Autonomic Provisioning with Self-Adaptive Neural Fuzzy Control for Percentile-Based Delay Guarantee

PALDEN LAMA and XIAOBO ZHOU, University of Colorado, Colorado Springs

Autonomic server provisioning for performance assurance is a critical issue in Internet services. It is challenging to guarantee that requests flowing through a multi-tier system will experience an acceptable distribution of delays. The difficulty is mainly due to highly dynamic workloads, the complexity of underlying computer systems, and the lack of accurate performance models. We propose a novel autonomic server provisioning approach based on a model-independent self-adaptive Neural Fuzzy Control (NFC). Existing model-independent fuzzy controllers are designed manually on a trial-and-error basis, and are often ineffective in the face of highly dynamic workloads. NFC is 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. We further enhance NFC to compensate for the effect of server switching delays. Extensive simulations demonstrate that, compared to a rule-based fuzzy controller and a Proportional-Integral controller, the NFC-based approach delivers superior performance assurance in the face of highly dynamic workloads. It is robust to variation in workload intensity, characteristics, delay target, and server switching delays. We demonstrate the feasibility and performance of the NFC-based approach with a testbed implementation in virtualized blade servers hosting a multi-tier online auction benchmark.

Categories and Subject Descriptors: D.4.8 [**Operating Systems**]: Performance-Modeling and prediction

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Resource allocation, multi-tier internet services, percentile-based delay guarantee, neural fuzzy control, self-adaptation, server virtualization

ACM Reference Format:
Lama, P. and Zhou, X. 2013. Autonomic provisioning with self-adaptive neural fuzzy control for percentile-based delay guarantee. *ACM Trans. Auton. Adapt. Syst.* 8, 2, Article 9 (July 2013), 31 pages.
DOI:http://dx.doi.org/10.1145/2491465.2491468

1. INTRODUCTION

Today, 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 [Urgaonkar et al. 2008]. Autonomic resource management for Internet services aims to reduce the degree of human involvement in the management of complex computing systems [Huebscher and McCann 2008]. It is critical to performance assurance and challenging due to rapidly growing scale and complexity of multi-tier Internet services. Dynamic server provisioning with virtualization is essential to

---

A preliminary version of the article appeared in Lama and Zhou [2010].
This research was supported in part by NSF CAREER award CNS-0844983 and research grant CNS-1217979.
Authors' address: P. Lama and X. Zhou (corresponding author), University of Colorado, Colorado Springs, CO; email: {plama, xzhou}@uccs.edu.

DOI:http://dx.doi.org/10.1145/2491465.2491468

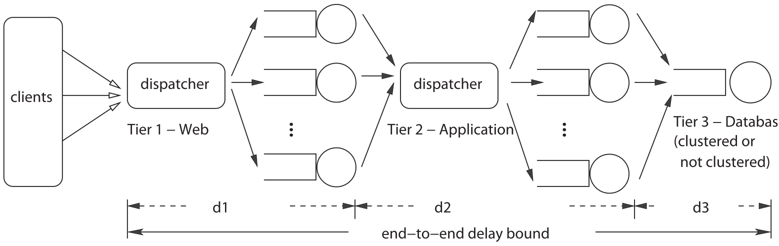


Fig. 1. End-to-end delay in a virtualized multi-tier Internet service architecture.

the performance assurance of multi-tier Internet services and resource utilization efficiency of the underlying computer systems. Recent research efforts relied on queuing-theoretic approaches [Padala et al. 2009; Urgaonkar et al. 2008; Villela et al. 2007] and control-theoretic approaches [Kamra et al. 2004] based on explicit system performance models for dynamic server allocation in multi-tier systems. However, it is complex and time consuming to accurately estimate system performance model parameters such as service time, workload distribution, etc. Furthermore, system parameter variation of virtual servers, nonstationary workload, and inherent nonlinearity of performance versus resource allocation introduce additional challenges to achieve an accurate system performance model [Litoiu 2007].

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 [Urgaonkar et al. 2008]. Figure 1 depicts a typical three-tier Internet service architecture. For load sharing, a tier is often replicated and clustered based on server virtualization techniques. The queueing-model-based approaches and feedback-control-based approaches can guarantee the average delay of requests. But they have no control on an important performance metric, percentile-based end-to-end delay of requests. A percentile-based performance metric such as the 95_{th}-percentile end-to-end delay is easy to reason about and to capture individual users' perception of Internet service performance [Mi et al. 2008; Welsh and Culler 2003].

It is very challenging to assure a percentile-based delay guarantee of requests of a multi-tier Internet 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. However, most queueing models are mean oriented. It is costly to compute various moments of the distribution of response times from a queueing model. 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. 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 [Lu et al. 2006]. Recent studies have seen highly dynamic workloads of Internet services that fluctuate over multiple time scales, which can have a significant impact on the processing demands imposed on data center servers [Mi et al. 2008, 2009; Singh et al. 2010]

In this article, we propose an autonomic server allocation approach based on a model-independent neural fuzzy controller for percentile-based end-to-end delay guarantee in virtualized multi-tier server clusters. There are model-independent rule-based fuzzy controllers that utilize heuristic knowledge for performance guarantee on Internet servers [Lama and Zhou 2009; Liu et al. 2003; Wei and Xu 2006]. They use a set of predefined 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 nonadaptive. They are not effective in the face of highly dynamic workloads. To overcome the drawbacks, 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 the neural fuzzy controller are as follows.

(1) It is robust to highly dynamic workload intensity as well as characteristics 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 and effectiveness 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 that learn from the supervised training data, it does not require offline training. Avoiding offline training saves significant amount of time and efforts required to collect a large set of representative training data and to train the system.

Furthermore, we address an important server switching cost issue. Server switching by addition and removal of a virtual server introduces nonnegligible latency to a multitier 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 [Chen et al. 2006]. 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 proactively compensate for the server switching effect.

For performance evaluation of the proposed server provisioning approach, we build a simulation model and conduct extensive simulations using synthetic heavy-tailed workloads. Simulation results demonstrate the effectiveness of the new approach in achieving the end-to-end delay guarantee for both stationary and highly dynamic workloads. We apply dynamic workloads with sudden step-changes similar to what was used in Urgaonkar et al. [2008] and with continuous changes similar to what was used in Chen et al. [2006]. Although our approach is designed in a way that is applicable to any percentile-based end-to-end delay guarantees, the experiments are conducted for assuring the 95_{th}-percentile and the median end-to-end delay targets. Results show that our approach is effective in provisioning percentile-base delay guarantee. We perform the sensitivity analysis of the neural fuzzy controller for various delay targets and compare its performance with the rule-based fuzzy controller used in Lama and Zhou [2009, 2012a]. The 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 impact of the 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 self-adaptivity based on the neural fuzzy control. Then, we show the robustness of the new server provisioning approach to server switching delays and study the impact of the control interval on the responsiveness of control actions. Furthermore, we demonstrate that our approach outperforms a commonly used proportional integral-control-based server provisioning technique in the face of dynamically varying workload intensity as well as characteristics.

Finally, we conduct a feasibility study with performance evaluation of the proposed server provisioning approach based on a testbed implementation of a virtualized server cluster hosting RUBiS application [Amza et al. 2002; RUBiS 2013], a multi-tier online auction Web site benchmark. The testbed is built on a cluster of HP ProLiant BL460C G6 blade server modules using VMware virtual machines. Experimental results show that the self-adaptive neural fuzzy-control-based server provisioning approach is able to assure the percentile-based end-to-end delay guarantee in the face of a highly dynamic workload with varying intensity and characteristics.

The rest of this article is organized as follows. Section 2 reviews related work in autonomic resource provisioning. Section 3 presents the design of self-adaptive neural fuzzy control for dynamic server provisioning. Section 4 describes the enhancement in the neural fuzzy controller for server switching delays. Section 5 presents simulation results and performance evaluation. Section 6 presents the case study based on a testbed implementation. Concluding remarks and discussions about the future work are given in Section 7.

2. 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 [Bennani and Menasce 2005; Diao et al. 2006; Karve et al. 2006; Liu et al. 2008; Singh et al. 2010; Stewart et al. 2007; Urgaonkar et al. 2005, 2008]. Diao et al. [2006] described a performance model based on $M/M/1$ queueing for differentiated services of multi-tier applications. Per-tier concurrency limits and cross-tier interactions were addressed in the model. Villela et al. [2007] studied optimal server allocation in the application tier that increase a server provider's profits. An optimization problem is constructed in the context of a set of application servers modeled as $M/G/1$ processor sharing queueing systems. Urgaonkar et al. designed a dynamic server provisioning technique on multi-tier server clusters [Urgaonkar et al. 2008]. The technique decomposes the per-tier average delay targets to be certain percentages of the end-to-end delay constraint. Based on a $G/G/1$ 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. Lama and Zhou [2009, 2012a] proposed an efficient server provisioning approach on multi-tier clusters based on an end-to-end resource allocation optimization based on a $M/G/1$ queueing model. It is able to minimize the number of virtual servers allocated to the system and satisfy the average end-to-end response time guarantee under stationary workloads. Singh et al. [2010] proposed a novel dynamic provisioning technique that handles both the nonstationarity in the workload and changes in request volumes when allocating server capacity in data centers. It is based on the k-means clustering algorithm and a $G/G/1$ queuing model to predict the server capacity for a given workload mix. These queueing-model-based techniques can provide the average response-time-based performance guarantee, but not the percentile-based.

Feedback control was used for service differentiation and performance guarantee on Internet servers [Abdelzaher et al. 2002; Jung et al. 2010; Kamra et al. 2004;

Leite et al. 2010; Liu et al. 2003; Lu et al. 2006; Padala et al. 2009; Wei and Xu 2006]. Sha et al. [2002] proposed to integrate, queuing model with feedback control for average response time control of Web systems. Kamra et al. [2004] designed a Proportional Integral (PI)-controller-based admission control proxy to bound the average end-to-end delay in a three-tier Web service. There are studies that argue model-dependent control techniques may 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. The estimated system model used for control would become inaccurate if the real workload range deviates significantly from those used for performance model estimation [Lu et al. 2006]. Padala et al. proposed AutoControl, a combination of an online model estimator and a multi-input multi-output controller [Padala et al. 2009]. The resource allocation system can automatically adapt to workload changes in a shared virtualized infrastructure to achieve the average response-time-based service-level objectives. The approaches can provide performance guarantees regarding the average response, but not percentile-based response time.

