Noname manuscript No.
(will be inserted by the editor)

PAPMSC: Power-Aware Performance Management Approach for Virtualized Web Servers via Stochastic Control

Xiaoyu Shi · Jin Dong · Seddik M. Djouadi · Yong Feng · Xiao Ma · Yefu Wang

Received: date / Accepted: date

Abstract As green computing is becoming a popular computing paradigm, the performance of energy-efficient data center becomes increasingly important. This paper proposes power-aware performance management via stochastic control method (PAPMSC), a novel stochastic control approach for virtualized web servers. It addresses the instability and inefficiency issues due to dynamic web workloads. It features a coordinated control architecture that optimizes the resource allocation and minimizes the overall power consumption while guaranteeing the service level agreements (SLAs). More specifically, due to the interference effect among the co-located virtualized web servers and time-varying workloads, the relationship between the hardware resource assignment to different virtual servers and the web applications' performance is considered as a coupled Multi-Input-Multi-Output (MIMO) system and formulated as an robust optimization problem. Then we propose a constrained stochastic linear-quadratic controller (cSLQC) in this paper to solve this problem by minimizing the quadratic cost function subject to constraints on resource allocation and applications' performance. Furthermore, a proportional controller is also integrated to enhance system stability. In the second layer, we dynamically manipulate the physical frequency for power efficiency using an adaptive linear quadratic regulator (ALQR). Experiments on our testbed server with a variety of workload patterns demonstrate that proposed control solution significantly outperforms existing solutions in terms of effectiveness and robustness.

Keywords Virtualization · Power efficiency · Resource managemen · Sochastic control · Web application

Xiaoyu Shi
School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, 611731, China.
E-mail: shixiaoyu0216@gmail.com

Jin Dong, Seddik M. Djouadi, Yefu Wang, Xiao Ma
Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN, 37996, USA.

Yong Feng
Chongqing Institude of Green and Intelligent Technology, Chinese Academy of Science, Chongqing, 400714, China.
E-mail: yongfeng@cigit.ac.cn

1 Introduction

Recently, energy-efficient computing is becoming a hot topic in the design and operation of enterprise servers and modern data centers. It is well known that the Internet services (e.g., online banking and online business, etc.), data storage and telecommunications industries are crucial parts of the American information economy, the repaid growth of these industries have led to an huge increase in electricity use [1]. As reported by the U.S. Department of Energy, the energy consumption for a data center can be 100 times higher than that of a typical commercial building. Furthermore, Gartner Inc. estimates that energy consumption will contribute more than 50% of the IT budget in the next few years [2]. Meanwhile, a single high performance sever can emit as much as 1,300kg CO_2 a year. Thus, the ultimate goal of this work is to improve those sectors energy efficiency, which will provide significant energy and cost saving, as well as reduce the carbon pollution.

Virtualization machine (VM) technology, as an important enabler of cloud computing [3], has been widely employed in modern data centers. The last decade has seen an explosion of the development in the area of virtualization (e.g., Amazon EC2 [4], Google AppEngine [5] and etc.), as it offers a promising approach to improve servers utilization and reduce power consumption via consolidating multiple applications to several powerful servers. However, the server farm is in a state of continuous expansion in recent

years, both in scale and complexity, as a result it is increasingly difficult for system administrators to maintain servers manually. Hence there is a growing interest on how to decrease human involvement in management of modern data centers through automatic computing [6].

Meanwhile, there are multi-faceted challenges for virtualized servers hosting multiple web applications. Firstly, the virtualization technology can support environment isolation, fault isolation for multiple virtual machines (VMs). These VMs can run different operating systems and applications as if they were running on the physical separate machines, but the contention of shared limited resources (i.e., processor, last level cache and bandwidth) among multiple customer applications lead to a performance interference effect [7]. Any change of one VM can adversely affect the performance of other VMs. Hence guaranteeing the performance isolation of each application is important for meeting the SLAs in virtualized servers. Secondly, the workloads of web applications show highly volatile variation and unexpected burst in terms of the request arrival rate [8]. It is infeasible to decide the resource allocation solution (e.g., how many resources should be allocated and how to assign them) in a fixed way. In such environments, the control approach needs to be robust and adaptive to dynamic variations in workloads. Lastly, the common approach to manage the power consumption in no-virtualized environment just simply adjusts the power state between different levels according to the record of power consumption. However, this approach cannot be directly applied to virtualized server environments [12]. Clearly, an energy-efficient virtualized server should minimize the power consumption without any performance loss.

