Autonomic Performance and Power Control for Co-Located Web Applications in Virtualized Datacenters

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

Abstract—In a datacenter, complex and time-varying interactions between various tiers and services of web applications, and the contention of shared resources among co-located virtual machines have significant impact on the user perceived performance and power consumption of the underlying system. We propose and develop APPLEware, an autonomic middleware for joint performance and power control of co-located web applications in virtualized datacenters. It features a distributed control structure that provides predictable performance and energy efficiency for large complex systems. It applies machine learning based self-adaptive modeling to capture the complex and time-varying relationship between the application performance and allocation of resources to various application components, in the face of highly dynamic and bursty workloads. The distributed controllers coordinate with each other and allocate resources to meet the service level agreements of applications in an agile and energy-efficient manner. Experimental results based on a testbed implementation with benchmark applications and large scale simulations demonstrate APPLEware's effectiveness, energy efficiency and scalability.

Index Terms—Joint performance and power control, autonomic systems, virtualized servers, distributed fuzzy MIMO control, co-located multi-service applications

1 Introduction

A modern datacenter utilizes virtualization technology to consolidate multiple customer applications onto high density servers for improving server utilization and reducing energy consumption costs [1], [2], [3], [4]. It also aims to satisfy the quality of service (QoS) needs of hosted applications for increasing datacenter revenue. There are growing interests in reducing the degree of human involvement in the management of these complex computing systems through autonomic computing [5]. However, the increasing scale, and complexity of the hosted applications and the contention of shared virtualized infrastructure pose significant and multi-faceted challenges in achieving the important goals of autonomic performance and power management.

In a datacenter, the user perceived performance is the result of a complex interaction of workloads in a very complex underlying system. Popular Internet services have multi-tier architecture in which various tiers in a pipeline invoke each other to process web requests, and multi-service architecture, which comprises of a complex set of disparate and collaborating services [6]. Due to complex performance dependencies between the application components, it is difficult to determine how the computing resources should be allocated to meet the application performance target. Furthermore, the increasing number, and complexity of the hosted applications, has a significant impact on the agility and scalability of performance and power management.

Server virtualization allows applications to share the underlying hardware by running in isolated virtual machines (VMs), which are configured with a certain amount of computing resources (such as CPU, memory, and I/O). For efficient resource usage, the capacity of VMs belonging to different applications need to be adjusted dynamically to match the time-varying resource demands. There are important and challenging issues related to performance and power management in virtualized computing environments. A fundamental problem is that application performance can change due to the existence of other VMs on a shared server [7]. Such performance interference arises due to the contention of resources such as the last level cache, memory bandwidth, etc, which are shared by co-located VMs [8], [9].

Recent studies observed highly dynamic and bursty workloads of Internet services that fluctuate over multiple time scales [10]. They have significant impact on the processing and power demands imposed on servers. Hence, joint performance and power management of modern virtualized computing environments needs to be autonomic. It needs to configure itself adaptively in the face of dynamic workloads to meet the performance objectives of the hosted applications. At the same time, it should optimize the allocation of resources to improve the energy efficiency of the virtualized server system.

- *P. Lama is with the Department of Computer Science, University of Texas, San Antonio, TX 78249. E-mail: palden.lama@utsa.edu.*
- *Y. Guo and X. Zhou are with the Department of Computer Science, University of Colorado, Colorado Springs, CO 80918. E-mail: {yguo, xzhou}@uccs.edu.*
- *C. Jiang is with the Department of Computer Science & Technology, Tongji University, Shanghai, China. E-mail: cjjiang@tongji.edu.cn.*

Manuscript received 27 Jan. 2015; revised 22 June 2015; accepted 27 June 2015. Date of publication 7 July 2015; date of current version 13 Apr. 2016. Recommended for acceptance by X. Gu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2453971

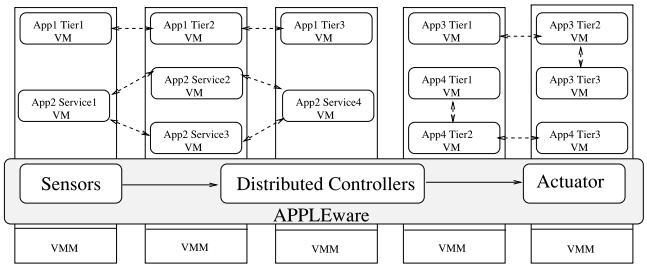


Fig. 1. APPLEware: Autonomic performance and power control for co-located web applications.

In this paper, we propose and develop APPLEware, an autonomic middleware for joint performance and power control of complex multi-tier and multi-service applications in virtualized computing environments. Fig. 1 shows APPLEware managing three multi-tier applications (App1, App3, App4) and one multi-service application (App2). For a system, a centralized controller at the fastest sampling rate gives the best achievable performance. However, implementing a centralized controller at the fastest sampling rate may not be feasible, due to operational constraints such as the overheads involved in measuring the current system states and computing the control decisions. This motivates us to design a distributed control approach for practicality.

APPLEware's core is a distributed model predictive control framework that scales well in large virtualized server systems. Each distributed controller optimizes the allocation of virtualized resources to meet an application performance objective in an energy efficient manner. It applies fuzzy modeling to capture the complex relationship between resource allocation and application's end-to-end response time, energy usage, and the coupling effects between neighboring applications that share the underlying physical resources. These models adapt in the face of dynamic workload variations using an online learning algorithm. The distributed controllers coordinate with each other to tackle the important problem of performance interference between co-located VMs. We design and implement APPLEware as a middleware solution for virtualized computing environments.

We evaluate APPLEware on a testbed of Dell PowerEdge servers using VMware virtual machines that host a mix of multi-tier and multi-service Internet applications. As many others in [11], [12], [13], we use RUBiS as the benchmark application in conducting the experiments. RUBiS is commonly used as a multi-tier benchmark application. We augmented it in multi-service forms as well. First, we compare APPLEware with a centralized control approach, which uses a single controller for managing the entire system. APPLEware achieves the improvement of 37 percent on average in terms of the relative error, which measures the deviation of application performance from its target. It also reduces the energy usage by 12 percent. This is due to its control agility, the ability to meet application performance targets and achieve the most energy-efficient system state within a short period of time.

Experimental results further demonstrate the effectiveness, and energy efficiency of APPLEware in the face of a dynamic workload derived from real web traces, and highly bursty workloads. For performance comparison, we consider two representative approaches. The first approach, PERFUME [11], is a traditional performance and power management technique that ignores the impact of performance interference between co-located applications. The second approach, PAC [14], represents a dynamic application consolidation technique for efficient resource sharing. PAC performs dynamic VM placement based on their resource usage patterns with the aim to load balance the server cluster while meeting each application's resource demands.

Compared with PERFUME, APPLEware achieves the improvement of 44 percent on average in terms of the relative error, while reducing the energy usage by 13 percent. Compared with PAC, APPLEware reduces the relative error by 65 percent while consuming merely 3 percent more energy. Our results also show that APPLEware is effective even in the presence of dynamic VM consolidation. Finally, we demonstrate APPLEware's scalability through testbed deployment as well as large-scale simulations.

Our contributions lie in the design and development of an autonomic performance and power control approach that: (1) significantly improves the performance and energy efficiency of co-located web applications in a virtualized server cluster, (2) is robust against dynamic and bursty workload variations, and performance interference among co-located VMs, (3) is robust against dynamic VM consolidation, and (4) is scalable due to its distributed control framework that decomposes the global control problem into local subproblems.

A preliminary version of the paper appeared in the Proc. of IEEE/ACM IWQoS'2013 [15]. In this extended manuscript, we enhanced the online learning component of APPLEware to address the complexity of multi-service architecture, developed a new mechanism to achieve robustness under dynamic VM consolidation, and conducted an extensive scalability analysis of the proposed approach. We observed that multi-service applications exhibited much higher deviation from their performance targets than traditional multi-tier applications in the face of dynamic workload variations. This is because the fuzzy model of multi-service applications, which require a larger set of fuzzy rules, is more sensitive to noise than the models of the multi-tier applications. To address this new challenge, we tuned APPLEware's online learning component (wRLS method) based on our sensitivity analysis of an important tuning parameter, the forgetting factor. We also conducted new experiments with a real workload trace, and compared the performance and energy efficiency with a representative application consolidation approach, PAC [14], and our previous work PERFUME [11].

In the following, Section 2 discusses related work. Section 3 presents the background and motivation of this work. Section 4 presents APPLEware architecture and design. Section 5 presents the testbed implementation. Section 6 provides the experimental results and analysis. Section 7 concludes the paper.