A percentile-based performance metric has the benefit that it is both easy to reason about and to capture individual users' perception of Internet service performance [Lama and Zhou 2009; Leite et al. 2010; Urgaonkar et al. 2008; Watson et al. 2010; Welsh and Culler 2003]. Welsh and Culler [2003] proposed to bound the 90_{th}-percentile response time of requests in an multistage Internet server. It is achieved by an adaptive admission control mechanism that controls the rate of request admission. The mechanism complements but does not apply to dynamic server provisioning in data centers.

Urgaonkar et al. [2008] proposed an interesting 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 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. One is that the approach is queueing 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 time consuming and importantly not adaptive online.

Lama and Zhou [2009] proposed a fuzzy-control-based server provisioning approach that can effectively bound the 95_{th}-percentile response time of an Internet service. The technique works well under stationary system workloads. However, it does not effectively adapt to the very dynamic nature of Internet workloads. The study in Singh et al. [2010] found that the nonstationarity in Internet application workloads can have a significant impact on the overall processing demands imposed on data center servers.

Leite et al. [2010] applied an innovative stochastic approximation technique to estimate the tardiness quantile of response time distribution, and coupled it with a Proportional-Integral-Derivative (PID) feedback controller to obtain the CPU frequency for single-tier servers that will maintain performance within a specified deadline. It is nontrivial to apply this approach to dynamic server allocation problem due to several reasons. First, it does not compensate for the effect of process delay in resource allocation, which is significant due to server switching costs. Second, it applies a PID controller without using any system performance model despite the fact that it is essentially a model-based control technique. The controller was designed solely based on response time measurement and manual tuning of controller parameters for a particular simulated workload. As a result, it may not be adaptive to highly dynamic workloads.

Watson et al. [2010] proposed an unique approach to model the probability distributions of response time, in terms of percentiles, based on CPU allocations on virtual machines. The performance model was obtained by offline training based on data collected from the system. It is not adaptive online to dynamically changing workloads. The work focuses on performance modeling without addressing issues related to adaptive resource provisioning such as process delay, system stability, performance assurance, etc.

Fuzzy theory and control were applied for Web performance guarantee due to its appealing feature of model independence, and used to model uncertain and imprecise information in applications [Zhou and Huang 2009]. Liu et al. [2003] used fuzzy control to determine an optimal number of concurrent child processes to improve the Apache Web server performance. Wei and Xu [2006] designed a fuzzy controller for provisioning guarantee of user-perceived response time of a Web page. Those fuzzy controllers were designed manually on a trial-and-error basis. Important design parameters such as input scaling factors, rule base, and membership functions are not adaptive. They are not very effective in the face of highly dynamic workloads. In this article, we design a self-adaptive neural fuzzy controller which is capable of automatically learning its structure and parameters using online measurement.

Statistical machine learning techniques have been used for measuring the capacity of Web sites [Liu et al. 2003; Rao and Xu 2011], for online system reconfiguration [Bu et al. 2009], and for resource allocation [Meng et al. 2010; Tesauro et al. 2006; Zhang et al. 2007]. For instance, Bu et al. [2009] proposed a reinforcement learning approach for autonomic configuration and reconfiguration of multi-tier Web systems.

In this article, 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 a significant amount of time and efforts 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.

3. A SELF-ADAPTIVE NEURAL FUZZY CONTROL

Our previous study in Lama and Zhou [2009] 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 three-tier server cluster. However, Internet workloads are often highly dynamic in nature [Mi et al. 2008, 2009; Singh et al. 2010]. We conducted simulation of the rule-based fuzzy control approach in the face of a highly dynamic workload that is illustrated in Figure 2. It is sudden step-changes based, similar to a workload scenario used in Urgaonkar et al. [2008].

Simulation results in Figure 3 show significant deviation of the 95_{th}-percentile end-to-end delay from its prespecified target 1400 ms. We observe a relative delay deviation and temporal target violation of 47% and 38%, respectively. Relative delay deviation is the ratio of the square root mean of delay errors to the end-to-end delay target. Temporal target violation is a measure of the percentage of times when the end-to-end delay target is violated within the measuring time frame. In the figure, the top of the error bar is the measured 95_{th}-percentile end-to-end delay, the bottom of the error bar is the measured 5_{th}-percentile end-to-end delay, and the data point is the mean.

The rule-based fuzzy controller is unable to adapt itself to the 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

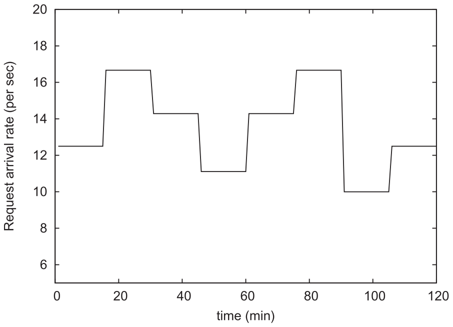


Fig. 2. A highly dynamic workload for a three-tier Internet service.

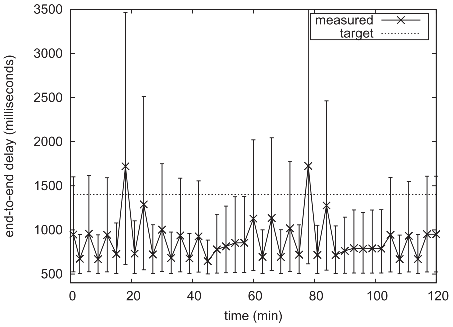


Fig. 3. End-to-end delay variation of a rule-based fuzzy controller.

fuzzy control approaches as well [Liu et al. 2003; Wei and Xu 2006]. For autonomic resource management and performance guarantee of Internet services, self-adaptive server provisioning is a critical and challenging issue.

We design a self-adaptive and self-constructing neural fuzzy controller as a hybrid of control-theoretical and machine learning techniques. Figure 4 shows the block diagram of the 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 percentile-based 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 percentile-based end-to-end delay in the k^{th} sampling period, which is the target delay minus the measured delay. The output of the controller is the server resource adjustment $\Delta m(k)$ for the next sampling period.

The decomposition of server resource adjustment $\Delta m(k)$ to the multiple tiers is performed in proportion to the per-tier delay observed from the controlled system. We choose per-tier delay rather than per-tier utilization because it directly affects the end-to-end response time. As the number of servers to be allocated to each tier can be a real value by the decomposition approach, there are two options. The first is at fine granularity. That is, a server capacity can be reconfigured according to the real value at runtime with the modern server virtualization technology. However, it requires that the hosting physical machine has available resources for the virtual server resizing. Otherwise, it may require server migrations, which are often costly [Isci et al. 2010; Jung et al. 2010]. The second is at the coarse granularity of a whole server. It uses the nearest integer value and the minimum is one. This option is feasible when it can

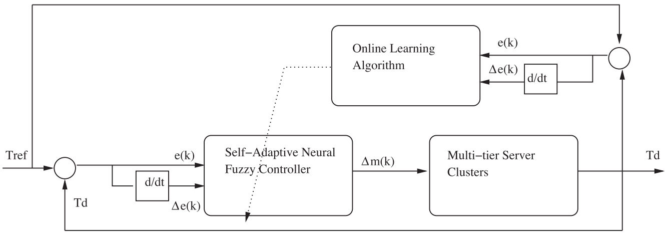

Online Learning Algorithm
e(k)
Δe(k)
d/dt
Tref
e(k)
d/dt
Td
Δe(k)
Self–Adaptive Neural Fuzzy Controller
Δm(k)
Multi–tier Server Clusters
Td

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

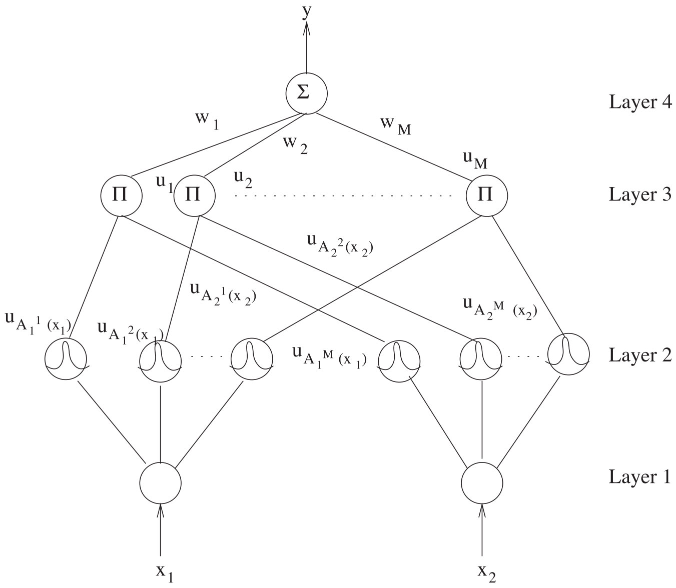


Fig. 5. Schematic diagram of the fuzzy neural network.

find a physical machine for the server replication. In a data center, the administrator can use either option or the hybrid. As related works in Lama and Zhou [2009, 2012a] and Urgaonkar et al. [2008], we adopt the second option in this work. But the neural fuzzy-control-based approach supports either option and the hybrid as well.

The neural fuzzy controller is designed to tolerate 5% error within the end-to-end delay target. As long as the error in delay is within the tolerance bound, it stops further control actions. The controller uses an online learning algorithm to automatically construct its structure and adapt its parameters. Online learning is a category of machine learning techniques which learns one instance at a time from each data collected from the system. This makes it suitable to be deployed at runtime.

3.1. 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. Table I summarizes important notations.

Table I. Notation Summary

Symbol	Description
T_{ref}	The percentile-based end-to-end delay target
T_d	The measured percentile-based end-to-end delay
k	Index of the control sampling interval
i	Index of the input variable in a fuzzy logic rule
j	Index of the fuzzy membership function
r	Index of the fuzzy logic rule
$e(k)$	Error for the control input
$\Delta e(k)$	Change in error for the control input
$\Delta m(k)$	The resource adjustment for the server system

The fuzzy neural network adopts fuzzy logic rules as follows.