Several solutions have been applied for the virtualized servers. Many of these solutions rely on feedback control theory [12,25,26,31]. They use the system identification theory to build their models for specific workloads in an offline way. Although these approaches can theoretically guarantee system stability and control accuracy within a range, they cannot adapt to a time-varying workload case such as Internet services. Recently, several methods have been proposed to address the limitations of using fixed offline models. They use various control approaches with online estimators/predictors to optimize the power consumption of virtualized servers with performance assurance [20,24,28]. The time-varying models have been adopted in their control architecture, but they consider control system as a deterministic system and ignore the error introduced by the model. Furthermore, the accuracy of the controller heavily depends on the assumption that the online estimator always provides correct model estimations for the controller. However, this assumption is not always valid, because the relationship between the workloads and costs is actually quite complex [9] and some *uncertainties* of real-world web servers are ignored in their models [10,11] (e.g., the bursty nature of web workloads can cause a significant noise in the system output, noise within instantaneous power measurements and the variability within processing times [45]). Compared with the deterministic control approaches, a key advantage of stochastic control approaches is that a noise term is considered in the model, which represents the unknown and uncertain elements in the system.

In this paper, we present a self-manage and power-efficient architecture that guarantees the performance of co-located web application in virtualized server. We design a stochastic control method with a robustness feature. Then, we implement it into a two-layer framework. The primary control layer adopts the stochastic control strategy to maintain the workload balancing among all of the virtual machines, therefore, the desired response time is guaranteed for different web applications. In order to enhance the system robustness, we also integrate a proportional (P) controller in this layer for dealing with the inaccuracy of estimated model. The secondary control layer then dynamically adjusts the CPU frequency to maintain the response time at a desired level for power efficiency by using ALQR approach. An online estimator is employed to identify system parameters in both of the two control layers using the computationally efficient Sliding Window Recursive Least Squares algorithm (SWRLS).

Specifically, the **contributions** of this paper are fourfold:

1. Due to the resource contention between co-located VMs and some uncertainties of real-world web servers, the web server system is considered as the stochastic time-varying system and formulate as a coupled MIMO control problem. Meanwhile, a time-varying Autoregressive-moving-average model with exogenous inputs (ARMAX) model is used for system modeling.
2. The load-balancing control problem is formulated as a robust dynamical optimization problem and the constrained stochastic linear quadratic control approach (cSLQC) is designed to solve this problem [18]. The quadratic cost function is subject to linear constraints on resource allocation and probabilistic constraints on the application performance.
3. A model-independent proportional controller is integrated to compensate for the degraded performance of online estimator, which enhances the system stability in the face of bursty workload case.
4. Empirical results are presented to demonstrate that the stochastic control approach effectively reduces the server power consumption by controlling the CPU frequency, while meeting a desired response time for each VM by adaptively adjusting resource allocation in face of dynamical and bursty workloads.

The rest of this paper is organized as follows: The overall architecture of stochastic control solution is described in Section 3. Section 4 and Section 5 present the design details of load balancing control layer and power saving control layer. Section 6 describes the implementation details of our testbed and each component of our control approach. The experimental results are analyzed and the control performance is evaluated in Section 7. Section 2 summarizes related work on power and performance control for virtualized servers. Finally, section 8 concludes the paper.

2 Related Work

Dynamic resource provisioning of Web applications in virtualized environment is an important and active research topic. Recently, a number of control approaches for the resource management in virtualized servers have been proposed [32–34]. In [35], the authors presented a queuing model for multi-tier web applications and a dynamic provisioning technique approach to handle dynamic workloads. Gandhi et al. [37] aimed to optimize the system utilization by adjusting the Apache server configuration, which built a MIMO system model using the system identification technique and used to design control-theory based controller. However, all the above solutions are not designed to reduce power consumption for virtualized servers.

Several groups [21,22,39] have successfully designed the adaptive control approach to satisfy the power saving target while meeting the SLAs of application in the face of high dynamical workload. The authors proposed an adaptive MIMO control algorithm based on fuzzy modeling technology to guarantee the performance and power targets with feasible trade off in the face of highly dynamic and bursty workload, showing the robustness of their Fuzzy control solution [24]. VCONF was a reinforcement learning based approach to automate the VM configuration (i.e., the CPU time, the memory of each VM and the number of virtual CPUs) [36]. In contrast to these adaptive control solutions that rely on heuristics, the key advantage of using *control theoretic* approach is that it can guarantee control accuracy and system stability from theoretical analysis and avoid the exhaustive manual tuning and testing.

