2010 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Autonomic Provisioning with Self-Adaptive Neural Fuzzy Control for End-to-end Delay Guarantee

Palden Lama and Xiaobo Zhou
Department of Computer Science
University of Colorado at Colorado Springs, CO 80918, USA
{plama, xzhou}@uccs.edu

***Abstract*—Autonomic server provisioning for performance assurance is a critical issue in data centers. It is important but challenging to guarantee an important performance metric, percentile-based end-to-end delay of requests flowing through a virtualized multi-tier server cluster. It is mainly due to dynamically varying workload and the lack of an accurate system performance model. In this paper, we propose a novel autonomic server allocation approach based on a model-independent and self-adaptive neural fuzzy control. There are model-independent fuzzy controllers that utilize heuristic knowledge in the form of rule base for performance assurance. Those controllers are designed manually on trial and error basis, often not effective in the face of highly dynamic workloads. We design the neural fuzzy controller as a hybrid of control theoretical and machine learning techniques. It is capable of self-constructing its structure and adapting its parameters through fast online learning. Unlike other supervised machine learning techniques, it does not require off-line training. We further enhance the neural fuzzy controller to compensate for the effect of server switching delays. Extensive simulations demonstrate the effectiveness of our new approach in achieving the percentile-based end-to-end delay guarantees. Compared to a rule-based fuzzy controller enabled server allocation approach, the new approach delivers superior performance in the face of highly dynamic workloads. It is robust to workload variation, change in delay target and server switching delays.**

I. Introduction

Popular Internet services employ a complex multi-tier architecture, with each tier provisioning a certain functionality to its preceding tier and making use of the functionality provided by its successor to carry out its part of the overall request processing [2], [3], [4], [5], [6], [19], [20], [23]. Autonomic resource management aims to reduce the degree of human involvement in the management of complex computing systems. It is critical and challenging due to rapidly growing scale and complexity of multi-tier Internet services. Recent research efforts relied on queuing-theoretic approaches [4], [19], [20] and control-theoretic approaches [1], [5] based on explicit system performance models for dynamic resource allocation. However, it is difficult and time consuming to accurately estimate system performance model parameters such as service time, workload distribution, etc. Furthermore, system parameter variation, workload uncertainty and inherent nonlinearity of performance versus resource allocation introduce additional challenges to achieve an accurate system performance model.

End-to-end system delay is the major performance metric of multi-tier Internet applications. It is the response time of a request that flows through a multi-tier computer system [8], [19], [22]. Figure 1 depicts a typical three-tier service architecture. For load sharing, each tier is often replicated and clustered based on server virtualization techniques. The queueing model based approaches in [4], [20] and feedback control based approaches in [1], [5] aim to guarantee the average delay of requests. But they have no control on an important performance metric, percentile-based end-to-end delay of requests. Using the average delay as a performance metric is unable to represent the shape of a delay curve [22]. Percentile-based performance metric such as the 95_{th}-percentile end-to-end delay, compared to the average delay, has the benefit that is both easy to reason about and to capture individual users' perception of Internet service performance [8], [19], [22].

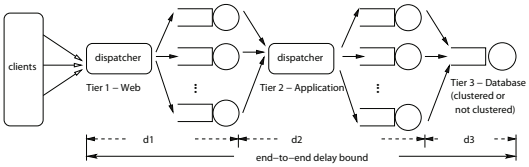


Fig. 1. End-to-end delay in a virtualized multi-tier server cluster.

However, it is very challenging to assure a percentile-based delay guarantee of requests of a multi-tier service. Compared with the average delay, a percentile delay introduces much stronger nonlinearity to the system performance model. Queueing theoretic techniques have achieved noteworthy success in providing average delay guarantee on multi-tier server systems [8], [19]. However, queueing models are mean oriented. Control theoretic techniques were applied to inherently nonlinear Web systems for performance guarantees by performing linear approximation of system dynamics and estimation of system parameters [1]. However, if the deployed system configuration or workload range deviates significantly from those used for system identification, the estimated system model used for control would become inaccurate [14].

The work in [19] proposed an innovative approach for assuring the 95_{th}-percentile delay guarantee. It uses an application profiling technique to determine a service time distribution whose 95_{th}-percentile is the delay bound. The mean of that distribution is used as the average end-to-end delay bound. It then applies the bound for the per-tier delay target decomposition and per-tier server provisioning based on a queueing model. There are two key problems, however.

1526-7539/10 $26.00 © 2010 IEEE
DOI 10.1109/MASCOTS.2010.24

One is that the approach is model dependent. The second is that the application profiling needs to be done offline for each workload before the server replication and allocation. Due to the very dynamic nature of Internet workloads, application profiling itself can be complex and time consuming.

In this paper, we propose an autonomic server allocation approach based on a model-independent neural fuzzy control technique for the percentile based end-to-end delay guarantees in virtualized multi-tier server clusters. Although we use the 95_{th}-percentile for the analysis and case study, note that the approach can be applied to any percentile based delay guarantee. Like others in [7], [15], [19], we consider server virtualization for high resource utilization efficiency and fast server switching. There are model-independent rule based fuzzy controllers that utilize heuristic knowledge for performance assurance [8], [13], [21]. They use a set of pre-defined rules and fuzzy membership functions to perform control actions in the form of resource allocation adjustment. These controllers have some drawbacks. First, they are designed manually on trial and error basis, using heuristic control knowledge. There is no specific guideline for determining important design parameters such as the input scaling factors, the rule base and the fuzzy membership functions. Second, those design parameters are non-adaptive. They are often not effective in the face of highly dynamic workloads. Therefore, we design a novel self-adaptive neural fuzzy controller as a hybrid of control theoretical and machine learning techniques.

The main advantages of the proposed server allocation approach based on neural fuzzy controller are as follows:

1) It is robust to highly dynamic workload variation and change in delay target due to its self-adaptive and self-learning capabilities.
2) It is model-independent. The parameter variations of the system performance and the unpredictability of dynamic workloads do not affect the validity of the proposed server allocation approach.
3) It is capable of automatically constructing the control structure and adapting control parameters through fast online learning. The controller executes resource allocation adjustment and learns to improve its performance simultaneously.
4) Unlike other supervised machine learning techniques, it does not require off-line training. Avoiding off-line training saves significant amount of time and efforts required to collect a large set of representative training data and to train the system.