R_r: IF x_1 is A_1^j .. and x_n is A_n^j, THEN y is b_r

Here R_r is the r_{th} fuzzy logic rule, x_i is an input, either to be $e(k)$ or $\Delta e(k)$, and y is the rule's output. A_i^j is the j_{th} linguistic term associated with the i^{th} input variable in the precondition part of the fuzzy logic rule R_r. 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_r is denoted as b_r. 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 $\Delta e(k)$. The net input and net output for the i^{th} node are

$$net_i^{(1)} = x_i,\ y_i^{(1)} = f_i^{(1)}(net_i^{(1)}) = net_i^{(1)}. \tag{1}$$

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. For the j^{th} fuzzy membership node, the net input and net output are

$$net_{ji}^{(2)} = -\frac{(x_i - m_{ji})^2}{\sigma_{ji}^2} \tag{2}$$

$$y_{ji}^{(2)} = u_{A_i^j} = f_j^{(2)}(net_{ji}^{(2)}) = \exp(net_{ji}^{(2)}). \tag{3}$$

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. They determine the position and shape of the input membership functions. 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 percentile-based end-to-end delay $e(k)$ is -40, the output of the node will be 0.77 from Eq. (2). 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, that is, the firing strength of a rule. For the r^{th} fuzzy rule node,

$$net_r^{(3)} = u_{A_1^j} \cdot u_{A_2^j} \dots \cdot u_{A_n^j}, \tag{4}$$

$$y_r^{(3)} = u_r = f_r^{(3)}(net_r^{(3)}) = net_r^{(3)}, \tag{5}$$

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 net input and the net output will be 0.77.

Layer (4). This layer acts a defuzzifier. It 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. The net input and net output are

$$net^{(4)} = \sum_{r=1}^{M} w_r \cdot u_r, \tag{6}$$

$$y^{(4)} = f^{(4)}(net^{(4)}) = net^{(4)}, \tag{7}$$

where the link weight w_r is the output action strength associated with the r^{th} rule and $y^{(4)}$ is the output of the neural fuzzy controller. For example, if the link weight w_r is 3, the output $\Delta m(k)$ of this layer will be 2.31 since u_r is 0.77. This result is intuitive because negative values of $e(k)$ and $\Delta e(k)$ imply that the percentile-based 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.

Note that the control framework can be extended for integration with the utility computing paradigm when needed. The error term $e(k)$ in the controller can be replaced by a utility function that captures the cost-benefit trade-off of server allocations.

3.2. 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 and described as follows.

3.2.1. Structure Learning Phase. For each input node in layer (1), 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 strengths 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 nodes at layer (2) have membership functions with a mean m_{ji}^{new} equal to the input x_i and standard deviation σ_{ji}^{new} equal to a prespecified or a 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 existing ones must be checked. We use the similarity measure proposed in Lin and Lee [1992] 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

$$E(A,B) = \frac{|A\bigcap B|}{\sigma_A\sqrt{\pi} + \sigma_B\sqrt{\pi} - |A\bigcap B|}. \tag{8}$$

Without loss of generality, assuming $m_A \geq m_B$,

$$|A\bigcap B|) = \frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A + \sigma_B))}{\sqrt{\pi}(\sigma_A + \sigma_B)}, \tag{9}$$

$$+\frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A - \sigma_B))}{\sqrt{\pi}(\sigma_B - \sigma_A)}, \tag{10}$$

$$+\frac{1}{2}\frac{h^2(m_B - m_A - \sqrt{\pi}(\sigma_A - \sigma_B))}{\sqrt{\pi}(\sigma_A - \sigma_B)}, \tag{11}$$

where $h(x) = max(0,x)$. In the case of scenario $\sigma_A = \sigma_B$,

$$|A\bigcap B|) = \frac{1}{2}\frac{h^2(m_B - m_A + \sqrt{\pi}(\sigma_A + \sigma_B))}{\sqrt{\pi}(\sigma_A + \sigma_B)}. \tag{12}$$

If the similarity measure between the new membership function and the existing ones corresponding to either input variable is less than a prespecified value, both new membership functions are adopted. Since the generation of membership functions in layer (2) 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 initialized with a random or prespecified value. The neural fuzzy controller applies an intuitive understanding of the system behavior for initializing the link weight. If the measured error in delay $e(k)$ is positive, the link weight is initialized with a randomly chosen small negative number and vice versa. For instance, a positive value of $e(k)$ indicates that the observed delay is smaller than the target delay. It is reasonable to reduce the number of servers allocated to the system by initializing a negative w^{new}. Thus, the controller takes corrective actions in the right direction from the beginning although the magnitude of the link weight has not been fully learned.

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 percentile-based 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, the neural fuzzy controller performs consistently well for a wide range of error and delay targets.

3.2.2. 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

$$E = \frac{1}{2}(T_{ref} - T_d)^2 = \frac{1}{2}(e(k))^2, \tag{13}$$

where T_{ref} and T_d are the target and measured values of the percentile-based 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 the chain rule method. The method is referred to as the backpropagation learning rule as the gradient vector is calculated in the direction opposite to the flow of the output of each node. The backpropagation learning algorithm is described as follows.

Layer (4). The error term to be propagated is computed as follows.

$$\delta^{(4)} = -\frac{\partial E}{\partial y^{(4)}} = \left[-\frac{\partial E}{\partial e(k)}\frac{\partial e(k)}{\delta y^{(4)}}\right] = \left[-\frac{\partial E}{\partial e(k)}\frac{\partial e(k)}{\partial T_d}\frac{\partial T_d}{\partial y^{(4)}}\right] \tag{14}$$

The link weight w_r is updated by the amount

$$\Delta w_r = -\eta_w \frac{\partial E}{\partial w_r} = \left[-\eta_w \frac{\partial E}{\partial y^{(4)}}\right]\frac{\partial y^{(4)}}{\partial net^{(4)}}\frac{\partial net^{(4)}}{\partial w_r} = \eta_w \delta^{(4)} u_r, \tag{15}$$

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

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

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 percentile-based end-to-end delay.

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

$$\delta_r^{(3)} = -\frac{\partial E}{\partial net_r^{(3)}} = \left[-\frac{\partial E}{\partial y^{(4)}}\right]\left[\frac{\partial y^{(4)}}{\partial net^{(4)}}\frac{\partial net^{(4)}}{\partial y_r^{(3)}}\frac{\partial y_r^{(3)}}{\partial net_r^{(3)}}\right] = \delta^{(4)} w_r. \tag{17}$$

Layer (2). The error term is computed as follows.

$$\delta_{ji}^{(2)} = -\frac{\partial E}{\partial net_{ji}^{(2)}} = \left[-\frac{\partial E}{\partial net_r^{(3)}}\right]\left[\frac{\partial net_r^{(3)}}{\partial y_{ji}^{(2)}}\frac{\partial y_{ji}^{(2)}}{\partial net_{ji}^{(2)}}\right] = \delta_r^{(3)} y_r^{(3)} = \delta_r^{(3)} u_r \tag{18}$$

The update law for m_{ji} is

$$\Delta m_{ji} = -\eta_m \frac{\partial E}{\partial m_{ji}} = 2\eta_m \delta_{ji}^{(2)} \frac{(x_i - m_{ji})}{(\sigma_{ji})^2}. \tag{19}$$

The update law for σ_{ji} is calculated as

$$\Delta \sigma_{ji} = -\eta_\sigma \frac{\partial E}{\partial \sigma_{ji}} = 2\eta_\sigma \delta_{ji}^{(2)} \frac{(x_i - m_{ji})^2}{(\sigma_{ji})^3}, \tag{20}$$

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.

$$m_{ji}(k+1) = m_{ji}(k) + \Delta m_{ji} \tag{21}$$

$$\sigma_{ji}(k+1) = \sigma_{ji}(k) + \Delta \sigma_{ji} \tag{22}$$

Thus, the position and the shape of the membership functions are adjusted dynamically. The exact calculation of the Jacobian of the system, $\partial T_d/\partial y^{(4)}$ in Eq. (14), 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 Lin et al. [1999] as follows.

$$\delta^{(4)} \equiv e(k) + \Delta e(k) \tag{23}$$

The proof of the convergence of the neural fuzzy controller using Eq. (23) is similar to that in Lin et al. [1999] and is omitted here.

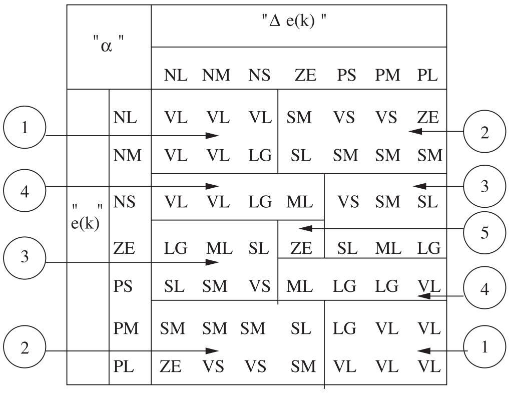
4. ENHANCEMENTS ON NEURAL FUZZY CONTROLLER

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 percentile-based end-to-end delay. This latency is caused by various factors including but not limited to the time required to start up new servers (say virtual machines). For example, long-lived workload sessions may continue to only use existing servers due to the locality issue while newly added servers are already started up, which would increase the measured server switching time. Furthermore, newly added servers do not process new session requests as quickly as the existing servers due to the cache warmup time. 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 percentile-based end-to-end 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 backpropagation. 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 proactively 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 online 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 that is shown in Figure 7. The preconditions of a rule are 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, respectively. These rules are applied only to adjust the scale of the neural fuzzy



"α"	"Δ e(k)"						
"e(k)"	NL	NM	NS	ZE	PS	PM	PL
NL	VL	VL	VL	SM	VS	VS	ZE
NM	VL	VL	LG	SL	SM	SM	SM
NS	VL	VL	LG	ML	VS	SM	SL
ZE	LG	ML	SL	ZE	SL	ML	LG
PS	SL	SM	VS	ML	LG	LG	VL
PM	SM	SM	SM	SL	LG	VL	VL
PL	ZE	VS	VS	SM	VL	VL	VL

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

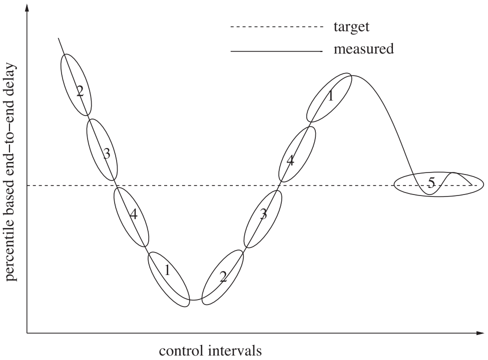


Fig. 7. Fuzzy control effect.

