Autonomic Performance and Power Control for Co-located Web Applications on Virtualized Servers

Palden Lama*, Yanfei Guo*, Changjun Jiang† and Xiaobo Zhou*
*Department of Computer Science
University of Colorado at Colorado Springs, CO 80918, USA
Email: {plama, yguo, xzhou}@uccs.edu
†Department of Computer Science & Technology
Tongji University, Shanghai, China
Email: cjjiang@tongji.edu.cn

***Abstract*—In a data center, various components of Web applications co-located on virtualized servers exhibit complex time-varying interactions and interference. It has a 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. It features a distributed control structure that provides performance assurance 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 presence of highly dynamic and bursty workloads and inter-application performance interference. The distributed controllers perform coordinated resource allocation 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 demonstrate APPLEware's effectiveness and energy efficiency.**

Keywords: Joint Performance and Power Control, Autonomic Systems, Virtualized Servers, Distributed Fuzzy MIMO Control

I. Introduction

A modern data center utilizes virtualization technology to consolidate multiple customer applications onto high density servers for improving server utilization and reducing energy consumption costs [2], [11], [24]. It also aims to satisfy the Quality of Service (QoS) needs of hosted applications for increasing data center revenue. There are growing interests in reducing the degree of human involvement in the management of these complex computing systems through autonomic computing [8]. However, the increasing scale, heterogeneity 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.

Today, popular Internet services have multi-tier architecture in which various components invoke each other to process web requests. Due to the complex inter-tier performance dependencies and heterogeneity of application workloads, it is difficult to determine how many and what type of computing resources should be allocated to each application component to achieve the performance assurance. Furthermore, the number as well as complexity of the applications being managed, have a significant impact on the agility and scalability of a performance and power management system.

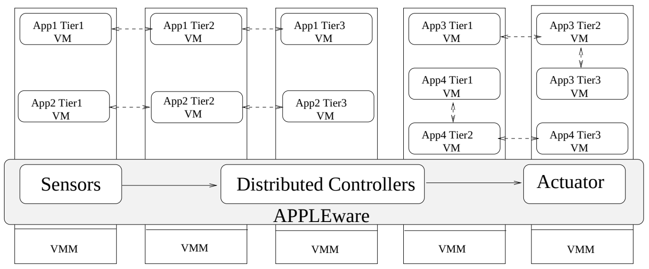


Fig. 1. APPLEware: Autonomic Performance and Power Control.

There are important and challenging issues related to performance management in virtualized computing environments. A fundamental problem is that when different applications are co-located on a shared server in the form of virtual machines (VMs), they exhibit performance interference effects [21]. Such interference arises due to the contention of resources such as the last level cache, memory bandwidth, etc, which are shared by co-located VMs [1], [25], [35].

Recent studies observed highly dynamic and bursty workloads of Internet services that fluctuate over multiple time scales [20], [26]. They have significant impact on the processing and power demands imposed on data center servers. Hence, joint performance and power management of modern virtualized computing environments needs to be autonomic: self-configuring and self-optimizing. 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 performance and energy efficiency of the virtualized server system in the presence of complex interactions between co-located Web applications.

In this paper, we propose and develop APPLEware, an autonomic middleware for joint performance and power control of co-located Web applications in virtualized computing environments. It dynamically allocates virtualized resources to various components of the hosted applications to meet their performance objectives in an agile and energy efficient manner. We use the terms power consumption and energy usage interchangeably since energy usage is measured over the same time period for all applications. Figure 1 shows APPLEware managing four multi-tier applications (App1, App2, App3, and App4) co-located on virtualized servers.

APPLEware's core is a distributed model predictive control framework that scales well in large virtualized server systems. It applies machine learning based self-adaptive modeling of application behavior based on the performance and power measurements collected by sensors. The distributed controllers exchange information and co-operate with each other to tackle the important problem of performance interference between co-located applications. Virtualized resources are allocated to an application through the actuator.

We design and implement APPLEware as a lightweight middleware solution, which is pluggable on existing virtualized computing environments. APPLEware is easily deployable as a virtual appliance on VMware infrastructure. The main contributions of APPLEware are as follows:

- It provides performance guarantee of co-located Web applications on virtualized servers.
- It is energy efficient. It reduces the energy consumption of virtualized servers while achieving the performance objectives.
- It is self-configuring. It adapts the system models to tackle time-varying system behavior in the face of highly dynamic and bursty workloads.
- It is self-optimizing. It optimizes the allocation of CPU and memory resources to improve the performance and energy efficiency of the virtualized server system.
- It is scalable to large systems. Its distributed Model Predictive Control framework decomposes the global performance and power management problem into local subproblems, which are handled through localized coordination among multiple controllers.

We evaluate APPLEware on a testbed of Dell PowerEdge servers using VMware virtual machines that host multi-tier Internet applications. As many others in [13], [15], [27], we use RUBiS as the benchmark application in conducting the experiments. RUBiS is commonly used as a multi-tier benchmark application. Experimental results demonstrate the effectiveness, and energy efficiency of APPLEware in the face of highly dynamic and bursty workloads. For performance comparison, we consider PERFUME [13], an example of the traditional performance and power management approach, which ignores the impact of performance interference between co-located applications. APPLEware achieves the improvement of 49% on average in terms of the relative deviation of application performance from its target while reducing the energy usage by 20%, compared to PERFUME [13].

