On Dynamic Resource Provisioning for Consolidated Servers in Virtualized Data Centers

Evangelia Kalyvianaki
Computer Laboratory
University of Cambridge
Email: ek264@cam.ac.uk

Themistoklis Charalambous
Department of Engineering
University of Cambridge
Email: tc257@cam.ac.uk

Abstract

Server consolidation based on Operating System (OS) virtualization technologies aims to increase the total resource utilization in data centers. However, efficiently capitalizing on the available resources requires tools to automatically monitor and allocate resources to the running servers as needed, while maintaining their Quality of Service (QoS) guarantees. In the presence of workload fluctuations that cause diverse and unforeseeable resource usages building such tools is challenging. In this paper, two novel techniques are presented that control the CPU allocation for running servers. Firstly, a Single Input Single Output (SISO) first-order Kalman filter is designed to dynamically allocate the CPU of individual Virtual Machines (VMs). The motivations to use Kalman filter are the nonlinear behavior of the system and the known fact that the Kalman filter is the optimum linear tracking controller in the sense that it minimizes the estimated error covariance when some presumed conditions are met. Secondly, a Multiple Input Multiple Output (MIMO) feedback controller that dynamically allocates CPU for multi-tier server applications is presented. The controller makes global decisions by coupling the resource usage of all components. By this way, a fast response to sudden changes in CPU usage is achieved, which causes tiers to saturate.

I. Introduction

In the last few years server consolidation using OS virtualization technologies has gained significant attention. Traditionally, each e-business server application runs on a set of dedicated machines. Although this approach offers performance and security guarantees, it results in under-utilization of the available resources. Workload fluctuations cause diverse resource usages across server machines, which are allocated in ways to cope with both the average and the worst-case workload scenario. In the average case however, they are under-utilized. Many enterprizes report only 10-15% processor utilization in their servers, as reported by [2] and the references therein. OS virtualization-based server consolidation aims to increase server resource utilization, by providing an abstract and efficient way for different applications to co-locate on machines and share resources in a secure and QoS provided manner.

OS virtualization is based on an abstraction of the hardware layer exported to the running OSes. Each instance of the virtualized OS, usually referred to as Virtual Machine (VM), runs on top of a hypervisor which manages resource sharing and isolation among them. The portion of the available resources allocated to each VM can be assigned upon its creation and can also be changed dynamically during its lifetime. In the context of server consolidation, a server application is deployed in one or more VMs, which may be located on the same or different machines. Ideally, resource allocation for each VM can now be managed dynamically to the point where each server component (or VM) is entitled to a portion of the available resources very close to its required usage with a certain safety margin. However, efficiently allocating the required resources while maintaining the applications' Service Level Agreements (SLAs) in the presence of dynamically changing workloads introduces several challenges and is the subject of this paper.

Dynamic allocation of resources for server applications has been widely studied ([4], [8], [14], [15]) for many different purposes (e.g. meeting QoS guarantees, achieving high resource usage). Existing resource allocation algorithms for server applications can be distinguished into different classes, such as queueing based models [6], reinforcement learning approaches [12] and methods based on control theory [9] and fuzzy logic [17]. Control theory has been used to design feedback controllers for computer systems' management (for example see [9] and references therein). More recent work addresses the issue of resource allocation on virtualized data centers by applying adaptive control theory. In [16] an adaptive integral controller has been designed for dynamic sizing of a server resource partition by trying to maintain the relative resource usage constant in spite of the changing demand. In [11] a two-layered nonlinear controller was developed using classical control theory to deal with two multi-tier applications running on a virtualized data center. In this paper, the problem of dynamically allocating the CPU resources to server applications running on VMs using control theory is addressed. The contribution of this paper is twofold, as summarized below.