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 the 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. We assume that a new server is fully started within a single control interval. However, the overall server switching delay can be much longer than the control interval.
(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.

Table II. Workload Characteristics A

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

When a server startup time is longer than the control interval, the neural fuzzy controller suspends its control actions until the newly allocated server is fully started. The rationale is that the interim period can show very unstable and unpredictable behavior no matter how good the control policy.

5. PERFORMANCE EVALUATION

We evaluate the server provisioning approach based on the self-adaptive neural fuzzy control in a typical three-tier server cluster with extensive simulations. In simulations, as others in Chen et al. [2006] we assume that the database tier can be replicated on-demand as it employs a shared architecture. The controlled multi-tier system has a number of servers preallocated initially. For choosing the initial server allocation, we first run a number of simulations for various server allocations to observe the percentile-based end-to-end delay in the face of a stationary workload of 12 requests per second. Then, we choose the server allocation that provided the end-to-end delay close to a target of 1500 ms. The setting is 1 Web server, 6 application servers, and 5 database servers. We use the same initial server allocation for all simulations to demonstrate that the dynamic provisioning approach based on the neural fuzzy controller is able to guarantee the percentile-based end-to-end delay target when the workload and delay targets vary dynamically. In practice, the initial server allocation in the controlled system may be decided by a data center administrator based on best practices. The research in Meng et al. [2010] proposed an interesting data center capacity planning approach that exploits statistical multiplexing among the workload patterns of multiple virtual machines. The research in the initial data center capacity planning is complementary to our research in dynamic server provisioning.

We generate a synthetic workload using Pareto distributions of request inter-arrival time and service time, following a $G/G/1$ FCFS queueing model. Pareto distribution representing a heavy-tailed traffic has close resemblance to real Internet traffic that is bursty in nature [Lama and Zhou 2009; Zhou et al. 2004]. Although the workload is generated according to a specific model and requests are processed in a specific principle, our self-adaptive neural fuzzy controller itself does not make such assumptions. We choose the workload characteristics of a three-tier application reported in Urgaonkar et al. [2008]. Table II gives the characteristics, in which s_i and σ_i^2 are the average service time and the variance of service time distribution of requests at tier i, respectively. We apply dynamic request arrival rates with sudden step-changes similar to what was used in Urgaonkar et al. [2008] and with continuous changes similar to what was used in Chen et al. [2006]. Each representative result reported is an average of 100 runs.

We use two performance metrics, relative delay deviation as in Wei and Xu [2006] and target violation. Relative delay deviation is based on the square root mean of delay errors. It reflects the transient characteristics of a control system and measures how closely the percentile-based delay of requests follows a given target for n sampling intervals. That is,

$$R(e) = \frac{\sqrt{\sum_{k=1}^{n} e(k)^2/n}}{T_{ref}}. \tag{24}$$

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

$$T(v) = \frac{\sum_{k=1}^{n} v(k)}{n}, \tag{25}$$

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}.

The neural network itself is empty initially. As a new node is added into the network, the neural fuzzy controller will choose the sign of the preconfigured link weight according to the observed error in delay. If the measured error in delay is positive, the link weight is initialized with a randomly chosen small negative number from the range of [-6, -3] and vice versa. Choosing initial link weight in the range [-6, -3] or [3, 6] is to attenuate the resource allocation adjustments made by the neural fuzzy controller at its initial phase. The controller will allocate or deallocate at least three and at most six virtual servers each iteration. As a result, the controller is able to take corrective actions right from the beginning although the magnitude of the link weight has not been fully learned. The neural network grows as the values of error and change in error are calculated based on the measured end-to-end delay, and the magnitude of the link weight is updated by the fast online learning.

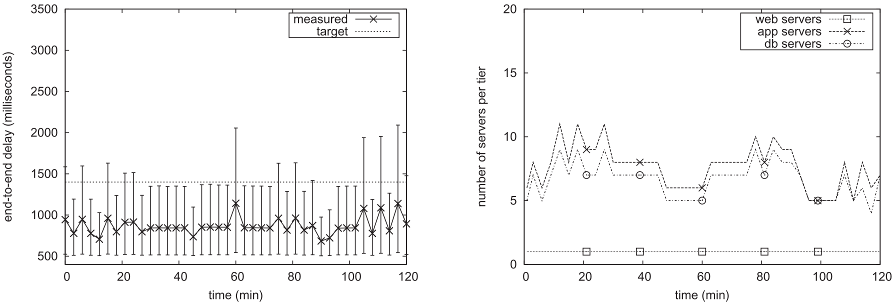
We initialize the neural fuzzy controller parameters as follows. When a new node is added in layer (2), its membership function has a standard deviation σ_{ji} equal to a prespecified value 20. The standard deviation of a fuzzy membership function determines the range of input values that will trigger a particular fuzzy rule. Since neural network parameters associated with the newly added node are not yet learned, it needs to restrict the effect of the new node to a smaller range of input values. The learning rates η_w, η_m, and η_{sigma} are set to 0.8 for aggressive initial learning. As the neural network grows in size, the learning rate is gradually decreased in proportion to the number of rule nodes for fast convergence.

The end-to-end delay of the system is measured periodically at a control interval of 3 minutes. The choice of control interval depends on the trade-off between responsiveness of the controller and robustness to measurement noise. The controller applies an exponential moving average method to further reduce the transient noise in delay measurement.

5.1. Effectiveness of Neural Fuzzy Control Approach

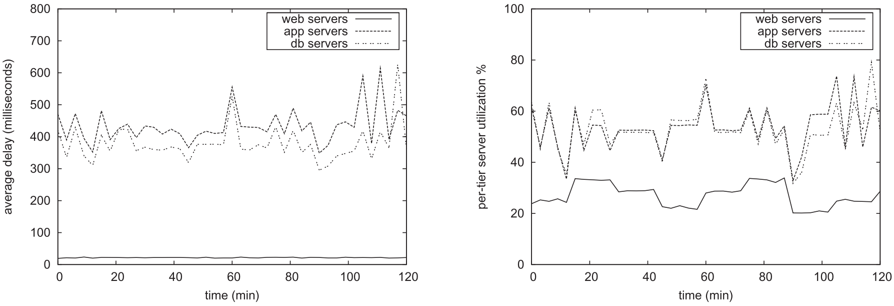
We evaluate the effectiveness of the new approach for performance guarantee under both dynamic and stationary workloads. First, we use a highly dynamic workload with sudden step-changes in request arrival rate as shown in Figure 2. We set the end-to-end delay bound to 1400 ms.

Figure 8(a) demonstrates the effectiveness of the neural fuzzy controller in assuring the 95_{th}-percentile end-to-end delay guarantee. The extent of server allocation and deallocation is crude at the beginning when the controller has gone through very few learning steps. The effect of adding or removing servers in such cases is usually not very good. Results show that during the first 24 minutes, the measured end-to-end delay oscillates significantly around the target delay of 1400 ms with the control actions. However, in some cases, a single control action may be able to bring the end-to-end delay very close to the target depending upon the existing server allocation. For example, at time 60_{th} minute, adding 2 application servers and 2 database servers to the system is just enough to bring the end-to-end delay close to the target.



(a) end-to-end delay assurance (b) server allocation

Fig. 8. End-to-end performance of neural fuzzy control for a dynamic workload.



(a) per-tier average delay (b) per-tier utilization

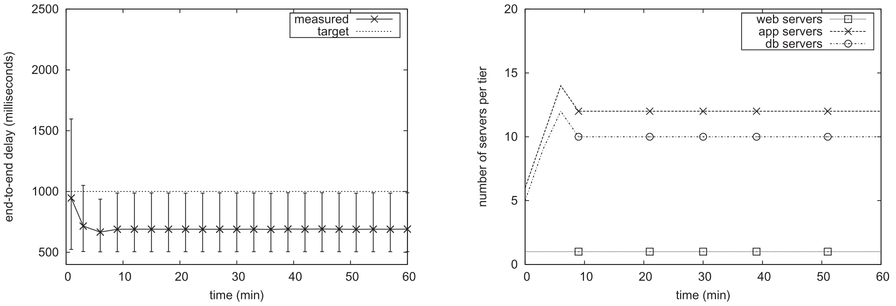
Fig. 9. Per-tier performance of neural fuzzy control for a dynamic workload.

Figure 8(b) shows the corresponding server allocations. The controller stops the control action as long as the error in the end-to-end delay is within the 5% delay tolerance bound. This improves the system stability. For example, during 30_{th} minute and 45_{th} minute, there is no change in the server allocation. Note that the number of servers allocated to the Web tier remains the same. This is due to the workload characteristics used in Table II. 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.

Figures 9(a) and (b) show the per-tier delay and server utilization under the highly dynamic workload. The per-tier delay of requests as well as server utilization vary across tiers mainly due to the characteristics of per-tier resource demands of the given workload as shown in Table II.

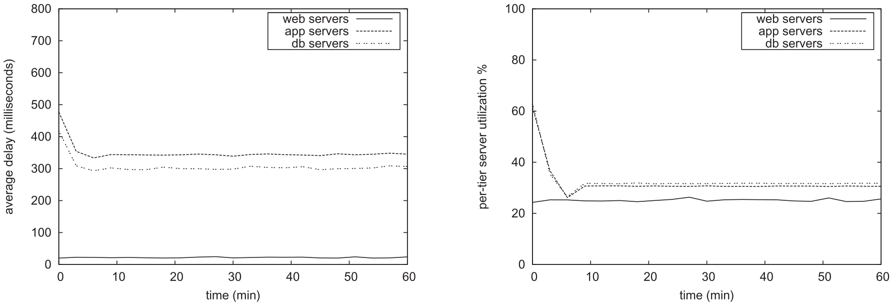
Overall, experimental results show that the new server provisioning approach achieves a small relative delay deviation of 14% and target violation of 17%, respectively. This is a significant improvement from the performance of the rule-based fuzzy controller for the same workload scenario in Figure 3, 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, it achieves the delay guarantee in a very responsive manner, usually in few intervals after there is a sudden step-change in the request arrival rate.



