Charalambous, T. & Kalyvianaki, E. (2010). A min-max framework for CPU resource provisioning in virtualized servers using $\mathcal{H}\infty$ Filters. Decision and Control (CDC), 2010 49th IEEE Conference on, pp. 3778-3783. doi: 10.1109/CDC.2010.5717375

A Min-Max Framework for CPU Resource Provisioning in Virtualized Servers using $\mathcal{H}_\infty$ Filters

Themistoklis Charalambous and Evangelia Kalyvianaki

Abstract— **Dynamic resource provisioning for virtualized server applications is integral to achieve efficient cloud and green computing. In server applications unpredicted workload changes occur frequently. Resource adaptation of the virtual hosts should dynamically scale to the updated demands (cloud computing) as well as co-locate applications to save on energy consumption (green computing). Most importantly, resource transitions during workload surges should occur while minimizing the expected loss due to mismatches of the resource predictions and actual workload demands. Our approach is to minimize the maximum expected loss using the same techniques as in two-person zero-sum games. We develop an $\mathcal{H}_\infty$ filter that minimizes the worst-case estimation and allocate resources fast. Through simulations our $\mathcal{H}_\infty$ filter demonstrates its effectiveness and good performance when compared against Kalman-based controllers.**

I. Introduction

Virtualization of data centers has given rise to important paradigms namely cloud and green computing. Cloud computing provides an execution platform of essential means (i.e. computing resources and software components) that applications can use on demand. Applications might include for example web and database servers which can be hosted within one machine or span across machines. In addition, green computing utilizes machines power in ways to reduce their energy consumption as well as power and cooling expenses.

Modern virtualization (e.g. [1]) is one of the key contributing factors to both paradigms. When virtualized, a machine is transformed into one or more virtual execution environments, called *virtual machines (VMs)* where applications can run in isolation and share the machine resources by runtime resource allocation. In cloud computing, virtualization is used to dynamically create VMs according to the application demands and online adjust their resource allocations to match their workload needs. Furthermore, virtualization is also used to reduce the required machines to host a certain number of server applications. This is achieved by application consolidation to a smaller group of hosting machines where unused machines are switched off for reduced energy consumption.

To achieve efficient virtualization it is essential to ensure performance guarantees for each co-located application and provide them with resources according to their demands and meet their Service Level Objectives (SLOs). However, applications exhibit highly changing workload demands which cause difficult to predict resource fluctuations [2], [3]. There is the need for *autonomic resource management* methods that adaptively allocate resources across virtualized applications with diverse workload and internal structure characteristics.

Autonomic resource management in a virtualized environment using control-based techniques has recently gained significant attention. The most widely-used approach to control the application performance is by controlling its CPU utilization within the VM, for example [4], [5]. This approach is further extended to dynamically distribute resources among co-located applications under conditions of contention [4], [6]. Finally, special attention has been given to model the resource coupling in multi-tier virtualized applications to provide timely allocations during workload changes [5], [7]. A key performance metric for these controllers is their responsiveness to sudden workload changes. Their parameters can be tuned to achieve transition phases of smaller duration, however, they do not provide any sort of control over the maximum error of the application performance.

The maximum error occurs under conditions of contention. In a virtualized application—resources are constantly updated to match the workload demands while freeing up resources for other applications—it is anticipated that the application exhibits short but very frequent periods of resource contention. It is thus important to optimize performance to recover after these periods. To the best of our knowledge, this is the first approach to control the maximum error of the performance of a virtualized application in conditions of contention.

This paper treats the problem as a game against nature and minimizes the maximum expected loss using the same techniques as in the two-person zero-sum games [8]. It presents a new discrete-time controller based on the $\mathcal{H}_\infty$ filter, which minimizes the worst-case estimation error (min-max). The $\mathcal{H}_\infty$ controller allocates CPU resources in virtualized applications and minimizes the maximum error in their performance as measured by the requests mean response time (*mRT*). Simulation results show that the $\mathcal{H}_\infty$ controller lowers mRT during saturation when compared against a Kalman controller.