Compared to a centralized control approach that considers the effects of performance interference, APPLEware improves the relative performance deviation and energy efficiency by 37% and 12% respectively. It is due to its control agility, the ability to meet application performance targets and achieve the energy-efficient system state within a short period of time.

In the following, Section II discusses related work. Sections III and IV present APPLEware architecture and design. Section V presents the testbed implementation. Section VI provides the experimental results and analysis. Section VII concludes the paper with future work remarks.

II. 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 [12], [14], [17], [23], [27], [31]. Urgaonkar et al. [27] proposed a dynamic server provisioning approach based on queueing models, which requires extensive application profiling for each workload. An approach proposed in [31] models the probability distributions of response time based on CPU allocations on virtual machines. However, the performance model is not adaptive online to dynamically changing workloads.

The multi-tier architecture forms server pipelines. Applying power management techniques such as Dynamic Voltage Scaling (DVS) independently to a particular tier will lead to inefficient usage of power for assuring an end-to-end delay guarantee due to the inter-tier dependency [7], [29]. Furthermore, traditional power management techniques are not easily applicable to virtualized environments where physical processors are shared by multiple virtual machines. For instance, changing the power state of a processor by DVS will inadvertently affect the performance of multiple virtual machines belonging to different applications [22], [30].

Recent studies have focused on jointly tacking the power and performance management of virtualized multi-tier servers for Cloud computing environments [3], [5], [10], [13], [16], [18], [19], [28], [33]. Lim et al. [18] proposed a combination of hardware-based and software-based power control techniques to coordinate the power distribution among a large number of VMs within given peak power capacity. Verma et al. [28] combined automatic VM resizing and live migration techniques to ensure that data centers can deal both with temporary outages that reduce the available power budget or with surges in workload.

The performance impact of shared resource contention in multi-core servers has been explored [6], [21], [34]. There are hardware and software resource partitioning based 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 [15]. Such a centralized technique is not scalable to large systems that host many complex multi-tier applications. It also has a dependability concern since a centralized controller acts as a single point of failure.

In this paper, we present a holistic middleware solution for all of the multi-faceted challenges discussed above.

III. APPLEware Architecture and Design

A. Design Goals and Motivations

APPLEware's key design issues are:

1) *Autonomic performance control*: A self-managing middleware for a complex virtualized server system requires an automated method to monitor its operating environment; to analyze and model the complex system behavior; to plan a sequence of actions that achieve performance goals; and to execute those actions. It is challenging to achieve performance assurance in the face of highly dynamic and bursty workloads, complex interactions between application components, and performance interference between co-located applications.
2) *Energy efficiency*: A common technique to reducing server energy consumption is to dynamically transition the hardware components from high power states to low-power states. However, changing the power state of a processor will affect the performance of multiple VMs running different applications in a virtualized computing

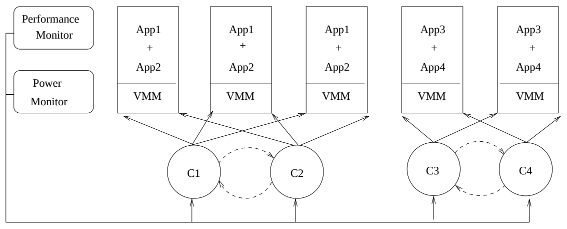


Fig. 2. APPLEware System Architecture.

environment. APPLEware achieves energy efficiency by controlling the virtualized resource usage of each VM, based on an accurate energy model.

3) *Scalability*: The computational complexity of an autonomic middleware increases significantly with the number as well as the complexity of the applications being managed. There are significant performance overheads in controlling a large system with many VMs spanning multiple server nodes. It is important but challenging to design a scalable middleware for autonomic performance and power management of large systems.

B. The Architecture

Figure 2 presents the APPLEware architecture. The computer system under control is a group of virtualized server nodes hosting multiple customer applications. We assume that each tier of a multi-tier application is deployed at a VM. Furthermore, VMs belonging to an application may span server nodes. An operator can specify the performance target and the priority of each application managed by APPLEware.

APPLEware employs a distributed control framework that decomposes the global performance and power management problem of the entire system into local sub-problems for scalability. Each controller executes a control loop on a sub-system, which comprises of the VMs belonging to one application. The control actions are the required adjustments in the allocation of virtualized resources to meet the performance target of an application. We observe that some sub-systems are inherently coupled with each other mainly due to shared resource contentions and performance interference in a virtualized computing environment. APPLEware addresses this challenge through co-ordination among neighboring controllers for effective performance assurance and energy efficiency.

An important feature of APPLEware is that increasing the number of applications hosted in the system has little impact on its scalability, because each local controller in its distributed control framework is responsible for controlling one application and coordinating with its neighbor controllers only.

APPLEware controls the power consumption by applying CPU usage limits on VMs hosted on a cluster of blade servers. It 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.