2 Related Work

Autonomic resource management for performance assurance of Internet applications is an important and active research topic. There are important studies in dynamic resource provisioning for delay guarantee in multi-tier Internet services [13], [16], [17], [18]. Urgaonkar et al. [13] proposed a dynamic server provisioning approach based on queueing models, which requires extensive application

profiling for each workload. An approach proposed in [18] models the probability distributions of response time based on CPU allocations on VMs in a datacenter. However, the performance model is not adaptive online to dynamically changing workloads.

Traditional power management techniques are not easily applicable to virtualized environments where physical processors are shared by multiple VMs. For instance, changing the power state of a processor by dynamic voltage scaling (DVS) will inadvertently affect the performance of VMs belonging to different applications [19]. Furthermore, applying DVS independently to a particular tier in a multi-tier application will affect the entire application due to the inter-tier dependency [20], [21].

There are recent studies on joint power and performance management of virtualized computing environments [11], [22], [23], [24], [25], [26], [27], [28], [29], [30]. Lim et al. [27] proposed a combination of hardware-based (e.g., DVS) and software-based (e.g., VM CPU time allocation) power control techniques to coordinate the power distribution among VMs within a peak power capacity. Verma et al. [29] combined VM resizing and live migration to ensure that datacenters can deal both with temporary power outages, and surges in workload. Gandhi et al. [23] proposed a hybrid approach that proactively allocates resources for the predictable demand pattern and leverages a reactive controller to deal with excess demand. Gong and Xu [31] applied the auto regressive moving average (ARMA) based modeling approach for power and performance control in virtualized servers. However, this approach performs a linear approximation of an inherently non-linear system, which results in modeling inaccuracies, and decrease in control effectiveness. PERFUME is a MIMO control based system for power and performance management of virtualized server [11]. It applies fuzzy modeling to capture the non-linear relationship of application performance with virtualized resources. However, it is interference-agnostic. Most related studies offer centralized techniques that are not scalable to large systems.

Shen et al. [32] presented, Cloudscale, an automatic resource scaling system for multi-tenant cloud. It integrates DVS technique, VM resource capping, and VM migration to meet application service level objectives (SLOs) with minimum resource cost and energy usage. However, it does not handle multi-tier application scaling, in which different tiers have interdependency and scaling on one tier can affect the others.

Jiang et al. [6] proposed an autonomous resource provisioning mechanism for multi-service applications. They apply queueing theoretical modeling for coarse-grained resource allocation on a standalone application. However, their approach does not address the performance interference between co-located applications.

Scalability is important for job scheduling and power management in data centers [33], [34]. Boutin et al. [33] proposed a distributed scheduling framework for data-parallel computation jobs over cloud-scale clusters. Keller et al. [34] developed a hierarchical control architecture that controls a data center's peak power consumption. However, these approaches do not control the performance of multi-tier and multi-service applications.

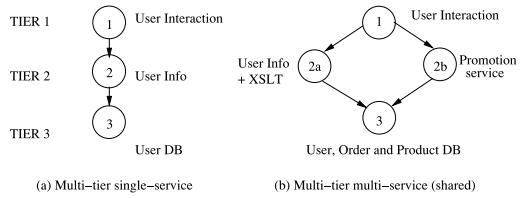


Fig. 2. Multi-tier and multi-service architectures.

The performance impact of shared resource contention in multi-core servers has been explored [7], [35], [36], [37]. There are resource partitioning techniques for performance isolation of applications running on a multi-core server. Lama and Zhou proposed a non-invasive performance isolation technique for virtualized servers in [12]. However, these studies did not address how to provide performance assurance for applications spanning multiple physical servers.

This paper addresses the multi-faceted challenges discussed above. It provides distributed control of virtualized resources for robust performance assurance, energy efficiency, and scalability in the presence of performance interference, highly dynamic, and bursty workloads.

3 Background and Motivation

3.1 Multi-Tier and Multi-Service Applications

Multi-tier web applications have a pipelined architecture in which each tier provides certain functionality to its preceding tier and uses the functionality provided by its successor to carry out its part of the overall request processing. This is illustrated by Fig. 2a. On the other hand, multi-service architecture comprises of a more complex set of disparate and collaborating services which are usually stateful and have interdependencies. Major web sites such as Amazon, eBay, etc have moved from 2-tier/3-tier architecture to a multi-service architecture for better scalability and manageability [6].

Fig. 2b shows an example of a multi-service application. The root service invokes the left branch for gathering user information, then the right branch for promoting product information to the same user. The User info service in turn accesses the shared data service, then invokes an external XSLT service to transform XML templates into HTML. The Promotion service first fetches users order histories from the shared data service, then searches for items related to users last orders using the Product data service in order to recommend further purchases. Finally, the root service combines the results from the two branches and returns it to the client.

Resource provisioning for performance management of web applications is challenging due to the complex inter-tier and inter-service relationships.

4 APPLEware Design

4.1 The Architecture

We design APPLEware to control a virtualized server cluster hosting multiple applications. We assume that each tier or service of a multi-tier multi-service application is deployed at a VM. Furthermore, VMs belonging to an application may span server nodes. APPLEware's core is a distributed control framework that decomposes the global control problem into local subproblems for scalability. Each

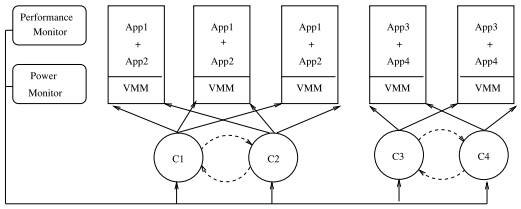


Fig. 3. APPLEware system architecture.

controller executes a control loop on a subsystem, which comprises of the VMs belonging to one application. The control actions are the adjustments in the virtualized resource allocation to meet the application performance target in an energy efficient manner.

The energy efficiency of APPLEware is mainly attributed to the fact that it applies *CPU* usage limits on VMs hosted on physical servers. This constrains the utilization of underlying physical processors thereby regulates power consumption. It is feasible due the idle power management of modern processors, which incorporate sleep states (C-states) to achieve substantive power savings when a processor is idle.

We observe that some subsystems are inherently coupled with each other due to shared resource contentions and performance interference. The action of one controller on a subsystem can affect the state of another subsystem. APPLEware addresses this challenge through coordination among neighboring controllers for effective performance assurance and energy efficiency.

Fig. 3 shows four distributed controllers, where each controller manages one application. Here, applications app1 and app2 span three server nodes and share the underlying physical resources. The controllers, C1 and C2, regulate the resource allocation of VMs belonging to app1 and app2 respectively. They coordinate with each other by exchanging information about their control decisions. Such coordination is important for control effectiveness, and system stability. This is because a control action taken by C1 or C2 affects the performance of both app1 and app2. Similarly the controllers, C2 and C4, coordinate with each other to control app3 and app4. In case of dynamic VM consolidation, the co-location of VMs may change at run time. We describe the mechanism to address this issue in Section 5.3.

4.2 System Modeling

We construct APPLEware's distributed model predictive control framework through various steps of system modeling and controller design. First, we consider a global system model that represents the behavior of multi-tier and multi-service applications spanning across virtualized server nodes. The inputs to the system are the *CPU* and *memory* usage limits at various tiers and services of the hosted applications. The outputs of the system are the performance and average energy usage of each application. We obtain two separate models for power and performance of the system, respectively. Table 1 summarizes the important notations.

The global system model is represented as follows:

$$\mathrm{Y}(k+1) = \mathrm{F} \cdot \mathrm{G(k)} + \mathrm{H} \cdot \mathrm{U(k)}, \tag{1}$$

where $\mathrm{U(k)}$ is a vector of resource allocations at sampling interval k. H is a matrix that represents the impact of current resource allocations on the system outputs, $\mathrm{Y}(k+1)$, at the next sampling interval. $\mathrm{G(k)}$ is a regression vector that contains the performance and energy usage values of each application in the current and previous sampling intervals. F is a matrix that represents the impact of regression vector on the system outputs.