(a) end-to-end delay assurance (b) server allocation

Fig. 10. End-to-end performance of neural fuzzy control for a stationary workload.



(a) per-tier average delay (b) per-tier utilization

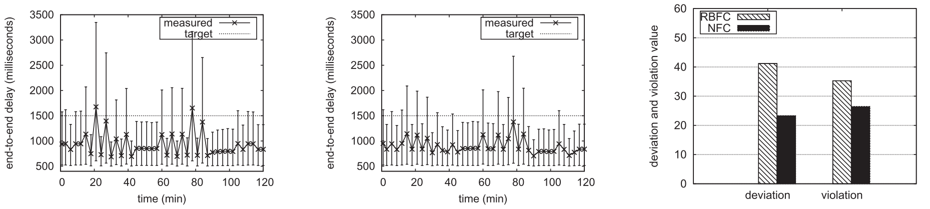
Fig. 11. Per-tier performance of neural fuzzy control for a stationary workload.

We also apply a stationary workload with an average request arrival rate of 12 requests per second. Figures 10(a) and 10(b) show the end-to-end delay variation and changes in server allocation. There is an overshooting in the number of application and database servers allocated at the beginning. It is due to the fact that the neural fuzzy controller is empty at the beginning. It learns how to make control decisions at runtime. As the learning process proceeds, the control behavior improves over time. Results show that the neural fuzzy controller is able to guarantee the 95_{th}-percentile end-to-end delay target of 1000 ms within a couple of 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 the fast online learning algorithm.

Figures 11(a) and 11(b) show the per-tier delay and server utilization under the stationary workload. The per-tier delay of requests as well as server utilization vary across tiers mainly due to the characteristics of per-tier resource demands of the given workload as shown in Table II. More stable results compared to those in Figures 9(a) and 9(b) are due to the fact that the workload is stationary rather than highly dynamic.

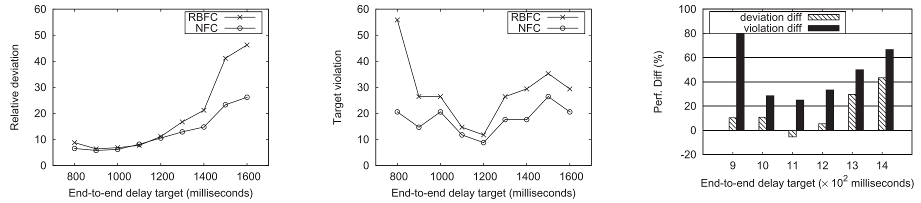
5.2. Comparison with Rule-Based Fuzzy Controllers

A rule-based fuzzy controller has shown its merits in achieving performance assurance through model-independent resource allocation and dynamic output scaling factor tuning [Lama and Zhou 2009, 2012a]. 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



(a) with RBFC (b) with NFC (c) performance comparison

Fig. 12. 95_{th}-percentile end-to-end delay assurance for a dynamic step-change workload (target 1500 ms).



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

Fig. 13. Performance comparison for various delay targets with dynamic step-change workload.

input scaling factor, rule base, and membership functions that are manually tuned for a particular workload and a delay target. In this section, we compare the performance of our Neural Fuzzy Controller (NFC) with a Rule-Based Fuzzy Controller (RBFC) used in Lama and Zhou [2009, 2012a]. We choose an input scaling factor of 1/500 as it shows good performance under a stationary workload for the rule-based fuzzy controller. For quantitative comparison, we take the performance of NFC as a baseline and define the performance difference between NFC and RBFC as

$$PD_{deviation} = \frac{R(e)_{RBFC} - R(e)_{NFC}}{R(e)_{NFC}} \tag{26}$$

$$PD_{violation} = \frac{T(v)_{RBFC} - T(v)_{NFC}}{T(v)_{NFC}}. \tag{27}$$

If PD is positive, the NFC has better performance than RBFC and vice versa.

5.2.1. Dynamic Workload with Sudden Step-Change Request Arrival Rate. First, we apply a dynamic workload with sudden step-changes in the request arrival rate as shown in Figure 2. Figure 12 shows that the self-adaptive neural fuzzy controller is more robust to the dynamic workload variation compared to the rule-based fuzzy controller 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 fewer 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 violations due to a small number of servers involved in the control action. The

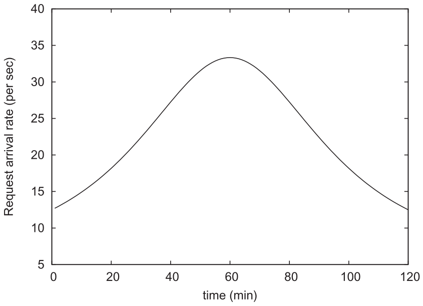
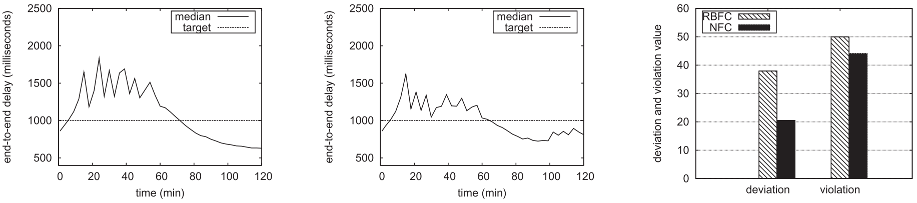


Fig. 14. A continuously changing dynamic workload for a three-tier Internet service.



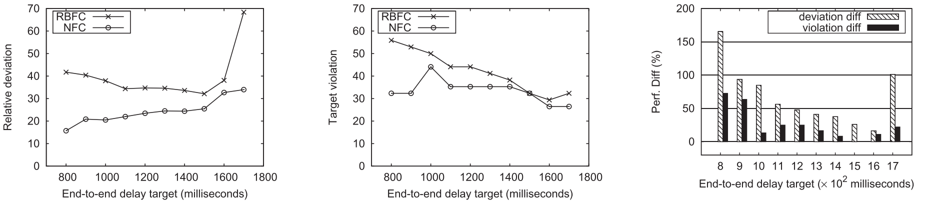
(a) with RBFC (b) with NFC (c) performance comparison

Fig. 15. Median end-to-end delay for a continuously changing workload (target 1000 ms).

targets in the lower range also result in larger target violations 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. 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 of the performance improvement is due to the fact that it adapts itself to accommodate various range of inputs instead of relying on a statically chosen input scaling factor.

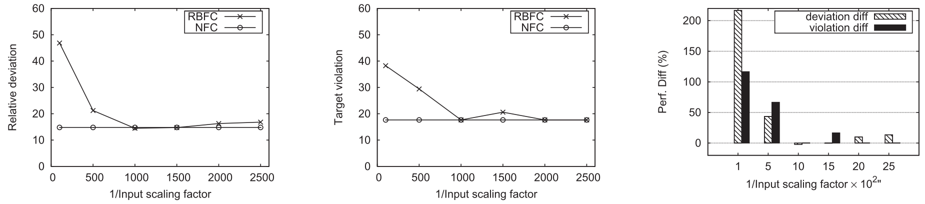
5.2.2. Dynamic Workload with Continuous Change in Request Arrival Rate. Our neural fuzzy-control-based server provisioning approach is applicable to any type of workload and delay guarantee metric due to its self-adaptive capability. We illustrate this by experimenting on a scenario with continuously changing load similar to what was used in Chen et al. [2006], in which the workload request arrival rate variation follows a sinusoid (sine) function. Figure 14 illustrates the workload scenario. As a case study, we aim to assure the median end-to-end delay guarantee instead of 95_{th}-percentile-based guarantee. This experiment is to demonstrate the capability of the neural fuzzy control in assuring other percentile-based end-to-end delay guarantee.

As shown in Figure 15, the median end-to-end delay is mostly above the target of 1000 ms as the workload continuously increases until the time 60 minutes. Both controllers allocate more servers to reduce the median end-to-end delay. After time 60 min, the median end-to-end delay is mostly below the target as the workload continuously decreases. In this case, both controllers deallocate servers to bring the end-to-end delay close to the target. Throughout the experiment, the self-adaptive NFC is able to keep the median end-to-end delay closer to the target as compared



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

Fig. 16. Performance comparison for various delay targets with continuously changing workload.



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

Fig. 17. Performance comparison for various input scaling factors with delay target 1400 ms in case of dynamic step-change workload.

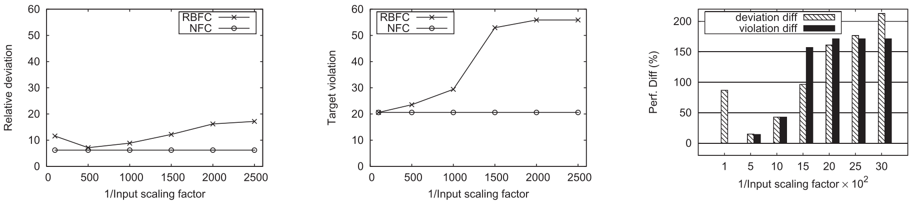
to RBFC. NFC outperforms RBFC by 85% and 14% in terms of the relative delay deviation and the target violation, respectively.

Next, we conduct sensitivity analysis of two controllers for various end-to-end delay targets. Figures 16(a), (b), and (c) show that the neural fuzzy controller consistently outperforms the rule-based fuzzy controller in terms of the relative delay deviation and target violation for various delay targets. Similar to the results observed in case of the sudden step-change workload, the performance difference varies depending on the range of delay targets for which the rule-based fuzzy controller is well tuned. For example, the performance differences in relative delay deviation and temporal target violation range from 17% to 42% and 0% to 17%, respectively, for delay targets between 1300 ms and 1600 ms. For delay targets outside this range, the performance difference is much larger. It is 48% to 165% for relative delay deviation and 14% to 73% for temporal target violation. On average, NFC outperforms RBFC by 68% and 26% in terms of delay deviation and target violation, respectively.