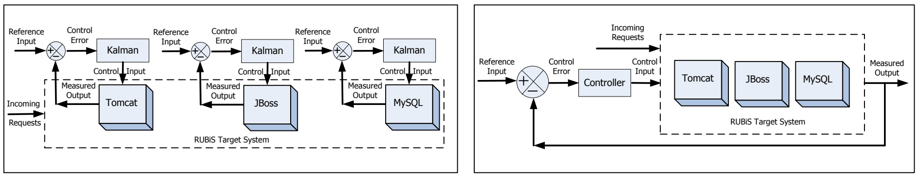
Firstly, resource allocation is formulated as a resource usage tracking problem. Thus, a Kalman filter-based Single Input Single Output (SISO) controller is developed to dynamically allocate the CPU to individual VMs. The controller adaptively adjusts the allocation to diverse resource usages while meeting the QoS guarantees. Our approach tracks the resource usage and uses this to control the resource allocation for a single component. The Kalman filter has been used extensively for tracking in many different areas, such as autonomous or assisted navigation, interactive computer graphics, and motion prediction. This was made possible due to advances in digital computing that made the use of the filter practical, but also to the relative simplicity and robust nature of the filter itself. The necessary conditions for optimality do not often actually exist and yet the filter works well for many applications in spite of this situation.

Secondly, a Multiple Input Multiple Output (MIMO) feedback controller that regulates the CPU allocations for multi-tier applications in the presence of varying number of requests is introduced. As server applications are distributed among many components deployed on different VMs, a global controller that allocates resources to all of them is required. By coupling the CPU usage of all components the controller adapts the CPU share for all, even when only a subset of them is saturated.

The rest of this paper is organized as follows. The prototype cluster along with the overall architecture employed in each case is presented in section II. In section III the design and performance evaluation of the Kalman filter are presented. Then, the MIMO controller is presented and its performance is evaluated in section IV. Both techniques are evaluated against a sample server running the RUBiS auction benchmark site [1]. Results show that efficient CPU allocation is achieved in an automatic and dynamic manner in the presence of varying number of clients while preserving the application's QoS guarantees.

II. Experimental Setup

The effectiveness of the two techniques is examined using a virtualized prototype data center that runs a sample RUBiS server. RUBiS is a prototype auction site modeled after the eBay.com. A three-components RUBiS EJB version is used. Figures 1(a), 1(b) illustrate the experimental setup with the RUBiS application deployed on three physical machines along with the SISO and MIMO controllers. Each of the three components, namely the Tomcat web server, the JBoss application server and the MySQL DB server, is deployed on a VM running on a separate physical machine. All machines are connected via a Gigabit Ethernet network. This simple setup enables us to study the impact of the controller's allocations on the server performance. Each machine runs the Xen virtualization infrastructure [5], version 3.0.2. Using Xen, we can dynamically change the CPU allocation (the portion of the available CPU capacity of the physical machine) for each VM. For all the experiments each VM is allocated memory as required. The network bandwidth is measured



(a) Kalman Filter. (b) MIMO controller.

Fig. 1. Overview of the two prototype control systems. In the SISO control system there is one Kalman filter for each RUBiS component. In the MIMO control system there is one controller for all components. In this figure, each RUBiS' component box corresponds to a VM running on a separate physical machine.

and is never a bottleneck to the application. Finally, the RUBiS client emulator is used to generate requests to the server machines. All requests belong to the browsing mix workload type. More information on the RUBiS emulator and the workload type can be found in [3].

III. SISO Controller

In this section the SISO controller based on the Kalman filter [10] is presented. In this case, the CPU allocation for each VM is controlled separately. Initially, the dynamics of the system are identified. Then a controller is designed and finally its performance is evaluated.