TABLE 1
Notation Summary

Symbol	Description
y_i	Per-application performance or energy usage
u_i	Per-application vector of CPU and memory usage limits on local and neighboring VMs
u_j	Per-VM vector of CPU and memory usage limits
ξ_i	Per-application regression vector
ζ_i	Vector representing the impact of ξ_i on y_i
η_i	Vector representing the impact of u_i on y_i
ϱ	Length of per-application regression vector
ref_i	Per-application performance target
k	Index of the control sampling interval
i	Index of the application
j	Index of the VM component
r	Index of the fuzzy logic rule
γ	The forgetting factor

Considering n applications in the system,

$$\mathrm{Y}(k) = [y_1(k), y_2(k), .., y_n(k)]^T, \tag{2}$$

$$\begin{aligned}\mathrm{U(k)} = [&u_1(k), .., u_{l_1}(k), u_{l_1+1}(k), .., \\ &u_{l_1+l_2}(k), u_{l_1+l_2+1}(k), .., u_{l_1+l_2+..+l_n}(k)]^T,\end{aligned} \tag{3}$$

$$\mathrm{H} = \begin{bmatrix} \eta_{1,1} & \eta_{1,2} & .. & \eta_{1,L} \\ \eta_{2,1} & \eta_{2,2} & .. & \eta_{2,L} \\ \vdots & & & \vdots \\ \eta_{n,1} & \eta_{n,2} & .. & \eta_{n,L} \end{bmatrix},$$

$$\begin{aligned}\mathrm{G(k)} = [&y_1(k), .., y_1(k-(\varrho-1)), \ldots, \ldots, \ldots, y_n(k).., y_n(k- \\ &(\varrho-1))]^T,\end{aligned} \tag{4}$$

$$\mathrm{F} = \begin{bmatrix} \zeta_{1,1} & \zeta_{1,2} & .. & \zeta_{1,n\times\varrho} \\ \zeta_{2,1} & \zeta_{2,2} & .. & \zeta_{2,n\times\varrho} \\ \vdots & & & \vdots \\ \zeta_{n,1} & \zeta_{n,2} & .. & \zeta_{n,n\times\varrho} \end{bmatrix}.$$

In Eq. (2), the output term $y_i(k)$ represents the average end-to-end response time of application i at sampling interval k. The output term for power modeling is the average energy usage of an application. In Eq. (3), the input term $u_j(k)$ represents the allocation of *CPU* and *memory* usage limits on a particular VM component j. A VM provides the functionality of a particular tier or service of a multi-tier or multi-service application. The total number of components in application i is denoted by l_i. The total number of VM components in the entire system is denoted by $L = \sum_{i=1}^{n} l_i$. In matrix H, the term $\eta_{*,j}$ represents the impact of resource allocation $u_j(k)$ on the application performance or energy usage. In Eq. (4), ϱ specifies the number of samples of output variable y_i that is used in the system model, including the current and previous control intervals. In the regression

matrix F, the term $\zeta_{*,k}$ represents the impact of the system outputs measured at the k_{th} previous sampling interval on the application performance and energy usage.

4.3 Problem Decomposition

Autonomic performance and power management of a virtualized server cluster containing many complex applications involves a large scale optimization and control process. For scalability, APPLEware decomposes the global control problem into localized subproblems.

A local control problem includes its local variables associated with the managed application, and neighbor variables associated with other applications that have an impact on the performance and energy usage of the managed application. The local variables include a managed application performance, energy usage and the amount of virtualized resources allocated. The neighbor variables include the amount of resources allocated to the VM components of other applications that affect the local application due to shared resource contentions and performance interference. They influence the control decisions on the local subsystem. The resulting local model is described as:

$$y_i(k+1) = \zeta_i \xi_i(k) + \eta_i \mathrm{u}_i(k). \tag{5}$$

Here, the output variable $y_i(k)$ represents the performance or energy usage of application i. $\xi_i(k)$ and ζ_i are subsets of regression vector $G(k)$ and regression parameter matrix F respectively. They represent the current and previous outputs of application i and their impact on the application output in the next control interval. $\mathrm{u}_i(k)$ is a subset of vector U (k) that represents the allocation of *CPU* and *memory* resources on the VM components belonging to application i and the neighbor applications. η_i is a subset of matrix H that reflects the impact of resource allocation on the application output. As an example, a local controller C_2 in Fig. 3 uses a local system model with the following input variables.

$$\mathrm{u}_2(k) = [u_1(k), u_2(k), u_3(k), u_4(k), u_5(k), u_6(k), u_7(k)]^T.$$

From C2's perspective, $[u_4(k), u_5(k), u_6(k), u_7(k)]$ are a set of local input variables, which represent four VMs of App2. $[u_1(k), u_2(k), u_3(k)]$ are the neighbor variables, which represent three VMs of App1 as shown in Fig. 1. The local system model predicts the performance of App2 in the presence of interference from App1.

4.4 Fuzzy Model To Capture System Non-linearity

A linear system model is often inadequate to accurately represent the complex behavior of inherently non-linear systems such as a multi-tier multi-service application hosted in a virtualized computing environment. APPLEware addresses this issue by constructing fuzzy models. The models include the local and neighbor variables of a subsystem according to APPLEware's problem decomposition approach. A key strength of fuzzy model is its ability to represent highly complex and nonlinear systems by a combination of inter-linked models with simple functional dependencies.

4.4.1 Model Formulation

A subsystem corresponding to application i is represented by a fuzzy model as follows:

$$y_i(k+1) = \mathbb{R}(\xi_i(k), \mathrm{u}_i(k)). \tag{6}$$

Similar to Eq. (5), $y_i(k)$ is the output variable. $\mathrm{u}_i(k)$ consists of the local and neighbor input variables. The regression vector $\xi_i(k)$ includes current and previous outputs of application i.

$$\xi_i(k) = [y_i(k), .., y_i(k-(\varrho-1))]^T. \tag{7}$$

$\mathbb{R}$ is a rule based fuzzy model consisting of Z fuzzy rules. Each fuzzy rule is described as follows:

$\mathbb{R}_r$: *If* $\xi_{i1}(k)$ *is* $\Omega_{r,1}$ *and ..* $\xi_{i\varrho}(k)$ *is* $\Omega_{r,\varrho}$ *and* $u_1(k)$ *is* $\Omega_{r,\varrho+1}$ *and ..* $u_m(k)$ *is* $\Omega_{r,\varrho+m}$ *then*

$$y_i(k+1) = \zeta_r \xi_i(k) + \eta_r \mathrm{u}_i(k) + \phi_r. \tag{8}$$

Here, Ω_r is a set of fuzzy values, which describe the elements of regression vector $\xi_i(k)$ and the current input vector $\mathrm{u}_i(k)$ for the fuzzy rule, $\mathbb{R}_r$. The numeric values of $\xi_i(k)$ and $\mathrm{u}_i(k)$ are mapped to fuzzy values by using the corresponding fuzzy membership functions. For example, the fuzzy membership function of $\Omega_{r,1}$ determines the degree to which it can accurately describe $\xi_{i1}(k)$. ϕ_r is the offset vector. $y_i(k+1)$ is the estimated model output according to the rule $\mathbb{R}_r$.

Note that the fuzzy membership functions may overlap with each other. As a result, multiple fuzzy rules can be triggered by a given set of input values. Each fuzzy rule describes a region of the complex non-linear system by using a simple functional relation given by the rule's consequent part. The contribution of each rule to the model output is determined by its firing strength, β_r. It is the product of the membership degrees of the antecedent variables in that rule. The final model output is calculated as the weighted average of the linear consequents in the Z individual rules as follows.

$$y_i(k+1) = \frac{\sum_{r=1}^{Z} \beta_r (\zeta_r \xi_i(k) + \eta_r \mathrm{u}_i(k) + \phi_r)}{\sum_{r=1}^{Z} \beta_r}. \tag{9}$$

The model output is expressed in the form of

$$y_i(k+1) = \zeta_r^* \xi_i(k) + \eta_r^* \mathrm{u}_i(k) + \phi_r^*. \tag{10}$$

The aggregated parameters ζ_r^*, η_r^* and ϕ_r^* are the weighted sum of vectors ζ_r, η_r and ϕ_r respectively.

4.4.2 Machine Learning Based Model Construction and Adaptation

APPLEware constructs initial fuzzy models by applying a subtractive clustering technique on performance and energy usage data collected from the system. The technique partitions the input-output space and determines the number of fuzzy rules and the shape of membership functions. For instance, the performance model obtained for the multi-service application, App2 in Fig. 1, consists of 14 clusters in a nine dimensional space. The dimensions correspond to four local variables $[u_4(k), u_5(k), u_6(k), u_7(k)]$, three neighbor variables $[u_1(k), u_2(k), u_3(k)]$, one regression vector $\xi_2(k)$, and one output variable $y_2(k+1)$. Each cluster center describes a fuzzy rule in which the fuzzy values Ω_r are represented by gaussian functions. The cluster centers determine the mean of the gaussian functions, and a

tunable parameter in the clustering technique determines the variance.

APPLEware applies an adaptive network based fuzzy inference system (ANFIS) [38] to further tune the fuzzy model parameters. It constructs an artificial neural network to represent a fuzzy model and tunes its parameters using a combination of back-propagation algorithm with a least squares method. This adjustment allows the fuzzy system to learn from the data it is modeling.

A static system model can not provide sufficient prediction accuracy of power and performance for all workload variations. Hence, APPLEware applies a computationally efficient wRLS method [11] to adapt the consequent parameters of its fuzzy models. The technique continuously samples new measurements from the system, and updates the model parameters in response to the prediction errors made by the existing models.