5.3. Impact of Input Scaling Factor on Controller's Self-Adaptivity

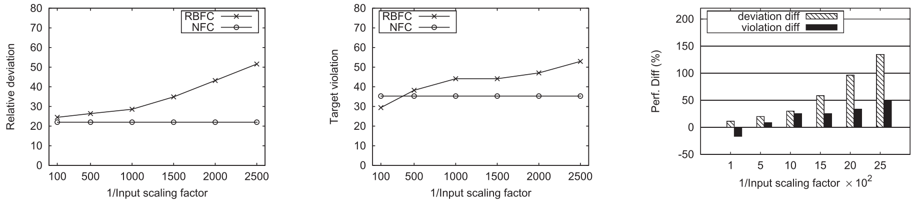
We now study the impact of the input scaling factor on the performance of the rule-based fuzzy controller, and compare its performance with our neural fuzzy controller. For RBFC, through trial and error we observed a certain ratio between the input scaling factors for $\Delta e(k)$ and $e(k)$ that gave the best performance. We found this ratio is about 1:3 for the given workloads. For RBFC in the experiments, the input scaling factor for $\Delta e(k)$ is chosen as one-third of the input scaling factor for $e(k)$. For example, when the input scaling factor for $e(k)$ is 1/500, it is 1/1500 for $\Delta e(k)$. RBFC works well when it is relatively less sensitive to the change in error.

Figures 17 and 18 show their relative delay deviation, target violation, and performance difference for end-to-end delay targets 1400 ms and 1000 ms, respectively in case of the dynamic step-change workload. 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



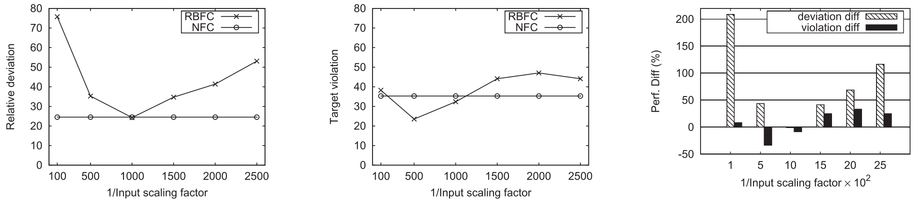
(a) relative delay deviation (b) temporal violation (c) performance difference

Fig. 18. Performance comparison for various input scaling factors with delay target 1000 ms in case of dynamic step-change workload.



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

Fig. 19. Performance comparison for various input scaling factors with delay target 1100 ms in case of continuously changing workload.



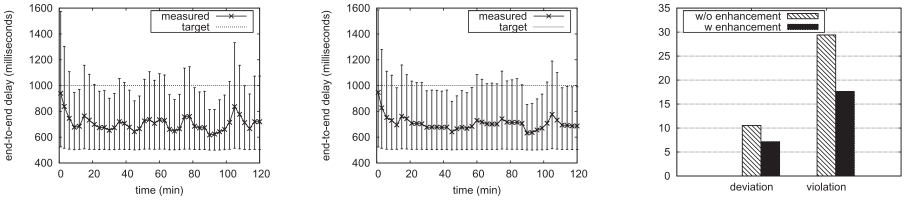
(a) relative delay deviation (b) temporal violation (c) performance difference

Fig. 20. Performance comparison for various input scaling factors with delay target 1300 ms in case of continuously changing workload.

better than the rule-based control. We observe similar results in case of the continuously changing workload as shown in Figures 19 and 20.

When the input scaling factor is chosen as 1/100, the rule-based fuzzy controller shows very good performance in terms of relative delay deviation and temporal target violation for one delay target (1100 ms), but very poor performance for another delay target (1300 ms). For the target 1100 ms and input scaling factor 1/100, its performance is similar to that of neural fuzzy controller. The neural fuzzy controller's relative delay deviation is slightly better by 12% and the temporal target violation is slightly worse by 16%. For all other scaling factors, the neural fuzzy controller significantly outperforms the rule-based fuzzy controller. For the target 1300 ms, the performance of two controllers is similar when the input scaling factor is chosen to be 1/1000 but the performance of the neural fuzzy controller is significantly better for most other scaling factors.

The rule-based fuzzy controller is sensitive to the choice of the input scaling factor, which attempts to partition the input fuzzy space nonadaptively. For optimal performance, it needs to be manually tuned each time for different delay targets. 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



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

Fig. 21. 95_{th}-percentile end-to-end delay assurance with neural fuzzy controller for a dynamic step-change workload due to server switching delays.

partitions as intended. Furthermore, the delay target can also change dynamically according to the service-level agreement. 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-adaptivity and online learning capability as compared to trial-and-error-based design of the rule-based fuzzy controller.

5.4. Impact of the Control Enhancements on Server Switching Delay

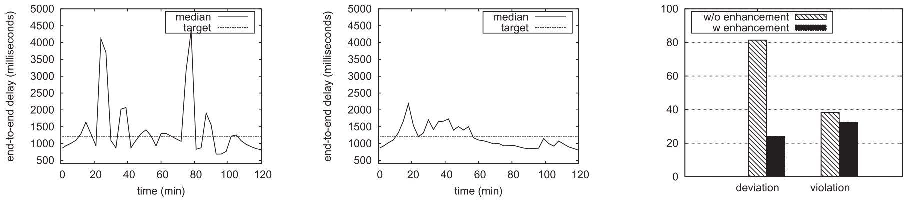
We demonstrate the impact of the two control enhancements designed in Section 4 on the performance of the neural fuzzy controller. We assume the times taken by addition and removal of one virtual server at any tier of an application are 16 and 8 seconds respectively. Due to the server switching delays, 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 and frequent overshoots and undershoots of the 95_{th}-percentile end-to-end delay from the given target. This is illustrated in Figure 21(a) where we apply a dynamic step-change workload.

Then, we compensate the server switching effect by the two proposed enhancements on the neural fuzzy controller. Figures 21(b) and 21(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 when learning the control parameters and they adaptively change the output of the neural fuzzy controller.

We observe similar impact of the enhancements when the neural fuzzy controller is used to assure the median end-to-end delay guarantee in case of a continuously changing workload as shown in Figures 22(a), (b), and (c). Figures 23(a) and (b) show the server allocation at various tiers of the multi-tier cluster managed by the neural fuzzy controller without and with the enhancements, respectively. Due to the presence of server switching delays, the controller without the enhancements abruptly assigns and removes more servers than necessary. This is avoided by the proposed enhancements on the neural fuzzy controller and thus there is the performance improvement with less deviation and violation.

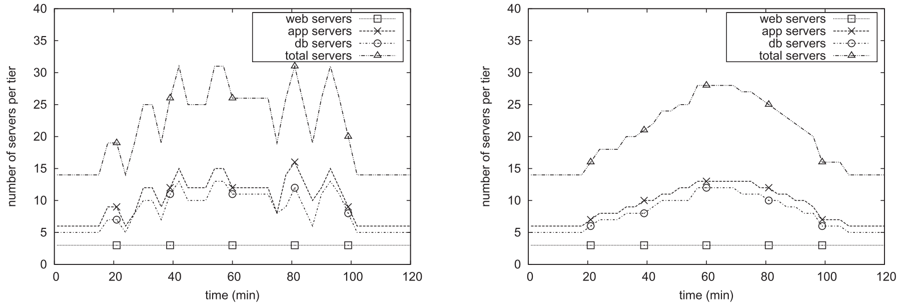
5.5. Comparison with a PI Controller under Varying Workload Characteristics

In this section, we evaluate the performance of our neural fuzzy-control-based server provisioning technique in case of varying workload characteristics. In practice, the



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

Fig. 22. Median end-to-end delay assurance with neural fuzzy controller for a continuously changing workload due to server switching delays.



(a) without enhancements (b) with enhancements

Fig. 23. Server allocations with the neural fuzzy controller due to server switching delays.

workload characteristic of a multi-tier application varies according to its workload mix. For example, the service time distribution of browsing mix-based workload requests is different than that of shopping mix-based workload requests in an e-commerce application [Singh et al. 2010]. Classical Proportional Integral (PI) controllers have been widely used for admission control and performance assurance in Internet servers [Kamra et al. 2004; Lu et al. 2006]. We use the classical PI control technique as the baseline for comparison with the neural fuzzy controller. The control interval is 3 minutes in both controllers.

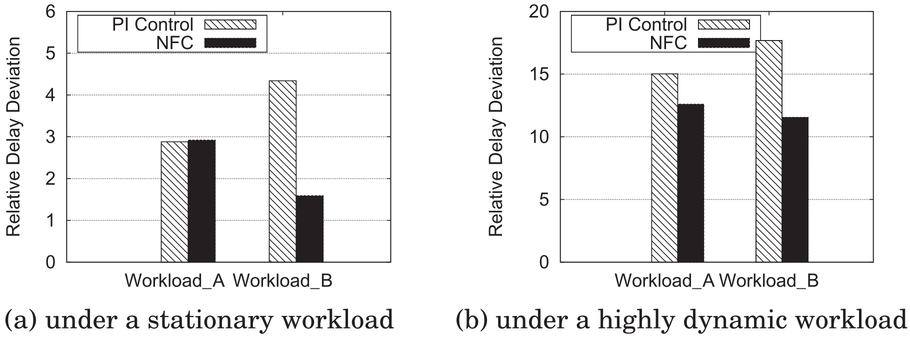
Since PI control is model based, we first obtain the system performance model of our virtualized multi-tier system by using a standard system identification technique. We use the workload characteristic A as shown in Table II and measure the end-to-end delay for various virtual server allocations. Based on the offline data collected from the system, we use the Least Squares Method (LSM) to estimate the parameters of the system model [Wang and Wang 2009]. The parameters are given by Eq. (28).

$$d'(k) = b_1 d'(k-1) + c_1 m'(k-1) \tag{28}$$

Here $b_1 = -0.06829$ and $c_1 = -0.5149$. The controlled variable in the system model is $d'(k) = d(k) - d$, where $d(k)$ is the end-to-end delay measured at sampling interval k. Note that the end-to-end delay measurement can be percentile based, mean, or median based. The manipulated variable is $m'(k) = m(k) - m$ where m(k) is the number of virtual servers allocated at sampling interval k. d and m are the end-to-end delay and virtual server allocation corresponding to a chosen operating point that is used to linearize the system model. We choose operating points in the system by selecting the middle value of a typical range of virtual server allocation as m, and then measure the resultant end-to-end delay as d. Based on the system model, the PI controller is

Table III. Workload Characteristics B

Parameter	WebTier	AppTier	DBTier
s_i	13.15 ms	272.67 ms	241.61 ms
σ_i^2	7.21	442.58	729.70



(a) under a stationary workload (b) under a highly dynamic workload

Fig. 24. Performance comparison with PI control for varying workload characteristics.

designed to achieve the desired control performance such as system stability and zero steady-state error. The transfer function of the designed PI controller is

$$F(z) = \frac{-0.34131(z+1)}{(z-1)}. \tag{29}$$

We compare the performance of server provisioning approach with our neural fuzzy controller and with the PI controller when the workload changes from characteristic *A* shown in Table II to characteristic *B* shown in Table III. Figures 24(a) and (b) show the relative delay deviations for the two workload characteristics with stationary intensity and with dynamically varying intensity, respectively. For stationary workload with characteristic A, our neural fuzzy controller shows similar performance as the PI controller. However, it significantly outperforms the PI controller for workload characteristic B. The PI controller is designed for a system model that is identified by using workload characteristic A. Hence, it shows poor performance when the workload characteristics change. On the other hand, our neural fuzzy controller is adaptive to varying workload characteristics. For highly dynamic workload, the neural fuzzy controller outperforms the PI controller for both workload characteristics A and B. This is because, unlike the PI controller, it is adaptive to variation in workload intensity as well as characteristics.

