On the use of Fuzzy Logic Controllers to Comply with Virtualized Application Demands in the Cloud

Kyriakos M. Deliparaschos, Themistoklis Charalambous, Evangelia Kalyvianaki and Christos Makarounas

***Abstract*— As virtualization technologies enable real-time CPU allocation, it is important to build controllers that adjust the allocation in a timely fashion avoiding resource saturation and hence, dissatisfaction of the end users of services. In this work, adaptive neuro-fuzzy inference, trained on Kalman and $\mathcal{H}_\infty$ filters, has been used to adjust the CPU allocations based on observations of past utilization. When evaluating the performance of the proposed controller it is demonstrated that it provides even better performance than the filters it is trained on. In addition, there are no assumptions on the noise characteristics and due to the fact that the neuro-fuzzy controller can, in general, capture non-linear level processes, our controller is more robust than linear model based approaches, such as the Kalman and the $\mathcal{H}_\infty$ filters.**

I. Introduction

The enabling technology to cloud computing is *virtualization* (see, for example, XEN [1] and VMware ESX [2], [3]) where a physical machine is transformed into one (or more) virtual machines (VMs) each capable of operating independently as a standalone physical server. Furthermore, VM live migration enables data center managers to move VMs across physical servers with only minimal application disruption [4]. In total, virtualization enables (a) application *consolidation* when multiple virtualized applications run on a single shared server; and (b) an application to span multiple physical resources across the data center. This technique is developed for efficient use of computer server resources in order to reduce the total number of servers required for a number of software implementations, since more resources are used than necessary to provide the functionality required. As a result, server loads can be aggregated in fewer number of servers and save energy by switching off idle servers or allow the providers accommodate more customers, helping them provide lower prices since the customers will not have to buy unnecessary cloud resources and the providers can increase their profit.

Consolidated applications share server resources, such as CPU time, memory and disk space. For each application to operate efficiently it is required that it is allocated with enough resources from the hosting server resources in order to meet its performance requirements, as measured, for example, by the requests mean Response Time (*mRT*). However, adjusting the shares of resources of consolidated applications while their demands change over time is challenging because (a) the workloads may vary significantly over time, (b) the workload of some applications may change tremendously over a very short period of time, and (c) resource allocation should be optimized for better performance.

Control-based techniques has been recently adopted for autonomic resource management in a virtualized environment. One way of improving the application performance is by controlling its CPU utilization within the VM; a key performance metric is the responsiveness of the controllers designed to workload changes, and especially sudden changes. Control-based approaches have designed controllers, such as Adaptive Kalman filters [5], [6], $\mathcal{H}_\infty$ filters [7] and nonlinear controllers [8], to continuously update the maximum CPU allocation based on CPU utilization measurements.

The focus of this paper is to build controllers that facilitate efficient consolidation as the resource demands of the consolidated applications change over time and are unknown. More specifically, this paper addresses the problem of efficient server consolidation by using an adaptive neuro-fuzzy inference system to adjust the CPU allocations based on observations of past utilization. With respect to existing works in the literature that use fuzzy logic in virtualized environments, our proposed controller differs both in the part of the application being applied as well as the methodology. The work that is closer to ours is by Sithu *et al.* [9] which also uses neuro-fuzzy control for controlling CPU utilization decisions of virtual machines level controllers. However, they do not use data from robust controllers to train their neuro-fuzzy controller; instead, they use CPU load profiles to train the neuro-fuzzy controller to only predict the usage with 100% allocation, without having any CPU allocation scheme to train on. More specifically, the contributions of this work are the following:

- A neuro-fuzzy controller is trained on data sets formed by the observed allocation and utilization when Kalman and $\mathcal{H}_\infty$ filters have been deployed to predict the demanded utilization and hence allocate the necessary resource.
- While the neuro-fuzzy controller is trained on linear filters (Kalman and $\mathcal{H}_\infty$), their operation in nonlinear regimes (i.e., at saturation points) was observed to enable the neuro-fuzzy controller to adapt better to the nonlinearities occurred and hence develop better response than the controllers it is trained on. This effect is highlighted in the evaluation of the performance of the proposed controller.