To apply the wRLS method, we express the fuzzy model output in Eq. (10) as follow:

$$y_i(k+1) = X_i\theta_i(k) + e_i(k), \tag{11}$$

where $e_i(k)$ is the error value between actual output of the system (i.e., measured performance or energy) and predicted output of the model. $\theta_i = [\theta_{i1}^T\theta_{i2}^T..\theta_{ip}^T]$ is a vector composed of the model parameters. $X_i = [w_1X_i(k), w_2X_i(k), \ldots, w_ZX_i(k)]$ where w_r is the firing strength of r_{th} rule and $X_i(k) = [\xi_i^T(k), \mathrm{u}_i(k)]$ is a vector containing current and previous outputs and inputs of the system. The parameter vector $\theta_i(k)$ is estimated so that the following cost function is minimized. That is,

$$Cost = \sum_{j=1}^{k} \gamma^{k-j} e_i^2(j), \tag{12}$$

where $0 < \gamma \leqslant 1$ is the *forgetting factor* which gives exponentially less weight to older error samples.

The fuzzy model parameters are updated as follows:

$$\theta_i(k) = \theta_i(k-1) + Q_i(k)X_i(k-1)[y_i(k) - X_i(k-1)\theta_i(k-1)]. \tag{13}$$

$$Q_i(k) = \frac{1}{\gamma}\left[Q_i(k-1) - \frac{Q_i(k-1)X_i(k-1)X_i^T(k-1)Q_i(k-1)}{\gamma + X_i^T(k-1)Q_i(k-1)X_i(k-1)}\right]. \tag{14}$$

Here $Q_i(k)$ is the updating matrix. The initial value of $\theta_i(0)$ is obtained in the off-line identification. And, the initial value of $Q(0)$ is equal to $(X_i^TX_i)^{-1}$.

4.4.3 Modeling Multi-Tier versus Multi-Service Application

We observe that our fuzzy model uses 14 fuzzy rules to represent the performance of the multi-service application, App2, with sufficient prediction accuracy. Whereas, the models use only 9 fuzzy rules in case of multi-tier applications (App1, App3, and App4). This is because the multi-service application exhibits more complex relationship between performance and resource allocations. Hence, they require more complex models for performance prediction. As a consequence, we found that the wRLS method was less effective in accurately adapting the fuzzy model for a multi-service application in response to dynamic workload variations. We address this issue by tuning the wRLS method based on our sensitivity analysis of the *forgetting factor* in Equation (12). Our experiments in Section 6.2.3 show the impact of this phenomenon, and the effectiveness of our solution.

4.5 Controller Design

4.5.1 Control Formulation

A local control objective of controller C_i is given by the following cost function:

$$V_i(k) = \sum_{p=1}^{H_p} ||ref_i - y_{i1}(k+p)||_P^2 + \sum_{p=1}^{H_p} ||y_{i2}(k+p)||_Q^2 + \sum_{c=0}^{H_c-1} ||\Delta \mathrm{u}_i(k+c)||_R^2. \tag{15}$$

Here, $y_{i1}(k)$ is the average end-to-end response time and $y_{i2}(k)$ is the average energy usage of application i at control interval k. The controller predicts both energy usage and performance over H_p control periods, called the *prediction horizon*. It computes a sequence of control actions $\Delta \mathrm{u}_i(k), \Delta \mathrm{u}_i(k+1), .., \Delta \mathrm{u}_i(k+H_c-1)$ over H_c control periods, called the *control horizon*, to keep the predicted performance close to its pre-defined target ref_i while minimizing the energy usage. The control action $\mathrm{u}_i(k)$ is the change in *CPU* and *memory* usage limits imposed on various tiers and services of the multi-tier multi-service applications. P and Q are the tracking error weights that determine the trade-off between power and performance. The third term in Eq. (15) represents the control penalty and is weighted by R. It penalizes big changes in control action for system stability.

The control problem is subject to the constraint that the sum of *CPU* and *memory* resources allocated to all VM components in the same physical server node must be bounded by its *CPU* and *memory* capacity.

4.5.2 Distributed Control Algorithm

APPLEware executes a controller C_i based on the distributed model predictive control approach, as shown in Algorithm 1.

Algorithm 1. Distributed Control Algorithm

1: **loop**
2: Measure the current state of the subsystem in terms of local and neighbor variables.
3: **repeat**
4: Optimize the local control objective (Eq. (15)).
5: Send control solutions to neighboring controllers.
6: Receive control solutions computed by neighboring controllers.
7: **until** The common variables in the control solutions converge to a steady value.
8: Execute control actions by adjusting *CPU* and *memory* resources assigned to application i.
9: **end loop**

For the proof of convergence of this algorithm, we refer the readers to [39].

4.5.3 Speeding Up Local Control

Although the decomposition of global system model into local subproblems reduces the computational complexity to a large extent, solving each local control problem still involves a non-convex [40] and time-consuming optimization as formulated in Eq. (15). APPLEware addresses this issue by transforming each local control problem into a standard quadratic programming problem. For this transformation, it linearizes the fuzzy model at the current operating point and represent it as a state-space linear time variant model as follows:

$$\begin{aligned} x_i(k+1) &= A(k)x_i(k) + B(k)\mathrm{u}_i(k). \\ \mathrm{y}_i(k) &= C(k)x_i(k). \end{aligned} \tag{16}$$

The state vector is defined as

$$x_i(k+1) = [\xi_i^T(k), 1]^T. \tag{17}$$

The matrices $A(k)$,$B(k)$ and $C(k)$ are constructed by freezing the parameters of the fuzzy model at a certain operating point $\mathrm{y}_i(k)$ and $\mathrm{u}_i(k)$ as follows. First, we calculate the degree of fulfillment β_r for the current inputs (i.e., *CPU* and *memory* usage limits) chosen for the application and compute the aggregated parameters ζ^*, η^* and ϕ^*. Comparing Eq. (10) and Eq. (16), the state matrices are computed as follows:

$$A = \begin{bmatrix} \zeta^*_{1,1} & \zeta^*_{1,2} & .. & .. & .. & \zeta^*_{1,\varrho} & \phi^*_1 \\ 1 & 0 & .. & & & 0 & 0 \\ & & & & & & \\ 0 & 1 & \vdots & & & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} \eta^*_{1,1} & \eta^*_{1,2} & .. & \eta^*_{1,m} \\ 0 & .. & .. & 0 \\ \vdots & & & \vdots \end{bmatrix} C = [1 \quad 0 \quad .. \quad .. \quad .. \quad .. \quad 0],$$

where ζ^*_{ij} and η^*_{ij} are the jth element of aggregate parameter vectors ζ^* and η^* for application i.

The MIMO control problem defined by Eq. (15) is transformed to a quadratic program:

$$\text{Minimize } \frac{1}{2}\Delta\mathrm{u}_i(k)^T H \Delta\mathrm{u}_i(k) + c^T \Delta\mathrm{u}_i(k) \tag{18}$$

subject to constraint $\Omega\Delta\mathrm{u}_i(k) \leq \omega$.

The matrices Ω and ω are chosen to formulate the constraints on *CPU* and *memory* resource usage. Here, $\Delta\mathrm{u}(k)$ is a matrix containing the *CPU* and *memory* usage limits on each VM over the entire control horizon H_c. In the minimization formulation,

$$H = 2(R_{1u}^T P R_{1u} + R_{2u}^T Q R_{2u} + R).$$

$$c = 2[R_{1u}^T P^T (R_{1x} Ax(k) - ref_i) + R_{2u}^T Q^T R_{2x} Ax(k)]^T. \tag{19}$$

The matrices R_{1u}, R_{1x} are associated with the performance models of hosted applications and matrices R_{2u}, R_{2x} are associated with the energy usage model.

$$R_{iu} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{H_p-1} \end{bmatrix}$$

$$R_{ix} = \begin{bmatrix} CB & 0 & .. & 0 \\ CAB & CB & .. & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{H_p-1}B & CA^{H_p-1}B & .. & CA^{H_p-H_c}B \end{bmatrix}.$$

4.5.4 Computational Complexity Analysis

The computation overhead of APPLEware is dominated by the quadratic programming problem. We choose a widely used algorithm, the interior point method, to solve this problem. The algorithm has a computational complexity of $O(N)$ Newton iterations [41]. Here N is the number of decision variables that need to be computed to solve the given problem. Since each Newton iteration requires $O(N^3)$ algebraic operations, the worst-case computation complexity of the quadratic program solver is cubic in the number of decision variables. APPLEware is able to significantly reduce the computation overhead by decomposing the global control problem into local subproblems. For APPLEware, the value of N depends on the local and neighbor variables only.