As an example, Figure 2 shows four distributed controllers. Each controller is responsible for the performance and energy efficiency of one application. Note that 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. At the same time, they co-ordinate with each other by exchanging information about their control decisions. Such co-ordination is important for the application performance assurance as well as system stability. This is due to the fact that a control action taken by C1 or C2 affects the performance of both app1 and app2. Similarly the controllers, C2 and C4, co-ordinate with each other to control app3 and app4.

Each autonomic controller performs the MAPE-K [8] control loop as follows:

1) *Monitor(M)*: The performance and power monitors periodically measure the average end-to-end response time of the managed application and the average energy usage of the underlying multi-core server respectively. Our design does not use any semantic information regarding the performance metric. It treats the performance values as raw data for modeling and control. Hence, APPLEware is applicable to any performance metric.
2) *Analyze(A)*: It constructs fuzzy models to analyze the complex system behavior in terms of the non-linear relationship between resource allocation and application's end-to-end response time as well as energy usage. It also captures the coupling effects between neighboring applications that share the underlying physical resources.
3) *Plan(P)*: It plans a sequence of control actions that regulate the allocation of virtualized resources for achieving the performance target and energy efficiency of the managed application. The control decisions are guided by Distributed Model Predictive Control theory. It involves optimizing a cost function that expresses the local control objectives and resource constraints over a time interval. The current state of the local sub-system and the control decisions made by neighboring controllers are taken into account to perform the optimization. This iterative process of optimization and communication with neighboring controllers is designed to converge to local control actions that lead to overall optimal performance.
4) *Execute(E)*: The optimization performed in the planning phase leads to a control action that brings the system closer to its performance target. The actuator executes the control actions in the form of adjustment in CPU and memory resources assigned to the application VMs.

IV. APPLEware Design

A. Global System Model

First, we consider a global system model that represents the performance and power consumption behavior of multi-tier applications spanning across a group of virtualized server nodes. The inputs to the system are the resource allocation in terms of CPU and memory usage limits at various tiers of the hosted applications. The outputs of the system are the measured performance and average energy usage of each application. We obtain two separate models for power and performance of the system, respectively. 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 U(k) is a vector of current 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. It also captures the coupling among applications due to performance interference in a shared virtualized environment. G(k) is a regression vector that contains the performance and power consumption values of each application in the current and previous sampling intervals. F is a regression matrix that represents the impact of regression vector on the system outputs.

Consider n applications in the system. The components of the global system model are described by the following equations:

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

$$\begin{aligned}\mathrm{U(k)} = [&u_1(k), .., u_{c_1}(k), u_{c_1+1}(k), .., \\ &u_{c_1+c_2}(k), u_{c_1+c_2+1}(k), .., u_{c_1+c_2+..+c_n}(k)]^T\end{aligned} \tag{3}$$

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

$$\begin{aligned}\mathrm{G(k)} = [&y_1(k), .., y_1(k-m_y), y_2(k), .., \\ &y_2(k-m_y), y_n(k).., y_n(k-m_y)]^T\end{aligned} \tag{4}$$

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

In Eq. (2), each 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), each input term $u_j(k)$ represents the allocation of CPU and memory usage limits on a particular VM component j in the virtualized server system. A VM component provides the functionality of a particular tier of a multi-tier application. The total number of components in an application i is denoted by c_i. The total number of VM components in the entire system is denoted by $C = \sum_{i=1}^{n} c_i$.

In the matrix H, the term $\eta_{i,j}$ represents the impact of resource allocation on the application performance or power consumption. $\eta_{i,j}$ has a non-zero value if the resource allocation input $u_j(k)$ has an impact on application i. In Eq. (4), m_y specifies the order of regression, which is the number of previous samples of output variable y_i that will be used in the system model. In the regression matrix F, the term $\zeta_{i,j}$ represents the impact of the system outputs measured at the j_{th} previous sampling interval on the performance and power consumption of application i.

B. Problem Decomposition

Autonomic performance and power management of an entire virtualized server system containing many complex applications involves a large scale optimization and control process. The computational complexity of the problem grows significantly with the number of applications being managed. To address this scalability issue, APPLEware decomposes the global performance and power control problem into a set of localized subproblems.

From a local controller's perspective, the goal of decomposition is to partition the set of system variables into three subsets, including local variables associated with the managed application, neighbor variables associated with other applications that have an impact on the performance and power consumption of the managed application, and all other variables in the system. The subproblem only includes its local and neighbor variables. The decomposition scheme reduces the number of variables involved in the control problem to improve system scalability. At the same time, it also captures the coupling among applications so that local controllers can achieve global system stability through coordination in their neighborhood.

The local variables in a control subproblem include a managed application's performance, power consumption and the amount of virtualized resources allocated. The neighbor variables include the amount of resources allocated to the VM components of other applications that have a coupling effect on the local application due to shared resource contentions and performance interference. Note that a local controller does not control the neighbor variables. Instead, these variables influence the control decisions on the local subsystem.

The local model obtained by decomposing the global system 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 power consumption 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_1 in Figure 2 uses a local system model with the following parameters.

$$\mathrm{u}_1(k) = [u_1(k), u_2(k), u_3(k), u_4(k), u_5(k), u_6(k), u_7(k)]^T \tag{6}$$

$$\eta(k) = [\eta_{1,1}, \eta_{1,2}, \eta_{1,3}, \eta_{1,4}, \eta_{1,5}, \eta_{1,6}, \eta_{1,7}] \tag{7}$$

From the controller C1's perspective, $[u_1(k), u_2(k), u_3(k)]$ are a set of local variables, which represent three VMs of App1. $[u_4(k), u_5(k), u_6(k), u_7(k)]$ are the neighbor variables, which represent four VMs of App2 as shown in Figure 1. The local system model predicts the performance of App1 in the presence of performance interference from App2.