K. M. Deliparaschos and C. Makarounas are with the Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, Limassol, Cyprus. Emails: {k.deliparaschos,christos.makarounas}@cut.ac.cy.

T. Charalambous is with the Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden. E-mail: themistoklis.charalambous@chalmers.se.

E. Kalyvianaki is with the School of Mathematics, Computer Science and Engineering, Department of Computer Science, City University London, UK. E-mail: evangelia.kalyvianaki.1@city.ac.uk.

The remainder of the paper is organized as follows. In Section II we provide the notation and preliminary results on Fuzzy set theory, which is key ingredient for the development of our results. Section III describes the models adopted for the resource utilization and the *mRT*. In Section IV, neuro-fuzzy controllers trained on Kalman and $\mathcal{H}_\infty$ filters are designed. Section V, the performance of the controllers designed are evaluated and compared with other state-of-the-art. Finally, in Section VI we conclude this work and draw directions for future work.

II. Preliminaries on Fuzzy Control Systems

A. Fuzzy Set Theory

Fuzzy Set Theory, firstly introduced in [10], proposes the fuzzy set concept by extending the two value logic, defined by the binary pair $\{0, 1\}$, to the whole continuous interval $[0, 1]$, introducing a gradual transition from falsehood to truth. A "crisp" set is closely related to its members, such that an item from a given universe of discourse is either a member (1) or not (0). Zadeh, suggested that many sets have more than a yes (1) or no (0) crisp criterion for membership, and he proposed a grade of membership μ in the interval $[0, 1]$, such that the decision whether an element x belongs or not to a set A is gradual rather than crisp (or binary). In other words, fuzzy logic replaces "true or 1" and "false or 0" with a continuous set of membership values ranging in the interval from 0 to 1. If X defines a collection of elements denoted by x, then a fuzzy set A in X is defined as a collection of ordered pairs $A = \{(x, \mu_A(x))|x \in X\}$. The elements x of a fuzzy set are said to belong to a universe of discourse, X, which effectively is the range of all possible values for an input to a fuzzy system. Every element in this universe of discourse is a member of the fuzzy set to some degree. The function $\mu_A(x)$ is called a membership function and assigns a degree of truth in the interval $[0, 1]$ to each element of x of X in the fuzzy set A.

B. System Modeling

Fuzzy modeling is well suited to model systems with complex behavior (see, e.g., [11]), making it suitable for the complex application we consider. The approach makes use of fuzzy set theory (briefly described in Section II-A) to automatically create rules, which are subsequently trained using the input and output of observed data from the system. The trained rules then represent the model of the system behavior and based on that model, inference can be deployed to compute the output for any given input.

III. Model description and formulations

A. mRT model

While there are several metrics for measuring server performance, a representative one is the client mean request response times (*mRT*). The *mRT* of server applications can be divided into three main regions [12]:

(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.

There are well known formulas to calculate the *mRT* of server requests. In this work, we adopt the model of *mRT* used in [7] to account for the characteristics aforementioned and to be able to compare their filters using the metrics used therein. Even though this is not an accurate *mRT* prediction model, it serves as the cost function to evaluate the performance of the designed controllers in a simulated environment. This model is solely based on well known characteristics of server applications as described above and in order to assimilate these characteristics into the system, the authors in [7] proposed the following *mRT* model defined in (1), where the *mRT* is given as a function of the ratio between workload demand d and allocation a:

$$mRT(d/a) = \begin{cases} e^{\gamma(\frac{d}{a}-\phi)}, & \text{if } \frac{d}{a} \leq \phi, \\ 1 + \gamma(\frac{d}{a} - \phi), & \text{if } \frac{d}{a} > \phi. \end{cases} \quad (1)$$