In addition, we address an important server switching cost issue. Server switching by addition and removal of a virtual server introduces non-negligible latency to a multi-tier service. It affects the perceived end-to-end delay of users. It takes time for a newly added server to adapt to the existing system. For example, an addition of database replica goes through a data migration and system stabilization phase [3]. A removal of a server does not happen instantaneously, since it has to process residual requests of an active session. To compensate for the server switching delay, we perform two enhancements on our neural fuzzy controller. First, we incorporate the effect of server switching with the online parameter learning. Second, we integrate a self-tuning component that adjusts its output to pro-actively compensate for the server switching effect.

For performance evaluation, we build a simulation model. We conduct extensive simulations to evaluate our server provisioning approach, using a synthetic heavy-tailed workload. Simulation results demonstrate the effectiveness of our new approach in achieving the 95_{th}-percentile end-to-end delay guarantee for both stationary and highly dynamic workloads. We perform the sensitivity analysis of our neural fuzzy controller for various delay targets and compare its performance with the rule based fuzzy controller used in [8], [13], [21]. The new neural fuzzy controller delivers consistently better performance for various delay targets. It, on average, outperforms the rule based fuzzy control approach by about 30% and 60% in terms of relative delay deviation and temporal target violation, respectively. We also demonstrate the effect of input scaling factor on the performance of the rule based fuzzy controller for different delay targets. There does not exist one single scaling factor that works best for different scenarios. It demonstrates the need of a self-adaptive controller based on neural fuzzy control. Finally, we show the robustness of the new server provisioning approach to server-switching delays.

The rest of this paper is organized as follows. Section II reviews related work in autonomic resource provisioning. Section III presents the design of self-adaptive neural fuzzy control for dynamic server provisioning. Section IV describes the enhancement in the neural fuzzy controller for server switching delays. Section V presents experimental results and performance evaluation. Concluding remarks and discussion about the future work are given in Section VI.

II. Related Work

Autonomic resource management for performance assurance in multi-tier Internet services is an important and challenging research topic. Recently, there are a few studies on the modeling and analysis of multi-tier servers with queueing foundations [4], [11], [12]. For instance, in [11], an analytical model of a three-tier Web services architecture was presented. Diao et al. described a performance model for differentiated services of multi-tier applications [4]. Per-tier concurrency limits and cross-tier interactions were addressed in the model. Urgaonkar et al. designed an important dynamic provisioning technique on multi-tier server clusters [19]. It sets the per-tier average delay targets to be certain percentages of the end-to-end delay constraint. Based on a queueing model, per-tier server provisioning is executed at once for the per-tier delay guarantees. There is however no guidance about the decomposition of end-to-end delay to per-tier delay targets. It relies on a queueing model with offline application profiling for the 95_{th}-percentile delay guarantee.

Feedback control was used for service differentiation and performance guarantee on Internet servers [1], [5], [8], [13], [14], [21]. For instance, a proportional integral controller based

admission control proxy was developed in [5] to maintain the average end-to-end delay target. An integration of queuing model with feedback control was applied for average response time control of web systems in [17]. However, using the average response time as the performance metric is unable to represent the shape of a response time curve [22]. Moreover, those control techniques suffer from the inaccuracy of modeling dynamic workloads in multi-tier systems. For instance, Lu et al. modeled a controlled Web server with a second order difference equation whose parameters were identified using the least square estimator [14]. The estimation was performed for a certain range and characteristics of workload. However, the estimated system model used for control would become inaccurate if the real workload range deviates significantly from those used for performance model estimation [14].

Fuzzy control was applied for Web performance guarantee due to its appealing feature of model independence. In [21], a fuzzy controller was designed for provisioning guarantee of user-perceived response time of a web page. It demonstrated that due to the model independence, the approach significantly outperforms linear proportional integral controllers. In [8], we designed a fuzzy controller for dynamic server provisioning with end-to-end delay guarantee in a multi-tier server architecture, together with a resource allocation optimization model. Those controllers were designed manually on trial and error basis. Furthermore, important design parameters such as input scaling factors, rule base and membership functions are not adaptive. Our preliminary research found that while those approaches provide performance guarantees under stationary workloads, they are not effective in the face of highly dynamic workloads. In this paper, we design a self-adaptive neural fuzzy controller which is capable of automatically learning its structure and parameters using online measurement of request response time.

Statistical machine learning techniques have been used for measuring the capacity of Internet websites [13], [16], for online hardware reconfiguration [2] and for autonomic resource allocation [18], [23]. For instance, the work in [2] proposed a reinforcement learning approach for autonomic configuration and reconfiguration of multi-tier web systems. In our work, we design a neural fuzzy controller as a hybrid of control theoretical and machine learning techniques for autonomic server allocation. It uses an online learning algorithm to self-construct its structure and adapt its parameters based on live incoming data. This saves significant amount of time and effort required to collect a large set of representative training data and to train the system. Furthermore, it avoids poor performance of a typical online training process due to the incorporation of feedback control.

III. A SELF-ADAPTIVE NEURAL FUZZY CONTROL

Our previous study in [8] found that a rule-based fuzzy control approach for server provisioning provides very good performance under stationary system workloads. It can assure the 95_{th}-percentile end-to-end delay guarantee on a typical three-tier server cluster. However, Internet workloads are often highly dynamic in nature [3], [19]. We conducted simulation of the rule based fuzzy control approach in the face of a highly dynamic workload that is illustrated in Figure 2. Simulation results in Figure 3 show significant deviation of the 95_{th}-percentile end-to-end delay from its pre-specified target 1400 ms. We observe a relative delay deviation and temporal target violation of 47% and 38% respectively. Temporal target violation is a measure of percentage of times when the end-to-end delay target is violated within the measuring time frame. The rule based fuzzy controller is unable to adapt itself to a highly dynamic workload since the rule base and fuzzy membership functions are fixed at the design time through trial and error. Moreover, its performance is sensitive to a statically chosen parameter, the input scaling factor. This problem exists for other rule-based fuzzy control approaches [13], [21]. For autonomic computing in large-scale data centers, self-adaptive server provisioning for performance guarantee is a critical issue. In the following, we design a self-adaptive and self-constructing neural fuzzy controller as a hybrid of control theoretical and machine learning techniques.