Feedback control approaches have been widely used to guarantee the performance and power request in virtualized environment. For example, Wang et al. [12] introduced a two-layer feedback control architecture to achieve the goal of response time optimization and power saving by controlling CPU resource allocation and CPU frequency in separate loops. Kusic et al. [26] proposed dynamically tuning control parameters such as the CPU allocation for each VM workload fractions, number of VM and etc. to maintain the desired quality-of-service (QoS) and energy efficiency. In [25], the authors presented a coordinated controller for power and performance control using model predictive control theory for the cluster level virtualized servers. However, the models of their systems were designed in a way that heavily relied on offline system identification for specific workloads. Although these approaches theoretically guaranteed the system stability and control accuracy within a range, they could not adapt to a time varying workload.

In [28], the authors proposed vPnP, a feedback control-based system to coordinately manage performance and power using a novel utilization function optimizer, and design a self-turning predictor to deal with high dynamic workloads. Compared to vPnP without constraints on SLA metrics, our work set the constraint on the SLA matrices (e.g., response time) in a probability way, which made our control action more robustness than vPnP approach. AutoControl was defined as an automatic control system for the multiple virtualized resources. It used an online estimator to adaptively model the relationship between application-level performance metrics and resource allocation for each VM. This estimator was then updated dynamically by the recursive least-square method [20]. Different with our control approach, AutoControl showed good control performance without consideration of the power efficient. Additionally, although all of the above deterministic control approaches showed good performance in coping with time varying workloads, the performance of control systems heavily relied on the online estimator, the control accuracy and system stability would degraded when the performance of the estimator decreased. In contrast to these approaches, which only focus on the chosen system identification model and ignored the error introduced by the chosen model, we integrated a model-independent Proportional controller to compensate for the degraded performance of online estimator.

3 Control Architecture and Design

In this section, we present a high-level description of the system architecture of PAPMSC. We attempt to achieve the trade-off objectives of high performance and low power consumption. The goal of PAPMSC is to dynamically adjust the resource allocation and scale the CPU frequency to reduce power consumption without performance degradation. As shown in Fig.1, to deal with the time-varying incoming workload, the primary control loop, i.e., the load balancing control loop, manages the performance of each web application to the desired level by adjusting the resources (i.e., fraction of CPU time) allocation to each VM. As a result, each VM can have roughly the same application-level performance (i.e., response time). To achieve the power saving target, the secondary control loop, i.e., the power saving control loop, then adjusts the physical CPU frequency to maintain the average response time at a desired level (e.g., 90 percent) for all web applications. Consequently, the target

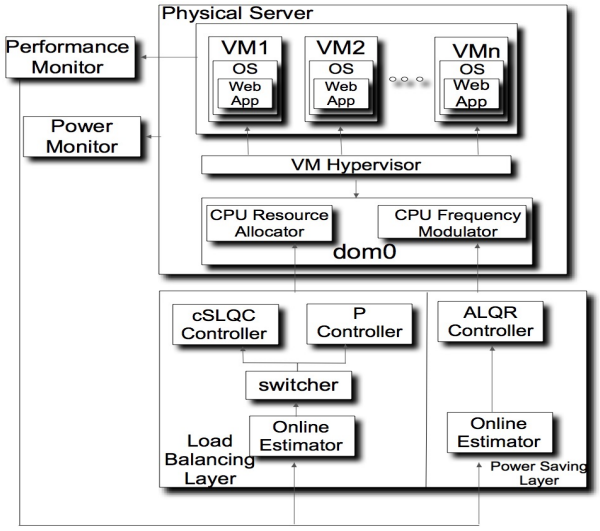


Fig. 1 Stochastic control architecture of PAPMSC

of power saving is achieved for the physical server while the desired response times are also met for web applications in all the VMs.

In this paper, the response time is chosen as a performance metric, because it is a user-perceived metric and can directly reflect the degree of users' satisfaction about Internet services. Meanwhile, the processor is our control objective for two reasons: 1). It is the main contributor to the total power consumption of a server [13]. The power consumption difference between the highest and lowest power states for the processor is large enough to compensate for the power variation of other components. Thus, the *power shifting* among different power states can provide an effective way for server power control. 2). The CPU-intensive business logic processing in multi-tier applications is often the bottleneck [14], hence we focus on CPU resources. Additionally, our design can be extended for I/O-intensive workloads by adding the I/O requirement as another *manipulated variable* in the model. We also choose the technique DVFS (i.e., processor dynamic frequency and voltage scaling) as our power control tool, since the DVFS has a smaller overhead than other actuation methods (e.g., turning on/off server or live migration), and is suitable for real time control of Internet services.

