This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Automated and Agile Server Parameter Tuning by Coordinated Learning and Control

Yanfei Guo, *Student Member, IEEE*, Palden Lama, *Student Member, IEEE*, Changjun Jiang, *Senior Member, IEEE*, Xiaobo Zhou, *Senior Member, IEEE*

Abstract—Automated server parameter tuning is crucial to performance and availability of Internet applications hosted in cloud environments. It is challenging due to high dynamics and burstiness of workloads, multi-tier service architecture, and virtualized server infrastructure. In this paper, we investigate automated and agile server parameter tuning for maximizing effective throughput of multi-tier Internet applications. A recent study proposed a reinforcement learning based server parameter tuning approach for minimizing average response time of multi-tier applications. Reinforcement learning is a decision making process determining the parameter tuning direction based on trial-and-error, instead of quantitative values for agile parameter tuning. It relies on a predefined adjustment value for each tuning action. However it is nontrivial or even infeasible to find an optimal value under highly dynamic and bursty workloads. We design a neural fuzzy control based approach that combines the strengths of fast online learning and self-adaptiveness of neural networks and fuzzy control. Due to the model independence, it is robust to highly dynamic and bursty workloads. It is agile in server parameter tuning due to its quantitative control outputs. We implemented the new approach on a testbed of virtualized data center hosting RUBiS and WikiBench benchmark applications. Experimental results demonstrate that the new approach significantly outperforms the reinforcement learning based approach for both improving effective system throughput and minimizing average response time.

Index Terms—Automated Server Parameter Tuning, Internet Applications, Autonomic Computing, Neural Fuzzy Control

1 Introduction

Internet applications have many configurable server parameters. For example, *Apache* web server has important parameters `MaxClients`, `KeepAliveTimeout`, `MaxSpareServers` and `MinSpareServers` that control server concurrency level, network link alive time, and worker process generating. These parameters are crucial to the performance of server applications and to the resource utilization of the underlying computer system. An improper configuration often leads serious consequences. According to the study [20], in average more than 50% of service failures is due to the misconfiguration. In some particular scenarios, misconfigurations caused almost 100% of service failures. However, server parameter tuning is a very complex and difficult task that highly relies on an administrator's experiences and understanding of the server system.

In Internet applications, user-perceived performance is the result of complex interaction of complex workloads in a complex underlying server system. The complexities are due to high workload dynamics and burstiness, multi-tier service architecture, and virtualized server infrastructure. Recent studies [18], [24] have observed significantly dynamic workloads of Internet applications that fluctuate over multiple time scales, which can have a significant impact on the processing and power demands imposed on data center servers. The burstiness in incoming requests in a server system can lead to significant server overload and dramatically degradation of server performance or even service unavailability.

In today's popular multi-tier Internet service architecture, a set of servers are divided by their functionality like a pipeline. Servers in each tier use the functionality provided by their successors and provide functionality to their predecessors. Incoming workloads are often unequally distributed across different tiers. Some tiers may run in the saturated or overloaded state while others are under-loaded. Highly dynamic workloads will also result in the bottleneck shifting across tiers [5]. The inter-tier and intra-tier performance dependences further complicate the configuration of a multi-tier server system. The complexities and high dynamics demand for automated and agile server parameter tuning.

Recently, Bu *at al.* proposed an automated server parameter tuning approach with reinforcement learning (RL) for multi-tier Internet applications [2], [3]. They demonstrated that the RL-based approach can effectively minimize the average response time of a multi-tier applications. However, there are two major limitations. First, the RL-based approach only considers tuning a single parameter. Our preliminary study found that multiple parameters have significant impact on the application performance. The RL-based approach can be extended for multiple parameter co-tuning. But it suffers from poor scalability in problems with a large state space that grows exponentially with the state variables (i.e. sever parameters). Second, RL is a decision making

- *Y. Guo, P. Lama, X. Zhou are with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918. Email: {yguo,plama,xzhou}@uccs.edu.*
- *C. Jiang is with the Department of Computer Science & Technology, Tongji University, Shanghai, China. Email: cjjiang@tongji.edu.cn.*
- *X. Zhou is the corresponding author.*

Digital Object Indentifier 10.1109/TPDS.2013.115

process for the tuning direction. It does not generate a quantitative result of the parameter value change. It needs a predefined adjustment value for each tuning action. However, it is nontrivial or even infeasible to find an optimal value for each tuning decision, particularly under highly dynamic and bursty workloads.

In the face of the challenges of highly dynamic and bursty workloads, parameter dependences, and various application characteristics, we design a neural fuzzy control that integrates the strengths of self-constructing online learning of neural networks and fast tuning of fuzzy control. The resulted approach is model-independent, robust and agile for automated server parameter tuning under highly dynamic and bursty workloads in virtualized environments. We use the new approach for improving both the effective system throughput and the average response time of multi-tier Internet applications. We implemented the new approach in a testbed of virtualized HP ProLiant blade system. We adopt RUBiS [23], an e-transactional benchmark application for performance evaluation. We use three different workload characteristics: stationary, bursty, and step-change [11], [27]. We also evaluate with WikiBench [1], [26], a real workload trace based application.

We conduct extensive experiments to compare the neural fuzzy control based and RL based approaches in automated server parameter tuning for improving performance of multi-tier Internet applications. Results find that the neural fuzzy control based approach can achieve more than 80% higher effective throughput than that due to default vendor configurations. It outperforms the RL based approach by average 10% to 20% in terms of effective system throughput. The improvement is mainly due to the agility of the new approach in finding ideal server parameter configurations. Importantly, under highly dynamic step-change workloads, the RL based approach may not converge in time in finding effective server parameter configurations. It results in significant performance penalties. Its achieved effective system throughput is just about 40% of that due to the neural fuzzy control based approach. And, its resulted average response time is about 35% higher that that due to the neural fuzzy control based approach.

Our contributions lie in the design and development of an automated and agile server parameter tuning approach that can significantly improve the performance of complex multi-tier Internet applications, the use of effective system throughput as the primary performance metric, the analysis of the weaknesses of reinforcement learning for server parameter tuning, and the implementation and evaluation of the new approach.