In Section II we provide further motivation and discuss related work. In Section III we provide the notation used throughout this paper. Section IV describes the models adopted for the resource utilization and the *mRT*. In Section V, the $\mathcal{H}_\infty$ filter is designed, while in Section VI, the performance of the $\mathcal{H}_\infty$ filter is evaluated. Finally, we conclude in Section VII.

Themistoklis Charalambous is at the Electrical and Computer Engineering Department, University of Cyprus, Cyprus and Evagelia Kalyvianaki is at the Department of Computing, Imperial College London, UK `themis@ucy.ac.cy, ekalyv@doc.ic.ac.uk`

II. Background

A. Motivation

Adequate provisioning for VMs' resources is crucial for a high performance data center. Consider a server consolidation example with two single-component applications hosted on a single physical server machine. Assume that each application has known workload requirements and the sum of resources from both applications does not exceed the total available physical resources for the server machine. The left diagram in Figure 1 illustrates two VMs, each one hosting an application with resources allocated as required. In this way, both applications are served adequately and the total resource utilization of the physical machine is now increased simply by co-locating two running servers.

Consider now the case where the workload in both applications changes, (middle diagram in Figure 1). In VM A it increases, therefore more resources are required, while in VM B it decreases so fewer resources are needed. In the case of VM A, the under-provisioning results in performance degradation, since the application does not have enough resources to serve its incoming requests. In the case of VM B, the over-provisioning does not affect the running application within the VM B. However it does reduce the free available resources for a third VM to be placed on the same machine. Therefore, in both cases, the resource allocation needs to adapt to the new resource demands (right most diagram in Figure 1).

This paper concentrates on the dynamic case of the resource adaptation while workload demands change. Our work makes use of modern virtualization platforms which export a user-level interface to bound the maximum resource allocation per VM at runtime.

B. Related Work

Autonomic resource management aims to adjust resource allocations across applications to meet their SLOs as measured by the application response times. In [9] and [10], the authors directly control application response times through runtime resource CPU allocation using an offline system identification analysis to model the relationship between the response times and the CPU allocations in regions where it is measured to be linear. However, as this relationship is application-specific and relies on offline identification performance models, other approaches, such as [11] and [10], control the response times in combination with the application CPU utilization.

The application performance can be controlled, by controlling is CPU utilization. As long as the utilization remains below the allocation by a certain threshold the application response times stay low [12]. Furthermore, when the utilization approaches the allocation, the response times increase dramatically and the application performance drops. Padala *et al.* [4] present a two-layer non-linear controller to regulate the utilization of the virtualized components of multi-tier applications. Kalyvianaki *et al.* [5] formulate the regulation problem as a CPU utilization tracking one and present

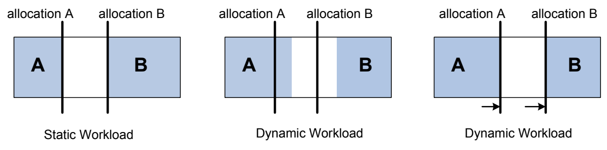


Fig. 1. Resource management example in virtualized applications. Shaded rectangles show resource utilizations and solid lines indicate allocated resources.

adaptive Kalman-based controllers to track and maintain the CPU utilization to a user-defined threshold. However, the Kalman filter provides an optimal estimate if the model and the noise statistics are known. Otherwise, it may perform poorly if there are errors in the system model or the assumed noise statistics.

In addition, [4], [6] control the resource allocations across consolidated virtualized applications under conditions of contention. When applications demand more resources than physically available the above controllers distribute resources among them in ways to respect their user-given priorities. Finally, in [7] and [5] Multi-Input-Multi-Output (MIMO) feedback controllers are presented. These controllers make global decisions by coupling the resource usage of all components of multi-tier server applications.

A key performance metric for controllers used for virtualized servers is their responsiveness to sudden workload changes. [5], [6] study the performance of their controllers across their parameters against significant resource fluctuations until the controllers stabilize to the new demands. The parameters of the controllers in [5] can be tuned to achieve transition phases of smaller duration, however, they do not provide any sort of control over the maximum error of the application performance. The $\mathcal{H}_\infty$ controller presented here is designed to minimize the max error caused by saturation periods.

