Applying Kalman Filters to Dynamic Resource Provisioning of Virtualized Server Applications

Evangelia Kalyvianaki
Computer Laboratory
University of Cambridge, UK
ek264@cl.cam.ac.uk

Themistoklis Charalambous
Department of Engineering
University of Cambridge, UK
tc257@eng.cam.ac.uk

Steven Hand
Computer Laboratory
University of Cambridge, UK
smh22@cl.cam.ac.uk

ABSTRACT

Resource management in virtualized data centres is important and challenging, particularly when dealing with complex multi-tier server applications and fluctuating workloads. In this paper, we use control theory to build two controllers based on Kalman filters which monitor and vary CPU allocations across application tiers. Our approach (a) tracks utilisation patterns over noisy data, (b) considers the resource coupling among tiers and collectively allocates resources to them, and (c) adapts to workload conditions through an on-line parameter estimation mechanism. An initial experimental evaluation on a multi-tier server application shows that our controllers work effectively.

Categories and Subject Descriptors

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

General Terms

Measurement, Performance.

Keywords

Kalman Filter, Feedback Control, Resource Provisioning, Virtual Machines.

1. INTRODUCTION

System-level virtualization enables a single physical machine to host multiple *virtual machines* (VMs) which may run any application. In modern data centres, virtualization is commonly deployed to provide the abstraction of an agile set of resources. This can create VMs on demand and dynamically allocate resources to them.

A key prerequisite to high performance is setting these allocations as required to meet server application demands. Server applications, however, exhibit variable resource usage over time due to workload fluctuations; a simple static resource allocation scheme will fail to react to these changes. If under-provisioned, the application is not properly equipped to serve incoming requests; its performance will drop and Service Level Agreement (SLA) violations may occur. If over-provisioned, incoming requests are adequately served, but physical resources are under-utilised, thus preventing additional applications from being run. A dynamic resource allocation scheme is required to address these challenges.

Recently control theory has been used to perform dynamic allocation of CPU resources in virtualized data centres. CPU is an important resource for data centres. Server machines' reported under-utilisation has various consequences such as increase in power consumption. Therefore it is critical to use efficiently CPU resources. Dynamic techniques can be used to construct an efficient scheme that adjusts allocations as workload changes occur, without using extensive *a priori* knowledge of the workload patterns or the applications' internal structure. For example, Wang et al [9] present a nonlinear gain-adaptive integral controller that regulates the relative utilisation at a target value for single resource containers. The same controller in combination with other controllers has been studied in virtualized environments in order to: (a) allocate the CPU resources for co-located multi-tier applications [6]; (b) maintain the server response time within user-specified limits [12]; and (c) regulate the response time to a reference value with the aid of a performance model based on transaction mixes to better estimate the utilisation across tiers [8]. Finally, Liu et al [5] present an optimal controller that computes the resource allocations for multi-tier co-located virtualized applications, providing QoS response time differentiation in overload.

This paper presents a feedback control scheme that uses Kalman filters to control CPU resource allocation for multi-tier server applications deployed across multiple VMs. We formulate the allocation problem as a CPU utilisation tracking one, where a controller aims to maintain the CPU allocation at a certain limit above the utilisation. Tracking the utilisation is an intuitive approach to resource provisioning as each VM is allocated resources as needed.

Although others have also tried to regulate the relative CPU utilisation to a reference value [9, 6], the contribution of this paper 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 an application performance model [11], we use Kalman filters as 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 and has good performance even when the conditions are relaxed.

This paper makes the following contributions: (a) a linear allocation controller for each VM based on tracking CPU usage; (b) an extended second controller that considers the resource coupling in multi-tier applications and enables faster allocations to saturated components; and (c) an on-line adaptation mechanism that modifies the filter parameters under different operating conditions without requiring any *a priori* knowledge.

2. PROTOTYPE VIRTUALIZED CLUSTER

Figure 1 illustrates the prototype virtualized cluster used to evaluate our controllers. The cluster, which consists of three machines running the Xen 3.0.2 hypervisor [2], hosts the Rubis server application [1]. Rubis is a prototype auction web server which models eBay.com. In this paper we use a 3-tier version of Rubis. Each one of the three server components — Tomcat web server, JBoss application server and MySQL DB server — is deployed on a separate VM running on a separate physical machine. A fourth machine hosts the Rubis Client Emulator used to generate requests[1] to the server. The Client Emulator also records the response times of requests and can be used to evaluate the performance of our controllers. All machines are connected on a Gigabit Ethernet network.