A. Controller Design

Figure 2(a) presents the CPU usage for all three components (Tomcat, JBoss and MySQL) when the number of clients varies from 100 to 1400 and the CPU allocation in all three VMs is 100%. When there are enough resources to accommodate all client requests (when the number of clients is less than 1200) the mean Response Time (mRT) stays well below 1 second and thus the server achieves good performance. However, when the clients increase above 1200 the mRT increases exponentially and the CPU usage approaches the total CPU allocation (100%) in the Tomcat component. Assuming that there are enough resources to serve clients[1], a controller needs to keep track of the CPU usage and adjusts accordingly the CPU allocation. The criterion used in this paper is similar to [16], i.e., keep the CPU usage at a percentage of the CPU allocation (80% for example), but here, our tracking error (e), at time instant k, is defined as:

$$e(k) = u(k) - ca(k) \tag{1}$$

where c is the desired percentage of the CPU allocation that the CPU usage must have, a is the CPU allocation defined as the percentage of the total CPU capacity of the physical machine allocated to a component and u is the CPU usage as the percentage of total CPU capacity of the physical machine used by a component. Hence, the problem is similar to tracking the time-varying CPU usage, as also mentioned in [11].

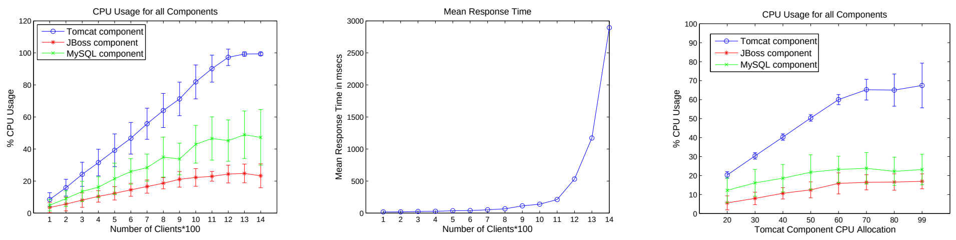
The time-varying CPU usage could be observed as a one-dimensional random walk. Therefore, the system is governed by the following linear stochastic difference equation:

$$a(k+1) = a(k) + z(k) \tag{2}$$

with a measurement u which directly relates to a, that is:

$$u(k) = ca(k) + w(k) \tag{3}$$

[1]In this experimental setup, the RUBiS server offers QoS guarantees (mRT is less than 1 second) to up to 1200 simultaneous clients.



(a) Components' CPU usage.

(b) Mean Client Response Time

(c) CPU usage when the Tomcat component is saturated.

Fig. 2. System Identification.

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, process noise variance (Q) might change in order to adjust to different dynamics. However, here we assume they stay constant during the filter operation. Whether or not we have a rational basis for choosing the parameters, sometimes we get superior performance by tuning these parameters. The tuning is usually performed off-line using system identification.

We define $\widetilde{a}(k)$ to be the *a priori* estimate of the CPU allocation and $\widehat{a}(k)$ to be the *a posteriori* estimate of the CPU allocation. The *a priori* estimate error variance then is $\widetilde{P}(k)$ and the *a posteriori* estimate is $P(k)$.

Hence, our prediction (time update) equations are given by a prediction of the likely state given what we already know,

$$\widetilde{a}(k) = \widehat{a}(k-1) \tag{6}$$

and a prediction of the estimated error variance (a priori estimate)

$$\widetilde{P}(k) = P(k-1) + Q \tag{7}$$

Our correction (measurement update) equations are given by the estimate of the Kalman Gain, K (which is the correction gain between the actual and the predicted observations),

$$K(k) = c\widetilde{P}(k)(c^2\widetilde{P}(k) + R)^{-1} \tag{8}$$

the estimate of the new state given prediction and correction from the observation

$$\widehat{a}(k) = \widetilde{a}(k) + K(k)(u(k) - c\widetilde{a}(k)) \tag{9}$$

and at last, estimate the error of estimated state (a posteriori estimate)

$$P(k) = (1 - cK(k))\widetilde{P}(k) \tag{10}$$

The Kalman filter is a linear controller and is always stable for all the possible values of the Kalman Gain.