Two tunable parameters, H_p and H_c affect the controller performance, and computation overhead. The prediction horizon H_p should be large enough so that the system converges to the reference target. However, larger value of H_p also increases the computation load. As the control horizon H_c increases, the dimension of the solution space increases, thereby improving the quality of control solutions. However, it also increases the time required to find the optimal solution. For our experiments presented in Section 6, H_p was tuned to 20, which was sufficiently large for stable control. H_c was tuned to 5, which provided good control performance.

5 System Implementation

5.1 Testbed

We built a testbed in a university prototype datacenter, which consists of Dell PowerEdge R610 servers. Each server has two Intel hexa-core Xeon X5650 CPUs and 32 GB memory. The servers are connected with 10 Gbps Ethernet. The testbed hosts three multi-tier applications (App1, App3 and App4) and one multi-service application (App2) as shown in Fig. 1. Each tier and service of an application is implemented on a VMware virtual machine with 1 VCPU, 1 GB RAM and 15 GB hard disk space. Each controller runs on a VM having 300 Mhz CPU usage limit and 128 Mb memory usage limit. The lightweight controllers do not interfere with the performance of the hosted applications.

For performance evaluation, we deploy the RUBiS benchmark in multi-tier and multi-service forms as shown in Fig. 1. A multi-tier deployment of RUBiS has a simple pipelined architecture consisting of web, application and database servers. Whereas a multi-service deployment is more complex. It has a web service at the front end to handle the HTML requests. There are two application services, one for processing the requests that read from the database, another for processing all requests that write to the database. The database is implemented as a single shared service. In our implementation, a tier and service at the front end of the application runs the Apache web server. The application tier and services run PHP servers. The database tier and the shared data service component run a MySQL server.

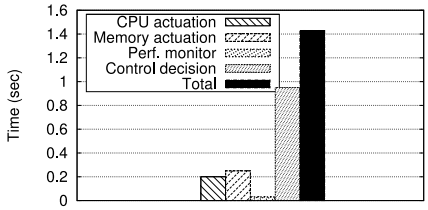


Fig. 4. APPLEware's average performance overhead.

5.2 APPLEware Components

Power monitor. The power monitor uses vSphere API to measure the average energy usage of VMs at each control interval. VMware ESX 4.1 gathers such data through its Intelligent Power Management Interface sensors.

Performance monitor. APPLEware collects the application response time values from the web-tier access logs, which are available in typical e-commerce applications. We inject an XML-RPC daemon program that runs at the web tier to measure the average end-to-end response time of requests. The performance monitor uses an XML-RPC client that communicates with RUBiS application to collect the performance statistics at each control interval.

System modeling. APPLEware invokes the Fuzzy Logic Toolbox in MATLAB to apply subtractive clustering and ANFIS modeling techniques on the data collected from the virtualized server system. It provides an API that allows the system administrator to load initial training data set for system modeling. At runtime, the wRLS algorithm updates the system models dynamically.

Distributed controller. Each controller invokes a quadratic programming solver, *quadprog*, in MATLAB to compute the local control solution. We used MATLAB Builder JA to create a Java class from the MATLAB program invoking *quadprog*. This Java class is integrated into APPLEware source code and deployed to each controller node. The distributed controllers communicate with each other using XML-RPC protocol.

Actuator. It uses vSphere API's *ReconfigVM_Task* method to set a VM's *CPU* and *memory* usage limits.

5.2.1 Performance Overhead

The performance overhead of APPLEware's distributed controllers is mainly affected by three factors: (1) time taken to collect performance statistics from the applications, (2) time required to compute a control decision, (3) actuation time. Fig. 4 shows the average time taken for each of these factors on our testbed hosting four applications. Throughout the paper, we set the control interval of APPLEware's distributed controllers to be 10 seconds, which is sufficiently large to overcome the control overheads and also avoid measurement noise.

5.3 Addressing Dynamic VM Consolidation

Dynamic VM consolidation is commonly used for load balancing a virtualized server cluster by migrating VMs from overloaded servers to lightly loaded servers, and also consolidating new VMs. As a consequence, the co-location of VMs may change dynamically. Hence, APPLEware's system models need to be updated accordingly. APPLEware addresses this challenge as follows.

5.3.1 Online Detection of Co-Location Changes

Each controller module in APPLEware's distributed control framework is launched with an application-specific performance model, which is obtained by offline training. We assume that the initial model does not include any information about co-located VMs. However, the controller module periodically fetches the list of VMs that are co-located with the VMs that belong to the application under control. For this purpose, the controller uses the vSphere API to first get the object reference to the host system that is responsible for running the application VMs. This information is available in the *VirtualMachineRuntimeInfo* property of the application VMs. Then, it gets the list of VMs associated with those hosts using their object references. Thus, APPLEware detects the changes in VM co-location.

5.3.2 Fuzzy Model Reconstruction

Unlike workload variations, changes in the VM co-location are relatively infrequent. However, they introduce new input parameters in the system model at run time. Hence, the whole structure of the system model needs to be updated to achieve interference-aware control. For this purpose, each controller gradually accumulates new data measured from the system since the last detection of a change in VM co-location. Meanwhile, the controller uses the application-specific performance model, which is obtained by initial offline training. Since the initial model does not include information about co-located VMs, it is interference-agnostic. This mechanism ensures that during the data accumulation phase, the controller performs no worse than an entirely interference-unaware control technique. We set the data accumulation phase to be five minutes, which is sufficient to collect the training data online. After collecting the data samples, APPLEware applies the subtractive clustering technique to reconstruct the fuzzy model with new rules, and parameters as described in Section 4.4.2. The training time required to reconstruct the fuzzy model is less than 10 seconds.

6 Performance Evaluation

We conduct all performance evaluation in our testbed of five physical servers, except for the experiments related to scalability analysis. In general, our experiments assume a static VM consolidation scenario. We evaluate APPLEware's robustness against dynamic VM consolidation as a separate experiment in Section 6.2.4.

6.1 Model Validation

We first validate APPLEware's system models using a multi-tier application App1 and a multi-service application App2. Note that various components of App2 are co-located with VMs belonging to App1. The initial models are obtained by using a training data of average end-to-end response time and energy usage measurements of the two applications subject to randomly varying *CPU* and *memory* usage limits. Each application faces a browsing workload mix of 600 concurrent users. The offline training session includes 10 minutes of data collection, followed by the training of fuzzy models per application, which completed within 10 seconds. For model validation, we use a different set of resource allocations that is not used for training the models.

TABLE 2
APPLEware's Model Validation for the Multi-Service Application (App2)

	Service1	Service2	Service3	Service4	Measured resp. time	Predicted resp. time	Measured energy	Predicted energy
CPU	400 Mhz	1100 Mhz	500 Mhz	800 Mhz	2532 ms	2658.6 ms	35 KJ	37.2 KJ
Mem	768 Mb	256 Mb	512 Mb	1024 Mb				
CPU	400 Mhz	700 Mhz	400 Mhz	1300 Mhz	1669 ms	1777.4 ms	31.2 KJ	32 KJ
Mem	256 Mb	128 Mb	768 Mb	768 Mb				
CPU	500 Mhz	1200 Mhz	500 Mhz	1000 Mhz	1304 ms	1277.9 ms	30 KJ	29 KJ
Mem	512 Mb	1024 Mb	512 Mb	1024 Mb				

Table 2 shows the performance and energy usage prediction results of APPLEware for App2. In this experiment, we statically allocate 1,000 GHz CPU and 512 MB memory at each VM component of the co-hosted App1. We compare the measured and predicted values of average end-to-end response time and average energy consumption over a period of one hour for various *CPU* and *memory* allocations on App2's VMs. Both prediction errors for performance and energy usage are under 7 percent, which confirms APPLEware's modeling accuracy.

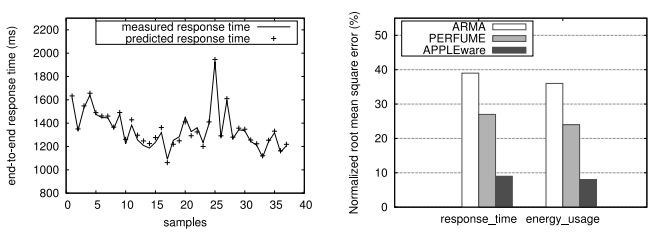
Next, we demonstrate APPLEware's accuracy in predicting the impact of performance interference between co-located VMs on the application performance. Fig. 5a shows the variations in the average end-to-end response time of App1 due to the interference caused by various resource allocations on App2. Note the resources allocated to App1 remains fixed. APPLEware is able to accurately predict application performance with a small normalized root mean square error (NRMSE) of 9 percent.