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

The current prototype controls CPU allocations per VM. To ensure that the server's performance depends solely on the controller(s) CPU allocations, certain actions are taken. All machines have two CPUs, and each one of the two VMs per physical machine is pinned on a separate CPU. This simple setup enables us to study the impact of the controller(s) allocations on the server performance, without any implications due to scheduling artifacts among running VMs sharing the same CPU. In future work we hope to demonstrate that our system performs well even when many VMs share a single CPU. Finally, for all the experiments each VM is allocated memory as required when first created and this allocation is kept constant throughout. The network bandwidth is also measured and is never a bottleneck to the application.

3. CONTROLLER DESIGN

Since first presented by R.E. Kalman in his seminal 1960 paper [3], the Kalman filter has been used in a large number of areas including autonomous or assisted navigation, motion prediction, and so on. 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 and in the maximum likelihood sense under the following assumptions: (a) the system is described by a *linear* model and (b) the process and measurement noise are *white* and *Gaussian*. It is also computationally attractive, due to its recursive computation, since the production of the next estimate only requires the updated measurements and the previous predictions.

In the rest of this section we present two controllers which make use of Kalman filters to dynamically allocate CPU resources to the various VMs comprising a multi-tier application. We define a component's *CPU allocation* a to be the percentage of the total CPU capacity of a physical machine allocated to a running VM; a component's *CPU usage* or *utilisation* v to be the percentage of the total CPU capacity of a physical machine actually used by that component; and u the measured/observed value of v.

[1]Due to space considerations, in this paper we limit ourselves to the read-only browsing mix [1].

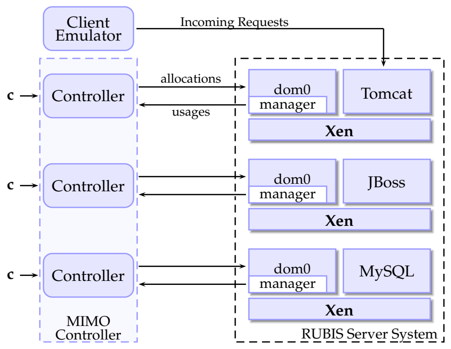


Figure 1: Virtualized prototype and control system overview. Solid lines between the controller modules and the Rubis Server System depict the three BC SISO controller systems. The PNCC MIMO controller is shown by the dashed rectangle.

3.1 Basic Controller

The Basic Controller (BC), also presented in [4], computes the allocations for each VM based on usage measurements from each tier separately. All metrics presented in this subsection are scalar and refer to a single component.

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

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

where the independent random variable t represents the process noise and is assumed to be normally distributed. Intuitively, in a server system the CPU usage v_{k+1} in the next interval will generally depend on the usage v_k of the previous interval as modified by changes, t_k, caused by request processing e.g., processes added to or leaving the system, additional computation by existing clients, lack of computation due to I/O waiting, and so on. Knowing the process noise and the usage v_k over the previous interval, one can predict the usage v_{k+1} for the next interval. The purpose of the controller is to maintain the allocation at a certain level $\frac{1}{c}$ of the usage, where c is customised for each server application or VM. In this way, allocations are updated as required and follow the workload fluctuations. The allocation is thus described by:

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

The allocation a is an unknown signal since the real usage v is also unknown. Therefore, we try to approximate a through the measurements u as:

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

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

$$p(z) \sim N(0, Q), \tag{4}$$

$$p(w) \sim N(0, R). \tag{5}$$

The measurement noise variance (R) might change with each time step or measurement. Also, the process noise variance (Q) might change in order to adjust to different dynamics. However, for the rest of this subsection they are assumed to be stationary during the filter operation. Later, we will present an approach which considers non-stationary noise.