Note that even for the same application running in the same environment, one can get different response times at different time moments (i.e., there is no stationarity). However, this model serves as a benchmark for the performance evaluation and does not affect the policies developed in this paper. Demand d, in terms of CPU, is given by

$$d[k+1] = d[k] - u[k] + i[k+1], \quad (2)$$

where $u[k]$ is the CPU utilization at time instant k and $i[k+1]$ is the CPU from the requests at time instant $(k+1)$. The role of the headroom value ϕ and the constant γ are to assign large response time values in regions close to saturation. 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*. To maintain good server performance, the operators (e.g., in data centers) aim to keep machine CPU utilization below 100% of the machine capacity by a certain value, which is usually called the *headroom*. 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 try to increase the server resources, in order to keep the response times low. Hence, the headroom values can be seen as the boundary between the second and the third *mRT* regions. Equation (1) along with the Root-Mean-Square-Error (RMSE) of the allocation will be used as a metric to evaluate the performance of the virtualized application when its CPU allocation is controlled by our designed controllers and the state-of-the-art used for comparison.

B. The CPU Usage and Allocation

We assume s single-tier server application composed of 1 component on a single VM, but the results can be extended

to N components, where each component runs on a different VM. The time-varying CPU utilization per component is modeled as a random walk (see [5]–[7]) given by the following linear stochastic difference equation:

$$u[k+1] = u[k] + r[k], \tag{3}$$

where $u[k] \in \mathbb{R}_+$ is the percentage of the total CPU capacity actually used by the application component during time-interval k. The independent random process $r[k] \in \mathbb{R}_+$ 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.

By $a[k] \in \mathbb{R}_+$ we denote the CPU capacity of a physical machine allocated to the VM, i.e., the maximum amount of resources a VM can use. By $x[k] \in \mathbb{R}_+$ we denote the total CPU utilization actually *observed* in the VM, i.e., $x[k]$ models the observed application utilization $u[k]$ in addition to any usage noise coming from other sources, such as the operating system, to support the application. The purpose of the any designed controllers is to control the allocation of the VM running a server application while observing its utilization in the VM, maintaining 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:

$$x[k] = ca[k] + v[k], \tag{4}$$

where $c \in (0, 1)$ denotes the *desired* percentage of the allocation being utilized; $v[k] \in \mathbb{R}_+$ denotes the utilization measurement noise.

To maintain good server performance, the allocation should adapt to the utilization. In [5], [6] and in [7] Kalman and $\mathcal{H}_\infty$ filters have been designed respectively, to track the CPU utilization per component of the server and thus allocate the correct amount of resources. Note that the filters do not only track the resource utilization of the system, but control the actual resources to be allocated and are therefore considered as controllers to the system, affecting its performance. For this reason in many cases we also refer to them as controllers. The output of any of these controllers at time k is the predicted value of the utilization, denoted by $y[k]$. An array of the input denoted by $x[0], \ldots, x[T]$ and output denoted by $y[0], \ldots, y[T]$ for some training period T, is used as inputs and outputs, respectively, during the training to our neuro-fuzzy controller. Note that after the training, the input to the controller at time k will be the observed utilization $x[k]$. For simplicity, throughout the paper, we use x as the input and y as the output of the neuro-fuzzy controller.

IV. Controller design

The methodology used in this work is as follows: the fuzzy modeling procedure learns information about a data set (i.e., observed utilization and predicted utilization that determines the CPU allocation), in order to identify the membership function parameters. These parameters, associated with the membership functions, change during the learning process and help the fuzzy inference system to track the given input/output data; this learning method (ANFIS [13]) uses a network-type structure similar to that of neural networks. The neuro-fuzzy inference consists of the following 5 layers (also depicted in Fig. 1):

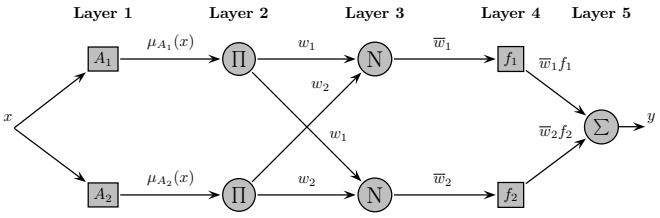

Fig. 1. 5-Layers SISO neuro-fuzzy inference structure.