A preliminary version of this paper appeared in the Proc. of IEEE IPDPS'2012 [6]. In this extended manuscript, we have extended the neural fuzzy control design from single-parameter tuning to multi-parameter co-tuning. We have also carried out new experiments and analysis with the extended approach, a larger capacity testbed and heavier workloads.

In the following, Section 2 discusses related work. Section 3 describes the server tuning problem. Section 4 gives the neural fuzzy control and reinforcement learning approaches. Section 5 introduces the testbed implementation. Section 6 presents the experimental results and analysis. Section 7 concludes the paper.

2 Related Work

Autonomic computing aims to reduce the degree of human involvement in the management of complex computing systems [8]. Recently, autonomic computing in modern data centers has become a very active and important research area. There are studies focused on capacity planning for virtual machines (VMs) co-location and distribution across a data center [10], [17], [28], VM provisioning for applications [11], [12], [25], [27], resource allocation in a VM [7], [9], [21], [22], and server parameter tuning [2], [3], [30].

Automated server parameter tuning is one key but challenging research issue. There were early studies that explored automated server parameter tuning problem on Web servers [4], [16], [29]. Those works studied how server application parameters can affect the user perceived performance and how to automatically tune those parameters. However, they focused the automated server parameter tuning problem in one server or one tier of servers. For example, Liu *et al.* focused on improving online response time of an Apache web server by tuning value of parameter `MaxClients` [16]. They applied rule-based fuzzy control for parameter tuning. As the rule-based fuzzy control is not model-independent, its pre-configured rules will determine the actions of the fuzzy control. To create the rules, they modeled the application server using queueing models. However, queueing theoretic approaches are often not effective in modeling workloads of complex multi-tier Internet applications due to the inter-tier dependences and per-tier concurrency limit [5], [11], [12].

A few recent studies focused on automated server parameter tuning at multiple-tier servers [2], [3], [22], [30]. A representative approach was proposed in [2], [3]. It used reinforcement learning for automated tuning of web-tier server parameters and application-tier server parameters in a coordinated manner. It aimed to minimize the average response time of a multi-tier online web application. Furthermore, it employed an online monitoring based Q-table switching mechanism, which can improve the adaptiveness of the tuning approach regarding various workload characteristics such as the TPC-W benchmark's ordering, shopping and browsing workload mixes. The work provided insights for the automated parameter tuning in complex multi-tier systems.

However, there are scalability and agility issues. First, the approach only considers tuning a single parameter `MaxClients`. In reality, multiple parameters have significant impact on server performance. When the approach is extended for multiple parameter co-tuning, it suffers

from poor scalability in problems with a large state space that grows exponentially with the state variables (i.e. sever parameters). Second, reinforcement learning itself is a decision making process. It only decides what tuning direction to be applied to a server parameter, i.e., increasing, decreasing or hold. It does not generate a quantitative result of the server parameter value change. In practice, a reinforcement learning based approach relies on a predefined adjustment value for each tuning decision. Finding a good predefined value is crucial to the performance of a reinforcement learning based approach. However, under highly dynamic and bursty workloads, finding such a good predefined value is nontrivial or even infeasible. To address the weaknesses of the reinforcement learning based approach, we design an enriched neural fuzzy control that integrates the strengths of fast online learning and self-adaptiveness of neural networks and fuzzy control.

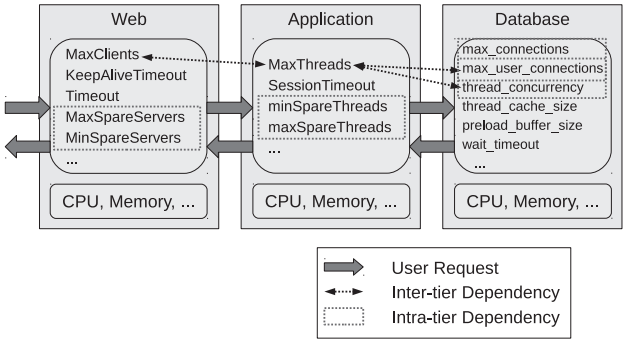
3 Automated Server Parameter Tuning

3.1 Challenges and Issues

Highly dynamic and bursty workloads require an agile and robust approach for automated server parameter tuning. Application level performance heavily depends on the characteristics of the workload. Server parameters must be tuned to match current system workloads. However, Internet workloads are highly dynamic and the workload characteristics keep changing. Online matching the server parameter configuration to the changes is a very challenging problem.

There are different parameter dependences of servers in a multi-tier application, which require a coordinated approach for automated server tuning across all tiers.

- Inter-tier parameter dependency: In a multi-tier application, each tier utilizes the functionality provided by its successor tier. Performance variation in one tier will affect user-perceived end-to-end performance. As Figure 1 shows, the concurrency capacity of *Apache* web server, *Tomcat* application server, and *MYSQL* database server are controlled by parameters `MaxClients`, `MaxThreads`, `max_user_connection`, and `thread_concurrency`, respectively. These parameters need to be configured carefully to match the workload distribution on all tiers. If we increase the concurrency capacity of web tier and leave no changes to other tiers, the web tier will try to process more user requests concurrently, which will result in more requests to the successor tiers. This will increase the response time at the successor tiers or even overload them, resulting in end-to-end performance degradation or even service outage.
- Intra-tier parameter dependency: There are intra-tier dependences of server parameters at each tier. For example, at the web tier, parameter `MaxSpareServers` must have a greater value than parameter `MinSpareServers`. It is also the case at the application tier between parameters `maxSpareThreads` and `minSpareThreads`. At the database tier, the intra-tier dependence is more complicated. Parameter `max_user_connections` has inter-tier dependency with parameter `MaxThreads` at its predecessor tier, but also intra-tier dependency with parameters `max_connections` and `thread_concurrency` at the same tier.



Parameter dependences for tuning in a multi-tier Internet server architecture

Fig. 1. Parameter dependences for automated tuning in a multi-tier Internet server architecture.

Dynamically changing application characteristics require a model-independent approach. The capacity of a web system is constrained by the underlying hardware resources. But the amount of hardware resources does not always provide the same capacity of a web system because it also depends on what application it hosts. For example, a web application based on dynamic pages needs more resources than one based on static pages. Performing server parameter tuning must consider the different characteristics of various web applications.