Given that the equations (eq.) (2) and (3) describe the system dynamics, the required allocation for the next interval is computed based on tracking the utilisation and is a direct application of the Kalman filter theory. $\widetilde{a}_k$ is defined as the *a priori* estimation of the CPU allocation, that is the predicted estimation of the allocation for the interval k based on previous measurements. $\widehat{a}_k$ is the *a posteriori* estimation of the CPU allocation, that is the corrected estimation of the allocation based on measurements. The predicted *a priori* allocation for the next interval $k+1$ is given by:

$$\widetilde{a}_{k+1} = \widehat{a}_k, \tag{6}$$

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

$$\widehat{a}_k = \widetilde{a}_k + K_k(u_k - c\widetilde{a}_k). \tag{7}$$

At the beginning of the $k+1$ interval the controller applies the *a posteriori* $\widehat{a}_k$ allocation. If the $\widehat{a}_k$ 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. Therefore, 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:

$$K_k = c\widetilde{P}_k(c^2\widetilde{P}_k + R)^{-1}, \tag{8}$$

where $\widetilde{P}_k$ is the *a priori* estimation error variance and is calculated based on the *a posteriori* error variance $\widehat{P}_{k-1}$:

$$\widehat{P}_{k-1} = (1 - cK_{k-1})\widetilde{P}_{k-1}, \tag{9}$$

$$\widetilde{P}_k = \widehat{P}_{k-1} + Q. \tag{10}$$

3.1.1 Modelling Variances

To obtain a good estimation of the allocation process noise variance Q it is enough to estimate the usage variance — since the allocation is considered to be proportional to the usage — and then evaluate it via the following formula:

$$var(a) \simeq var(\frac{u}{c}) = \frac{1}{c^2}var(u). \tag{11}$$

The usage process noise corresponds to the evolution of the usage signal in successive time frames. Estimating its variance is difficult, since the usage signal itself is an unknown signal and it does not correspond to any physical process well described by a mathematical law. The usage variance is calculated from measurements of the CPU utilisation. When the current BC controller is applied, the stationary process variance Q is computed off-line before the control process and remains the same throughout.

Finally, the measurement noise variance R corresponds to the confidence that the measured value is very close to the real one. Once more it is difficult to compute the *exact* amount of CPU usage. However, given the existence of relatively accurate measurement tools, a small value (e.g. $R = 1.0$ is used throughout the paper) can act as a good approximation of possible measurement errors.

3.2 Process Noise Covariance Controller

The Process Noise Covariance Controller (PNCC) further extends the BC controller by considering the resource coupling between multi-tier applications (MIMO controller in Figure 1, usages from all tiers are forwarded to all controllers). In multi-component servers, there is a correlation between the utilisation of the various tiers. Server applications are usually modelled with queues in tandem. If any of the tiers is inadequately provisioned, the overall server performance is affected. In fact, when workload fluctuations happen in resource provisioned server components, the saturation point can be moved from one component to another, causing prolonged poor server performance [7, 10].

To address this problem, the PNCC controller considers the coupling between the components. The allocation for each component is adjusted based on the errors of the current component in addition to the errors caused in the other components, as explained below. If n is the number of application components, then the PNCC Kalman filter equations for stationary process and measurement noise take the form:

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{Z}_k, \tag{12}$$

$$\mathbf{U}_k = \mathbf{C}\mathbf{A}_k + \mathbf{W}_k, \tag{13}$$

$$\widehat{\mathbf{A}}_k = \widetilde{\mathbf{A}}_k + \mathbf{K}_k(\mathbf{U}_k - \mathbf{C}\widetilde{\mathbf{A}}_k), \tag{14}$$

$$\mathbf{K}_k = \mathbf{C}\widetilde{\mathbf{P}}_k(\mathbf{C}\widetilde{\mathbf{P}}_k\mathbf{C}^T + \mathbf{R})^{-1}, \tag{15}$$

$$\widehat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{C}\mathbf{K}_k)\widetilde{\mathbf{P}}_k, \tag{16}$$