The key components in the first control loop consist of a switcher, a cSLQC controller, a P controller and a CPU resource allocator while there are an ALQR controller and a CPU frequency modulator in the second control layer. Additionally, the online estimator is designed in each control layer to dynamically adjust the model parameters.

3.1 Power and performance monitors

The power monitor measures the overall power consumption of the physical server in the last control interval. The performance monitor located in each VM measures the average response time of the web application in the last control interval. Please note that our control architecture can also be applied to control the percentile-based response time (e.g. 90 or 95 percentile), since our design does not use any semantic information regarding the performance metric and uses the performance values as raw data for modeling and control.

3.2 Load balancing control layer

In this loop, the response time of each VM is considered as the *controlled variable* and the average response time of all VMs as the *reference*. The amount of resource to be allocated to each VM is the *manipulated variable*. Due to the complexity of the computer system and dynamical nature of web workloads, we define the system as a stochastic time-varying system. The relationship between the control variables and manipulated variables is considered as a coupled MIMO system and formulated in the form of an Autoregressive-moving-average model with exogenous inputs (ARMAX) model. In each control interval, the online estimator adaptively tunes the system parameters by using the computational efficient wRLS algorithm. Then, the switcher chooses the controller based on the performance index of the online estimator (e.g., the fitting percentage). If the fitting percentage outnumbers the defined threshold, the switcher sends the updated model information to the cSLQC controller, where the cSLQC controller calculates the optimal resource allocation for the next control period through minimizing a quadratic cost function with constants; Otherwise, the P controller is adopted and assigns the resource to each VM based on the last optimal result. The CPU resource allocator works as an actuator in control system. It enforces the calculated resource allocation solution in order to maintain the response time of each VM close to the reference response time. Our control solution can be extended to the multi-tier web services case by modeling the relationship between the SLA matrices and resource allocation on different tiers.

3.3 Power saving control layer

In this loop, the online estimator receives the average response time of all VMs from the monitors, which is the *controlled variable* of this loop. Then the LQR controller adjusts the CPU frequency (i.e., *manipulated variable*) to manage the average response time to a desired level (e.g.,

90 percent) of all VMs. Next, the CPU modulator changes the frequency level of processors according to the result of the LQR controller. If we set the load balancing control loop to run on a smaller timescale and enter its steady state by the end of each control period of power saving control loop, all VMs would have the same relative response time. For power efficiency, when the average response time of all applications exceeds the desired level, the controller automatically reduces the CPU frequency to a lower level. Conversely, the controller adjusts the CPU frequency to a higher level. Similarly, the relationship between the average response time and CPU frequency is considered as the ARMAX model adapted recursively. For this purpose, it uses the online estimator to calculate the model parameters iteratively according to collected the temporal data.

Consequently, the goal of power saving can be achieved for the physical server while maintaining a more effective resource utilization for each web application hosted on VMs. Clearly, the two control loops need to be worked in a coordinate way since they use the same object (i.e., response time) as the controlled variable. In order to decouple the two control loops and design them independently, we employ a similar decoupling technique as in [12] where the period of response time control loop is supposed to be much longer than the setting time of load balancing control loop. Therefore, the frequency and average response time are fixed constants in the first control loop, which guarantees that the load-balancing loop can always enter its steady state within one control period of the power saving control loop. As a result, both of them work in a coordination way.

In practice, a bottleneck can switch between multiple resources (e.g., CPU, memory or disk) depending on the workload patterns. To provide response time guarantees to a multi-tier application, our control architecture can be extended to deal with a multiple-resources case. For the load-balancing layer, we treat the other type of resource (e.g., the memory or the I/O disk) as an additional manipulated variable, and rebuild the ARMAX model by modeling the correlations among all the resource types and SLAs metrics (i.e., the response time or throughput). Then the online estimator also dynamically adjusts the ARMAX model parameters in each control period. Based on the updated model, the controller decides how many resources are allocated to VM to achieve performance assurance. For the power-saving layer, our control solution can automatically adjust the CPU frequency to a relatively lower level to meet the goal of power saving, since the CPU speed is no longer the main factor as a constraint on system performance under the memory or I/O disk bottleneck case. We will test our thoughts in the three-tier system case as the part of our future work.

4 Load Balancing Control Design

In this section, we first present the model for the load balancing control, and introduce the design of cSLQC controller and P controller.