3.2 Effective System Throughput

We propose to maximize the effective system throughput via automated server parameter tuning. Effective system throughput is defined as the number of requests that meet the service level agreement (SLA) requirement on the response time. While the average response time of requests is important to individual users, the effective system throughput is more important to the application provider in clouds [19].

We use a SLA with two response time bounds, hard response time and soft response time. The *absolute effective throughput* is the number of requests that are processed within the SLA time bounds. If a request is processed between the hard and soft response time bounds, its effective throughput is measured according to a utility decaying function, e.g., linear decaying, exponential decaying, and logarithmic decaying [6]. The *relative effective throughput* is the ratio of the absolute effective throughput to the total number of incoming requests.

The work in [2] aimed to minimize the average response time of all requests. However, minimizing the

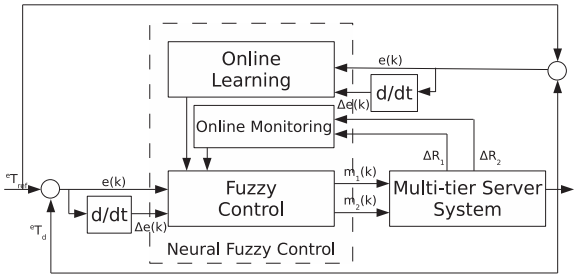


Fig. 2. The block diagram of the neural fuzzy control.

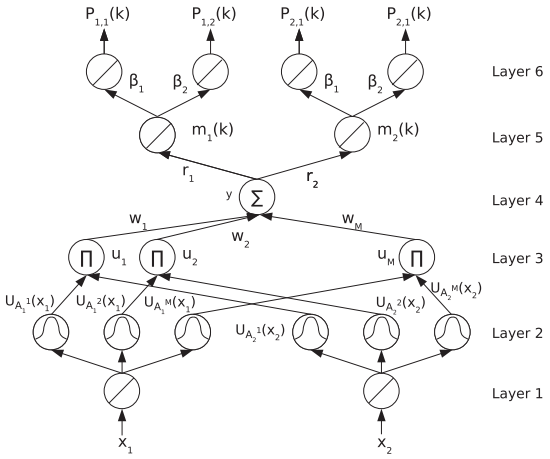


Fig. 3. Schematic diagram of the neural fuzzy control.

average response time of requests does not necessarily maximizing the effective throughput of the system. To minimize the average response time, server parameter configurations will be tuned to devote more resources processing each request, which in turn would make resource scarce when needed.

4 Approaches by Learning and Control

4.1 An Enriched Neural Fuzzy Control Approach

4.1.1 Architecture and Features

A general rule-base fuzzy control consists of a fuzzification stage, a rule-base stage, and a defuzzification stage. The fuzzification stage maps numerical inputs into linguistic fuzzy values by appropriate membership functions. The rule-base stage invokes fuzzy logic rules and combines the results of those invoked rules in a linguistic output value. Finally, the defuzzification stage converts the output value back into a numerical output value for the controlled system.

To tackle the challenges of highly workload dynamics and burstiness, inter-tier and intra-tier parameter dependences, and various application characteristics, we integrate a neural network with the general rule-base fuzzy control. This enables the integrated control to automatically construct its neuron structures and adapt its parameters. Figure 2 shows the block diagram and control flows of the control. The task of control is to adjust server configurable parameters on a multi-tier system in order to improve the performance metric ${}^eT_d$ such as the effective system throughput or the average response time of requests.

The neural fuzzy control has two inputs: error notated as $e(k)$ and error changing rate notated as $\Delta e(k)$. We define error as the difference between the achieved performance and the tuning objective notated as ${}^eT_{ref}$ during the k_{th} tuning period. That is, $e(k) = {}^eT_d - {}^eT_{ref}$. The output is the parameter value $e_i(k)$ for the next tuning period. To support coordinated server parameter tuning for multi-tier applications, we enrich the neural fuzzy control with an online monitoring component that keeps monitoring real-time workload distributions at each tier. Based on the monitoring data, the control updates the server parameter values at all tiers in proportion to their workload distributions.

The neural fuzzy control has following features:

- Model-independence: Fuzzy control is suitable for nonlinear, time-variant, and model-incomplete systems. The workload changes will not affect the functionality of automated server parameter tuning.
- Self-construction: The structure of the neural fuzzy control is automatically generated. The neurons and weights are dynamically changed during the server parameter tuning process.
- Robustness: Because the self-construction and model-independence, the neural fuzzy control can adjust itself to match dynamic workload variations. This results in the robustness of the control.
- Cross-tier coordination: The neural fuzzy control treats the multi-tier system as a whole. According to parameters inter-tier dependency and intra-dependency, each tuning will be applied to related parameters at each tier at once.

4.1.2 Design of the Neural Fuzzy Control

The design of the neural fuzzy control is shown in Figure 3. We develop a six-layer neural network. The interconnected neurons play the role of membership functions and the rule base as in a traditional fuzzy control. However, unlike a traditional fuzzy control, the neural fuzzy control has multiple outputs, one output for one tier in a multi-tier system. Figure 3 shows a two-tier example. The control is initialized in a neural network with eight neurons. This minimal structure only contains two membership functions and one rule. More membership functions and rules will be dynamically constructed and adapted as the network grows and learns. The design details of each layer are as follows:

Layer 1: This is the input layer. At this layer, each neuron is associated with one input variable. There are two neurons for inputs $e(k)$ and $\Delta e(k)$, respectively. The activation functions of these two neurons pass the inputs to the next layer for fuzzification.

Layer 2: This is the fuzzification layer. At this layer, each neuron represents a linguistic term. The activation functions of neurons determine how to transform in-

put values into linguistic terms. We set the activation functions using a Gaussian function Eq. (1). We cluster the linguistics terms into two groups, i.e., error $e(k)$ and change in error $\Delta e(k)$. The Gaussian function uses the average of inputs m_{ji} and the standard deviation of inputs σ_{ji} to determine the j_{th} linguistic term.