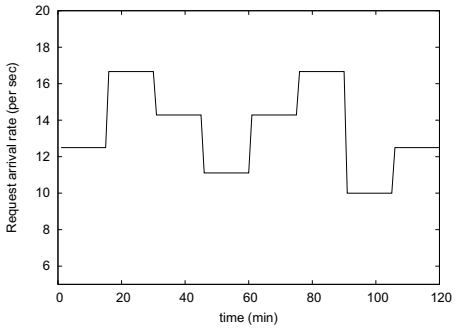


Fig. 2. A highly dynamic workload for a three-tier Internet service.

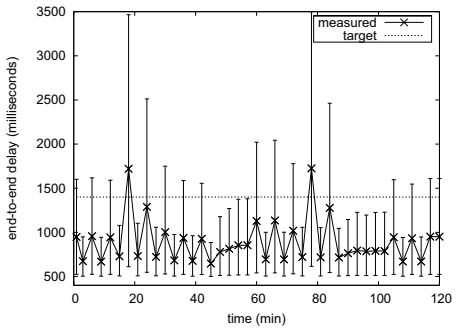


Fig. 3. End-to-end delay variation of a rule-based fuzzy controller.

Figure 4 shows the block diagram of a dynamic server provisioning approach with a self-adaptive neural fuzzy control. The task of the controller is to adjust server provisioning on multi-tier clusters in order to bound the 95_{th}-percentile end-to-end delay T_d to a specified target T_{ref}. The controller has two inputs; error denoted as $e(k)$ and change in error denoted as $\Delta e(k)$. Error is the difference between the target and the measured value of the end-to-end delay in the k^{th} sampling

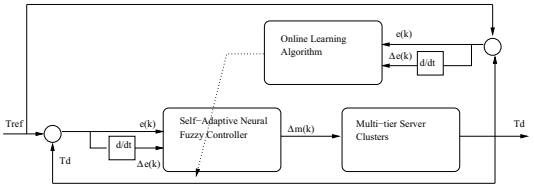


Fig. 4. Block diagram of a self-adaptive neural fuzzy control.

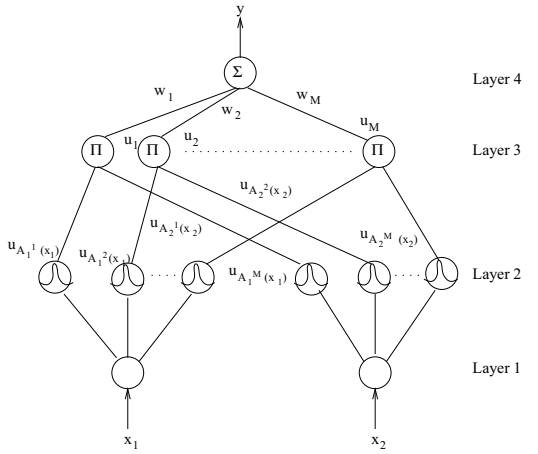


Fig. 5. Schematic diagram of the fuzzy neural network.

period, which is target delay minus measured delay. The output of the controller is the resource adjustment $\Delta m(k)$ for the next sampling period. The controller uses online learning algorithms to automatically constructs its structure and adapts its parameters.

A. Design of neural fuzzy controller

We design the neural fuzzy controller using a general four-layer fuzzy neural network as shown in Figure 5. The various layers of the neural network and their interconnections provide the functionality of membership functions and rule base of a fuzzy controller. Unlike a rule based fuzzy controller, the membership functions and rules dynamically construct and adapt themselves as the neural network grows and learns. Hence, the proposed controller is robust to highly dynamic workload variation. The fuzzy neural network adopts fuzzy logic rules as follows:

R_j: *IF* x_1 *is* A_1^j .. *and* x_n *is* A_n^j, *THEN y is* b_j

where x_i is an input, either to be $e(k)$ or $\Delta e(k)$. y is the rule's output. A_i^j is the linguistic term associated with the i^{th} input variable in the precondition part of the fuzzy logic rule R_j. Linguistic terms are fuzzy values such as "positive small", "negative large", etc. They describe the input variables with some degree of certainty, determined by their membership functions $u_{A_i^j}$. The consequent part or outcome of the rule R_j is denoted as b_j. Each rule contributes to the controller output, denoted as $\Delta m(k)$ according to its firing strength.

The functions of the nodes in each layer are as follows:

Layer 1: Each node in this layer corresponds to one input variable. These nodes only pass the input signal to the next layer. The proposed neural fuzzy controller has two input nodes corresponding to $e(k)$ and $\Delta e(k)$.

Layer 2: Each node in this layer acts as a linguistic term assigned to one of the input variables in layer 1. These nodes use their membership functions to determine the degree to which an input value belongs to a fuzzy set. A Gaussian function is adopted as the membership function as follows.

$$u_{A_i^j} = \exp(-\frac{(x_i - m_{ji})^2}{\sigma_{ji}^2}). \quad (1)$$

Here, m_{ji} and σ_{ji} are the mean and standard deviation of a Gaussian function of the j^{th} linguistic term associated with i^{th} input variable. As shown in Figure 5, let a node represent a linguistic term A_1^1 for the input variable x_1, which is $e(k)$. Assume that its membership function $u_{A_1^1}$ has a mean m_{11} and standard deviation σ_{11} of -50 and 20 respectively. A_1^1 is a fuzzy value such as "negative small", "negative large", etc. that corresponds to the numeric value of -50 with absolute certainty. The degree of certainty is calculated by using the membership function $u_{A_1^1}$. If the measured error in the 95_{th}-percentile end-to-end delay $e(k)$ is -40, the output of the node will be 0.77 from Equation 1. Similarly, let another node represent a linguistic term A_2^1 for the input variable x_2, which is $\Delta e(k)$. Assume that its membership function $u_{A_2^1}$ has a mean and standard deviation of -30 and 10 respectively. If the change in error $\Delta e(k)$ is -30, the output of the node is 1.

Layer 3: Each node in this layer represents the precondition part of one fuzzy logic rule. Each node multiplies the incoming signals and outputs the product result, i.e., the firing strength of a rule. The output of the j^{th} rule node u_j is obtained as follows,

$$u_j = u_{A_1^j} \cdot u_{A_2^j} ... \cdot u_{A_n^j} \quad (2)$$

where n is the number of input variables. The outputs of Layer 2 will be the inputs to this layer. From the previous example, the inputs to a node in this layer are 0.77 and 1. As a result, the output of the node will be 0.77.