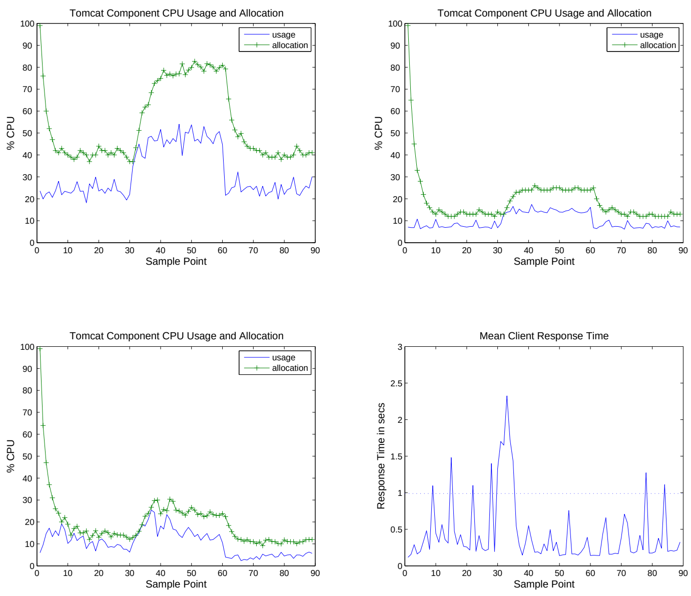


Fig. 3. Kalman Filter CPU allocation and measured mRT.

B. *Results*

The SISO controller designed is validated through its effectiveness in a variety of situations and sudden changes in the workload. A Kalman filter is applied to each one of the three RUBiS components, as shown in Figure 1(a). The CPU usage variance of each component differs with the MySQL component's usage being the most variable. By this way, the Kalman filter is tested against diverse CPU usage signals. The aim of the controller is to achieve a set usage target for the individual components, guaranteeing that QoS is maintained.

In our experiments, we set $c = 0.6$, which corresponds to a CPU usage of 60% of the allocated CPU. The allocation of the CPU is initially assigned to 100% for all three components. Under a browsing mix, we set the number of clients to 300 for the duration of the experiment. After 300 seconds, 300 more clients are added for a duration of another 300 seconds. Each Kalman filter adjusts the allocation for each component every 10 seconds. During the 10 seconds interval the allocation and the CPU usage are measured every second and their average value at the end of the interval is used by the Kalman filter to compute new allocations.

Figures 3 illustrates the measured CPU usage u of all the components (Tomcat, JBoss and MySQL) and how the CPU allocation is adjusted by each controller, such that a 40% margin is always maintained above the usage. Every sample point corresponds to a 10 seconds interval and the average CPU usage and allocation for that interval is shown. Each controller tracks the usage satisfactorily and assigns the required CPU allocation automatically for any demand. Even with the noisy data (especially the MySQL component) acquired in this experiment the controller is very effective.

Figure 3 also shows the client mRT for this experiment. The mRT stays below 1 second for the duration of the experiment with some exceptions where in the worst case it reaches 2 seconds. The spikes in Figure 3 are caused when the usage of one or more components approaches the allocation. As shown in Figure 2(b) the mRT can be very low when the allocation of resources is well above the actual usage. As the allocation reaches the usage, the mRT increases and even exceeds 1 second. At sample point 30 for example, the number of clients doubles to 600. However, each component is allocated CPU for only 300 clients at that point. As a result, the CPU reaches the current allocation and the mRT increases to more than 1 second. When c is set to 0.6 the resulting difference between the allocation and the usage results on a low mRT. In general, the mRT increases when the usage is closer to the allocation as can be seen by the spikes. This observation can be further studied and the controller can be extended to provide finer QoS guarantees.

IV. MIMO CONTROLLER

In this section the MIMO controller is presented. In this case, the CPU allocation for all VMs is adjusted by one controller. Initially, the system's dynamics are discussed. Then the MIMO controller design is presented and finally its performance is evaluated.