C. Fuzzy Modeling To Capture System Non-linearity

The prediction accuracy of the system model has a significant impact on the control performance. A linear model is often inadequate to accurately represent the complex behavior of inherently non-linear systems such as a multi-tier application hosted in a virtualized computing environment. APPLEware addresses this issue by constructing Fuzzy models that capture the performance and power consumption behavior of a local subsystem. 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 subsystems with simple functional dependencies.

1) Model Formulation: A local sub-system is represented by a fuzzy model as follows:

$$y_i(k+1) = \mathrm{R}(\xi_i(k), \mathrm{u}_i(k)). \tag{8}$$

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.

$$\xi_i(k) = [y_i(k), .., y_i(k - m_y)]^T \tag{9}$$

where m_y specifies the order of regression.

R is a rule based fuzzy model consisting of K fuzzy rules. Each fuzzy rule is described as follows:

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

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

Here, Ω_r is the antecedent fuzzy set of the r_{th} rule which describes elements of regression vector $\xi_i(k)$ and the current input vector $\mathrm{u}_i(k)$ using fuzzy values such as 'large', 'small', etc. ζ_r and η_r are vectors containing the consequent parameters and ϕ_r is the offset vector. ϱ denotes the number of elements in the regression vector $\xi_i(k)$. The model output is calculated as the weighted average of the linear consequents in the individual rules. That is,

$$y_i(k+1) = \frac{\sum_{r=1}^{K} \beta_r(\zeta_r \xi_i(k) + \eta_r \mathrm{u}_i(k) + \phi_r)}{\sum_{r=1}^{K} \beta_r} \tag{11}$$

where the degree of fulfillment for the r_{th} rule β_r is the product of the membership degrees of the antecedent variables in that rule. Membership degrees are determined by fuzzy membership functions associated with the antecedent variables. The model output is expressed in the form of

$$y_i(k+1) = \zeta_r^* \xi_i(k) + \eta_r^* \mathrm{u}_i(k) + \phi_r^*. \tag{12}$$

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

$$\begin{aligned} \zeta_i^* &= \frac{\sum_{r=1}^{K} \beta_r \cdot \zeta_r}{\sum_{r=1}^{K} \beta_r}. \\ \eta_i^* &= \frac{\sum_{r=1}^{K} \beta_r \cdot \eta_r}{\sum_{r=1}^{K} \beta_r}. \\ \phi_i^* &= \frac{\sum_{r=1}^{K} \beta_r \cdot \phi_r}{\sum_{r=1}^{K} \beta_r}. \end{aligned}$$

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. Each obtained cluster represents a certain operating region of the system, where input-output data values are highly concentrated. The clustering process partitions the input-output space and determines the number of fuzzy rules and the shape of membership functions. APPLEware applies an adaptive network based fuzzy inference system (ANFIS) [9] 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.

Dynamic and bursty Internet workloads have significantly varying resource demands at multiple tiers of hosted applications. A static system model can not provide sufficient prediction accuracy of power and performance for all possible variations in the workload. APPLEware applies a wRLS (weighted Recursive Least Squares) method [13] to adapt the consequent parameters of its fuzzy models as new measurements are sampled from the system at runtime. It applies exponentially decaying weights on the sampled data so that higher weights are assigned to more recent observations.

D. Controller Design

Each local controller applies the distributed model predictive control principle to regulate a sub-system's dynamic behavior towards the performance targets while minimizing the energy usage.

1) Control Formulation: A local control objective of controller C_i is given by the following cost function:

$$\begin{aligned} V_i(k) = \sum_{p=1}^{H_p} ||r_i - y_{i1}(k+p)||_P^2 + \sum_{p=1}^{H_p} ||y_{i2}(k+p)||_Q^2 \\ + \sum_{c=0}^{H_c-1} ||\Delta \mathrm{u}(k+c)||_R^2. \end{aligned} \tag{13}$$

Here, $y_{i1}(k)$ is the average end-to-end response time and $y_{i2}(k)$ is the average power consumption of application i at control interval k. The controller predicts both power 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 targets r_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 of the multi-tier applications. P and Q are the tracking error weights that determine the trade-off between power and performance. The third term in Eq. (13) represents the control penalty and is weighted by R. This term penalizes big changes in control action and contributes towards high 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 the total CPU and memory capacity of the server.

For our experiments presented in Section VI, the value of H_p was tuned to 20, which was sufficiently large for stable control. The value of H_c was tuned to 5, which was able to provide good control performance. Due to space limitation, we did not include the sensitivity analysis.

2) Distributed Control Algorithm: APPLEware's distributed control algorithm is shown in Algorithm 1.