Layer 4: This layer acts a defuzzifier, which converts fuzzy conclusions from Layer 3 into numeric output in terms of resource adjustment $\Delta m(k)$. The single node in this layer sums all incoming signals to obtain the final inferred result. That is,

$$y = \sum_{j=1}^{M} w_j \cdot u_j \quad (3)$$

where the link weight w_j is the output action strength associated with the j^{th} rule and y is the output of the neural fuzzy controller. For example, if the link weight w_j is 3, the output $\Delta m(k)$ of this layer will be 2.31 since u_j is 0.77. This result is intuitive because negative values of $e(k)$ and $\Delta e(k)$ imply that the 95_{th}-percentile end-to-end delay is greater than its target and the situation is further worsening. Thus, the neural fuzzy controller allocates more servers to reduce the error. The magnitude of resource adjustment depends on various parameters and interconnections of the neural fuzzy controller,

which are determined and adapted dynamically as described in the next section.

B. Online Learning of Neural Fuzzy Controller

The neural fuzzy controller combines fuzzy logic's reasoning with the learning capabilities of an artificial neural network. It is capable of automatically learning its structure and parameters using online request response time measured from a live system. Initially, there are only input and output nodes in the neural network. The membership and the rule nodes are generated dynamically through the structure and parameter learning processes described as follows.

1. Structure Learning Phase: The structure learning technique decides to add a new node in layer 2 and the associated rule node in layer 3, if all the existing rule nodes have firing strength smaller than a certain degree threshold. Low firing strength of rule nodes imply that the input data pattern of error and change in error is not recognized by the existing neural network. Hence, the neural network needs to grow. We use a decaying degree threshold to limit the size of the neural network. The new node at layer 2 will have a membership function with a mean m_i^{new} equal to the input x_i and standard deviation σ_i^{new} equal to a pre-specified or randomly generated value.

To avoid the newly generated membership function being too similar to the existing one, the similarities between the new membership function and the exiting ones must be checked. We use the similarity measure proposed in [9] to check the similarity of two membership functions. Suppose $u_A(x)$ and $u_B(x)$ are two Gaussian membership functions with means m_A, m_B and standard deviations σ_A, σ_B respectively. Then the similarity measure $E(A,B)$ is given by:

$$E(A,B) = \frac{|A \bigcap B|}{\sigma_A\sqrt{\pi} + \sigma_B\sqrt{\pi} - |A \bigcap B|}. \tag{4}$$

Assuming $m_A \geq m_B$,

$$|A \bigcap B|) = \frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A + \sigma_B))}{\sqrt{\pi}(\sigma_A + \sigma_B)} \tag{5}$$

$$+\frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A - \sigma_B))}{\sqrt{\pi}(\sigma_B - \sigma_A)} \tag{6}$$

$$+\frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A - \sigma_B))}{\sqrt{\pi}(\sigma_A - \sigma_B)}. \tag{7}$$

where $h(x) = max(0,x)$. If the similarity measure between the new membership function and all existing ones are less than a pre-specified value, the new membership function is adopted. Since the generation of a membership function corresponds to the generation of a new fuzzy rule, the link weight, w^{new}, associated with a new fuzzy rule has to be decided. Generally, the link weight is selected with random or pre-specified constant.

The structure learning phase dynamically determines proper input space fuzzy partitions and fuzzy logic rules, depending on the measured error and change in error in the 95_{th}-percentile end-to-end delay. This is in contrast to a rule based fuzzy controller with heuristically designed rules, which uses input scaling factors and a fixed set of membership functions to statically determine the input space fuzzy partitions. Hence, our neural fuzzy controller performs consistently well for a wide range of error and delay targets.

2. Parameter Learning Phase: The parameter learning is used to adaptively modify the consequent part of existing fuzzy rules and the shape of membership functions to improve the controller's performance in the face of highly dynamic workload variation. The goal of performance improvement is expressed as a problem of minimizing an energy function,

$$E = \frac{1}{2}(T_{ref} - T_d)^2 = \frac{1}{2}(e(k))^2 \tag{8}$$

where T_{ref} and T_d are the target and measured values of the 95_{th}-percentile end-to-end delay. The learning algorithm recursively obtains a gradient vector in which each element is defined as the derivative of the energy function with respect to a parameter of the network. This is done by means of the chain rule. The method is referred to as the backpropagation learning rule, because the gradient vector is calculated in the direction opposite to the flow of the output of each node. The parameter learning algorithm based on backpropagation is described in the following.

Layer 4: The error term to be propagated is computed as

$$\delta^4 = -\frac{\delta E}{\delta y} = \left[-\frac{\delta E}{\delta e(k)}\frac{\delta e(k)}{\delta y}\right] = \left[-\frac{\delta E}{\delta e(k)}\frac{\delta e(k)}{\delta T_d}\frac{\delta T_d}{\delta y}\right]. \tag{9}$$

The link weight w_j is updated by the amount

$$\Delta w_j = -\eta_w\frac{\delta E}{\delta w_j} = -\eta_w\frac{\delta E}{\delta y}\frac{\delta y}{\delta w_j} = \eta_w\delta^4 u_j \tag{10}$$

where η_w is the learning rate of the link weight. The weights in layer 4 are updated according to the following equation.

$$w_j(k+1) = w_j(k) + \Delta w_j \tag{11}$$

where k denotes the current sampling interval. Thus, the output action strength or consequence associated with each fuzzy rule is adjusted in order to reduce the error in the 95_{th}-percentile end-to-end delay.

Layer 3: Only the error term needs to be calculated and propagated in this layer. That is

$$\delta_j^3 = -\frac{\delta E}{\delta u_j} = \left[-\frac{\delta E}{\delta y}\right]\left[\frac{\delta y}{\delta u_j}\right] = \delta^4 w_j. \tag{12}$$

Layer 2: The error term is computed as follows,

$$\delta_{ji}^2 = -\frac{\delta E}{\delta u_{A_i^j}} = \left[-\frac{\delta E}{\delta u_j}\right]\left[\frac{\delta u_j}{\delta u_{A_i^j}}\right] = \delta_j^3\frac{u_j}{u_{A_i^j}}. \tag{13}$$

The update law for m_{ji} is

$$\Delta m_{ji} = -\eta_m\frac{\delta E}{\delta m_{ji}} = 2\eta_m\delta_{ji}^2\frac{(x_i - m_{ji})}{(\sigma_{ji})^2}. \tag{14}$$

The update law for σ_{ji} is calculated as

$$\Delta\sigma_{ji} = -\eta_\sigma\frac{\delta E}{\delta\sigma_{ji}} = 2\eta_\sigma\delta_{ji}^2\frac{(x_i - m_{ji})^2}{(\sigma_{ji})^3} \tag{15}$$

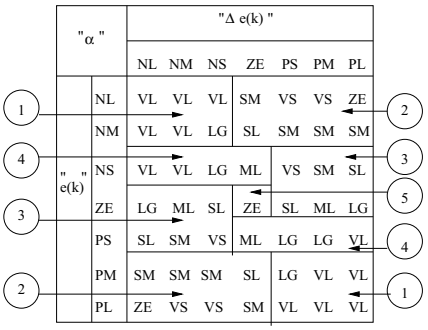


Fig. 6. The fuzzy rule base for scaling factor α.