Fig. 5b compares APPLEware's performance and energy usage prediction accuracy with PERFUME [11] and a representative modeling approach ARMA [31]. APPLEware's superior prediction accuracy is due to its fuzzy modeling that captures the non-linear relationship of performance and energy with virtualized resources, while considering the impact of performance interference among co-located VMs. In contrast, the modeling approach used in PERFUME is interference-agnostic, and the ARMA modeling approach performs a linear approximation of the inherently non-linear system.

6.2 Performance Control and Energy Efficiency

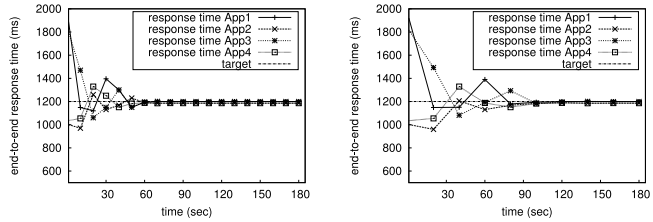
6.2.1 Control Agility

We evaluate APPLEware's effectiveness and agility in assuring the performance and reducing the energy usage of co-located web applications. For comparison, we develop a centralized controller that uses a global system model as described in Section 4.2. As a performance metric, we use the average end-to-end response time, which represents the user-perceived performance of web applications. We apply a stationary workload of 600 concurrent users, and set the service-level-agreement (SLA) target of 1,200 ms to each application.



(a) Performance (App1). (b) Prediction accuracy.

Fig. 5. APPLEware's prediction accuracy.



(a) with APPLEware. (b) with the centralized control.

Fig. 6. Performance assurance.

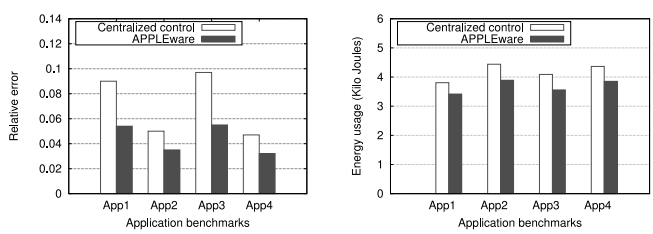
Fig. 6a shows that APPLEware brings the average end-to-end response time of the applications close to their respective SLA targets within 60 seconds. This is due to the agile and effective adjustments in the *CPU* and *memory* resources of the multi-tier multi-service applications by APPLEware's distributed controllers. On the other hand, Fig. 6b shows that a centralized controller takes 100 seconds to meet the SLA target.

In this experiment, we found the worst-case control overhead of the centralized controller to be seven seconds. As a result, its control interval needs to be much larger than APPLEware's control interval of 10 seconds. Hence, APPLEware provides better control agility than a centralized controller. Furthermore, the control solutions of APPLEware converge close to the optimal solutions obtained by the centralized controller. The convergence takes place in each control interval.

To quantify APPLEware's effectiveness in assuring application performance, we use relative error as the metric. The relative error for performance is $|y(k) - r|/r$, where $y(k)$ is the average end-to-end response time of an application at time interval k and r is the application's SLA target. Figs. 7a and 7b show that APPLEware reduces the relative error and the energy usage of each application, compared with the centralized controller. On average, the improvement in the relative error by APPLEware is 37 percent. It is also 12 percent more energy efficient than the centralized controller. The improvement in energy efficiency is due to the fact that APPLEware drives the system towards optimal operating conditions more quickly than the centralized controller does.

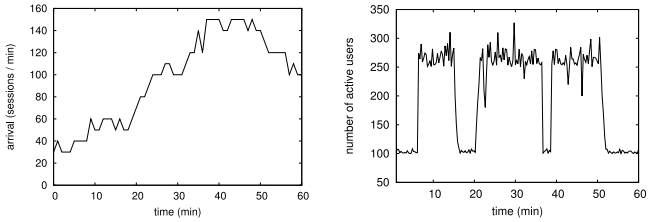
6.2.2 Robustness under Workload Variations.

We now evaluate APPLEware's robustness in the face of a dynamic workload based on real web traces and a highly bursty workload. We modify the RUBiS client to generate workload based on the web traces from the 1998 Soccer World Cup site [42]. These traces contain the number of arrivals per minute to this website over an eight-day period. As a case study, we choose the workload trace of a



(a) Performance. (b) Energy usage.

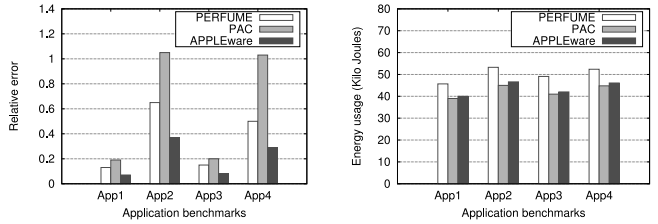
Fig. 7. Performance and energy efficiency improvement due to APPLEware's distributed control.



(a) Web traces from the 1998 Soccer World Cup site.

(b) a bursty workload.

Fig. 8. Workloads to evaluate the robustness of APPLEware.



(a) Performance.

(b) Energy usage.

Fig. 10. Comparison between PERFUME, PAC, and APPLEware under bursty workload and real workload trace.

moderately busy day and compress the original 24-hour long trace to 1 hour, similar to the related work in [13]. The resulting workload, which is shown in Fig. 8a, is applied to App1 and App3. Furthermore, we apply a bursty workload shown in Fig. 8b to App2 and App4. Bursty workloads, which are common in real world web applications, manifest sudden and huge increase in the web traffic. We inject burstiness into the arrival process of RUBiS clients according to the index of dispersion, I. The dispersion index modulates the think times of users between submission of consecutive requests [10]. The larger the value of I, the longer the duration of the traffic surge. We set the value of I to 4,000 and the maximum concurrent users to 1,000.

For performance comparison, we consider the PERFUME [11] and PAC [14] approaches. PERFUME applies a Model Predictive control technique to assure the performance of web applications based on their SLA targets. On the other hand, PAC relies on the resource usage patterns of various applications to consolidate them without degrading their performance. In this experiment, we used PAC to match the CPU and memory usage signature of application VMs with the residual resource usage patterns of the available hosts. In contrast to APPLEware, both PERFUME and PAC ignores the impact of performance interference on co-located VMs.

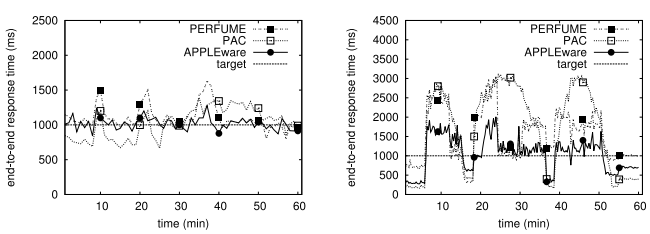
As shown in Fig. 9a, APPLEware is able to keep the average end-to-end response time of App1 close to the SLA target of 1,000 ms in spite of the significant increase in the trace-based workload from time 20 minute onwards. On the other hand, both PERFUME and PAC show significant fluctuations in the application performance. APPLEware is more robust than the other two approaches due to its ability to accurately adapt its fuzzy model in response to dynamic system behavior, and interference effects between co-located VMs. As shown in Fig. 9b, the fluctuations in the average response time of App2, which faces a bursty workload, is more significant. This is mainly due to two reasons. First, the impact of performance interference is more significant in case of a bursty workload. Since PERFUME and PAC are interference-agnostic, their effectiveness is further reduced in case of App2. Second, a bursty workload increases the performance loss among the consolidated applications, even at modest CPU utilization levels [10]. Hence, the underlying principle of PAC, which consolidates VMs until the servers are overloaded, is ineffective in meeting the application's SLA target.

Figs. 10a and 10b show the average relative errors and the energy usage of the hosted applications under the dynamic and bursty workloads. Compared with PERFUME, there is the improvement of 44 percent on average in terms of relative error by APPLEware. At the same time, APPLEware improves the energy efficiency by 13 percent. Compared with PAC, APPLEware reduces the relative error by 65 percent while consuming merely 3 percent more energy. This is because PAC tends to underprovision resources in the face of performance interference among co-located applications and burstiness in the workload.

6.2.3 Impact of wRLS Method on the Performance of Multi-Tier and Multi-Service Applications

APPLEware's wRLS method described in Section 4.4.2, plays an important role in its robustness under workload variations. We compare APPLEware's performance with and without the online learning capability in the face of dynamic and bursty workloads. Fig. 11a shows that the online learning capability reduces the average relative errors of the multi-tier applications (App1, App3, and App4) by 45 percent. In case of the multi-service application (App2), the relative error is reduced by 23 percent. The performance gain achieved by the wRLS method is less pronounced for multi-service application. This is because the fuzzy model of the multi-service application, which consists of a larger set of fuzzy rules, is more sensitive to noise than the models of the multi-tier applications. Hence, the wRLS method needs to be tuned differently for multi-tier and multi-service applications.