III. Notation

Vectors are denoted by small, bold letters whereas matrices are denoted by capital letters. A^T and A^{-1} denote the transpose and inverse of matrix A respectively. For two symmetric matrices A and B, $A \succ B$ means that $A - B$ is positive definite and $A \succeq B$ means that $A - B$ is semi-positive definite. By I we denote the identity matrix. $\hat{\mathbf{a}}_k$ denotes the estimate of random vector $\mathbf{a}_k$ for time instant k. P_k denotes the matrix P at time instant k. The norm of a vector or a matrix is given by $\|\cdot\|$. $\mathbf{a} \in \mathbb{R}_+^{N\times 1}$ represents a vector with N nonnegative real entries and $A \in \mathbb{R}_+^{N\times N}$ represents a nonnegative matrix, i.e. all entries in the matrix are nonnegative.

IV. System Model

A. mRT model

One of the most widely used metrics for measuring server performance is the client mean request response times (*mRT*). There are well known formulas to calculate the *mRT* of server

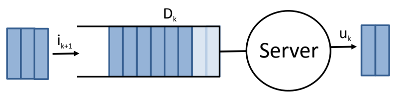


Fig. 2. Model of the demand D in a server.

requests. For instance, in the case of a single server M/M/1, this is given by: $mRT = s/(1-u)$ (Little's law), where s is the mean service time and u the mean utilization. This model predicts that when the utilization reaches saturation the *mRT* goes to infinity. This is the case where the server has no more resources to serve new requests, and hence these are kept in the input queues and their waiting times grow indefinitely. However, in a virtualized environment when the server is saturated, we can dynamically increase its resource allocations and therefore provide more resources to serve new incoming requests and those already waiting in the queue. In this way, as long as the server is saturated the requests *mRT* will grow, however, it will drop when the server exits saturation and has enough resources to server all requests. This section provides a function of the *mRT* which models the above characteristics. We do not aim to provide an accurate *mRT* prediction model, rather, we use this model as the cost function to evaluate the performance of the $\mathcal{H}_\infty$ controller in a simulated environment. This model is solely based on well known characteristics of server applications described below.

It is very difficult to predict the exact values of the *mRT* of server applications across operating regions and different applications and workloads. However, it is known to have certain characteristics [12]. Generally, its values can be divided into three regions: (a) when the application is provisioned with abundant resources all requests are served as they arrive and the response times are kept low; (b) when the utilization approaches 100% (e.g. around 70-80% on average) the *mRT* increases above the low values from the previous region because there are instances at which the requests peak and approach 90-100%; (c) however, when resources are scarce and very close to 100%, requests compete for limited resources, they wait in the input queues and their response times increase dramatically to relatively high values.

It is often the case, for instance in data centers that to maintain good server performance the operators aim to keep machine CPU utilization below 100% of the machine capacity by a certain value, which is usually called *headroom*. Headroom values denote the boundary between the second and the third *mRT* regions. At these values the server is well provisioned and response times are kept low. If the utilization approaches 100% due to increased workload demands, operators increase the server resources.

In order to assimilate these characteristics into the system we propose the following *mRT* model defined in (1), where the *mRT* is given as a function of the ratio between workload

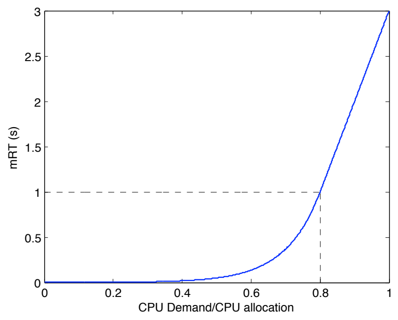


Fig. 3. *mRT* model with respect to CPU usage, where $\gamma = 10$ and $\phi = 0.8$.

demand D and allocation a:

$$mRT(D/a) = \begin{cases} e^{\gamma(\frac{D}{a}-\phi)}, & \text{if } \frac{D}{a} \le 1 \\ 1+\gamma(\frac{D}{a}-\phi), & \text{if } \frac{D}{a} > 1 \end{cases} \tag{1}$$

First, we define the demand D, in terms of CPU, at time instant k. This is given by: $D_{k+1} = D_k - u_k + i_{k+1}$, where u_k is the CPU utilization at time instant k and i_{k+1} is the CPU from the requests at time instant $k+1$. A graphical illustration of the demand model in a server is given in Figure 2. We then include the headroom value ϕ and we also use the constant γ to assign some large response time values in regions close to saturation.