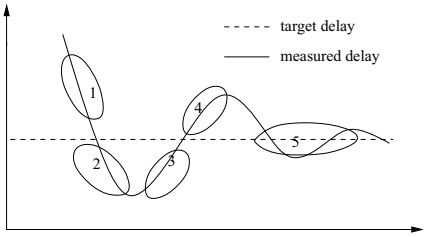


Fig. 7. Fuzzy control effect.

where η_m and η_σ are the learning-rate parameters of the mean and the standard deviation of the Gaussian function, respectively. The mean and standard deviation of the membership functions in this layer are updated as following.

$$m_{ji}(k+1) = m_{ji}(k) + \Delta m_{ji}. \tag{16}$$

$$\sigma_{ji}(k+1) = \sigma_{ji}(k) + \Delta \sigma_{ji}. \tag{17}$$

Thus, the position and the shape of the membership functions are adjusted dynamically. The exact calculation of the Jacobian of the system, $\frac{\delta T_d}{\delta y}$ in Eq. (9), cannot be determined due to the unknown dynamics of the multi-tier server clusters. To overcome this problem, we apply a delta adaptation law proposed in [10] as follows,

$$\delta^4 \equiv e(k) + \Delta e(k). \tag{18}$$

The proof of the convergence of the neural fuzzy controller using Eq. (18) is similar to that in [10] and is omitted here.

IV. Enhancements on Neural Fuzzy Controller

One major challenge in controlling a physical process is its inherent process delay. For the dynamic server provisioning process, it is the latency between allocating servers and measuring the effect of the server provisioning on the end-to-end delay. To compensate for the server switching delay, we propose two enhancements for the neural fuzzy controller.

The first enhancement is on the parameter learning phase. In the neural fuzzy controller, the parameter adjustment depends on the measured error in the 95_{th} percentile delay, current weights and outputs of the fuzzy neural network nodes at various layers. However, due to the server switching delay, the current measurement of delay error may actually be caused by the weights and outputs of the neural fuzzy controller that existed a few sampling intervals earlier. In our enhancement, we store the weights and outputs of the neural fuzzy controller at each sampling interval. After a few sampling intervals equivalent to the server switching delay, the stored values are utilized for parameter learning using back propagation. This enhancement ensures that the controller's parameters are adjusted considering the effect of server switching.

We further enhance the neural fuzzy controller by integrating a self-tuning component that adjusts its output to pro-actively compensate for the server switching effects. We introduce an output scaling factor α in the range [0,1]. It is multiplied by the output of the neural fuzzy controller to determine the actual adjustment in server allocation.

Figure 6 shows the rule base for the scaling factor controller α. The rule base is designed to perform on-line gain variation of the neural fuzzy controller based on instantaneous behavior of the system. The table shows the rules corresponding to various regions of the system behavior as shown in Figure 7. The preconditions of a rule is described by the linguistic values of "e(k)" and "Δe(k)", such as NL, NM, NS, ZE, PS, PM, and PL. They stand for negative large, negative medium, negative small, zero, positive small, positive medium and positive large respectively. The outcome of a rule is described by linguistic values of α, such as ZE, VS, SM, SL, ML, LG and VL. They stand for zero, very small, small, small large, medium large, large and very large. These rules are applied only to adjust the scale of the neural fuzzy controller's output. The granularity of output in terms of server provisioning is still determined by the self-adaptive neural fuzzy controller. A few important considerations for the rule design are as follows:

1) When the error is large but has the same sign as the change in error, α should be made very large to prevent from further worsening the situation. This will amplify the corrective action suggested by the neural fuzzy controller in terms of server provisioning.
2) If the server switching delay is high, the controller may not achieve expected output after allocating required number of servers, and hence, may overreact by assigning too many servers in the next sampling period. In such situations, usually the error is big but has the opposite sign as compared to the change in error. This is compensated by adjusting the output scaling factor to a small value.
3) To improve the controller performance under load disturbance, α should be sufficiently large around the steady state. For example, if the error is small and has the same sign as a large change in error, α should be large to bring the system back to steady state within a short time.
4) At a steady state, when the error is small and the change in error is also small, α should be very small to avoid oscillations around the equilibrium point.

V. Performance Evaluation

We evaluate the server provisioning approach based on the self-adaptive neural fuzzy control in a typical three-tier

TABLE I
WORKLOAD CHARACTERISTICS.

Parameter	WebTier	AppTier	DBTier
s_i	20 ms	294 ms	254 ms
σ_i^2	848	2304	1876

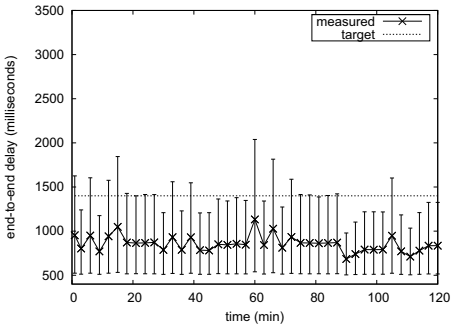


Fig. 8. End-to-end delay assurance for a dynamic workload.

server cluster with extensive simulations. As others in [3], we assume that the database tier can be replicated on-demand as it employs a shared architecture. We generate a synthetic G/G/1 workload using Pareto distributions of request inter-arrival time and service time. Pareto distribution representing a heavy-tailed traffic has close resemblance to real Internet traffic that is bursty in nature [8], [24]. We choose the workload characteristics of a three-tier application reported in [19]. Table I gives the characteristics. s_i and σ_i^2 are the average service time and the variance of service time distribution of requests at tier i, respectively. The workload is measured periodically on a "control interval" of 3 minutes. Each representative result reported is an average of 100 runs.