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 and time-consuming optimization as formulated in Eq. (13). 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

Algorithm 1 Distributed Control Algorithm.

```
1: loop
2:   A local controller $C_i$ measures the current state of the sub-system in terms of local and neighbor variables.
3:   It compares the measured values of performance and power consumption of application $i$ with the predictions made by its Fuzzy models.
4:   if A significant error in the prediction of performance and power consumption is detected then
5:     It updates the Fuzzy models using its online learning algorithm.
6:   end if
7:   repeat
8:     $C_i$ executes Model Predictive Control algorithm to solve local subproblem.
9:     $C_i$ sends its control solutions to neighboring controllers. In addition, it receives the control solutions computed by its neighbors.
10:  until The control solutions converge to a steady value.
11:  It executes the control actions in the form of adjustment in CPU and memory resources assigned to the application $i$.
12: end loop
```

variant model in the following form:

$$\begin{aligned} x_i(k+1) &= A(k)x_i(k) + B(k)\mathrm{u}_i(k). \\ \mathrm{y}_i(k) &= C(k)x_i(k). \end{aligned} \quad (14)$$

The state vector for the state-space description is defined as

$$x_i(k+1) = [\xi_i^T(k), 1]^T. \quad (15)$$

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. (12) and Eq. (14), 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 j^{th} element of aggregate parameter vectors ζ^* and η^* respectively for application i.

The MIMO control problem defined by Eq. (13) is transformed to a quadratic program:

$$\text{Minimize } \frac{1}{2}\Delta\mathrm{u}(k)^T H \Delta\mathrm{u}(k) + c^T \Delta\mathrm{u}(k) \quad (16)$$

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

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 virtual machine over the entire control horizon H_c. In the minimization formulation,

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

$$c = 2[R_{1u}^T P^T (R_{1x} A x(k) - r) + R_{2u}^T Q^T R_{2x} A x(k)]^T. \quad (17)$$

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 power model of the resource pool.

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

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

4) 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 [32]. 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 sub-problems. For APPLEware, the value of N depends on the number of local and neighbor input variables only.

V. System Implementation

A. Testbed

We built a testbed in a university prototype data center, which consists of Dell PowerEdge R610 servers. Each server has 2 Intel hexa-core Xeon X5650 CPUs and 32 GB memory. The servers are connected with 10 Gbps Ethernet. The testbed hosts multi-tier Web applications as shown in Figure 1. Each tier of an application is implemented on a VMware virtual machine with 1 VCPU, 1 GB RAM and 15 GB hard disk space. All VMs use Ubuntu server 10.04 with Linux kernel 2.6.35. Each controller runs on a VM having 300 Mhz CPU usage limit and 128 Mb memory usage limit. We focus on using lightweight controllers that do not interfere with the performance of the hosted applications and that leave a small footprint on the energy usage of system.

For performance evaluation, we deploy the RUBiS benchmark as shown in Figure 1. RUBiS is an open source multi-tier Internet benchmark application. It provides a web auction application that is 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. A multi-tier deployment of RUBiS has a simple pipelined architecture consisting of web, application and database servers. In our implementation, a tier at the front end of the hosted application runs the Apache web server. The application tier runs PHP servers, and the database tier runs a MySQL server.

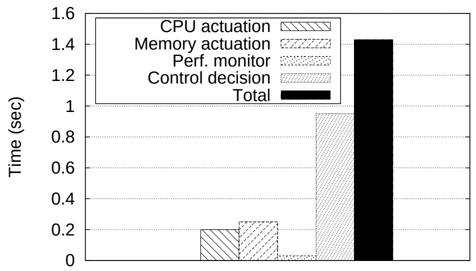


Fig. 3. APPLEware's Average performance overhead.

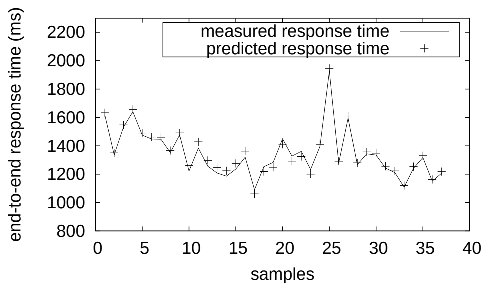


Fig. 4. APPLEware's performance prediction (APP1) in the presence of interference effects.

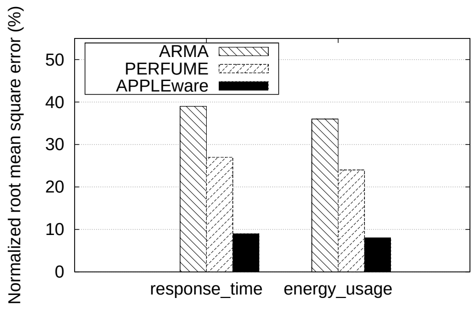


Fig. 5. Prediction accuracy comparison.

B. APPLEware Components

We implemented the components of APPLEware as software modules that interact with each other as shown in Figure 2. In this paper, APPLEware is deployed as a lightweight virtual appliance, which is distributed over separate machines on VMware infrastructure.