$$\widetilde{\mathbf{A}}_{k+1} = \widehat{\mathbf{A}}_k, \tag{17}$$

$$\widetilde{\mathbf{P}}_{k+1} = \widehat{\mathbf{P}}_k + \mathbf{Q}, \tag{18}$$

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; $\mathbf{C} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with the target value c for each component along the diagonal; $\widetilde{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ and $\widehat{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ are the *a priori* and *a posteriori* error covariance matrices; $\mathbf{K}_k \in \mathbb{R}^{n\times n}$ is the Kalman gain matrix and $\mathbf{R} \in \mathbb{R}^{n\times n}$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are the measurement and process noise matrices respectively. For matrices $\mathbf{Q}$ and $\mathbf{R}$ the diagonal elements correspond to the process and measurement noise for each component. The non-diagonal elements of the matrix $\mathbf{Q}$ correspond to the process noise covariance between different components. Similarly, the non-diagonal elements of the $\mathbf{K}_k$ matrix correspond to the gains between different components and are computed based on their covariances (eq. (15), (16), and (18)). For example, in a 3-tier application, the *a posteriori* $\widehat{\mathbf{A}}_k(1)$ estimation of the allocation of the first component at time k is the result of the *a priori* estimation $\widetilde{\mathbf{A}}_k(1)$ of the allocation plus the corrections from all components' innovations, given by:

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

3.2.1 Modelling Covariances

Similarly to the computation of the allocation variances, the covariances between the components' allocations are computed off-line 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:

$$cov(a_i, a_j) \simeq cov(\frac{u_i}{c}, \frac{u_j}{c}) = \frac{1}{c^2} cov(u_i, u_j). \qquad (19)$$

3.3 Process Noise Adaptation

So far we have assumed stationary process and measurement noises. Both controllers can be easily extended to adapt to operating conditions by considering non-stationary noises. For example in the case of the PNCC controller, all formulae are as before but instead of the stationary $\mathbf{Q}$, the dynamic $\mathbf{Q}_k$ is now used. In this case, $\mathbf{Q}_k$ is updated every several intervals with the latest computations of variances and covariances from CPU utilisation measurements over the last iterations. For simplicity reasons, we consider the measurement noise variance $\mathbf{R}$ to be always stationary, ($\mathbf{R}_k = \mathbf{R}$).

4. RESULTS

In this section, the controllers' performance is studied in a variety of situations. System identification experiments performed on our benchmark server application show that when all components are adequately provisioned the client mean response time (mRT) is kept below 1 second (s). If one or more components are saturated, however, then the mRT exceeds 1s. Therefore, for evaluation purposes, the performance of the controller is examined against the mRT.

For all experiments the controller interval is set to 5s; this interval enables the controller to react quickly to workload changes. The parameter c, which denotes the level of the CPU usage to the allocation, is set to 60%; this enables us to study the controllers' performance without any implications from the benchmark application. When the CPU usage approaches its allocation, the mRT exceeds 1s, because of the large CPU usage variance exhibited by the Tomcat and MySQL components.

4.1 Basic Controller

We first evaluate the performance of the BC controller. Initially, the stationary process and measurement variances are computed. To compute the allocation noise variance Q for each component, the usage variances are first measured, eq. (11). To this end, the following experiment is performed: 600 clients issue requests to the server for 200s, and each component is allocated to 100% of its CPU. The same experiment is repeated 10 times for statistically confident results. The estimated usage variances are: var(Tomcat, JBoss, MySQL) $= (28.44, 4.75, 47.43)$. We refer to the set of these values as Q_0 and to the data set of this experiment a D1. These values are an estimation of the variances in the case of 600 clients, when no allocation restrictions are applied to the components. Experiments in this paper use 600 clients.

Figure 2 illustrates the BC controller allocations across all components, when the workload intensity changes; the number of clients doubles from 300 to 600 at the 20^{th} sample point, and drops to 300 at the 40^{th}, we refer to this type of experiment as E1. The controller tracks the usage fluctuations and it tries to maintain the allocations at $\frac{1}{c}$ of the usages, (Figures 2(a), 2(b), 2(c)). The server's performance (Figure 2(d)) is sustained at good levels, as the mRT for the majority of the intervals stays well below 1s. It exceeds this value when the usage of one or more components becomes very close to its allocation for that interval; this is standard Rubis server behaviour.