Figure 3 graphically illustrates the response time model when $\gamma = 10$ and $\phi = 0.8$. It is easy to distinguish the three regions aforementioned: (a) When $D/a < 0.7$, the *mRT* is much lower than 1 second (s). For the rest of this paper we will use the 1s threshold to denote a timely completed request; (b) when $0.7 \le D/a \le 0.8$, then the *mRT* increases, but since there are still resources it remains below the threshold; (c) when $D/a > 0.8$ we assume that some requests remain in the input queue due to the fluctuations in demand and hence, the *mRT* is varying linearly with the demand. Note that $D/a > 1$ means that the maximum amount of resources has been allocated and queues are growing in the input, making the demand even bigger, thus increasing the *mRT*.

For the rest of this paper, we will use (1) along with the Root-Mean-Square-Error (RMSE) of the allocation to evaluate the performance of the virtualized application when its CPU allocation is controlled by our $\mathcal{H}_\infty$ filter and the other filters we use for comparison. Without loss of generality this enables us to evaluate the performance of the $\mathcal{H}_\infty$ filter across operating regions.

B. The CPU Usage and Allocation

The purpose of the $\mathcal{H}_\infty$ controller is to control the allocation of the VMs running a server application while observing its utilization across VMs. We assume multi-tier server applications composed of N components, where each component runs on a different VM. We start by modeling the time-varying CPU utilization per component as a random walk given by the following linear stochastic difference equation:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{r}_k, \tag{2}$$

where $\mathbf{x}_k \in \mathbb{R}_+^{N\times 1}$ is the vector of the percentages of the total CPU capacity actually used by the application components during interval k; each row corresponds to a component. The independent random vector $\mathbf{r}_k \in \mathbb{R}_+^{N\times 1}$ represents the process noise. The process noise models the utilization between successive intervals caused by workload changes, e.g. requests being added, doing work from previous intervals, or leaving the server.

We denote $\mathbf{a}_k \in \mathbb{R}_+^{N\times 1}$ as the CPU capacity of a physical machine allocated to the VMs; each row corresponds to a component. $\mathbf{a}_k$ shows the maximum amount of resources a VM can use. We denote $\mathbf{u}_k \in \mathbb{R}_+^{N\times 1}$ as the total CPU utilization actually *observed* in the VMs; again each row corresponds to an application tier. $\mathbf{u}_k$ models the observed application utilization $\mathbf{x}_k$ in addition to any usage noise coming from other sources, such as the operating system, to support the application.

The purpose of the $\mathcal{H}_\infty$ controller is to maintain good server performance in the presence of workload changes. This is achieved by adjusting the allocation to values above the utilization. For each time-interval k the desired relationship between the two quantities is given by:

$$\mathbf{u}_k = C\mathbf{a}_k + \mathbf{v}_k, \tag{3}$$

where $C \in \mathbb{R}_+^{N\times N}$ is a diagonal matrix with the target value c_i for each component i along the diagonal, and denotes the gap between the allocation and the utilization; $\mathbf{v}_k \in \mathbb{R}_+^{N\times 1}$ denotes the utilization measurement noise at each component. To maintain good server performance, the allocation should follow the utilization and therefore is also modeled as a random walk. The allocation for the next time-interval $(k+1)$ is given by:

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \mathbf{w}_k, \tag{4}$$

where $\mathbf{w}_k \in \mathbb{R}_+^{N\times 1}$ denotes the process noise of the allocation signal.