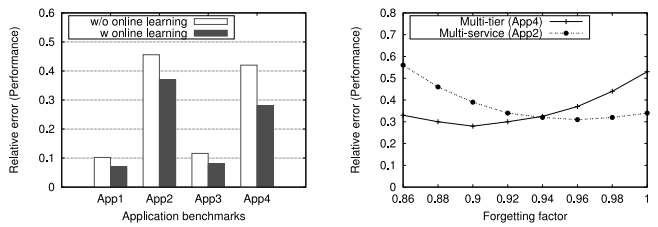
An important design parameter of the wRLS method is the *forgetting factor* γ. If γ is large, the algorithm forgets the



(a) With real workload trace (App1).

(b) With a bursty workload (App2).

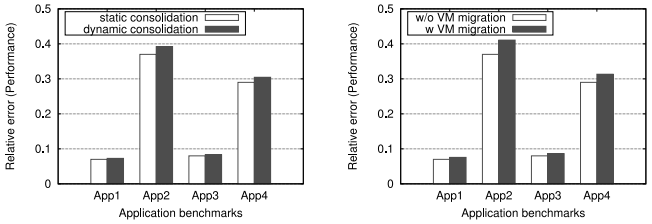
Fig. 9. Performance under workload variations.



(a) Robustness under workload variations.

(b) Multi-tier vs. multi-service application.

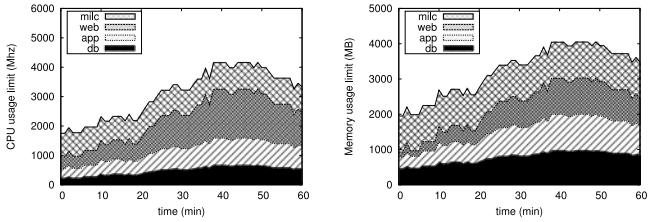
Fig. 11. Impact of online learning and forgetting factor.



(a) Impact of dynamic consolidation.

(b) Impact of VM migration.

Fig. 12. Robustness under dynamic VM consolidation.



(a) CPU allocation.

(b) Memory allocation.

Fig. 13. Interference-aware resource allocation.

past data less aggressively while updating the fuzzy model, which leads to better stability and robustness to noise, but slower convergence rate. A small value of γ has the opposite effect. Fig. 11b shows the impact of the *forgetting factor* on the relative errors of multi-tier and multi-service applications. We observe that the optimal values of γ for the multi-tier and multi-service applications are 0.9, and 0.96 respectively. Since the fuzzy model of multi-service application is more sensitive to noise, a higher value of γ is suitable for it. However, if $\gamma > 0.96$, the relative error increases due to slower convergence rate of the wRLS method.

Our results suggest that the forgetting factor should be large enough to counter the effect of an applications noise sensitivity, but not too large to maintain a fast convergence rate for the wRLS method. One potential mechanism to detect an application's noise sensitivity, and to automatically tune the forgetting factor is to analyze the correlation between a chosen forgetting factor and the estimation error of the fuzzy model. For instance, when the initial value of the forgetting factor is set to a small value ($\ll 1$) and the estimation error measured over a set of sampling intervals is also small, it may be concluded that the application is not sensitive to noise. In this case, there is no need to change the forgetting factor. However, if the estimation error is large over a set of sampling periods, then it indicates that the application is sensitive to noise, and hence the forgetting factor should be increased to suppress noise sensitivity. We will explore auto-tuning of forgetting factor in future work.

6.2.4 Robustness under Dynamic VM Consolidation

To evaluate the robustness of APPLEware in the presence of dynamic VM consolidation, we enabled APPLEware's capability to reconstruct its fuzzy models as described in Section 5.3, and conducted two types of experiments. First, we consolidated each application (App1, App2, App3 and App4) on the virtualized server cluster one by one in 10 minute intervals. The application VMs were randomly placed on the physical servers, and they were allowed to run for an hour. We repeated the experiment 20 times, and measured the average relative error for each application performance. Fig. 12a compares each application's average relative error under dynamic VM consolidation with its relative error under a static consolidation scenario, where the co-location of VMs are known apriori. The increase in relative error due to dynamic VM consolidation is under 4 percent on average.

Next, we consolidated the four applications as shown in Fig. 1 and randomly migrated a VM from one physical server to another on 10 minute intervals. In practice, VM migration for load balancing is done conservatively due to the overheads involved with migration, and the dynamic nature of resource demands. For this experiment, we used VMware's vMotion capability, which took 20 seconds on average to complete a live VM migration. We repeated the experiment 20 times, and measured the average relative error for each application performance. As shown in Fig. 12b, the increase in relative error due to VM migration is under 6 percent on average. Overall, our results show that dynamic VM consolidation has little impact on the APPLEware's effectiveness.

6.2.5 Co-Locating Memory Intensive Application

We evaluate APPLEware's ability to control CPU and memory allocation for performance assurance, when a memory intensive application is co-located with a multi-tier web application. We consolidate a VM running the MILC benchmark with another VM that runs the web tier of RUBiS application. The application and database tiers of RUBiS are running separately on different servers. MILC is a memory intensive benchmark of the SPEC CPU2006 benchmark suite [9]. We measure its performance in terms of IPC (instruction per cycle) normalized to the IPC value when it is running solo. The performance target is set to be one. The RUBiS application faces a dynamic workload, shown in Fig. 8a, with a browsing workload mix. Its average response time target is set to be 1,000 ms.

Figs. 13a, and 13b show that the CPU and memory allocation at the web, application, and database tiers increase until time 40 minutes, and then decrease after that. This is because APPLEware dynamically adjusts the CPU and memory usage limits of the application VMs in response to dynamic workload variations. Importantly, we observe that the increase in CPU allocation at the web tier is larger than that at the other tiers. This additional increase in CPU allocation is required to mitigate the performance interference imposed by the MILC application on the web tier. On the other hand, the CPU allocation for MILC does not change significantly. This is because MILC suffers negligible performance degradation due to the co-located RUBiS application. Previous studies have shown that MILC does not suffer from extra cache misses when it is co-located with other applications, since it hardly ever reuses its cached data [9]. As a result of APPLEware's dynamic resource allocation, the RUBiS application is able to maintain a small relative error of 0.08 in its performance with respect to its target. The normalized performance of MILC is close to one.

6.3 Scalability Analysis

We analyze APPLEware's scalability through testbed deployment as well as large-scale simulations. Due to the

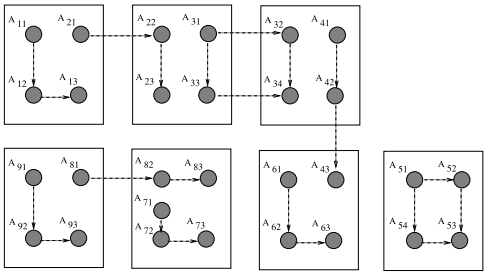
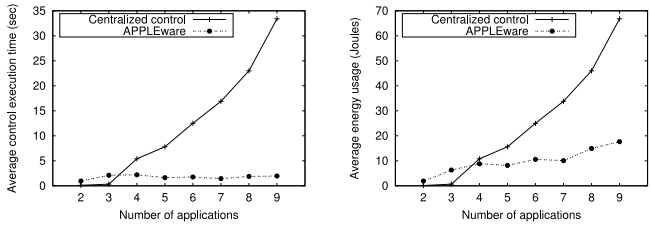


Fig. 14. Testbed for scalability analysis.



(a) Per-controller overhead.

(b) Total energy usage of the controllers.

Fig. 15. APPLEware overhead analysis.

space limitation, the results of our large-scale simulations are presented in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2015.2453971.

6.3.1 Testbed Deployment

For scalability analysis, we build a testbed of seven physical servers running a total of 29 VMs hosting nine multi-tier applications, as shown in Fig. 14. A VM running the jth tier of an application i is denoted by A_{ij}. APPLEware uses nine distributed and lightweight controllers. Each controller C_i runs on a VM that has 300 Mhz CPU usage limit and 128 Mb memory usage limit.

First, we measure the performance overhead of the controllers when the total number of applications in the system are increased. Fig. 15a shows that the per-controller execution time of APPLEware becomes significantly smaller than that of the centralized controller with increasing number of applications. It should be noted that although the control overhead of centralized controller is close to $5\sim6$ seconds (for four applications), its control interval needs to be larger than that of APPLEware (> 10 sec). This is because there should be sufficient time gap between the control action actuated in terms of CPU and memory resource allocation, and the next sampling measurement of application performance. This is necessary in order to capture the impact of control action accurately, and to reduce measurement noise. We observe that in cases of two and three applications, the overhead of APPLEware is slightly larger than that of the centralized controller. This is because the overhead caused by the coordination among APPLEware's distributed controllers overshadows the pure computation overhead, when few applications are being managed.