1) Power Monitor: The average energy usage of the virtualized server is measured at the VM level by using VMware ESX 4.1. VMware gathers such data through its Intelligent Power Management Interface sensors. The power monitor module uses vSphere API to collect the energy usage data of a multi-tier application at each control interval. Energy usage is measured in terms of Kilo Joules (KJ).
2) Performance Monitor: APPLEware collects the application response time values from the Web-tier access logs, which are commonly 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. APPLEware's performance monitor module consists of an XML-RPC client that communicates with RUBiS application to periodically collect the performance statistics at each control interval.
3) Performance and Power modeling: We use MATLAB's Fuzzy Logic Toolbox to apply subtractive clustering and ANFIS modeling technique on the data collected from the server system. At runtime, the performance and power models are updated according to new measurements collected from the system using the wRLS algorithm.
4) Distributed Controller: Each controller module 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 local controller node. The distributed controllers communicate with each other in a peer-to-peer manner using XML-RPC protocol.
5) Actuator: It uses vSphere API to impose CPU and memory usage limits on the VMs. The vSphere module provides an interface to execute a method *ReconfigVM_Task* to modify a VM's resource usage limit.

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. Figure 3 shows the average time taken for each of these factors on our testbed hosting four applications. Note that control overheads play an important role in determining the control interval. 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.

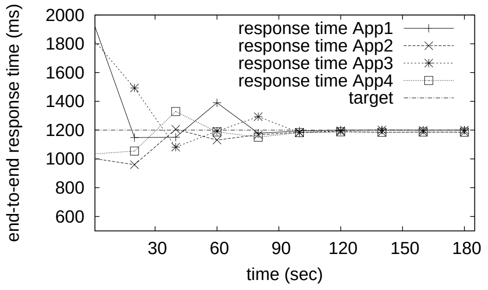
VI. Performance Evaluation

A. Model validation

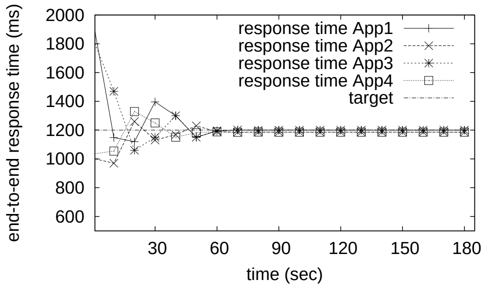
The accuracy of the system model has a significant impact on effective control of power and performance. We first validate APPLEware's system models using multi-tier applications App1 and App2. Note that various tiers of App2 are co-located with VMs belonging to App1. The initial models are obtained by using a training data set that consists of the 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 workload of a browsing mix of 600 concurrent users. For model validation, we use a different set of resource allocations that is not used for training the system models.

In this experiment, we demonstrate APPLEware's performance prediction accuracy in the presence of interference between co-located applications. Figure 4 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%. NRMSE is a standard metric for deviation.

Figure 5 compares APPLEware's performance and energy usage prediction accuracy with the PERFUME [13] and a representative modeling approach ARMA [4]. In contrast to APPLEware, the modeling approach used in PERFUME does not capture the performance interference effects between applications co-located on virtualized servers. On the other hand,

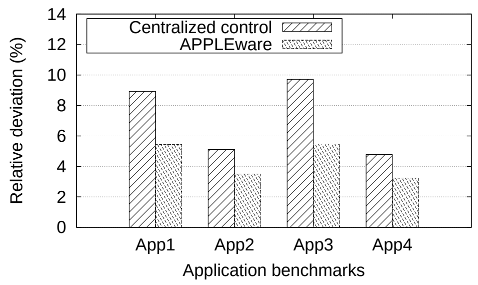


(a) Centralized controller.

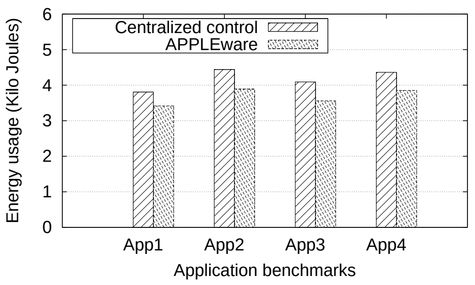


(b) APPLEware.

Fig. 6. Performance assurance under a stationary workload.



(a) Performance.



(b) Energy usage.

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

the ARMA modeling approach performs a linear approximation of the inherently non-linear system.

APPLEware's superior prediction accuracy is due to its fuzzy modeling that captures the non-linear relationship of performance and energy with multiple virtualized resources. Furthermore, it considers the impact of performance interference among co-located VMs. For instance, the prediction model for App1 includes the resources allocated to co-located VMs as predictor variables, in addition to the resources allocated to its own VMs. We observe that our fuzzy models use 9 fuzzy rules to represent the performance of a multi-tier application with sufficient prediction accuracy.

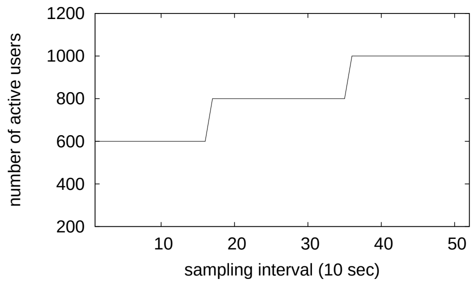
B. Autonomic Performance Control and Energy Efficiency

1) Control Agility: We evaluate the effectiveness and agility of APPLEware in assuring the performance and reducing the energy usage of co-located multi-tier applications. As a performance metric, we use the average end-to-end response time, which represents the user perceived performance of interactive Internet applications. The SLA target for all the applications is set to 1200 ms. We apply a stationary workload of 600 concurrent users to each application. Figure 6(b) shows that APPLEware is able to bring the average end-to-end response time of each application close to their respective SLA targets within 60 seconds. It is due to the agile and effective adjustments in the CPU and memory resources of the multi-tier applications by APPLEware's distributed controllers. On the other hand, Figure 6(a) shows that a centralized controller takes around 100 seconds to meet the performance target.