V. The $\mathcal{H}_\infty$ Controller

We formulate the allocation problem as a state estimation problem. Kalman filters [13] are commonly used to estimate the states of a dynamic system and they have also been used for the allocation problem [5]. Their attractiveness relies on the fact that the Kalman filter is the optimal linear filter when minimizing at each time step the two-norm of the expected values of the estimation error. However, they do not provide any guarantees in terms of limiting the maximum estimation error. Alternatively, $\mathcal{H}_\infty$ filters minimize the worst-case estimation error—they are called *minimax* filters—and can be used to incorporate more robustness into the state estimation problem.

The cost function for our problem formulation is given by:

$$J = \frac{\sum_{k=0}^{N-1} \|\mathbf{a}_k - \hat{\mathbf{a}}_k\|_2^2}{\|\mathbf{a}_0 - \hat{\mathbf{a}}_0\|_{P_0^{-1}}^2 + \sum_{k=0}^{N-1}\left(\|\mathbf{w}_k\|_{Q_k^{-1}}^2 + \|\mathbf{v}_k\|_{R_k^{-1}}^2\right)} \tag{5}$$

where $P_0 \in \mathbb{R}^{N\times N}$, $Q_k \in \mathbb{R}^{N\times N}$ and $R_k \in \mathbb{R}^{N\times N}$ are symmetric, positive definite matrices defined by the problem specifications, i.e. P_0 is the initial error covariance matrix, Q_k and R_k are the process and measurement covariance matrices for time interval k, respectively; $\hat{\mathbf{a}}_k$ is the estimate of the CPU allocation. The direct minimization of J in (5) is not tractable, and as a result we choose a performance bound and our controller is designed based on that threshold. In our problem, the target is to keep the *mRT* below a certain threshold (e.g. less than a second). Therefore, our controllers are designed based on the fact that $J < 1/\theta$, where θ is specified such that the desired *mRT* is less than a certain user-specified threshold. Considering (5), the steady-state $\mathcal{H}_\infty$ filter bounds the following cost function:

$$J = \lim_{N\to\infty} \frac{\sum_{k=0}^{N-1} \|\mathbf{a}_k - \hat{\mathbf{a}}_k\|_2^2}{\sum_{k=0}^{N-1}\left(\|\mathbf{w}_k\|_{Q_k^{-1}}^2 + \|\mathbf{v}_k\|_{R_k^{-1}}^2\right)}. \tag{6}$$

Let $G_{\hat{\mathbf{a}}\mathbf{e}}$ be the system that has $\mathbf{e} = [\mathbf{w}\ \mathbf{v}]^T$ as its input and $\hat{\mathbf{a}}$ as its output. Since the $\mathcal{H}_\infty$ filter makes the cost (6) less than $1/\theta$ for all $\mathbf{w}_k$ and $\mathbf{v}_k$, then according to [14]:

$$\|G_{\hat{\mathbf{a}}\mathbf{e}}\|_\infty^2 = \sup_\zeta \frac{\|\mathbf{a} - \hat{\mathbf{a}}\|_2^2}{\|\mathbf{w}\|_{Q^{-1}}^2 + \|\mathbf{v}\|_{R^{-1}}^2} \le \frac{1}{\theta}, \tag{7}$$

where ζ is the phase of $\|\mathbf{w}\|_{Q^{-1}}^2 + \|\mathbf{v}\|_{R^{-1}}^2$ comprised by the sampling time of the system and the frequency of the signals. Since we want the *mRT* to be less than a certain value (usually around 1 second), we have to keep the CPU usage to less than a threshold set by our mRT model. Therefore, using (7) we want:

$$\sup_\zeta \frac{\|\Phi - C\|_2^2}{\|\mathbf{w}\|_{Q^{-1}}^2 + \|\mathbf{v}\|_{R^{-1}}^2} \le \frac{1}{\theta}, \tag{8}$$

which is equivalent to:

$$\theta \le \inf_\zeta \frac{\|\mathbf{w}\|_{Q^{-1}}^2 + \|\mathbf{v}\|_{R^{-1}}^2}{\|\Phi - C\|_2^2}. \tag{9}$$

where Φ and C are diagonal matrices with the headroom values ϕ_i and target values c_i for each component, respectively, along the diagonal. Inequality (9) suggests that a higher value of θ can be accommodated when the system is very noisy or the CPU usage u is very closed to the headroom value ϕ. Note, however, that the necessary condition to ensure that P_k remains positive definite and the system retains stability for the above $\mathcal{H}_\infty$ filter is that:

$$I - \theta P_k + C^T R_k^{-1} C P_k \succ 0. \tag{10}$$

To design the controller we consider inequalities (9) and (10).

