transactions on Autonomic and Adaptive Systems (TAAS'08)* 3, 1 (2008), 1:1–1:39.

Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. 2002. Resource overbooking and application profiling in shared hosting platforms. *SIGOPS Operating Systems Review* 36, SI (Dec. 2002), 239–254. DOI:http://dx.doi.org/10.1145/844128.844151

Zhikui Wang, Xue Liu, Alex Zhang, Christopher Stewart, Xiaoyun Zhu, Terence Kelly, and Sharad Singhal. 2007. AutoParam: Automated control of application-level performance in virtualized server environments. In *Proceedings of the 2nd International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'07)*.

Zhikui Wang, Xiaoyun Zhu, and Sharad Singhal. 2005. Utilization and SLO-Based control for dynamic sizing of resource partitions. In *Proceedings of the 16th IFIP/IEEE Ambient Networks International Conference on Distributed Systems: Operations and Management (DSOM'05)*. Springer-Verlag, Berlin, 133–144. DOI:http://dx.doi.org/10.1007/11568285_12

Greg Welch and Gary Bishop. 1995. *An Introduction to the Kalman Filter*. Technical Report 95-041. University of North Carolina at Chapel Hill, Department of Computer Science.

Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. 2008. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*. Springer-Verlag, New York, NY, 366–387.

Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2007. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation (NSDI'07)*. USENIX Association, Berkeley, CA, 229–242.

Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, and Mazin Yousif. 2007. On the use of fuzzy modeling in virtualized data center management. In *Proceedings of the International Conference on Autonomic Computing (ICAC'07)*. IEEE Computer Society, Washington, DC, 25.

Wei Xu, Xiaoyun Zhu, Sharad Singhal, and Zhikui Wang. 2006. Predictive control for dynamic resource allocation in enterprise data centers. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'06)*. 115–126.

Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. 2007. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the 4th International Conference on*

Autonomic Computing (ICAC'07). IEEE Computer Society, Washington, DC, 27. DOI:http://dx.doi.org/10.1109/ICAC.2007.1

Tao Zheng, Jinmei Yang, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. 2005. Tracking time-varying parameters in software systems with extended Kalman filters. In *Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research (CASCON'05)*. IBM Press, 334–345.

Xiaoyun Zhu, Zhikui Wang, and Sharad Singhal. 2006. Utility-driven workload management using nested control design. In *Proceedings of the American Control Conference (ACC'06)*. 6.

Received August 2013; revised March 2014; accepted April 2014