We use two performance metrics, relative deviation as in [21] and target violation. Relative deviation is based on square root mean of delay errors. It reflects the transient characteristics of a control system and measures how closely the 95_{th}-percentile delay of requests follows a given target for n sampling intervals. That is,

$$R(e) = \frac{\sqrt{\sum_{k=1}^{n} e(k)^2/n}}{T_{ref}}. \quad (19)$$

The relative deviation, however, does not differentiate whether the actual end-to-end delay is greater than or less than the target. It is indeed desirable that an actual end-to-end delay is less than the target. To measure the temporal violation of delay target, we define a metric of target violation

$$T(v) = \frac{\sum_{k=1}^{n} v(k)}{n} \quad (20)$$

where $v(k)$ is one if the actual end-to-end delay is greater than the target T_{ref}, and zero if it is less than or equal to T_{ref}.

A. Effectiveness of Neural Fuzzy Control Approach

We evaluate the effectiveness of the new approach for performance guarantee under both dynamic and stationary workloads. First, we use the highly dynamic workload shown in Figure 2 and set the end-to-end delay bound to 1400 ms.

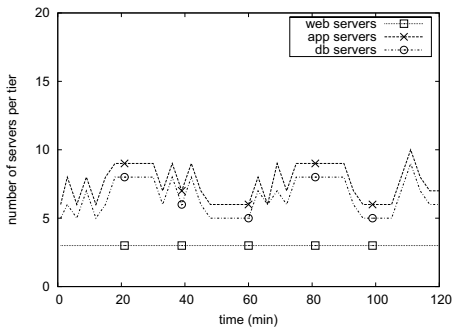


Fig. 9. Server allocation for a dynamic workload.

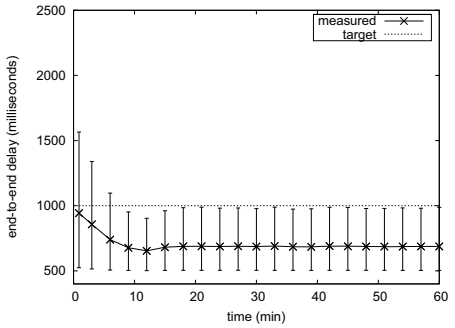
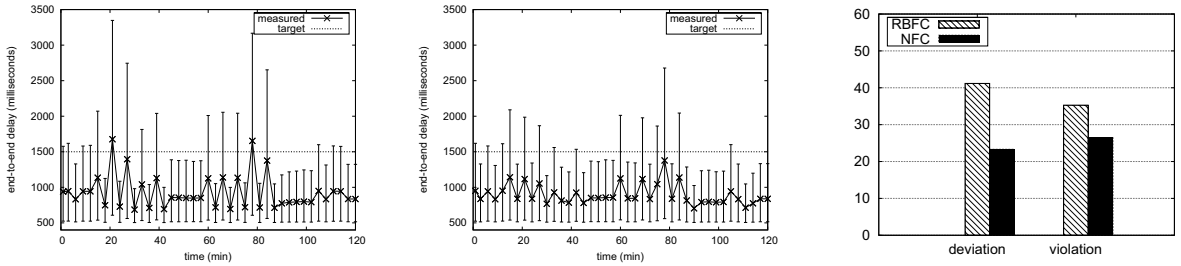


Fig. 10. End-to-end delay assurance for a stationary system workload.

Figure 8 demonstrates the effectiveness of the neural fuzzy controller in assuring the 95_{th}-percentile end-to-end delay guarantee. Figure 9 shows the corresponding server allocation. Note that the number of servers allocated to the web tier remains fixed. This is due to the workload characteristics used in Table I. The web tier has relatively small resource demand compared to the application and database tiers. Hence, the controller allocates more servers to the application tier and database tier. New server provisioning approach achieves a small relative delay deviation of 14% and target violation of 17% respectively. This is a significant improvement from the performance of the rule based fuzzy controller for the same workload scenario in Figure 3, where the relative delay deviation and the target violation are 47% and 38% respectively.

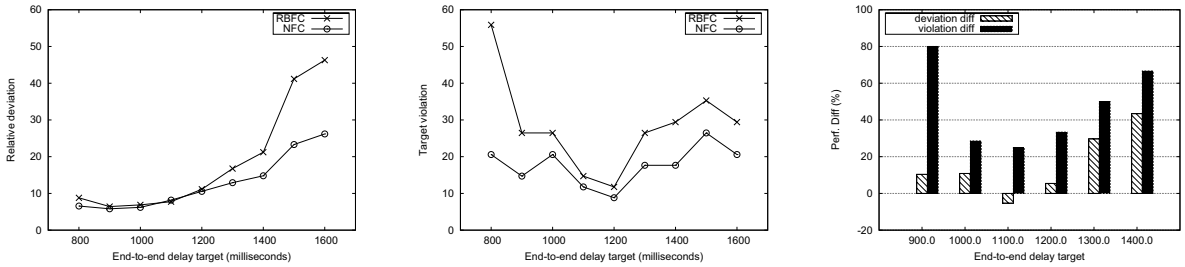
The neural fuzzy controller is robust to highly dynamic workload variation due to its self-adaptive capability. There are a few spikes in the end-to-end delay due to sudden changes in the applied workload. However, the neural fuzzy controller achieves the delay guarantee in a very responsive manner.

Next, we apply a stationary workload with an average request arrival rate of 12 requests per second. Figures 10 and 11 show the end-to-end delay variation and changes in server allocation. The neural fuzzy controller is able to guarantee the 95_{th}-percentile delay target of 1000 ms within a few sampling intervals, in spite of the fact that the controller starts its operation with an empty structure. This is due to its capability to self-construct its structure and to adjust its parameters through fast online learning algorithm.



(a) with rule based fuzzy control. (b) with neural fuzzy control. (c) performance comparison.

Fig. 12. End-to-end delay assurance for a dynamic workload (target 1500 ms).



(a) relative delay deviation. (b) temporal target violation. (c) performance difference.

Fig. 13. Performance comparison for various delay targets.

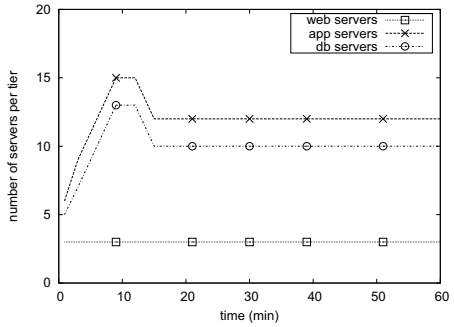


Fig. 11. Server allocation for a stationary system workload.