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

Layer 3: This is the rule-base layer. At this layer, each neuron represents one fuzzy logic rule. The activation functions of neurons are given by Eq. (2). The output of a layer 3 neuron is the result of r_{th} fuzzy logic rule.

$$u_r = \prod_{i=1}^{n} u_{A_i^j} \quad (2)$$

Layer 4: This layer is the defuzzification layer. It converts results of fuzzy logic rules from layer 3 into a numeric parameter value. There is only one neuron in this layer. It sums all results of fuzzy logic rules from Layer 3 and obtains the numeric parameter value. The w_r in function Eq. (3) is the link weight of r_{th} rule. It is adapted from the online learning process.

$$y = \sum_{r=1}^{M} w_r \cdot u_r \quad (3)$$

Layer 5: This is the tier distribution layer. It converts the outcome of defuzzification layer into normalized parameter $m_v(k)$ for each tier. In the implementation, we consider the parameters for web and application tiers. Thus, there are two neurons in layer 5 generating the output values. The activation function of each node is given by Eq. (4). The weight r_v is determined by the workload distribution that was obtained through the online monitoring.

$$m_v(k) = y \cdot r_v \quad v = 1, 2. \quad (4)$$

Layer 6: This is the parameter denormalization layer. It converts the normalized parameter from layer 5 into server parameters $p_{v,q}(k)$ of a tier. The activation function of each node is given by Eq. (5). The weight β_q is the denormalization factor. It is normalized and proportional to the request arrival rate of different tiers. It can be obtained by monitoring the workload intensity and the interval of connection activities.

$$p_{v,q}(k) = m_v(k) \cdot \beta_q \quad q = 1, 2. \quad (5)$$

We use a threshold to determine when the neural fuzzy control stops the server parameter tuning process. A complete tuning iteration consists of two steps. First, the neural fuzzy control generates server parameter values by forwardly feeding inputs through the five layers. Second, after new parameter values are applied to the multi-tier system, the neural fuzzy control evaluates performance changes. If the new server parameters cause performance improvement, there is no change on the parameters and weights as they are making the positive effect. But if the new server parameters cause performance degradation, the neural fuzzy control will amend its parameters and weights using online learning. After several tuning iterations, the neural fuzzy control stops parameter tuning when either error $e(k)$ or change in error $\Delta e(k)$ is less than the threshold. We set the threshold value to be 10%, same as that in work [2].

Please refer to the supplement for the learning process of the neural fuzzy control and the reinforcement learning based approach.

5 System Implementation

5.1 The Testbed

We implemented a multi-tier web system in a university prototype data center, where each HP ProLiant BL460C G6 blade server is equipped with 2-way Intel quad-core Xeon E5530 CPUs and 32GB memory. The blade servers are connected with 10 Gbps Ethernet. VMware vSphere 5.0 is used for server virtualization.

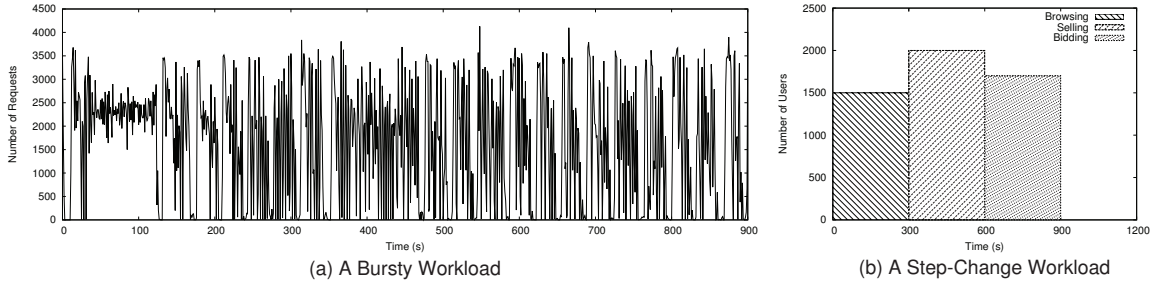
We construct the web system with three VMs, *Apache* web server in the first, *PHP* application server in the second, and *MYSQL* database server in the third. We configure the web server and applications server with 2 VCPUs and 1 GB memory each. We set the CPU usage cap to 1 GHz for stationary workloads and 2 GHz for bursty and step-change workloads. We allocate another three VMs to emulate clients. All VMs run Ubuntu server 10.04 with Linux kernel 2.6.35.

We use RUBiS [23] e-transactional benchmark application for multi-tier Internet applications. RUBiS provides a web auction application modeled in a similar way of ebay.com. It characterizes the workload into three categories: seller, visitor, and buyer. They have different combinations of selling, browsing, and bidding requests. RUBiS client emulates user requests at different concurrent levels. We use multiple clients to emulate the dynamics in workloads. To provide the runtime performance monitoring, we modify the original RUBiS client to retrieve performance statistics of requests. We also use WikiBench [1], [26], a real workload trace.

5.2 Workloads

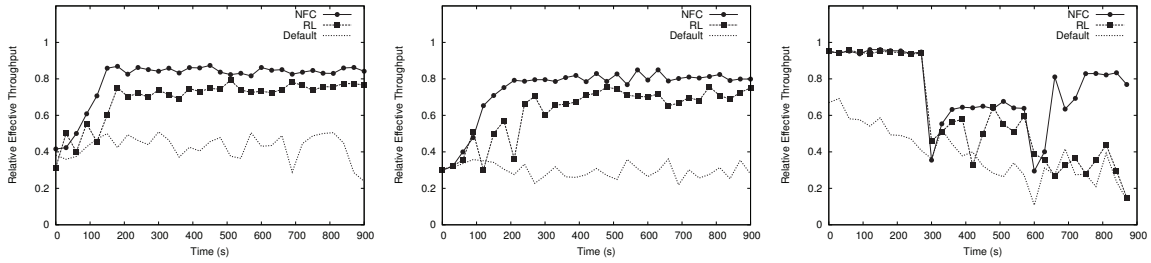
We use three workloads with different densities: a stationary workload, a bursty workload, and a step-change workload [27]. The stationary workload is generated to emulate 2000 concurrent users. For the bursty workload, we implement a workload generator for RUBiS benchmark using the approach proposed in [18]. Figure 4(a) shows the generated bursty workload generated. It emulates a bursty workload in a 900-second time span from 2000 concurrent users with the average think time of 7 seconds. We record the trace of the bursty workload and reuse it for each experiment under the bursty workload.