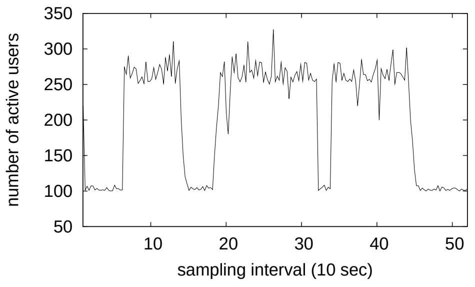
For any system, a centralized controller at the fastest sampling rate gives the best achievable performance. However, implementing centralized controller at the fastest sampling rate may not be feasible, due to operational constraints such as the overheads involved in measuring the current system states and computing the control decisions. In such cases, a distributed control approach presents an opportunity to obtain superior control performance. 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's distributed controllers are able to converge close to the optimal solutions obtained by the centralized controller. Note that the convergence takes place within each control interval.

To quantify the performance of APPLEware, we use the relative deviation from a target as the metric. The relative deviation 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 SLA target for that application. Figures 7(a) and 7(b) show that APPLEware is able to improve the relative performance deviation as well as the energy efficiency of each application, compared to the centralized controller. On average, the improvement in the relative deviation by APPLEware is 37%. It is also 12% 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.

2) Robustness under dynamic and bursty workloads: Next, we evaluate the robustness of APPLEware under dynamic and bursty workloads. As a case study, we apply a step-change workload as shown in Figure 8(a) to App1 and App3. On the other hand, we apply a bursty workload to App2 and App4. We inject burstiness into the arrival process of RUBiS clients according to the index of dispersion. The dispersion index modulates the think times of users between submission

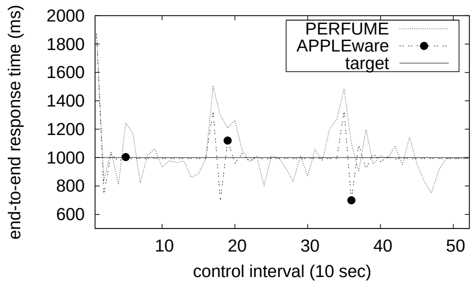


(a) a dynamic workload.

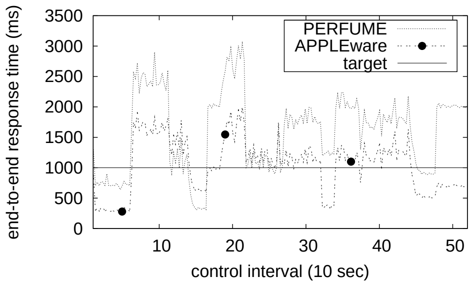


(b) a bursty workload.

Fig. 8. Robustness of APPLEware under dynamic and bursty workloads.



(a) under dynamic workload (App1).



(b) under a bursty workload (App2).

Fig. 9. APPLEware performance assurance under dynamic and bursty workloads.

of consecutive requests. We set the index of dispersion to 4000 and the maximum number of concurrent users to 1000. Figure 8(b) shows a bursty workload.

Figures 9(a) and 9(b) compare the performance assurance capability of APPLEware with that of PERFUME in the face of the dynamic and bursty workloads. In Figure 9(a), both PERFUME and APPLEware show some fluctuations in the average response times at the control intervals 17 to 21 and 34 to 38. It is due to the abrupt changes in the workload starting at control interval 17 and 34. However, APPLEware is able to able to adapt itself more effectively so that the average end-to-end response time of App 1 converges to the SLA target of 1000 ms within few control intervals. The fluctuations in the average response time of App2 is more significant mainly due to burstiness in the workload, which is more difficult to handle. However, compared to PERFUME, APPLEware is able to keep the response time closer to the SLA target. The robustness of APPLEware under dynamic and bursty workloads is attributed to its fast online learning algorithm, which updates the performance model by observing dynamic system behavior. Importantly, self-configuration of APPLEware is effective due to its awareness of the performance interference effects.

Figures 10(a) and 10(b) show the average relative deviations and the energy usage of various multi-tier applications under the dynamic and bursty workloads. Compared with PERFUME, there is the improvement of 49% on average in terms of relative deviation by APPLEware. At the same time, APPLEware improves the energy efficiency by 20%. We observe that both relative deviation and energy usage of App2 and App4 are larger than that of App1 and App3. It is due to the fact that these applications face a bursty workload, which demonstrate abrupt workload variations. However, APPLEware still shows significant improvement in performance and energy efficiency compared with PERFUME.