Note that the Kalman filter gain is smaller than the $\mathcal{H}_\infty$ filter gain for $\theta > 0$, meaning that the $\mathcal{H}_\infty$ filter relies more on the measurement and less on the system model. As θ goes closer to zero, the $\mathcal{H}_\infty$ filter gain goes closer to the Kalman filter gain [14].

For the cost function (5), the $\mathcal{H}_\infty$ filter is thus given by:

$$K_k = P_k[I - \theta P_k + C^T R_k^{-1} C P_k]^{-1} C^T R_k^{-1} \tag{11}$$

$$\hat{\mathbf{a}}_{k+1} = \hat{\mathbf{a}}_k + K_k(\mathbf{u}_k - C\hat{\mathbf{a}}_k) \tag{12}$$

$$P_{k+1} = P_k[I - \theta P_k + C^T R_k^{-1} C P_k]^{-1} + Q_k \tag{13}$$

where K_k is the gain matrix and P_k is the error covariance matrix and it is positive definite (since P_0 is positive definite and if P_k is positive definite, then from (13) positive definiteness is preserved in P_{k+1}).

VI. Performance Evaluation

In this section, we evaluate the performance of the $\mathcal{H}_\infty$ filter using a simulated virtualized environment which we have built using MATLAB. For simplicity, a Single-Input-Single-Output (SISO) system is used, in order to highlight the characteristics of our controller. For the current evaluation we measure the performance of the controller around the *mRT* using (1) and the RMSE of the allocation. We compare our controller against the conceptually similar Kalman controller from [5]. Both controllers use the same utilization model, however, the Kalman controller aims to minimize the mean prediction error, while our controller minimizes the maximum error.

We evaluate the $\mathcal{H}_\infty$ controller across two workload conditions. First, we simulate gradual workload variations to decreasing and increasing demand. Results are shown in Section VI-A. Second, we simulate a flash crowd, where the workload demand repeatedly peaks for a very short time following a saw-tooth pattern. Results are shown in Section VI-B. In both cases, we study the performance of the $\mathcal{H}_\infty$ controller when the process $\mathbf{w}_k$ and the measurement noise $\mathbf{v}_k$ are either normally or uniformly distributed.

Finally, the parameters used in the current evaluation are the following: c is set to 0.95 which is above the headroom value $\phi = 0.8$. This makes our system to constantly operate near conditions of contention, in order to better observe the $\mathcal{H}_\infty$ filter performance. θ is set to the high value of 0.7 because we set the system to be noisy, which is depicted by the value of Q that we set to 4. Different values for the rest of the parameters showed similar results and therefore they are not presented here.

A. Gradual workload changes

The performance of the $\mathcal{H}_\infty$ controller is measured when the utilization exhibits gradual changes towards decreasing from an initial 60% to 30% and then increasing again to 60%. We study the controller allocations when the process (w_k) and measurement (v_k) noises are taken to be normally distributed in Figure 4 and when these noises are uniformly distributed in Figure 5. Numerical results for the *mRT* and the RMSE of the allocation for the duration of the experiments are shown in Table I. The RMSE measures the error between the observed allocation and the predicted allocation, given the utilization for each interval and the headroom value c. It measures the variation of the controller's allocations with respect to the reference value c.

Figures 4(a) and 5(a) illustrate the allocations of the $\mathcal{H}_\infty$ controller as the utilization demand varies. The same figures also show the allocations of the Kalman controller for comparison purposes. Although both controllers adjust their allocations to match the workload demands, the $\mathcal{H}_\infty$ controller allocates resources faster during conditions of contention than the Kalman controller. This is better shown for the intervals 50 to 80 where the workload demand increases gradually and saturation here causes the *mRT* to jump to high values as shown in Figures 4(b) and 5(b).