To examine the adaptiveness of the neural fuzzy control under highly dynamic workloads, we design a step-change workload that contains dynamics in both number



(a) A Bursty Workload (b) A Step-Change Workload

Fig. 4. Highly dynamic and bursty RUBiS workloads used in the experiments.



(a) Under Stationary Workload (b) Under Bursty Workload (c) Under Step-change Workload

Fig. 5. Performance of two automated server tuning approaches compared with the default vendor configuration.

of users and workload characteristics. Figure 4(b) shows the workload. Each time when the number of users changes, the workload mix also changes. At the 300_{th} second, the workload mix changes from browsing to selling. At the 600_{th} second, the workload mix changes from selling to bidding.

5.3 Performance Metrics

We conduct experiments using the effective throughput of the system and the average response time of requests as performance metrics. We use both the absolute effective throughput and the relative effective throughput. In the experiments, a linear decaying function is used.

Each parameter tuning iteration is executed every 30 seconds for all parameter tuning approaches. To demonstrate the agility of the two automated approaches, we compare their converging times before the configurable parameters reach their stable states, 1) in terms of the number of iterations, and 2) in terms of the total execution times of the control algorithms.

6 Performance Evaluation

6.1 Automated Tuning on the Effective Throughput

We use the default vendor provided configuration of *Apache* web server as the baseline of the performance evaluation. Please refer to the supplement for the default vendor configurations and the performance analysis.

We compare two automated tuning approaches, the neural fuzzy control (NFC) based and the reinforcement learning (RL) based, with the default configuration. For the NFC based approach, we feed the measured effective system throughput directly into the control. The control starts with the default values of the server parameters. It generates new parameter values to reconfigure the server until the effective system throughput change meets the threshold-based target. The RL based approach followed the same process. As the work [2], our work sets the key parameters of the RL algorithm as follows: the learning rate value 0.1, the discount factor value 0.9, and no exploration probability.

First, we choose parameter `MaxClients` for tuning as it was found to have most significant impact on server performance [2], [3]. Figure 5 shows the relative effective throughput achieved by three approaches under the stationary workload, bursty workload and step-change workload. Both the NFC based and the RL based approaches can significantly improve the relative effective throughput. Under the stationary, bursty, and step-change workloads, the NFC based approach achieves 95.1%, 141.9%, and 89.7% higher relative effective throughput than the default configuration does, respectively. The RL based approach achieves 70.7%, 106.5%, and 51.3% higher relative effective throughput than the default configuration does, respectively.

Figure 6 shows the absolute effective throughput and the relative effective throughput due to the three approaches. Both automated approaches significantly improve the achieved performance due to the default configuration. The results demonstrate the significance of automated parameter tuning in performance improvement of complex multi-tier applications. Under the stationary, the bursty, and the step-change workloads, we observe that there are significant differences in achieved performance between the two automated approaches.

The absolute effective throughput is the number of

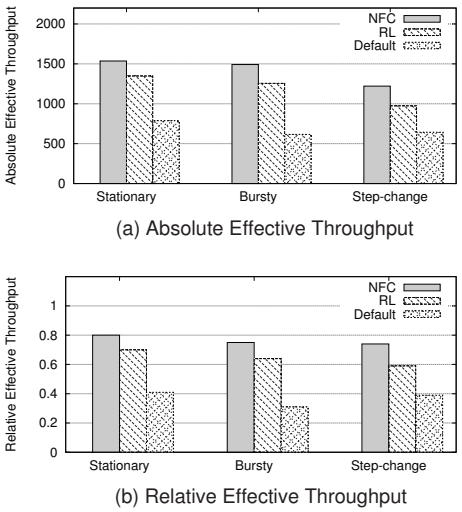


Fig. 6. Performance of three server tuning approaches.

TABLE 1
Converging time of two automated tuning approaches.

Workload type	NFC		RL	
	Number of iterations	Total exec. time (ms)	Number of iterations	Total exec. time (ms)
Stationary	5	855	12	3684
Bursty	8	1368	19	5833
Step-change Browsing	1	171	3	921
Step-change Selling	5	855	8	2456
Step-change Bidding	7	1197	10	3070

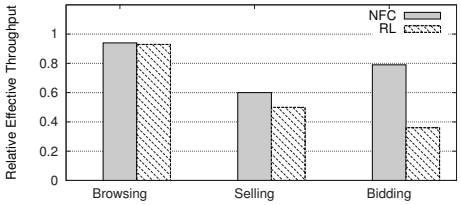


Fig. 7. Average relative effective throughput under the step-change workload.

requests that were responded within the time bounds. Essentially, RUBiS benchmark uses a closed-loop request generator. The request generation rate is affected by the responsiveness of requests. High response time will slow down the workload generation rate, which would result in fewer requests incoming to the system. When using different approaches, the number of total incoming requests indeed could be different. In this case, the absolute effective throughput cannot truly reflect the performance of different approaches. Therefore, in the following, we use the relative effective throughput as the major performance metric.

6.2 Comparison of Two Automated Approaches

In this section, we focus on the performance difference between the NFC based and the RL based approaches for improving the effective system throughput and minimizing the average response time. We also compare their converging times during the parameter tuning process. For the step-change workload, we compare their converging times for each workload stage separately.

6.2.1 Improving the Effective Throughput

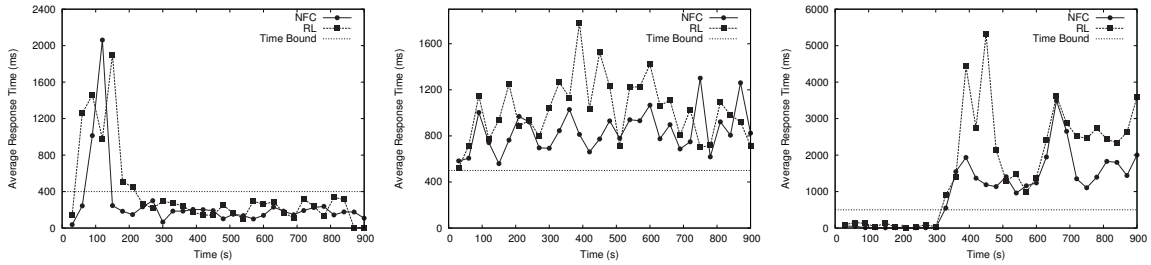
From Figure 5, we observe that the NFC based approach achieves on average 14.3%, 17.2%, and 12.5% higher relative effective throughput than the RL based approach under stationary, bursty, and step-change workloads.