(a) Layer 1 is the *fuzzification* layer. Each neuron in this layer represents the fuzzy sets expressing the linguistic terms in the antecedents of the fuzzy rules. A fuzzification neuron performs a domain transformation essentially mapping a conventional (crisp) input, x of universe of discourse, X into a fuzzy set interval [0, 1] of a fuzzy set A. A membership function describes the degree of membership of the element, x in the fuzzy set, A, usually denoted by $\mu_A(x)$. For the antecedent part, triangular membership functions were used, i.e.,

$$\mu_A(x; l, c, r) = \begin{cases} 1 - \frac{c-x}{c-l}, & \text{if } l < x \leq c, \\ 1 - \frac{x-c}{r-c}, & \text{if } c < x < r, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

(b) Layer 2 forms the rule layer, where each neuron is analogous to a single Sugeno type fuzzy rule. In the T-S inference rule, the state space of a nonlinear system is divided into different fuzzy regions with a local linear model being used in each region, and hence the conclusion is expressed in the form of linear functions. T-S Single-Input-Single-Output (SISO) rules have the following form:

$$R_i : \text{IF } x \text{ IS } A_i \text{ THEN } y_i, \tag{6}$$

where x is the input variable, A^i is the input fuzzy set, $y^i = c_0^i + c_1^i x_1$, $i = 1, 2, ..., m$ is the output of each rule for m number of rules, and c_k^i are constants. Each rule neuron in Layer 2 receives fuzzified inputs from the fuzzification neurons in Layer 1, to consequently calculate the firing strength of the rule, μ_{ij}. The conjunctions of the rule antecedents are evaluated by the product operator $w_i = \prod_{j=1}^{k} \mu_{ij}$, which reduces to $w_i = \mu_i$, since we use a SISO structure.

(c) Each neuron in the normalization Layer 3 receives inputs from Layer 2 and determines the normalized firing strength or the contribution $\overline{w}_i$ of any given rule to the final result as,

$$\overline{w}_i = \frac{\mu_i}{\sum_{j=1}^{n} \mu_i}, \tag{7}$$

(d) Layer 4 represents the deffuzzification layer, where each neuron calculates a weighted consequent value of a given rule such as,

$$y_i = \sum_{i=1}^{n} \overline{w}_i f_i, \tag{8}$$

where for a first-order model f_i denotes the output of each rule expressed as a first order polynomial $f_i = c_{0i} + c_{1i}x_1 + ... + c_{ji}x_j$, $i = 1, ..., n$ and $j = 1, .., m$. A zero-order model arises (simplified T-S functional reasoning), if we only use constants c_0^i (singletons) at the output. In our case for a SISO zero-order model f reduces to $f_i = c_{0i}$, $i = 1, ..., n$.

(e) Layer 5 sums all outputs (moving singletons) from the defuzzification neurons and produces the overall ANFIS *de-fuzzified* crisp output y of universe of discourse, Y as a weighted average of the contribution of each rule,

$$y = \sum_{i=1}^{n} y_i = \frac{\sum_{i=1}^{n} w_i f_i}{\sum_{i=1}^{n} w_i}, \tag{9}$$

The ANFIS training algorithm uses a hybrid learning algorithm presented in [13]. The algorithm uses the least-squares method in the forward pass to identify the consequent parameters on Layer 4, while In the backwards pass the errors are propagated and the antecedents parameters are updated by gradient descent. The SISO ANFIS is depicted in Figure 1. The fuzzy estimator proposed in this paper is based on the Takagi Sugeno (T-S) zero-order type fuzzy model [11], [14]. It is well known that the T-S fuzzy model can provide an effective representation of complex nonlinear systems in terms of fuzzy sets and fuzzy reasoning. The T-S method is actually quite simple as a method, it leads to fast calculations and it is relatively easy to apply. In general, the fuzzy inference system performs a non-linear mapping from its input space (an input data vector, also referred to as crisp inputs) to its output space (scalar output data, also referred to as crisp output).