A. Controller Design

As illustrated in Figure 2(a), different components consume different amounts of CPU, with Tomcat consuming the largest amount of resources while JBoss consumes the least. It also suggests that the components' CPU usage are coupled and more specifically in this case they are linearly related. Intuitively, this should be the case since the workload on each component is affected by the workload on the rest. If one of the components is a bottleneck, then the rest of the components cannot process the requests of more clients. In Figure 2(c) the CPU allocation for the Tomcat component is varied, while the number of clients is kept constant at 800. The JBoss and MySQL components' allocations are kept to 100%. The required allocation for the Tomcat component to process all 800 clients is around 60%. As the Tomcat CPU allocation is kept below the required 60%, the CPU usage of the other two components is also kept low indicating that less than 800 clients are being served. Similar observations are obtained when either the JBoss or the MySQL component are saturated. In the case of a bottleneck, the increase in the allocation of one component eventually leads to the increase of the CPU usage of the other components, suggesting that their allocations should be increased as well. Therefore, a controller that takes into account the CPU usage of all the components and assigns the CPU allocation to all components accordingly would be appropriate.

Firstly, the relationships between the different components is extracted. Data is collected (CPU usages for all three components at different workloads) off-line and then processed with the aid of the MATLAB Curve Fitting Toolbox [13]. The CPU usage for all components (denoted by u_1, u_2 and u_3) are found to be related by the following formulae:

$$u_1 = \gamma_1 u_2 + \delta_1, \tag{11}$$

$$u_2 = \gamma_2 u_3 + \delta_2, \tag{12}$$

$$u_3 = \gamma_3 u_1 + \delta_3, \tag{13}$$

where γ_i, δ_i (for all $i \in \{1, 2, 3\}$) are the coefficients found[2].

Secondly, a controller is required for each component that takes into account the behavior of the rest of the components. In addition, when a component has enough resources then its CPU usage is independent of the CPU allocation (Figure 2(c)) and the mRT (Figure 2(b)). Therefore, in order to tackle this problem, a controller is proposed that utilizes the measured CPU usage.

Next, the dynamic allocation of resources is considered by maintaining the difference between the CPU allocation (denoted by $\mathbf{a} = [a_1\ a_2\ a_3]^T$) and the CPU usage ($\mathbf{u} = [u_1\ u_2\ u_3]^T$) at a reference value, $\mathbf{r} = [r_1\ r_2\ r_3]^T$ (the $\mathbf{r}$ values can be assigned in many different ways, in order to

[2]Two of the equations are adequate to describe the relationships between all three components, but we retain them all for notational simplicity.

be higher for noisy workloads and lower for more predictable workloads). Hence, we define the tracking error at time k as:

$$\mathbf{e}(k) = |\mathbf{r} - (\mathbf{a}(k) - \mathbf{u}(k))|. \tag{14}$$

We make use of the absolute error since we always want to provide more resources than the CPU usage. If the difference between the allocation and the usage at time k equals the reference value, the tracking error becomes zero. To always allocate more CPU than the usage, we introduce:

$$\mathbf{P} = \begin{pmatrix} p_1 & 0 & 0 \\ 0 & p_2 & 0 \\ 0 & 0 & p_3 \end{pmatrix}$$

as the lowest ratio of the usage that the allocation is assigned to. The ratio $\mathbf{P}$ values must be set to a number larger than 1. The controller then is defined by:

$$\mathbf{a}(k+1) = \mathbf{P}\mathbf{u}(k) + \lambda\mathbf{M}\mathbf{e}(k), \tag{15}$$

where, λ is a tunable parameter, $M \in \mathbb{R}^{3\times 3}$ and is given by:

$$\mathbf{M} = \begin{pmatrix} 1 & \gamma_1 & \gamma_1\gamma_2 \\ \gamma_2\gamma_3 & 1 & \gamma_2 \\ \gamma_3 & \gamma_1\gamma_3 & 1 \end{pmatrix}$$

It has been proved that λ does not depend on $\gamma_1, \gamma_2, \gamma_3$ and the controller is globally stable if $-1/3 < \lambda < 1/3$.