The difference between the $\mathcal{H}_\infty$ and the Kalman controller is better shown in the case where the noises are normally distributed. In this case, the server is saturated for many intervals (i.e. intervals 50-60) and the *mRT* increases a lot. However, as soon as the demand starts to stabilize the server is able to serve new incoming requests and those already left in its input queue. The $\mathcal{H}_\infty$ controller manages to serve all requests faster than the Kalman controller as shown by the lower maximum *mRT*. The Kalman controller also serves all requests, but its *mRT* increases to higher values. Overall, the $\mathcal{H}_\infty$ controller achieves better performance for the duration of the experiment as also shown in Table I.

When the process and measurement noise are uniformly distributed the workload utilizations vary less than in the case of the normally distributed noises. When the noise is uniform, the $\mathcal{H}_\infty$ controller also recovers faster after a period of contention; the *mRT* of the $\mathcal{H}_\infty$ controller is lower than the Kalman *mRT* around the intervals $50-60$ in Figure 5(b). However, its overall performance for the duration of the experiment is very close to the Kalman performance, Table I.

The most important aspect of the $\mathcal{H}_\infty$ filter that differentiates it from other controllers is that is minimizes the maximum error. When the noise is normal, it operates very well when compared against the Kalman controller which is designed under this assumption. In addition, the $\mathcal{H}_\infty$ filter performs better near the maximum error for uniformly distributed noise.

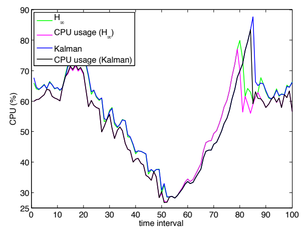
B. Saw-tooth demand for CPU usage

In this case we vary the utilization in a saw-tooth structure. This is a very demanding workload, where the utilization changes rapidly from very large to very small values. In this case it is very important for the controller to adapt the allocations in a timely fashion. To achieve overall good performance the error during contention should be minimized.

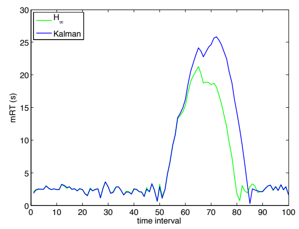
Figures 7 and 6 illustrate the performance of the $\mathcal{H}_\infty$ controller against the Kalman filter in cases where the process (w_k) and measurement (v_k) noises are taken to be normally and uniformly distributed, respectively. Numerical results for the duration of the simulations, given in Table II, show that the $\mathcal{H}_\infty$ controller keeps the *mRT* in both cases to lower values than the Kalman controller. The $\mathcal{H}_\infty$ controller allocates resources faster during periods of contentions as also shown by its increased utilization when the workload demand increases in Figure 7(a) and 6(a).

VII. Conclusions

We have used a minimax framework and developed an $\mathcal{H}_\infty$ filter for CPU resource provisioning in virtualized servers. As virtualization technologies enable the runtime CPU allocation, it is important to build controller that adjust the allocation in a timely fashion and avoid resource saturation. To this end, we adopt a minimax approach that minimizes

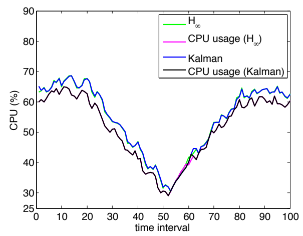


(a) CPU allocations and utilizations

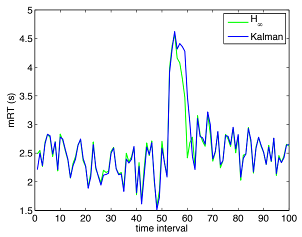


(b) *mRT*

Fig. 4. Gradual Workload Changes: $\mathcal{H}_\infty$ controller performance when the process (w_k) and the measurement (v_k) noise are *normally* distributed.



(a) CPU allocations and utilizations



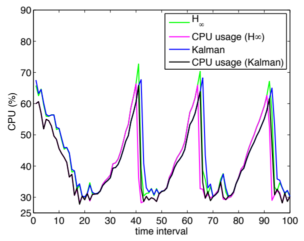
(b) *mRT*

Fig. 5. Gradual Workload Changes: $\mathcal{H}_\infty$ controller performance when the process (w_k) and the measurement (v_k) noise are *uniformly* distributed.