The SISO fuzzy estimator used in this work uses only 5 triangular type membership functions on the aggregate part, 5 singletons on the consequence part, and a rule base of 5 rules, maintaining a low complexity of the controller structure. The membership function parameters of the fuzzy inference system were trained for 20 epochs to adopt to the training data from the Kalman and $\mathcal{H}_\infty$ filters. Since we use a SISO fuzzy inference the second layer reduces to a unity gain multiplier and it can be safely omitted further lowering the complexity of the controller.

V. Performance Evaluation

In this section, the performance of the designed fuzzy controllers is evaluated via a simulated virtualized environment built in MATLAB™. A SISO system is used for simplicity, in order to highlight the distinct features of the designed controllers. The performance of the controller is evaluated via the *mRT* using (1) and the RMSE of the allocation.

We compare our controllers against the controllers used for training, i.e., the Kalman controller proposed in [5], [6] and the $\mathcal{H}_\infty$ controller proposed in [7]. All controllers use the same utilization model. Note that the Kalman controller aims to minimize the mean prediction error, while the $\mathcal{H}_\infty$ controller aims to minimize the maximum error. The performance of the fuzzy controllers are evaluated for the cases for which the process $\mathbf{w}_k$ and the measurement noise $\mathbf{v}_k$ are either normally or uniformly distributed.

A. Training data

For the designed fuzzy controllers to work efficiently, the training data set needs to be representative of the data the trained model is intended to emulate. For this reason, we chose a scheme that includes most of the range of values of % CPU demanded and with different rates of changes (see Figure 2).

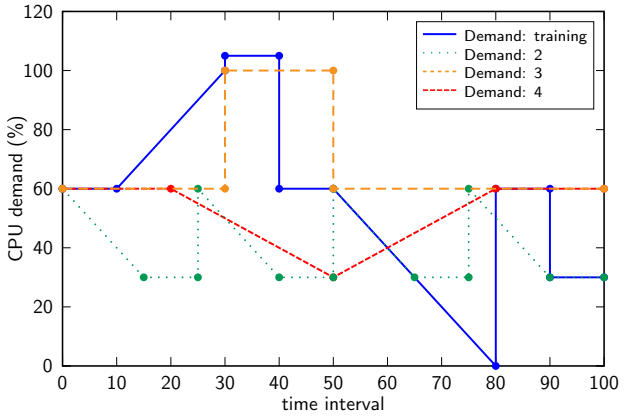

Fig. 2. The % CPU demand (without fluctuations) on which our designed controllers have been trained and tested. The range of values of % CPU demand is $[0, 105]$, wide enough to include most of the range of values of % CPU demanded.

B. Evaluations for demands similar to the training data

Firstly, as a first level of evaluation, we tested the fuzzy controller against data that fluctuate around the training data. At the training, both the Kalman and the $\mathcal{H}_\infty$ filters were not able to follow the required demand, increasing the *mRT* considerably. The neuro-fuzzy controller has been trained on these responses. When we checked the controller on the same exact data, the neuro-fuzzy controller performed slightly better than the Kalman and $\mathcal{H}_\infty$ filters, as shown in the column titled "training" in Table I. When the data changed (i.e., we used a different random vector to create different noise, thus constructing datasets 1, 2 and 3), then the **RMSE** and *mRT* time have been improved. As aforementioned, the neuro-fuzzy controller is able to capture some nonlinearities that the Kalman and $\mathcal{H}_\infty$ filters cannot due to their linear nature. The *mRT* response of the proposed controller for *dataset 2* (shown in Table I) is shown in Figure 3. As it is shown, the workload demand changes are so abrupt that neither the Kalman nor