We note that performance variations during the parameter tuning process are very different due to the two automated approaches. As Figures 5(a) and 5(b) show, the performance due to the RL based approach varies significantly during the initial parameter tuning process (the first 300 seconds). That results in significantly lower effective throughput compared with that due to the NFC based approach. There are two major reasons that the RL based approach suffers performance penalty from high variations. First, a parameter value change is upper bounded by the predened adjustment value. This limits the capability of the approach in agilely adjusting sever parameter values. Second, in RL, the Q-table is initialized by the offline training data. The training outcome may not accurately describe the complex interaction between parameter tuning and performance outcome.

Figure 7 shows the performance comparison of two automated approaches under different stages of the step-change workload. Under the bidding workload, the NFC based approach doubles the achieved relative effective throughput by the RL based approach. This can be attributed to the fact that the NSC based approach is more agile in server parameter tuning.

Table 1 shows the converging time of two automated tuning approaches. Under the stationary and the bursty workloads, the NFC based approach converges about 2.4 times faster than the RL based approach in terms of the number of tuning iterations. The improvement is more significant if it is measured in terms of the total execution time of the control algorithms. This is due to that the NFC based approach takes less time in one algorithm execution than the RL based approach does. Under the stationary and the bursty workloads, it converges about 4.3 times faster than the RL based approach.

Note that as shown in Figure 5(c), at the bidding stage the RL based approach does not converge during the last 300-second experimental period. This results in significantly lower relative effective throughput compared with that due to the NFC based approach. Figure 5(c) also shows that the RL based approach suffers from high variations in the system stability due to the slow convergence under the highly dynamic step-change workload.



(a) Under the Stationary Workload (b) Under the Bursty Workload (c) Under the Step-change Workload

Fig. 8. The average response time due to the two automated server parameter tuning approaches.

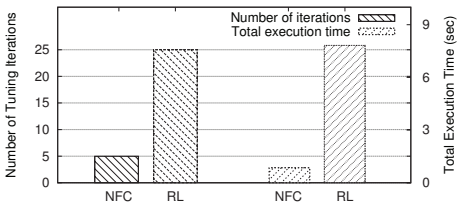


Fig. 9. Converging time in minimizing the average response time.

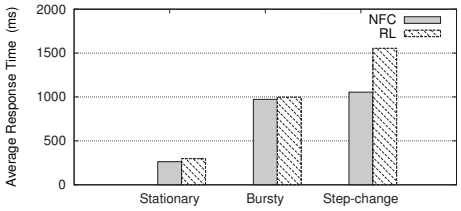


Fig. 10. Average response time comparison.

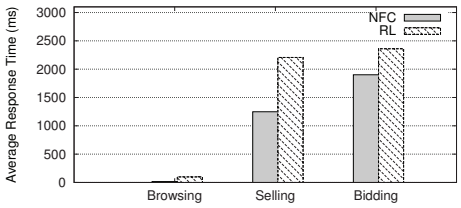


Fig. 11. Average response time under the three-stage step-change workload.

6.2.2 *Minimizing Average Response Time*

Figure 8 shows the average response time of the multi-tier application due to the NFC based approach and the RL based approach. For the stationary and bursty workloads, the achieved average response time of the two automated approaches are close. Under the stationary workload, both approaches are able to decrease the average response time below the predefined time bound. Note that the bound is needed for the RL approach to generate the reward. Under the bursty workload, both approaches cannot assure that the average response time below the predefined time bound. But both maintain the average response time close to the bound. The main difference between the two approaches is the convergence time for minimizing the average response time. Figure 9 shows that the NFC based approach converges much faster than the RL based approach does, in terms of both the number of tuning iterations and the total execution time of the control algorithms.

Figure 10 shows the average response time comparison between the two automated parameter tuning approaches. Under the step-change workload, the NFC based approach achieves more than 30% lower average response time than the RL based approach does.

Figure 11 illustrates significant differences in the achieved average response time by the NFC based approach and by the RL based approach during three stages of the step-change dynamic workload.

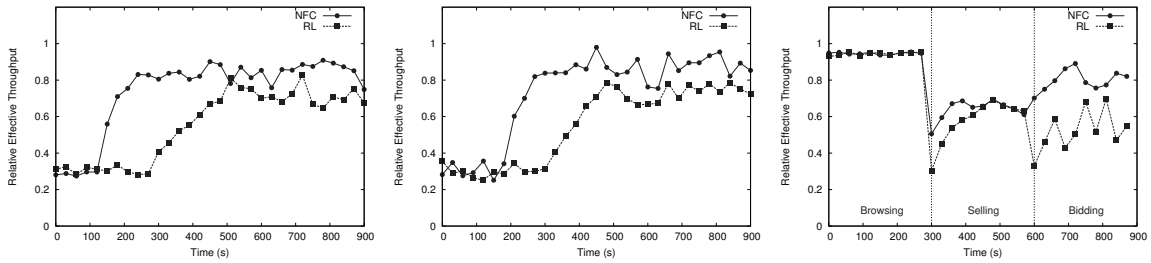
6.2.3 *Analysis*

The experiments have shown that the NFC based approach outperforms the RL based approach for maximizing the relative effective throughput and minimizing average response time. The differences in performance and agility are mainly due to the weaknesses of the RL algorithm used for automated server parameter tuning.

Essentially, RL is a decision making process. It does not directly generate the actual parameter value for server parameter configuration. It needs a predefined adjustment value for each parameter tuning iteration. Experimental results show that during the server parameter tuning process, some configurations that the RL based approach chooses is not as effective as those chosen by the NFC based approach. This is due to the fact that the predefined adjustment value by the RL based approach will limit the number of states in its state space. This will make it possible that some effective parameter configurations are not reachable by the RL based approach. On the other hand, the NFC based approach does not have such a constraint on the reachable configurations.