B. Comparison With Rule Based Fuzzy Controllers

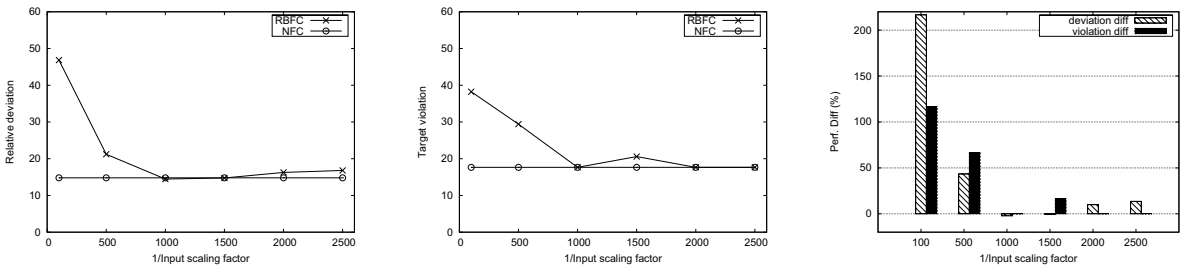
A rule based fuzzy controller has shown its merits in achieving performance assurance through model-independent resource allocation and dynamic output scaling factor tuning [8], [21]. However, it shows inconsistent delay guarantee and significantly more target violations in case of highly dynamic workloads. That is mainly due to the fact that the rule based fuzzy controller applies statically chosen input scaling factor, rule base and membership functions that are manually tuned for a particular workload and a delay target.

We compare the performance of our neural fuzzy controller with a rule based fuzzy controller used in [8], [21]. We choose an input scaling factor of 1/500 as it shows good performance under a stationary workload for the rule based fuzzy controller. Figures12 shows that the self-adaptive neural fuzzy controller (NFC) is more robust to the dynamic workload variation compared to the rule based fuzzy controller (RBFC) in assuring the 95_{th}-percentile end-to-end delay guarantee (1500 ms). NFC outperforms RBFC by 76% and 33% in terms of the relative delay deviation and the target violation respectively. Its robustness to highly dynamic workloads is due to the self-adaptive and self-learning capabilities.

Next, we conduct sensitivity analysis of two controllers for various end-to-end delay targets. Figure 13(a) shows that the relative delay deviation tends to increase with the increase in the end-to-end delay target (from 800 ms to 1600 ms). This is due to the fact that larger delay targets require few servers for allocation, making it more difficult to achieve fine-grained control on the 95_{th}-percentile end-to-end delay. As shown in Figure 13(b), the temporal target violation is small for medium range of delay targets between 1000 ms to 1400 ms. The delay targets higher than this range show more target violation due to a small number of servers involved in the control action. The targets in the lower range also results in larger target violation, due to the fact that a controller takes more control intervals to reach very low delay targets. Compared to the rule based fuzzy controller, the new neural fuzzy controller consistently achieves less delay deviation and target violation for various delay targets. For quantitative comparison, we take the performance of NFC as a baseline and define the performance difference between the NFC and RBFC as

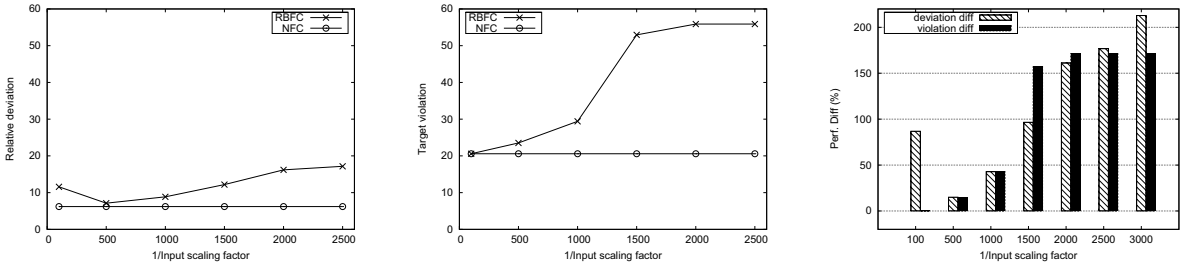
$$PD_{deviation} = \frac{R(e)_{RBFC} - R(e)_{NFC}}{R(e)_{NFC}} \quad (21)$$

$$PD_{violation} = \frac{T(v)_{RBFC} - T(v)_{NFC}}{T(v)_{NFC}} \quad (22)$$



(a) relative delay deviation. (b) temporal target violation. (c) performance difference.

Fig. 14. Performance comparison for various input scaling factors with delay target 1400 ms.



(a) relative delay deviation. (b) temporal target violation. (c) performance difference.

Fig. 15. Performance comparison for various input scaling factors with delay target 1000 ms.

If PD is positive, the NFC has better performance than RBFC and vice versa. Figure 13 (c) shows that NFC outperforms RBFC for all delay targets except 1100 ms. For that case, the rule based fuzzy controller has slightly better performance in terms of delay deviation because it is well suited for that particular delay target. On average, NFC performs better than RBFC by 32% and 59% in terms of delay deviation and target violation respectively. The main reason is due to the fact that it adapts itself to accommodate various range of inputs instead of relying on statically chosen input scaling factor.

C. Effect of Input Scaling Factor

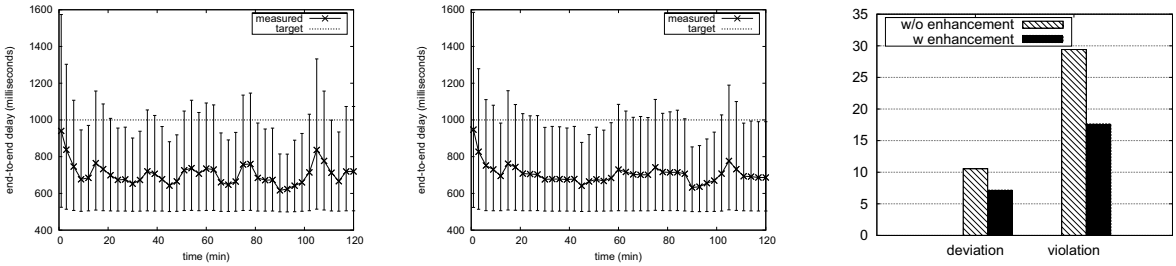
We now study the impact of the input scaling factor on the performance of the rule based fuzzy controller, and compare its performance with our neural fuzzy controller. Figures 14 and 15 show their relative delay deviation, target violation and performance difference for end-to-end delay targets 1400 ms and 1000 ms respectively. Results demonstrate that increasing the scaling factor may improve the performance of the rule based fuzzy controller for one delay target (1400 ms), but it may degrade the performance for another delay target (1000 ms). In both cases, the performance of the neural fuzzy controller is consistently better than the rule based control.