It is even more useful to build a controller which corrects its error and is not so strongly affected by transient fluctuations. This is achieved by tuning the variances which affect the Kalman gain. According to eq. (8), K_k monotonically increases with Q, and monotonically decreases with R. 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 (the difference between the predicted state and the measurement), rather than the current prediction, in order to keep up with the highly variable measurements. Therefore, the Kalman gain in this case is relatively large. This is also the case for the gain in Figure 2(e), where the variance values Q_0 are relatively larger than the R values. On the other hand, when the measurement noise variation R relatively increases to the Q value, the new measurements are biased by the included measurement error. The filter should then decrease its confidence in the new error and the Kalman gain stabilises to smaller values.

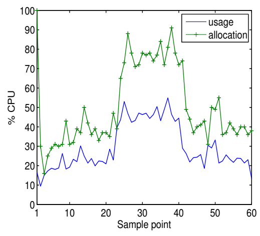
To better illustrate this behaviour, an E1 type of experiment with workload fluctuations is carried out, but where only a fraction of the initially computed values Q_0 are considered; the new variances are set to $Q_0/400$, which are closer to the R values. Results are shown in Figure 3. In this case, the values of the Kalman gains (Figure 3(e)) are smaller than before (Figure 2(e)), and therefore the filter has more confidence in the predicted values than the new measurements. Figures 3(a), 3(b), and 3(c) show that the allocation signal still adapts to the usage changes; however, the behaviour is now smoother than before. In fact, the overall server's performance is better (Figure 3(d)), with fewer spikes above 1, since the controller is not affected by transient usage fluctuations. Tuning the Q values enables the controller to act fast to workload changes without being so strongly affected by transient fluctuations. With further system identification analysis, Q values can be set to appropriate levels for specific applications.

The slow responsiveness of the controller to workload changes is mainly the result of two factors. Firstly, when the controller is configured to make smooth allocations, it responds slowly to sudden workload changes. Secondly, the BC controller controls the allocations for each component separately, ignoring any resource coupling between them. The PNCC controller presented next, considers this issue.

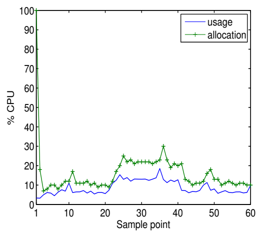
4.2 Process Noise Covariance Controller

In the current example server application, each component's usage is the result of its own workload processing. However in multi-tier applications, if any component is not adequately provisioned, all component utilisations can be affected [4]. To incorporate the other components' usages, the usage covariance between the components is now used; the new allocations, as explained in Section 3.2, are therefore adjusted according to all components' errors.

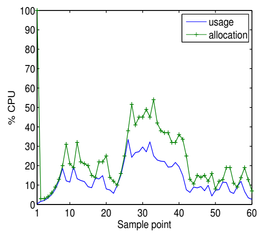
The PNCC controller is now evaluated using an exper-



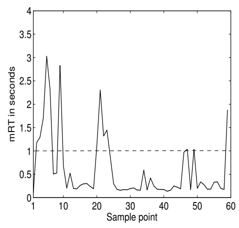
(a) Tomcat



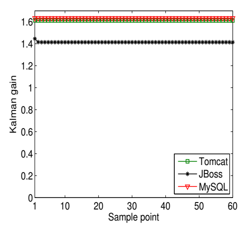
(b) JBoss



(c) MySQL

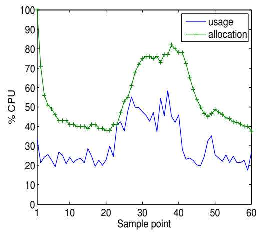


(d) Response

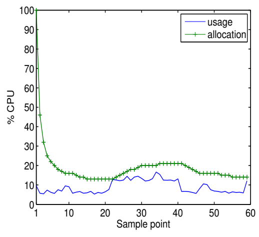


(e) Kalman gains

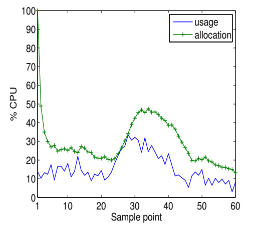
Figure 2: BC controller performance, stationary Q_0 variances.