At each iteration, the RL based approach can only move from one state to another one. Its converging speed is dependent on the size of the state space. With the same range of parameter tuning, the smaller the adjustment value is, the larger the state space is, which often leads to longer converging time. Finding a good adjustment value for the RL based approach is a difficult problem. In

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Under the Stationary Workload (b) Under the Bursty Workload (c) Under the Step-change Workload

Fig. 12. Relative effective throughput due to two automated approaches in multiple parameter co-tuning.

contrast, the NFC based approach is designed to search all possible states and generate the proper parameter configuration value at runtime.

Moreover, the RL based approach is initialized by the training data and updated during the learning process. During the server parameter tuning process, the relation among workloads, parameters, and resulted performance is very complex and non-linear. The training result does not necessarily characterize the complex interactions between the performance and parameters. Therefore, the RL based approach can be misled by the training data.

6.3 Multiple Parameter Co-Tuning

We further study the effectiveness of the NFC based approach in tuning multiple server parameters simutaneously. For the experiment, we use the neural fuzzy control to automatically tune both `MaxClients` and `KeepAliveTimeout` parameters. We compare the relative effective throughput with the RL based approach under stationary, bursty, and step-change workloads.

Figure 12 shows the relative effective throughput due to the NFC based and the RL based approaches under different workloads. The NFC based approach achieves 13.3% and 16.4% higher performance than the RL based approach under the stationary and bursty workloads, respectively. Under the step-change workload, the NFC based approach achieves similar performance with the RL based approach in the browsing workload mix. In the selling and bidding workload mixes, the NFC based approach outperforms the RL based approach by 8.3% and 80%. In most scenarios, both approaches are able to converge to stable parameter settings. However, the RL based approach does not reach a stable setting for parameters in the bidding workload mix of the step-change workload. It significantly affects the application's achieved effective throughput.

Figure 13 shows the performance comparison of multiple parameter co-tuning and single parameter tuning. The relative effective throughput due to the NFC based approach with multiple parameter co-tuning is 8.6%, 9.0% and 6.7% higher than the single parameter tuning under stationary, bursty and step-change workloads, respectively. Improvement due to the multiple parameter co-tuning is also observed with the RL based approach.

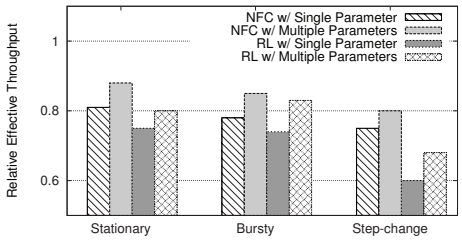


Fig. 13. Performance comparison of multiple parameter co-tuning and single parameter tuning.

This demonstrates the significance of multiple parameter co-tuning. Note that the NFC based approach outperforms the RL based approach.

Figure 14(a) shows the converging time in terms of the number of tuning iterations for multiple parameter co-tuning and single parameter tuning due to the two approaches. With the NFC based approach, there is little difference of the converging time between multiple parameter co-tuning and single parameter tuning. It shows good agility and scalability. But with the RL based approach, the converging time of multiple parameter co-tuning is 75%, 47%, and 41% higher than that of single parameter tuning in stationary, bursty, and step-change workloads, respectively. The increase in the number of state variables (server parameters) results in an exponential growth in the RL state space, which significantly increases the converging time of parameter tuning. Figure 14(b) shows the converging time in terms of the total execution times of the control algorithms. The observation is similar to that in Figure 14(a).

Please refer to the supplement for more experimental results and analysis of the impact of RL predefined adjustment value, selection of server parameters for tuning, overhead and scalability, and server tuning on WikiBench applications.

7 Conclusions

In this paper, we tackle the important but challenging problem of automated server parameter tuning in virtualized server environments. We proposed and developed an automated and agile approach that integrates the strengths of fast online learning and control to maximize

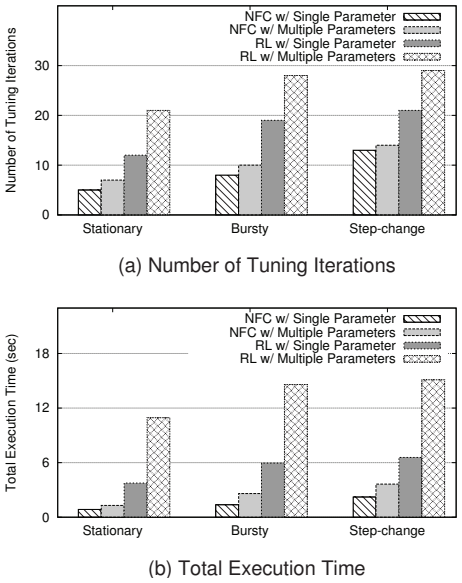


Fig. 14. Converging time comparison of multiple parameter co-tuning and single parameter tuning.

the effective system throughput of multi-tier Internet applications. We implemented the approach on a testbed of virtualized HP ProLiant blade servers. Experimental results based on multi-tier benchmark applications demonstrated that the proposed approach significantly outperforms a reinforcement learning based approach for both improving the effective throughput and minimizing the average response time. We analyzed the weaknesses of the reinforcement learning based approach due to the use of a predened adjustment value and inaccurate training of Q-table. Our developed neural fuzzy control based approach avoids the weaknesses, providing the self-management capability for automated and agile server parameter tuning under highly dynamic and bursty workloads. It is scalable and agile for multiple parameter co-tuning.

Our future will be coordinating server parameter tuning with VM provisioning in Cloud environments.

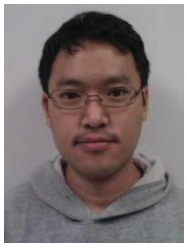
Acknowledgement

This research was supported in part by U.S. National Science Foundation research grant CNS-1217979 and CAREER Award CNS-0844983. The authors thank the anonymous reviewers for their valuable suggestions.