VII. Conclusion

The user perceived performance of Internet applications and the power consumption 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 co-located multi-tier applications pose significant challenges to autonomic performance and power management. APPLEware is an autonomic middleware for joint performance and power control in virtualized computing environments. It is easily deployable as a lightweight virtual appliance on VMware infrastructure. As demonstrated by modeling, analysis and experimental results based on testbed implementation, its main contributions are robust performance assurance and energy efficiency in the presence of VM performance interference as well as highly dynamic and bursty workloads.

The proposed and developed middleware solution is based on a distributed MAPE-k control framework. It integrates the strengths of machine learning based adaptive system modeling and a distributed control algorithm to achieve self-configuring, self-optimizing and self-scaling capabilities. Our future work will extend APPLEware's compatibility to XenServer.

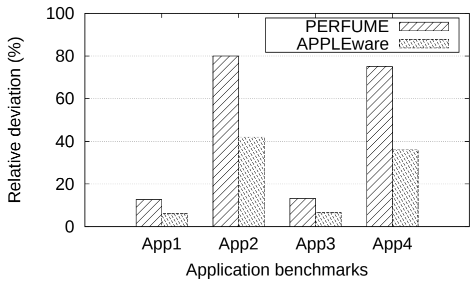
VIII. Acknowledgement

This research was supported in part by NSF CAREER award CNS-0844983 and research grant CNS-1217979.

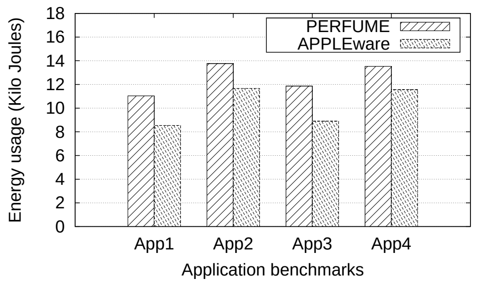
References

[1] A. Fedorova, M. Seltzer, and M. D. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *Proc. Int'l Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2007.

[2] D. Gmach, J. Rolia, and L. Cherkasova. Resource and virtualization costs up in the cloud: Models and design choices. In *Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*, 2011.



(a) Performance.



(b) Energy usage.

Fig. 10. Performance and energy efficiency improvement due to APPLEware.

[3] 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 EuroSys Conference (EuroSys)*, 2012.

[4] J. Gong and C.-Z. Xu. vpnp: Automated coordination of power and performance in virtualized datacenters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2010.

[5] Y. Guo and X. Zhou. Coordinated vm resizing and server tuning: Throughput, power efficiency and scalability. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.

[6] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proc. ACM/IFIP/USENIX Int'l Conference on Middleware*, 2006.

[7] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier Web servers with end-to-end delay control. *IEEE Trans. on Computers*, 56(4):444–458, 2007.

[8] M. C. Huebscher and J. A. McCann. A survey of autonomic computing: Degrees, models, and applications. *ACM Computing Surveys*, 40(3), 2008.

[9] J.-S. Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):665 –685, 1993.

[10] 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'l Conference on Distributed Computing Systems (ICDCS)*, 2010.

[11] G. Jung, K. 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'l Conference on Dependable Systems and Networks (DSN)*, 2010.

[12] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 151–160, 2010.

[13] P. Lama and X. Zhou. PERFUME: Power and performance guarantee with fuzzy mimo control in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, pages 345–354, 2011.

[14] 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, 19 pages*, 23(1), 2012.

[15] P. Lama and X. Zhou. Ninepin: Non-invasive and energy efficient performance isolation in virtualized servers. In *Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*, 2012.

[16] 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'l Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[17] 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'l Conference on Autonomic computing (ICAC)*, 2010.

[18] H. Lim, A. Kansal, and J. Liu. Power budgeting for virtualized data centers. In *Proc. USENIX Annual Technical Conference*, 2011.

[19] 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. *ACM SIGMETRICS*, 2012.

[20] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proc. ACM/IFIP/USENIX Int'l Middleware Conference (Middleware)*, 2008.

[21] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proc. 5th ACM European conference on Computer systems (EuroSys)*, 2010.

[22] R. Nathuji and K. Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, pages 265–278, 2007.

[23] 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 EuroSys Conference (EuroSys)*, pages 13–26, 2009.

[24] 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 Int'l Conference on Dependable Systems and Networks (DSN)*, 2011.

[25] J. Rao, K. Wang, X. Zhou, and C.-Z. Xu. Optimizing virtual machine scheduling in numa multicore systems. In *Proc. IEEE Int'l Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[26] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, pages 21–30, 2010.

[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] 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 1Int'l Conference on Middleware*, 2010.

[29] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. on Parallel and Distributed Systems*, 22(2), 2011.

[30] Y. Wang, X. Wang, M. Chen, and X. Zhu. Partic: Power-aware response time control for virtualized Web servers. *IEEE Trans. on Parallel and Distributed Systems*, 21(4), 2010.

[31] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. Probabilistic performance modeling of virtualized resource allocation. In *Proc. IEEE Int'l Conference on Autonomic computing (ICAC)*, 2010.

[32] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley and Sons, 1997.

[33] 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'l Conference on Autonomic Computing (ICAC)*, 2012.

[34] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *Proc. 4th ACM European conference on Computer systems (EuroSys)*, 2009.

[35] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proc. Int'l Conference on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2010.