Dataset	*training*		*dataset 1*		*dataset 2*		*dataset 3*	
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*	**RMSE**	*mRT*	**RMSE**	*mRT*
Kalman	5.0253	19.1831	4.4395	18.1157	4.6042	14.2144	4.4086	16.4540
$\mathcal{H}_\infty$	4.7666	18.1666	4.2344	17.6559	4.7786	12.8696	4.1035	15.3220
Neuro-fuzzy	**4.7237**	**16.403**	**4.1978**	**15.8500**	**4.5843**	**11.6313**	**4.0414**	**14.3288**

TABLE I

RMSE AND *mRT* VALUES FOR WORKLOAD DEMAND SCHEMES SIMILAR TO THE TRAINING SCHEME.

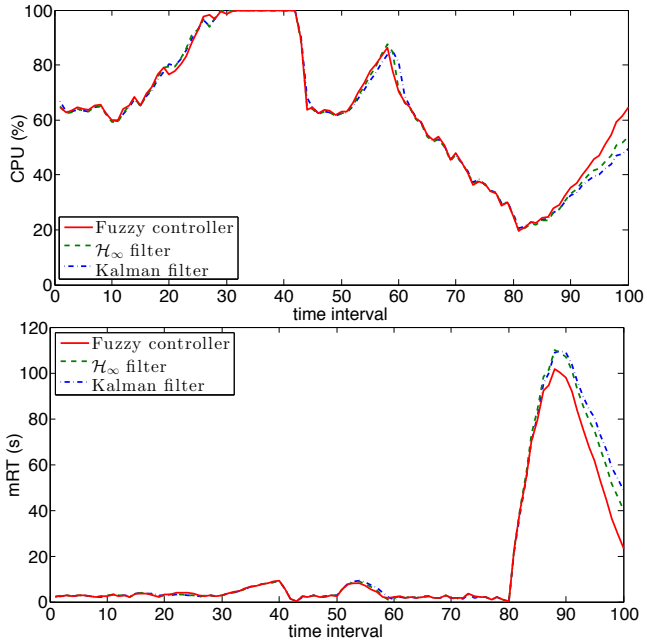

Fig. 3. The CPU allocation and *mRT* response of the proposed controller when evaluated with data (*dataset 2*) that is different than the training data, but follow the same % CPU demand. The neuro-fuzzy controller is trained with the data of a Kalman filter and outperforms both the Kalman and $\mathcal{H}_\infty$ filters.

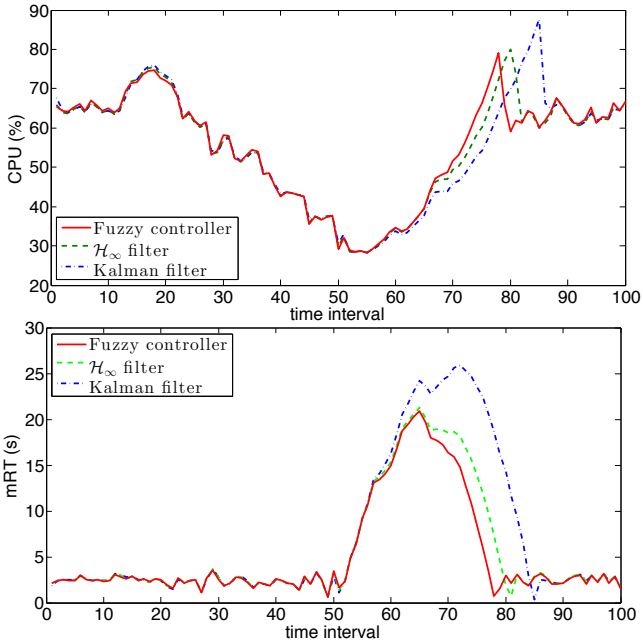

Fig. 4. The CPU allocation and *mRT* response of the proposed controller when evaluated with data (*dataset 4*) that is different than the % CPU demand in the training data. The neuro-fuzzy controller is trained with the data of a Kalman filter first, and overall, it outperforms both the Kalman and $\mathcal{H}_\infty$ filters.