As a summary, compared with the PI controller, there is an overall performance improvement of 61% by the use of the neural fuzzy controller. This value is the average of the performance difference metrics defined in Eq. (26) and Eq. (27), for the four scenarios: (1) stationary workload A, (2) stationary workload B, (3) dynamic workload A, and (4) dynamic workload B.

6. A CASE STUDY BASED ON THE TESTBED IMPLEMENTATION

6.1. The Testbed

We conduct a feasibility study with performance evaluation of the proposed neural fuzzy-control-based server provisioning approach in a prototype data center, which consists of 12 HP ProLiant BL460C G6 blade server modules and a 40TB HP EVA storage area network with 10Gbps Ethernet and 8Gbps Fibre/iSCSI dual channels. Each blade server is equipped with Intel Xeon E5530 2.4 GHz quad-core processor and

32GB DDR3 memory. Virtualization of this cluster is enabled by VMWare's vSphere 4.1 Enterprise edition. vSphere controls the disk space, memory, and CPU share (in MHz) allotted to the virtual machines, and also provides an Application Programming Interface (API) to support the remote management of virtual machines. Our controller uses VMware's VIX API 1.10 as an actuator to dynamically instantiate or deinstantiate VMs on the hosts and to assign CPU shares to the virtual machines.

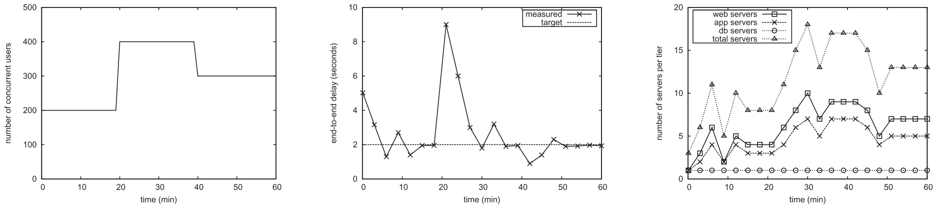
We have implemented a virtualized multi-tier server cluster architecture as shown in Figure 1. The database tier is not replicable in our testbed implementation. Each server in the multi-tier cluster is hosted inside a VMware virtual machine. The configuration of each virtual machine for the Web and application tiers is 1 vCPU, 2GB RAM, and 15GB hard disk space. We use 4 vCPUs, 2GB RAM, and 15GB hard disk space for the database server to perform the case study where the database tier is not the bottleneck. Otherwise, the database tier server needs to be reconfigured with more resources or replicated. The guest operating system used is Ubuntu Linux version 10.04. Load balancers are used to distribute requests among virtual machines at the Web and application tiers. An Apache module, *mod_proxy_balancer*, is used for load balancing while taking into account session affinity.

The neural fuzzy controller interacts with the VM Manager (VMM) through the vSphere Management API. To start a VM, the controller issues the VM start request using the *PowerOnVM_Task* method. It submits a virtual machine power-on task to the VMM. The controller obtains the task start time from the task information structure. When the power-on task has been completed, the controller obtains the task completion time. Using the task start and completion time, the controller calculates the time used for starting the VM. Similarly, it can obtain the time used for removing a VM. Our experiments show that adding or removing a VM in the testbed cluster takes approximately 7 seconds, which is quite small compared to the control interval of 3 minutes used for delay measurement.

As with many related studies [Lama and Zhou 2012b; Padala et al. 2009; Urgaonkar et al. 2008; Watson et al. 2010], this work uses an open-source multi-tier application benchmark, RUBiS [2013], in the experimental study. RUBiS implements the core functionality of an eBay-like auction site: selling, browsing, and bidding. It implements three types of user sessions, has nine tables in the database, and defines 26 interactions that can be accessed from the clients' Web browsers. The application contains a Java-based client that generates a session-oriented workload. RUBiS sessions have an average duration of 15 minutes and the average think time is 5 seconds. It defines two workload mixes: a browsing mix made up of only read-only interactions and a bidding mix that includes 15% read-write interactions. We configure the RUBiS clients to submit workloads of different mixes as well as workloads of time-varying intensity. Each RUBiS client also provides a sensor that measures the client-perceived QoS metrics such as average response time and throughput over a period of time. We modify the client to measure the percentile-based response time required by our experiments. Our implementation of RUBiS application is done with Apache 2.2.14, PHP 5.3.2, and MySQL 5.1 servers for the Web, application, and database tiers.

6.2. End-to-End and per-Tier Delays by the Neural Fuzzy Controller

For performance evaluation, we apply a dynamic workload to our multi-tier virtualized server cluster as shown in Figure 25(a). Initially, the workload consists of a bidding mix of 200 concurrent users. After 20 minutes we double the workload intensity to 400 concurrent users with browsing workload mix. Another 20 minutes later, we decrease the workload to 300 concurrent users. The reported results are from a single run. The 95_{th}-percentile end-to-end delay target is set to be 2 seconds.



(a) a dynamic workload (b) delay assurance (c) server allocation

Fig. 25. End-to-end performance of neural fuzzy control based on a testbed implementation hosting RUBiS.

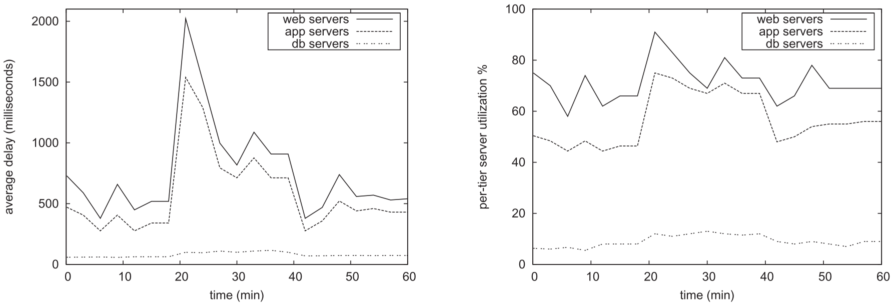
Figure 25(a) shows that the self-adaptive neural fuzzy controller is able to guarantee the 95_{th}-percentile delay target of 2 seconds within a few sampling intervals. The multi-tier system is initially provisioned with one virtual server at each tier. As the controller starts allocating virtual servers at the Web and application tiers, it applies online learning to tune its neural network structure and parameters based on the measured percentile-based end-to-end delay of requests. We observe that the 95_{th}-percentile delay approaches the target of 2 seconds within the first 15 minutes of the experiment as a result of dynamic server allocations. The oscillation of the delay around its target is mainly due to the fact that the neural fuzzy controller needs to learn how to control the system by exploring different server allocations. As time progresses, the controller becomes more effective in achieving the end-to-end delay guarantee.

There is a spike in the measured end-to-end delay at time 20th minute due to the sudden increase in the workload intensity and the change in the workload mix. However, the neural fuzzy controller achieves the delay guarantee in a responsive manner, usually in three to four control intervals. Similarly, there is a sudden drop in the measured delay at time 40th minute due to the decrease in workload from 400 to 300 concurrent users. The neural fuzzy controller effectively removes virtual servers from different tiers to bring the end-to-end delay close to the target. Results show that the controller is robust to dynamically varying workload intensity as well as characteristics due to its self-adaptive capability.

Figure 25(b) shows the change in the allocation of virtual servers at various tiers. The controller allocates servers at individual tiers in proportion to the per-tier average delay measurement. Note that in the testbed implementation, the server allocation adjustments are only distributed between the Web and application tiers as the database tier is not replicated.

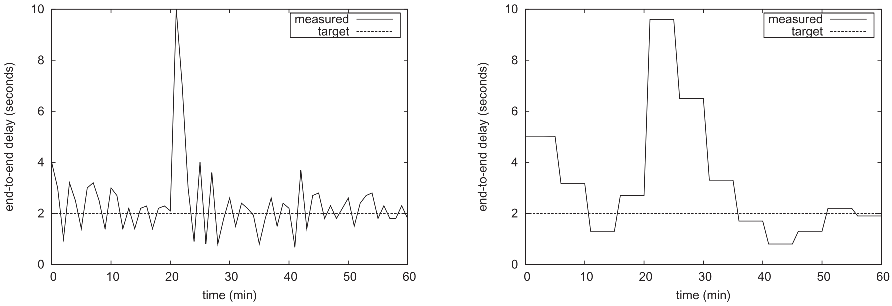
We also show the per-tier average delay and server utilization. Obtaining per-tier delays of a request is nontrivial. At the Web tier (Apache) and the application tier (PHP), we can obtain the response time of each request from its log. However, at the database tier, the free community version MySQL 5.1 does not provide such a functionality for obtaining a query's response time. That means, we can obtain the delay at the Web tier and the total delay, but not the per-tier delay at the application and the database tiers. To timestamp each query issued from the application tier, we make an instrumentation of RUBiS application by modifying its PHP scripts. Thus, we are able to measure the per-tier delays. Note that the per-tier delay measured by this approach may not be perfectly accurate since it includes the time spent inside the application tier while constructing the query before sending it to the database tier.