The rule based fuzzy controller is sensitive to the choice of the input scaling factor, which attempts to partition the input fuzzy space non-adaptively. In practice, a highly dynamic and realistic workload increases the possibility that the inputs to the fuzzy controller (i.e., error and change in error) may not fit into the input space fuzzy partitions as intended. A change in the end-to-end delay target further worsens the situation. Hence, there does not exist one single scaling factor that works best for different scenarios. Since the rule base and fuzzy membership functions are also fixed at the design time through trial and error, the rule based fuzzy controller is unable to adapt itself to a highly dynamic workload. Thus, we need a self-adaptive controller designed based on neural fuzzy control. The main reason behind the superior performance of the neural fuzzy controller in assuring the end-to-end delay guarantee is its self-adaptive and online learning capability as compared to trial and error based design of the rule based fuzzy controller.

D. Effect of the Server Switching Delay

We demonstrate the impact of the two control enhancements designed in Section IV on the performance of the neural fuzzy controller. We assume the times taken by addition and removal of one virtual server at any tier of an application are 16 seconds and 8 seconds respectively. Due to the server switching delays, the controller may not achieve the expected output after adding or removing servers from a multi-tier cluster. Hence, it may overreact by assigning or removing too many servers in the next control interval. This results in large overshoot and undershoot of the 95_{th}-percentile end-to-end delay from the given target as illustrated in Figure 16 (a).

Then, we compensate the server switching effect by the two enhancements on the neural fuzzy controller. Figure 16 (b) and (c) show that the integrated neural fuzzy controller provides more consistent assurance of the 95_{th}-percentile end-to-end delay guarantee. The improvements in relative delay deviation and target violation are 47% and 66% respectively. It is due to the fact that the enhancements consider the effect of server switching delay to learn the parameters and to adaptively change the output of the neural fuzzy controller.



(a) without the enhancements. (b) with the enhancements. (c) performance comparison.

Fig. 16. End-to-end delay assurance with neural fuzzy control for a target of 1000 ms due to server switching delays.

VI. Conclusion

In this paper, we have designed a novel self-adaptive neural fuzzy control based server provisioning approach to guarantee the 95_{th}-percentile end-to-end delay of requests flowing through a multi-tier server cluster. The major contributions lie in the design and evaluation of a model-independent and self-adaptive control system for dynamic server provisioning. We combine the strength of both machine learning and control theoretic techniques for robust performance assurance of Internet applications in the face of highly dynamic and unpredictable workloads. We further enhance the neural fuzzy controller to compensate for the effect of server switching delays.

Simulation results demonstrate that the neural fuzzy controller is robust to highly dynamic workloads and changes in delay target. Compared to the rule based fuzzy controller, it shows superior performance in achieving the end-to-end delay assurance. While the simulations were conducted for 95_{th}-percentile end-to-end delay assurance, it can be easily tailored for the average and other percentile delay targets. Importantly, the neural fuzzy control demonstrated its promise of being a self-adaptive approach for autonomic computing in virtualized data centers. Our future work will be on the implementation and evaluation of the approach in a prototype data center.

Acknowledgement: This research was supported in part by NSF CAREER award CNS-0844983 and grant CNS-0720524.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Trans. on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system auto-configuration. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 670–673, 2009.

[3] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content Web servers. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2006.

[4] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, M. Surendra, and A. Tantawi. Modeling differentiated services of multi-tier web applications. In *Proc. IEEE Int'l Symp. on Modeling, Analysis, and Simulation table of contents (MASCOTS)*, 2006.

[5] A. Kamra, V. Misra, and E. M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2004.

[6] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *Proc. ACM WWW*, 2006.

[7] A. Kochut and K. Beaty. On strategies for dynamic resource management in virtualized server environments. In *Proc. IEEE Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2007.

[8] P. Lama and X. Zhou. Efficient server provisioning for end-to-end delay guarantee on multi-tier clusters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2009.

[9] C. Lin and C. S. Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Trans. on Computers*, 40(12):1320–1336, 1991.

[10] F.-J. Lin, R.-J. Wai, and C.-C. Lee. Fuzzy neural network position controller for ultrasonic motor drive using push-pull dc-dc converter. *Control Theory and Applications*, 146(1):99–107, 1999.

[11] X. Liu, J. Heo, and L. Sha. Modeling 3-tiered web applications. In *Proc. IEEE Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[12] X. Liu, J. Heo, L. Sha, and X. Zhu. Queueing-model-based adaptive control of multi-tiered web applications. *IEEE Transactions on Network and Service Management*, 5(3):157–167, 2008.

[13] X. Liu, L. Sha, and Y. Diao. Online response time optimization of apache web server. In *Proc. Int'l Workshop on Quality of Service (IWQoS)*, 2003.

[14] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. Feed back control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. on Parallel and Distributed Systems*, 17(9):1014–1027, 2006.

[15] D. A. Menascé and M. N. Bennani. Autonomic virtualized environments. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2006.

[16] J. Rao and C. Xu. Online measurement of the capacity of multi-tier websites using hardware performance counters. In *Proc. IEEE Int'l Conference on Distributed Computing Systems (ICDCS)*, 2008.

[17] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queueing model based network server performance control. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, 2002.

[18] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2006.

[19] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1):1–39, 2008.

[20] D. Villela, P. Pradhan, and D. Rubenstein. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. on Internet Technology*, 7(1):1–23, 2007.

[21] J. Wei and C.-Z. Xu. eQoS: provisioning of client-perceived end-to-end QoS guarantee in Web servers. *IEEE Trans. on Computers*, 55(12):1543–1556, 2006.

[22] M. Welsh and D. Culler. Adaptive overload control for busy Internet servers. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[23] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier Internet applications. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2007.

[24] X. Zhou, J. Wei, and C.-Z. Xu. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, pages 88–97, 2004.