The controller along with the tracking error can be interpreted as follows: the assigned allocation for each component a_i will always be larger than or equal to $p_i u_i$ providing at least the minimum QoS guarantees. The reference value $\mathbf{r}$ here serves as a second parameter, which among the rest, ensures that the CPU allocation for small CPU usage possesses a specific absolute margin. Hence the problem of having a bottleneck of low CPU usage but high variance is also tackled.

B. *Results*

Figure 4 illustrates the CPU allocations made by the controller as the number of clients varies. The reference value $\mathbf{r}$ is set to $[30\ 30\ 30]^T$, the ratio values p_1, p_2 and p_3 to (1.3, 1.3, 1.3) and λ to 0.2. Initially, each component is allocated with 100% of CPU and 300 clients issue requests to the server for the first 100 seconds. After 105 seconds the number of clients increases to 600 for another 100 seconds. Finally, 300 clients issue requests for the last 100 seconds. The time interval the controller makes new allocations is set to 5 seconds. The allocation and the CPU usage over any time interval is computed as in the Kalman filter case and described earlier. As shown in Figure 4, at the beginning the controller decreases the CPU allocation from the initial 100% for each component to an amount close to the CPU usage. The CPU allocations follow the changes to the CPU usage for the duration of the experiment, even when the number of clients increases or decreases substantially as can be seen around sample points 22 and 42. A change in the number of clients gives different CPU usage changes for each component. Since the controller employs a multi-component approach, it allocates resources in a manner that is aligned to all components' usage as defined by (11), (12) and (13). Similarly, although the initial reference value is set to 30 for all components, the final differences between the usage and the allocation varies for each component.

The mean client response time (mRT) for this experiment is also shown in Figure 4. The server meets its QoS guarantees as the mRT stays well below 1 second throughout the experiment, with only a few spikes close to 1 second. The spikes are caused when the usage of one or more components reaches its allocation.

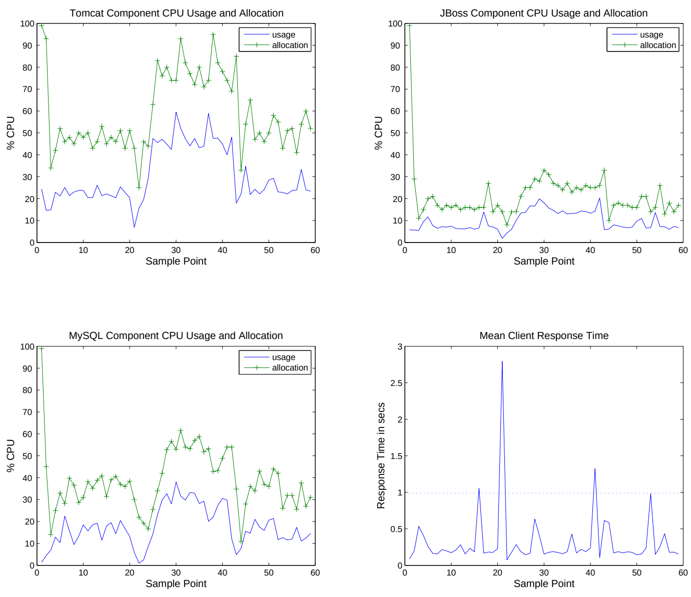


Fig. 4. MIMO Controller CPU allocation and measured mRT.

In this section, a MIMO feedback controller that dynamically allocates CPU for multi-component server applications was presented. The controller allocates CPU in a way that reflects the relative CPU usage among components obtained via off-line analysis. The authors in [11] also address the same problem. They employ a multi-controller approach with one controller per component, as we also do in the case of the SISO controller for the three-components RUBiS application. Each controller computes the next CPU allocation based on the previous CPU usage and a reference value; the usage must be 80% of the allocation. The approach presented in this paper allocates resources to all components, even when only some of them are saturated. By this way the controller increases its responsiveness when sudden increases to CPU usage happen. This can be easily observed if we compare the mRT for the SISO controller at sample point 30 with the mRT for the MIMO controller at the sample point 20. Even though the number of clients for both systems increases from 300 to 600 clients, in the MIMO controller the mRT increases above 1 second for just a period of one sample (which means 5 seconds), whereas in the SISO controller the mRT stays above 1 second for a period of about 7 samples (which means about 70 seconds).