Figures 26(a) and (b) show the per-tier average delay of requests and server utilization at various control intervals. The per-tier delay of requests as well as server utilization vary across tiers mainly due to the characteristics of per-tier resource demands of the given workload. In our experiment, the resource utilization at the database tier



(a) per-tier delays (b) per-tier server utilization

Fig. 26. Per-tier performance of neural fuzzy control based on a testbed implementation hosting RUBiS.



(a) with 1-minute control interval (b) with 5-minute control interval

Fig. 27. End-to-end performance of neural fuzzy control with different control intervals.

is small compared to the resource utilization at the Web and application tiers mainly due to the high server capacity at the database server.

6.3. Impact of Control Interval on Control Robustness and Delay Guarantee

In the experiments reported so far, the end-to-end delay of the system is measured periodically at a control interval of 3 minutes. The choice of control interval depends on the trade-off between robustness to delay measurement noise and responsiveness of control actions. We evaluate the impact of choosing different control intervals on the end-to-end delay guarantee and control robustness to measurement noise.

Figures 27(a) and (b) show the 95_{th}-percentile delay achieved by the neural fuzzy controller when the control intervals are chosen as 1 minute and 5 minutes, respectively. We observe that using a short control interval of 1 minute results in significant oscillations in the measured delay. It is due to the fact that the controller reacts ineffectively in response to the measured noise in delay. It is necessary to use a longer control interval so that the controller can allocate servers effectively based on more accurate and consistent measurements of delay.

On the other hand, using a long control interval of 5 minutes reduces the measurement noise, but also reduces the responsiveness of the controller in achieving the end-to-end delay target. In such cases, the control actions of adding or removing virtual servers are performed less frequently. Hence, it takes longer time to bring the end-to-end delay close to the target as shown in Figure 27(b).

In the case study, the control interval of 3 minutes is chosen to balance the trade-off between the neural fuzzy controller's responsiveness and robustness to noise. The length of the control interval is a very interesting and important issue. There might not exist a control interval that works best for different workload scenarios and different applications.

7. CONCLUSIONS AND FUTURE WORK

Performance assurance is critical and challenging to data centers that host popular Internet services. In this article, we have designed a novel self-adaptive neural fuzzy-control-based server provisioning approach to guarantee the end-to-end delay of requests flowing through multi-tier server clusters. The approach can effectively provide any percentile- and the mean-based delay guarantee. 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 in the face of highly dynamic workloads. We further enhance the neural fuzzy controller to compensate for the effect of server switching delays. We have also studied the impact of input scaling factor on controller's self-adaptivity and the impact of the control interval on the system robustness.

Our approach is capable of automatically constructing the controller's structure and adapting control parameters through fast online learning. Simulation results have demonstrated that the neural fuzzy controller is robust to highly dynamic workloads and changes in delay target. Compared to the rule-based fuzzy controller and a classical PI controller, the new approach has shown superior performance in achieving the end-to-end delay assurance, particularly under highly dynamic workloads. Importantly, we have demonstrated the feasibility and excellent performance of the new approach in a testbed implementation of virtualized server cluster. The neural fuzzy controller demonstrated its promise of being a self-adaptive approach for autonomic computing in virtualized data centers.

In the future work, we will continue to tackle the autonomic performance control problem from the following three perspectives. First, we aim to enrich the neural fuzzy controller by developing self-tuning components for choosing the control interval and other key parameters. The control interval choice can affect performance measurement noise and responsiveness of control actions. We want to extend the controller to support multiple heterogeneous applications with varying workload characteristics. Second, we will consider integrating a reinforcement-learning-based component for resource allocation optimality. Third, we want to integrate admission control with dynamic server allocation when the resources are being exhausted in the system.

ACKNOWLEDGMENTS

The authors thank the National Institute of Science, Space and Security centers for providing blade server equipments for conducting the case study. The authors are so grateful to the associate editor Dr. Jeffrey Kephart and the anonymous reviewers for their valuable suggestions for revising the manuscript.

REFERENCES

Abdelzaher, T. F., Shin, K. G., and Bhatti, N. 2002. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst. 13,* 1, 80–96.

Amza, C., Chanda, A., Cox, A., Elnikety, S., Gil, R., Rajamani, K., Zwaenepoel, W., Cecchet, E., and Marguerite, J. 2002. Specification and implementation of dynamic web site benchmarks. In *Proceedings of the IEEE International Workshop on Workload Characterization (WWC'02)*. 3–13.

Bennani, M. N. and Menasce, D. A. 2005. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'05)*.

Bu, X., Rao, J., and Xu, C.-Z. 2009. A reinforcement learning approach to online web system autoconfiguration. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'09)*.

Chen, J., Soundararajan, G., and Amza, C. 2006. Autonomic provisioning of backend databases in dynamic content web servers. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'06)*.

Diao, Y., Hellerstein, J. L., Parekh, S., Shaihk, H., Surendra, M., and Tantawi, A. 2006. Modeling differentiated services of multi-tier web applications. In *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'06)*.

Huebscher, M. C. and McCann, J. A. 2008. A survey of autonomic computing: Degrees, models, and applications. *ACM Comput. Surv. 40,* 3.

Isci, C., Hanson, J. E., Whalley, I., Steinder, M., and Kephart, J. O. 2010. Runtime demand estimation for effective dynamic resource management. In *Proceedings of the Network Operations and Management Symposium (NOMS'10)*.

Jung, G., Hiltunen, M. A., Joshi, K. R., Schlichting, R. D., and Pu, C. 2010. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'10)*.

Kamra, A., Misra, V., and Nahum, E. M. 2004. Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS'04)*.

Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., and Tantawi, A. 2006. Dynamic placement for clustered web applications. In *Proceedings of the ACM International Conference on World Wide Web*.

Lama, P. and Zhou, X. 2009. Efficient server provisioning for end-to-end delay guarantee on multi-tier clusters. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS'09)*.

Lama, P. and Zhou, X. 2010. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'10)*. 151–160.

Lama, P. and Zhou, X. 2012a. Efficient server provisioning with control for end-to-end delay guarantee on multi-tier clusters. *IEEE Trans. Parall. Distrib. Syst. 23,* 1, 78–86.

Lama, P. and Zhou, X. 2012b. NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers. In *Proceedings of the IEEE/IFIP Conference on Dependable Systems and Networks (DSN'12)*. 1–12.

Leite, J. C. B., Kusic, D. M., Mosse, D., and Bertini, L. 2010. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'10)*.

Lin, C. and Lee, C. S. G. 1992. Real-time supervised structure/parameter learning for fuzzy neural network. In *Proceedings of the IEEE International Conference on Fuzzy Systems*. 1283–1291.

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

Litoiu, M. 2007. A performance analysis method for autonomic computing systems. *ACM Trans. Auton. Adapt. Syst. 2,* 1.

Liu, X., Sha, L., and Diao, Y. 2003. Online response time optimization of apache web server. In *Proceedings of the International Workshop on Quality of Service (IWQoS'03)*.

Liu, X., Heo, J., Sha, L., and Zhu, X. 2008. Queueing-model-based adaptive control of multi-tiered web applications. *IEEE Trans. Netw. Service Manag. 5,* 3, 157–167.

Lu, C., Lu, Y., Abdelzaher, T. F., Stankovic, J. A., and Son, S. H. 2006. Feed back control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. Parall. Distrib. Syst. 17,* 9, 1014–1027.

Meng, X., Isci, C., Kephart, J., Zhang, L., and Bouillet, E. 2010. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceedings of the International Conference on Autonomic Computing (ICAC'10)*.

Mi, N., Casale, G., Cherkasova, L., and Smirni, E. 2008. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*.

Mi, N., Casale, G., Cherkasova, L., and Smirni, E. 2009. Injecting realistic burstiness to a traditional client-server benchmark. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'09)*.

Padala, P., Hou, K.-Y., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A. 2009. Automated control of multiple virtualized resources. In *Proceedings of the EuroSys Conference (EuroSys'09)*. 13–26.

Rao, J. and Xu, C. 2011. Online capacity identification of multitier websites using hardware performance counters. *IEEE Trans. Parall. Distrib. Syst. 22,* 3, 426–438.

RUBiS. 2013. Rice university bidding system. http://www.cs.rice.edu/CS/Systems/DynaServer/rubis.

Sha, L., Liu, X., Lu, Y., and Abdelzaher, T. 2002. Queueing model based network server performance control. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'02)*.

Singh, R., Sharma, U., Cecchet, E., and Shenoy, P. 2010. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'10)*. 21–30.

Stewart, C., Kelly, T., and Zhang, A. 2007. Exploiting nonstationarity for performance prediction. In *Proceedings of the EuroSys Conference (EuroSys'07)*. 31–44.

Tesauro, G., Jong, N. K., Das, R., and Bennani, M. N. 2006. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'06)*.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. 2005. An analytical model for multi-tier internet services and its applications. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'05).*

Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., and Wood, T. 2008. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst. 3,* 1, 1–39.

Villela, D., Pradhan, P., and Rubenstein, D. 2007. Provisioning servers in the application tier for e-commerce systems. *ACM Trans. Internet Technol. 7,* 1, 1–23.

Wang, X. and Wang, Y. 2009. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS'09)*.

Watson, B. J., Marwah, M., Gmach, D., Chen, Y., Arlitt, M., and Wang, Z. 2010. Probabilistic performance modeling of virtualized resource allocation. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'10)*.

Wei, J. and Xu, C.-Z. 2006. eQoS: Provisioning of client-perceived end-to-end QoS guarantee in Web servers. *IEEE Trans. Comput. 55,* 12, 1543–1556.

Welsh, M. and Culler, D. 2003. Adaptive overload control for busy Internet servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*.

Zhang, Q., Cherkasova, L., and Smirni, E. 2007. A regression-based analytic model for dynamic resource provisioning of multi-tier Internet applications. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC'07)*.

Zhou, D. and Huang, W. W. 2009. Using a fuzzy classification approach to assess e-commerce web sites: An empirical investigation. *ACM Trans. Internet Technol. 12,* 9, 3.

Zhou, X., Wei, J., and Xu, C.-Z. 2004. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'04)*. 88–97.

Received February 2011; revised November 2011; accepted September 2012