the $\mathcal{H}_\infty$ filters is able to accommodate. The neuro-fuzzy controller responds a bit better by allocating slightly more resources than the other controllers, thus reducing the *mRT* overshoot. Note that the graph shows how well the demand is tracked. *mRT* increases when the allocation provided cannot handle the excess demand, as this is computed via the *mRT* model (1). At time period 30-40 the increasing demand can be initially met, but when the demand exceeds 100% of the available capacity there are not enough resources allocated and the *mRT* increases; however, the limited time there are not enough resources is small and the increase is relatively small. At time period 80-100, the allocation can not track the abrupt changes in demand and as a result a lot of requests are not handled on time, thus increasing the *mRT* considerably.

C. Evaluations for demands distinct to the training data

In order to have a clearer picture of the performance of our controller, the usage demanded throughout the evaluation phase should be sufficiently distinct from the training data set, in order to avoid rendering the evaluation process trivial. In this subsection we evaluate the performance of the proposed controller on different demands that are within the range of training of the controller.

In Figure 4 we compare the CPU allocation and *mRT* response of the proposed controller when evaluated with sawtooth-like data (*dataset 4*) that is different than the % CPU demand in the training data. The neuro-fuzzy controller is trained with the data of a Kalman filter and it is shown to outperform both the Kalman and $\mathcal{H}_\infty$ filters.

Next, we compare the performance of the proposed neuro-fuzzy controller for different datasets (the structure of the datasets is shown in Fig. 2). Table II shows the performance of the neuro-fuzzy controller with respect to the controllers it is trained on. Its performance is in general comparable. It can be slightly worse when there are no saturations due to the very good performance of the controllers in the linear regime. However, when the demand is such that it forces the controllers to saturation, the neuro-fuzzy controller performs better. The numerical results suggest that the neuro-fuzzy controller performs well and in general better at saturation points where fast response is very critical in such applications. However, it would be of great interest to find a way to response even better to abrupt changes and avoid saturation while other controllers, such as neuro-fuzzy controller, are unable to respond satisfactorily.

Distribution	Normal		Uniform	
Workload demand scheme: *training, normal distr.*				
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	4.1035	15.322	4.2672	13.1319
Kalman	4.4086	16.454	4.4074	14.3625
Fuzzy (Kalman)	4.0762	14.2733	4.0593	12.2306
Fuzzy ($\mathcal{H}_\infty$)	4.0464	14.774	4.0557	12.4655
Workload demand scheme: 2				
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	2.9893	5.7186	1.728	2.5317
Kalman	3.5798	7.2353	1.73	2.5241
Fuzzy (Kalman)	3.0416	5.2092	1.7629	2.4882
Fuzzy ($\mathcal{H}_\infty$)	3.2394	5.2848	1.748	2.5264
Workload demand scheme: 3				
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	2.0617	2.501	1.2527	2.4887
Kalman	2.0512	2.499	1.2578	2.4814
Fuzzy (Kalman)	2.0416	2.5287	1.3605	2.4195
Fuzzy ($\mathcal{H}_\infty$)	2.072	2.4985	1.283	2.4809
Workload demand scheme: 4				
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	5.3188	15.4991	4.644	12.1519
Kalman	5.1037	16.3308	4.9031	13.472
Fuzzy (Kalman)	5.2146	15.3073	5.0899	11.4303
Fuzzy ($\mathcal{H}_\infty$)	5.3637	15.2786	4.9622	11.4128
Workload demand scheme: 5				
Filter	**RMSE**	*mRT*	**RMSE**	*mRT*
$\mathcal{H}_\infty$	4.6487	5.0589	4.5682	4.9661
Kalman	4.8048	5.0824	4.7083	4.9897
Fuzzy (Kalman)	4.6263	5.1043	4.5548	5.0056
Fuzzy ($\mathcal{H}_\infty$)	4.6072	5.0975	4.5332	4.9965