Fig. 15b compares the total energy usage of APPLEware's controllers themselves with that of the centralized controller. The energy usage of the centralized controller increases rapidly with increasing number of applications. On the other hand, APPLEware's energy usage shows a gradual increase. This is because the total energy usage is dominated by the execution time of the controllers. Energy usage is a product of the average power consumption and control execution time.

7 Conclusions

The user perceived performance of Internet applications and the energy usage of hardware resources is the result of a complex interaction of various workloads in a very complex underlying system. The increasing scale and complexity of virtualized server systems hosting multi-tier multi-service applications pose significant challenges to performance and power management. APPLEware is a scalable middleware for autonomic performance and power control in virtualized data centers. The proposed and developed middleware is based on a distributed control framework. It integrates the strengths of machine learning based adaptive system modeling and a distributed control algorithm. As demonstrated by modeling, analysis and experimental results based on testbed implementation, its main contributions are robust performance assurance, energy efficiency and scalability in the presence of performance interference, highly dynamic and bursty workloads, and dynamic VM consolidation.

Our future work will extend APPLEware's compatibility to XenServer, and analyze the impact of various control knobs.

Acknowledgments

This research was supported in part by US National Science Foundation (NSF) CAREER award CNS-0844983 and research grant CNS-1217979, and NSF of China research grant 61328203. The authors thank the anonymous reviewers for their valuable suggestions for revising the manuscript. X. Zhou is the corresponding author.

References

[1] D. Gmach, J. Rolia, and L. Cherkasova, "Resource and virtualization costs up in the cloud: Models and design choices," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 395–402.

[2] G. Jung, K. R. Joshi, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2010, pp. 497–506.

[3] C. Pham, D. Chen, Z. Kalbarczyk, R. Iyer, S. Sarkar, and R. Hosn, "Cloudval: A framework for validation of virtualization environment in cloud infrastructure," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 189–196.

[4] R. Singh, D. Irwin, P. Shenoy, and K. K. Ramakrishnan, "Yank: Enabling green data centers to pull the plug," in *Proc. USENIX Conf. Netw. Syst. Design Implementation*, 2013, pp. 143–156.

[5] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing: Degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, p. 7, 2008.

[6] D. Jiang, G. Pierre, and C.-H. Chi, "Autonomous resource provisioning for multi-service web applications," in *Proc. ACM 19th Int. World Wide Web Conf.*, 2010, pp. 471–480.

[7] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2010, pp. 237–250.

[8] A. Fedorova, M. Seltzer, and M. D. Smith, "Improving performance isolation on chip multiprocessors via an operating system scheduler," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2007, pp. 25–38.

[9] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proc. Int. Conf. Archit. Support Programm. Lang. Oper. Syst.*, 2010, pp. 129–142.

[10] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Injecting realistic burstiness to a traditional client-server benchmark," in *Proc. IEEE 6th Int. Conf. Autonomic Comput.*, 2009, pp. 149–158.
[11] P. Lama and X. Zhou, "Coordinated power and performance guarantee with fuzzy mimo control in virtualized server clusters," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 97–111, Jan. 2015.
[12] P. Lama and X. Zhou, "NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers," in *Proc. 42nd IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2012, pp. 1–12.
[13] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier Internet applications," *ACM Trans. Auton. Adaptive Syst.*, vol. 3, no. 1, pp. 1–39, 2008.
[14] Z. Gong and X. Gu, "PAC: Pattern-driven application consolidation for efficient cloud computing," in *Proc. IEEE Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst.*, 2010, pp. 24–33.
[15] P. Lama, Y. Guo, and X. Zhou, "Autonomic performance and power control for co-located web applications on virtualized servers," in *Proc. IEEE/ACM Int. Symp. Quality Service*, 2013, pp. 1–10.
[16] J. C. B. Leite, D. M. Kusic, D. Mossé, and L. Bertini, "Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2010, pp. 41–50.
[17] 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. ACM Eur. Conf. Comput. Syst.*, 2009, pp. 13–26.
[18] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang, "Probabilistic performance modeling of virtualized resource allocation," in *Proc. IEEE 7th Int. Conf. Auton. Comput.*, 2010, pp. 99–108.
[19] R. Nathuji and K. Schwan, "Virtualpower: Coordinated power management in virtualized enterprise systems," in *Proc. ACM Symp. Oper. Syst. Principles*, 2007, pp. 265–278.
[20] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE Trans. Comput.*, vol. 56, no. 4, pp. 444–458, Apr. 2007.
[21] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 2, pp. 245–259, Feb. 2011.
[22] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, and L. Zhang, "A hierarchical approach for the resource management of very large cloud platforms," *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 5, pp. 253–272, Sep./Oct. 2013.
[23] A. Gandhi, C. Yuan, D. Gmach, M. Arlitt, and M. Marwah, "Minimizing data center SLA violations and power consumption via hybrid resource provisioning," in *Proc. Int. Green Comput. Conf. Workshops*, 2011, pp. 1–8.
[24] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenhadoop: Leveraging green energy in data-processing frameworks," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2012, pp. 57–70.
[25] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 62–73.
[26] K. Le, J. Zhang, J. Meng, R. Bianchini, Y. Jaluria, and T. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2011, p. 22.
[27] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *Proc. USENIX Annu. Tech. Conf.*, 2011, p. 5.
[28] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Model. Comput. Syst.*, 2012, pp. 175–186.
[29] A. Verma, P. De, V. Mann, T. Nayak, A. Purohit, G. Dasgupta, and R. Kothari, "Brownmap: Enforcing power budget in shared data centers," in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*, 2010, pp. 42–63.
[30] Q. Zhang, F. Mohamed, S. Zhang, Q. Zhu, B. Raouf, and L. Joseph, "Dynamic energy-aware capacity provisioning for cloud computing environments," in *Proc. ACM Int. Conf. Auton. Comput.*, 2012, pp. 145–154.
[31] J. Gong and C.-Z. Xu, "vPnP: Automated coordination of power and performance in virtualized datacenters," in *Proc. IEEE Int. Workshop Quality Service*, 2010, pp. 1–9.
[32] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, p. 5.
[33] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proc. USENIX Symp. Oper. Syst. Design Implementation*, 2014, pp. 285–300.
[34] T. W. Keller, C. Lefurgy, M. Chen, and X. Wang, "Ship: A scalable hierarchical power control architecture for large-scale data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 168–176, Jan. 2012.
[35] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*, 2006, pp. 342–362.
[36] X. Zhang, S. Dwarkadas, and K. Shen, "Towards practical page Coloring-based multicore cache management," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2009, pp. 89–102.
[37] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi2: Cpu performance isolation for shared compute clusters," in *Proc. ACM Eur. Conf. Comput. Syst.*, 2013, pp. 379–391.
[38] J.-S. R. Jang, "Anfis: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man Cybern.*, vol. 23, no. 3, pp. 665–685, May/Jun. 1993.
[39] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Syst.*, vol. 22, no. 1, pp. 44–52, Feb. 2002.
[40] S. Mollov, R. Babuska, J. Abonyi, and H. B. Verbruggen, "Effective optimization for fuzzy model predictive control," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 5, pp. 661–675, Oct. 2004.
[41] Y. Ye, *Interior Point Algorithms: Theory and Analysis*. New York, NY, USA: Wiley, 1997.
[42] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," HP Labs., Palo alto, CA, USA, Tech. Rep. HPL-99-35R, 1999.

Palden Lama received the BTech degree in electronics and communication engineering from the Indian Institute of Technology, in 2003. He received the PhD degree in computer science from the University of Colorado, Colorado Springs, in 2013. He is currently an assistant professor in the Department of Computer Science, University of Texas, San Antonio. His research interests include the areas of Cloud computing, sustainable computing, autonomic resource, and power management. He is a member of the IEEE.

Yanfei Guo received the BS degree in computer science and technology from the Huazhong University of Science and Technology, China, in 2010. He is currently working toward the PhD degree in computer science at the University of Colorado, Colorado Springs. His research interests include MapReduce workloads, and autonomous resource management for Cloud computing. He won the Best Paper Award of the 10th USENIX ICAC, 2013. He is a student member of the IEEE.

Changjun Jiang received the PhD degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995. He is currently 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. 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. He is currently a professor and the chair of the Department of Computer Science, University of Colorado, Colorado Springs. His research lies broadly in computer network systems, more specifically, autonomic and sustainable computing in datacenters, cloud computing, server virtualization, and scalable Internet services and architectures. His research was supported in part by the US National Science Foundation (NSF) and Air Force Research Lab. He received the NSF CAREER AWARD in 2009, and the University Faculty Award for Excellence in Research in 2011. He is a senior member of the IEEE.