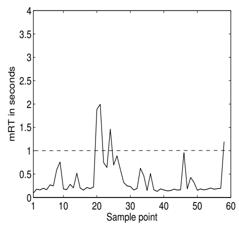
(a) Tomcat



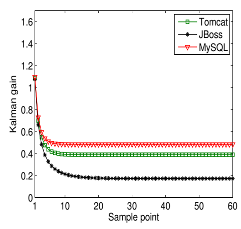
(b) JBoss



(c) MySQL



(d) Response



(e) Kalman gains

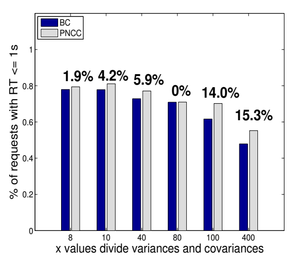
Figure 3: BC controller performance, stationary $Q_0/400$ variances.

iment of type E1 with workload fluctuations; results are shown in Figure 4. **Q** variance values are set from the off-line estimated Q_0 and we also estimate the covariances from the same data set, D1. Figure 4(d) shows the server's mRT, which is maintained at good levels (below 1s). Note that the PNCC controller responds more quickly to the increased number of clients, as shown by the fewer mRT spikes at around the 20^{th} interval compared with Figure 3(d).

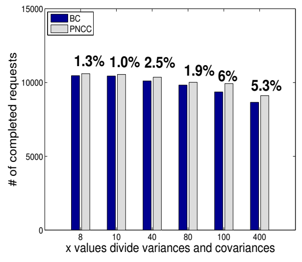
To better compare the performance of the PNCC and BC controllers when workload changes, we perform the next experiment: 200 clients issue requests for 60 intervals; at the 30^{th} interval another 600 are added for the rest of the experiment, quadrupling the total number of clients. The same experiment is performed for variances and covariances divided by x values drawn from $X = (x \in \{8, 10, 40, 80, 100, 400\})$. Each experiment is performed from 20 to 40 times for statistically confident results. Results are shown in Figure 5, where the controllers are evaluated against two metrics: the percentage of requests with response time (RT) $\leq$ 1s (Figure 5(a)), and the number of completed requests (Figure 5(b)). To emphasise on the actual workload change, all metrics are calculated for the duration of the workload increase until the server settles down to the new increased number of clients. Results show that the PNCC controller improves the BC controller's performance; larger improvement is achieved as the values of the variances and covariances decrease. In general, the small overall improvement observed is due to the fact that each component has much more confidence in its own error (e.g. large Tomcat gain for the Tomcat allocations in Figure 4(e)) than the errors coming from other components (e.g. small gains between different components in Figure 4(e)). Again, with system identification analysis on specific applications, the PNCC controller can be tuned to achieve different performance.

4.3 PNCC Adaptive

So far we have presented results with stationary variances



(a) Low Response Requests

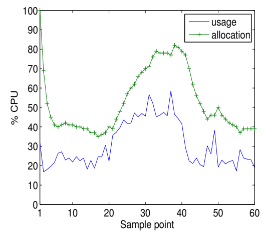


(b) Completed Requests

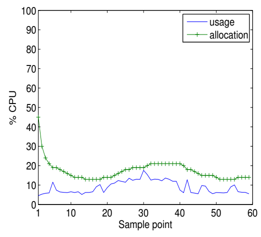
Figure 5: Comparison between BC and PNCC controllers. Percentages in each case show the metric difference of the PNCC controller over the BC with a 90% confidence interval (CI).

and covariances computed off-line. As server workloads can change frequently, it is impossible to compute off-line the variances and covariances for every possible combination of number of clients and request type mixes. In Section 3.3 we discussed how each controller is formulated in the case of non-stationary process variances. In this section, we evaluate the PNCC controller augmented with an adaptation mechanism which computes the process variances and covariances in an on-line fashion.