REFERENCES

[1] WikiBench. http://www.wikibench.eu/.
[2] X. Bu, J. Rao, and C.-Z. Xu. A reinforcement learning approach to online web system auto-configuration. In *Proc. IEEE ICDCS*, 2009.
[3] X. Bu, J. Rao, and C.-Z. Xu. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Trans. on Parallel and Distributed Systems*, 24(4), 2013.
[4] I.-H. Chung and J. K. Hollingsworth. Automated cluster-based web service performance tunning. In *Proc. IEEE HPDC*, 2004.
[5] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, and M. Surendra. Controlling quality of service in multi-tier Web applications. In *Proc. IEEE ICDCS*, 2006.
[6] Y. Guo, P. Lama, and X. Zhou. Automated and agile server parameter tuning with learning and control. In *Proc. IEEE IPDPS*, 2012.
[7] Y. Guo and X. Zhou. Coordinated VM Resizing and Server Tuning: Throughput, Power Efficiency and Scalability. In *Proc. IEEE MASCOTS*, 2012.
[8] M. C. Huebscher and J. A. McCann. A survey of autonomic computing–degrees, models, and applications. *ACM Computing Surveys*, 40(7):1–28, 2008.
[9] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *Proc. IEEE ICAC*, 2008.
[10] M. Korupolu, A. Singh, and B. Bamba. Coupled placement in modern data centers. In *Proc. of IEEE IPDPS*, 2009.
[11] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE/ACM MASCOTS*, 2010.
[12] P. Lama and X. Zhou. Efficient server provisioning with control for end-to-end delay guarantee on multi-tier clusters. *IEEE Transactions on Parallel and Distributed Systems*, 23(1):78–86, 2012.
[13] P. Lama and X. Zhou. Ninepin: Non-invasive and energy efficient performance isolation in virtualized servers. In *Proc. IEEE/IFIP DSN*, 2012.
[14] C. Lin and C. S. G. Lee. Real-time supervised structure/parameter learning for fuzzy neural network. In *Proc. IEEE FUZZ*, 1992.
[15] F.-J. Lin, R.-J. Wai, and C.-C. Lee. Fuzzy neural network position controller for ultrasonic motor drive using push-pull dc-dc converter. *Control Theory and Applications*, 146(1), 1999.
[16] X. Liu, L. Sha, and Y. Diao. Online response time optimization of apache web server. In *Proc. IEEE IWQoS*, 2003.
[17] X. Meng, C. Isci, J. Kephart, L. Zhang, and E. Bouillet. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proc. IEEE ICAC*, 2010.
[18] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *Proc. IEEE ICAC*, 2009.
[19] J. Moses, R. Iyer, R. Illikkal, S. Srinivasan, and K. Aisopos. Shared resource monitoring and throughput optimization in cloud-computing datacenters. In *Proc. of IEEE IPDPS*, 2011.
[20] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *Proc. USENIX ITS*, 2003.
[21] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of ACM EuroSys*, 2009.
[22] J. Rao, X. Bu, C. Z. Xu, L. Wang, and G. Yin. Vconf: A reinforcement learning approach to virtual machines auto-conguration. In *Proc. IEEE ICAC*, 2009.
[23] RUBiS. Rice university bidding system. http://www.cs.rice.edu/CS/Systems/DynaServer/rubis.
[24] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. IEEE ICAC*, 2010.
[25] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proc. IEEE ICAC*, 2006.
[26] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.
[27] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1):1–39, 2008.
[28] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *Proc. IEEE INFOCOM*, 2011.
[29] Y. Zhang, W. Qu, and A. Liu. Automatic performance tuning for J2EE application server systems. *Proc. WISE*, 2005.
[30] W. Zheng, R. Bianchini, and T. Nguyen. Automatic configuration of internet services. In *Proc. ACM EuroSys*, 2007.

Yanfei Guo received the BS degree in Computer Science and Technology from the Huazhong University of Science and Technology, China, in 2010. Currently, he is working towards the PhD degree in Computer Science at the University of Colorado, Colorado Springs. His research interests include autonomous performance and resource management for Cloud computing. He is a student member of the IEEE.



Palden Lama received the BTech degree in electronics and communication engineering from the Indian Institute of Technology Roorkee, India, in 2003. He has worked as a software engineer in Qwest Software Services, India. Currently, he is a PhD candidate in Computer Science at the University of Colorado, Colorado Springs. His research interests include the areas of Cloud computing, sustainable computing, autonomic resource and power management, and big data processing in the Cloud. He is a student member of the IEEE.

Changjun Jiang received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995 and conducted post-doctoral research at the Institute of Computing Technology, Chinese Academy of Sciences, in 1997. Currently he is a Professor with the Department of Computer Science and Engineering, Tongji University, Shanghai. He is also a council member of China Automation Federation and Artificial Intelligence Federation, the Director of Professional Committee of Petri Net of China Computer Federation, and the Vice Director of Professional Committee of Management Systems of China Automation Federation. He was a Visiting Professor of Institute of Computing Technology, Chinese Academy of Science; a Research Fellow of the City University of Hong Kong, Kowloon, Hong Kong; and an Information Area Specialist of Shanghai Municipal Government. His current areas of research are concurrent theory, Petri net, and formal verification of software, concurrency processing and intelligent transportation systems. He is a senior member of the IEEE.

Xiaobo Zhou received the BS, MS, and PhD degrees in Computer Science from Nanjing University, in 1994, 1997, and 2000, respectively. He was a Postdoc researcher at the University of Paderborn in 2000. Currently he is an Associate Professor and the Chairperson of the Department of Computer Science, University of Colorado, Colorado Springs. He was a General Co-Chair of ICCCN 2012, TPC Co-Chair of ICCCN 2011, a TPC Vice Chair of the IEEE/ACM CCGrid 2014, GLOBECOM 2010, ICCCN 2009, HPCC 2008, and IEEE/IFIP EUC 2008. He serves on the editorial boards of the Elseviers Computer Communications and Journal of Network and Computer Applications. His research lies broadly in computer network systems, more specifically, autonomic and sustainable computing in datacenters, Cloud computing, server virtualization, scalable Internet services and architectures, and computer network security. His research was supported in part by the US NSF and Air Force Research Lab. He was a recipient of the NSF CAREER AWARD in 2009, the University Faculty Award for Excellence in Research in 2011. He is a senior member of the IEEE