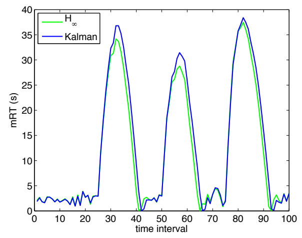
Distribution	Normal		Uniform	
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	2.9893	5.7186	1.6433	2.5633
Kalman	3.5798	7.2353	1.7168	2.5823

TABLE I

RMSE AND *mRT* VALUES FOR GRADUAL WORKLOAD CHANGES



(a) CPU allocations and utilizations



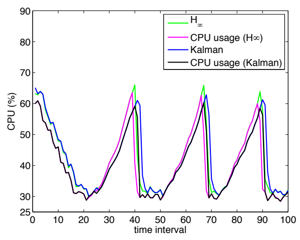
(b) *mRT*

Fig. 6. Saw-Tooth Workload Changes: $\mathcal{H}_\infty$ controller performance when the process (w_k) and the measurement (v_k) noise are *normally* distributed.

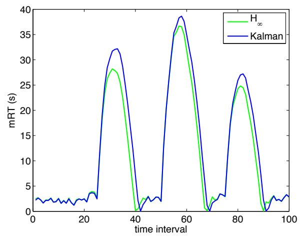
the maximum error during conditions of contention. In these conditions, the $\mathcal{H}_\infty$ controller provides better performance than other approaches. But, there are no assumptions on the noise characteristics and hence the $\mathcal{H}_\infty$ controller is more robust than other controllers.

References

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



(a) CPU allocations and utilizations



(b) *mRT*

Fig. 7. Saw-Tooth Workload Changes: $\mathcal{H}_\infty$ controller performance when the process (w_k) and the measurement (v_k) noise are *uniformly* distributed.

Distribution	Normal		Uniform	
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	6.0463	11.5674	4.575	10.5759
Kalman	5.7544	12.7232	5.0099	12.0326

TABLE II

RMSE AND *mRT* VALUES FOR SAW-TOOTH-LIKE WORKLOAD CHANGES

[2] M. Arlitt and T. Jin, "A Workload Characterization Study of the 1998 World Cup Web Site," *IEEE Network*, vol. 14, no. 3, pp. 30–37, May/June 2000.
[3] A. Iyengar, J. Challenger, D. Dias, and P. Dantzig, "High-Performance Web Site Design Techniques," *IEEE Internet Computing*, vol. 4, no. 2, pp. 17–26, Mar/Apr 2000.
[4] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2007, pp. 289–302.
[5] E. Kalyvianaki, T. Charalambous, and S. Hand, "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)*. New York, NY, USA: ACM, 2009, pp. 117–126.
[6] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated Control of Multiple Virtualized Resources," in *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys '09)*. New York, NY, USA: ACM, 2009, pp. 13–26.
[7] E. Kalyvianaki, T. Charalambous, and S. Hand, "Resource Provisioning for Multi-Tier Virtualized Server Applications," *Computer Measurement Group (CMG) Journal*, vol. 126, pp. 6–17, 2010.
[8] T. Basar and P. Bernhard, *$\mathcal{H}_\infty$ - Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach*, 2nd ed. Boston, MA: Birkhäuser, 1995.
[9] Z. Wang, X. Zhu, and S. Singhal, "Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions," in *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, October 2005, pp. 133–144.
[10] X. Zhu, Z. Wang, and S. Singhal, "Utility-Driven Workload Management using Nested Control Design," in *Proceedings of the American Control Conference (ACC)*, 2006, pp. 6033–6038.
[11] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal, "AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments," in *Proceedings of the IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, 2007.
[12] L. Kleinrock, *Queueing Systems, Volume 1, Theory*. Wiley-Interscience, 1975.
[13] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transaction of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
[14] D. Simon, *Optimal State Estimation: Kalman, H-infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.