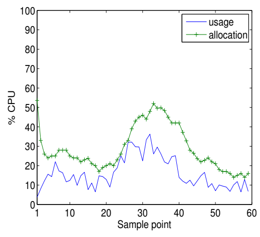
The adaptation mechanism is tested again sudden workload changes using the following experiment: initially 300 clients issue requests to the server for 120 intervals; at the 40^{th} interval another 300 are added for 40 intervals so that for the last 40 intervals the number of clients is dropped to the initial 300. Variances and covariances are updated every 10 controller intervals using the usage measurements. The same experiment is repeated for different **Q** values divided by $x \in X$. We compare against the PNCC controller, configured with the stationary variances and covariances as com-



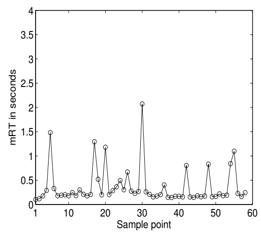
(a) Tomcat



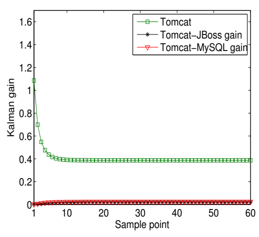
(b) JBoss



(c) MySQL

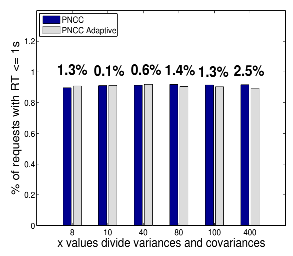


(d) Response

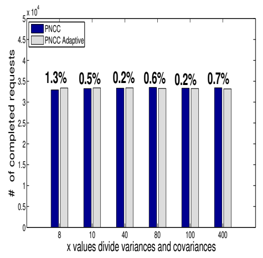


(e) Kalman gains

Figure 4: PNCC controller performance, stationary $\mathbf{Q}/400$ variances and covariances. Figure 4(e) shows the Kalman gains used for calculating the allocations for the Tomcat component only.



(a) Low Response Requests



(b) Completed Requests

Figure 6: Comparison between PNCC and PNCC Adaptive controllers. Percentages in each case show the metric difference of the PNCC Adaptive controller over the PNCC with a 90% CI.

puted previously. Results shown in Figure 6; all experiments are repeated 5 times. The PNCC adaptive performs equally well to the non-adaptive PNCC, as shown by the number of requests with RT $\leq$ 1s, and the number of completed requests for the duration of the experiment. To conclude, the adaptation mechanism is very powerful, as it eliminates the need for off-line computations and so the controller can adapt to unknown workload changes.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented two feedback controllers that control the CPU allocations of multi-tier virtualized server applications based on the Kalman filtering method. Our results showed that the controllers dynamically adapt to workload fluctuations. By considering the resource coupling among tiers, a further improvement is achieved. In addition, the controllers can be tuned to adapt more quickly or more slowly to workload changes. Finally, the controllers' parameters can be computed on-line, avoiding therefore extensive off-line computations.

Part of our ongoing research is the extension of this work by combining particle filtering with Kalman filters. This should let us to track variables with noise that is not essentially normally distributed, and also allow us to cope better with multi-modal models.

Finally, we would like to thank the anonymous reviewers for their useful comments and suggestions on this paper.

6. REFERENCES

[1] C. Amza, A. Chanda, E. Cecchet, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and Implementation of Dynamic Web Site Benchmarks. In *Proc. of the 5th Annual IEEE Int. WWC Worskhop*, 2002.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. of the 19th ACM SOSP*, 2003.

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

[4] E. Kalyvianaki and T. Charalambous. On Dynamic Resource Provisioning for Consolidated Servers in Virtualized Data Centres. In *Proc. of the 8th Int. PMCCS Workshop*, 2007.

[5] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In *Proc. of the 46th IEEE Conf. on Decision and Control*, 2007.

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

[7] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Groyal. Dynamic Provisioning of Multi-tier Internet Applications. In *Proc. of the 2nd Int. ICAC*, 2005.

[8] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal. AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments. In *Proc. of the 2nd IEEE Int. FeBID Workshop*, 2007.

[9] Z. Wang, X. Zhu, and S. Singhal. Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions. In *Proc. of the 16th IFIP/IEEE Int. DSOM Workshop*, 2005.

[10] Q. Zhang, L. Cherkasova, and E. Smirni. A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. In *Proc. of the 4th IEEE Int. ICAC*, 2007.

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

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