TABLE II

RMSE AND *mRT* VALUES FOR DIFFERENT WORKLOAD DEMAND SCHEMES. THE NORMAL DISTRIBUTION IS THE ONE USED FOR THE TRAINING.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

A. Conclusions

Virtualization technologies enable the runtime CPU allocation, where it is of paramount importance to build controllers that adjust the allocation in a timely fashion avoiding resource saturation. An adaptive neuro-fuzzy inference system has been used to adjust the CPU allocations based on observations of past utilization. The neuro-fuzzy controller is, in general, designed for non-linear level processes in order to improve the control performance better than the conventional Kalman and $\mathcal{H}_\infty$ filters that work well only if the model assumptions of the system hold. Hence, it is difficult for the conventional Kalman or $\mathcal{H}_\infty$ filters to operate well as controllers for variable and complex systems. On the contrary, fuzzy logic control does not require any precise mathematical model and works well for complex applications as well, provided the neuro-fuzzy controller is properly trained. The nonlinear mapping in the fuzzy logic generally allows the fuzzy controller to perform better than the linear Kalman or $\mathcal{H}_\infty$ controller. The performance improvement is justified in our simulations for the paradigm of CPU allocations in data centers, in which at nonlinearities due to saturation the neuro-fuzzy controller performs better than the controllers it was trained on.

B. Future Directions

This work is an initial step towards the integration of control and learning in virtualization technologies. Workload consolidation should be performed while taking care of the resource coupling in multi-tier virtualized applications to provide timely allocations during workload changes [5], [6], [15]; i.e., resource needs across multiple dimensions such as compute, storage, IO, network bandwidth, etc. For this reason, current research concentrates on using neuro-fuzzy control to extract the coupling between the resource needs for more efficient workload consolidation. Furthermore, we wish to evaluate our controllers against complex consolidation scenarios in modern multi-core systems with the latest resource schedulers and compare the behavior of the system with the system model proposed herein.

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.

[2] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2002, pp. 181–194.

[3] D. Liu, J. Tai, J. Lo, N. Mi, and X. Zhu, "VFRM: Flash Resource Manager in VMware ESX Server," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–7.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Berkeley, CA, USA: USENIX Association, May 2005, pp. 273–286.

[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] ——, "Adaptive resource provisioning for virtualized servers using Kalman filters," *ACM Transaction on Autonomous and Adaptive Systems*, vol. 9, no. 2, pp. 10:1–10:35, July 2014.

[7] T. Charalambous and E. Kalyviannaki, "A min-max framework for CPU resource provisioning in virtualized servers using $\mathcal{H}_\infty$ Filters," in *49th IEEE Conference on Decision and Control (CDC)*, Dec. 2010, pp. 3778–3783.

[8] 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.

[9] M. Sithu and N. L. Thein, "A resource provisioning model for virtual machine controller based on neuro-fuzzy system," in *2011 The 2nd International Conference on Next Generation Information Technology (ICNIT)*, Jun. 2011, pp. 109–114.

[10] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965.

[11] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. on Syst., Man, and Cyber.*, vol. 15, no. 1, pp. 116–132, Feb. 1985.

[12] L. Kleinrock, *Queueing Systems, Volume 1, Theory*. Wiley-Interscience, 1975.

[13] J.-S. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 3, pp. 665–685, May 1993.

[14] T. Takagi and M. Sugeno, "Derivation of fuzzy control rules from human operator's control actions," in *in Proc. of the IFAC Symp. on Fuzzy Inf. Repr. and Dec. Anal.*, 1984, pp. 55–60.

[15] 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.