V. Conclusions and Future Work

The area of resource management for virtualized server applications is now gaining significant attention. In this paper two novel techniques that automatically allocate CPU resources for server applications running on virtualized data centers are presented. Firstly, a Kalman filter that tracks the CPU usage for individual VMs and then allocates new resources is presented. Secondly, a MIMO controller that allocates CPU resources for multi-component server applications is discussed. Each controller design is tested in a variety of scenarios against the RUBiS server application running on a prototype cluster of three machines. Each controller allocates resources efficiently as the number of clients varies while maintaining a good application performance.

The work presented in this paper is ongoing research. The theoretical basis of the SISO controller suggests that it is a very efficient controller. After taking into account the various exogenous factors that affect its performance and estimating on-line the changing dynamics of the system that are handled in the Kalman Gain (equation 8), the controller is predicted to become extremely efficient. We would like to extend this work by a high-level implementation approach based on the Condensation algorithm [7], which can track variables with noise that is not essentially normally distributed and copes better with multi-modal models.

Further variations of the MIMO controller are under study. In particular, an integral controller is more desired since it eliminates the steady state error in the system. A new controller that is still under development, uses integral action and considers the relationships between the amounts of usage only when one or more components are bottleneck. Hence, when a bottleneck component is allocated CPU, the rest of the components are also allocated the corresponding CPU. Therefore, the controller intelligently uses the coupling of the amounts of usage to allocate the available resources and simultaneously eliminates any steady state error. In the case, where none of the components are saturated a number of SISO controllers such as the Kalman filter can be used. Finally, in this paper an off-line analysis is performed to derive the relationship among the components CPU usage. Online regression models can be used to dynamically derive this relationship. This approach also permits the controller to dynamically adapt to different workload types.

References

[1] RUBiS: Rice University Bidding System. http://rubis.objectweb.org.

[2] System z9: The Value of Mainframe. http://www-03.ibm.com/systems/z/about/charter/value_utilization.html.

[3] 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 *Fifth Annual IEEE International Workshop on Workload Characterization (WWC-5)*, 2002.

[4] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 90–101, 2000.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[6] M. Bennani and D. Menasce. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC0-05)*, pages 229–240, 2005.

[7] M. J. Black and A. D. Jepson. Recognizing Temporal Trajectories Using the Condensation Algorithm. In *Proceedings of the 3rd. International Conference on Face and Gesture Recognition (FG)*, pages 16–21, 1998.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centres. In *18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, 2001.

[9] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. A Control Theory Foundation for Self-Managing Computer Systems. *IEEE Journal on Selected Areas in Communications*, 23(12):2213–2222, December 2005.

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

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

[12] G. Tesauro, N. Jong, R. Das, and M. Bennani. A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation. In *Proceedings of the IEEE International Conference on Autocomic Computing (ICAC-06)*, pages 65–73, 2006.

[13] M. C. F. Toolbox. *http://www.mathworks.com/products/curvefitting*.

[14] B. Urgaonkar and P. Shenoy. Sharc: Managing CPU and Network Bandwidth in Shared Clusters. *IEEE Transactions on parallel and distributed ststems*, 15(1):2–17, 2004.

[15] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 239–254, 2002.

[16] Z. Wang, X. Zhu, and S. Singhal. Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions. In *Distributed Systems Operations and Management Workshop DSOM*, pages 133–144, October 2005.

[17] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *Proceedings of the Fourth International Conference on Autonomic Computing (ICAC)*, 2007.