4.1 System Modeling

For a virtualized server that hosts n virtual machines, T_1 is the control period. $rt_i(k)$ is the average response time of VM_i in the time interval $[(k-1)T_1, kT_1]$. $r_i(k)$ is the relative response time of VM_i, namely, $r_i(k) = \frac{rt_i(k)}{d_i}$, where d_i is the maximum allowed response time of VM_i. The reference $\hat{r}(k)$ denotes the average relative response time of all VMs, namely, $\hat{r}(k) = \sum_{i=1}^{n} r_i(k)/n$. As the manipulated variable u, we use *weight* to assign the CPU resource to virtual machine, which is a key parameter in Xen credit scheduler [15]. Specifically, the amount of CPU resource allocation to each VM is proportional to the weight value.

For designing an effective controller, it is important to model the dynamical of the controlled system. However, the complexity and nonlinearity make it impossible to obtain a well-established physical model for most computer systems, hence we treat the problem as a black-box problem and *System identification* [16] is used to build our model. Due to the resource contention between co-located VMs and there are some uncertainties of real-world web servers(e.g. the variability of the processing times[45], the bursty nature of web workloads can cause a significant noise in the system output[10]), we consider the web server system is a stochastic time-varying system and formulate as a coupled MIMO control problem. Furthermore, different with the existing works consider the web server as the time-invariant system or ignores the disturbances in their model[12, 20, 25, 28], we adopt the time-varying AMRAX model to build our system model. The basic disadvantage with the ARMA model is that the lack of adequate freedom in describing the properties of the disturbance, which is the popular and recently used technique for modeling web services. The AMRAX model adds flexibility to the system model by describing the equation error as a moving average of white noise. Specifically, let $\mathbf{u}(k) = [u_1(t), u_2(t), \ldots, u_n(t)]^T$ be the input vector and $\mathbf{r}(k) = [r_1(t), r_2(t), \ldots, r_n(t)]^T$ be the output vertor. Instead of directly using $\mathbf{u}(k)$ and $\mathbf{r}(k)$, we use their differences $\Delta\mathbf{r}(k) = \mathbf{r}(k) - \hat{\mathbf{r}}(k)$ and $\Delta\mathbf{u}(k) = \mathbf{u}(k) - \hat{\mathbf{u}}$, where $\hat{u}$ is a typical value in Xen. The system model is described as:

$$\Delta\mathbf{r}(k+1) = \mathbf{A}(k)\Delta\mathbf{r}(k) + \mathbf{B}(k)\Delta\mathbf{u}(k) + \mathbf{C}(k)\omega(k) \quad (1)$$

where $\omega(k) \in \mathbb{R}^n$ is the disturbance vector (i.e., an unknown quantity), the matrices $\mathbf{A}(k) \in \mathbb{R}^{n\times n}$, $\mathbf{B}(k) \in \mathbb{R}^{n\times n}$, and $\mathbf{C}(k) \in \mathbb{R}^{n\times n}$ respectively. The typical assumption is that

the disturbances ω are independent and normally distributed (i.i.d Gaussian white noise)[16], i.e., $\omega \sim \mathcal{N}(0,I)$.

It is important to dynamically update the system model based on the most recent measured data, since the workload of web application varies quickly and unexpectedly. The Sliding Window Recursive Least Square algorithm is used [16]. In this algorithm, it only uses the W_c most recent observations (equally weighted) for parameter estimation. The sliding window corresponds to moving a constant-length (i.e. W_c) window along the data sequence. Since the model is updated from the sliding window instead of being computed from the entire history, a lot of computational complexity is reduced. In practice, we set $W_c = 20$ based on a comprehensive consideration of the experiment (e.g., estimation accuracy and execution efficiency).

4.2 cSLQC Controller Design

We apply the Constrained Stochastic Linear-quadratic control (cSLQC) theory to design the controller, since cSLQC is a tractable control technique that deals with stochastic systems in the presence of linear constrains on the control input and probability constraint on the states[18,19]. This characteristic makes the cSLQC well suited for web servers.

4.2.1 Cost Function

Compared with the existing works consider the load-balancing control as a dynamical optimization problem, we consider it as an robust dynamical optimization problem, where the relative response time vector $\mathbf{r}(k)$ is required to stay within certain bounds of a constant around reference $\hat{\mathbf{r}}(k)$ in the presence of a disturbance vector $\omega(k)$, and it is defined over a time interval (i.e., N is called *prediction horizon*) in the future. Equivalently we hope to minimize the state $\Delta\mathbf{r}(k)$ to keep the response time of each VM (i.e., $r_1(k), r_2(k), \cdots, r_n(k)$) close to the desired value $\hat{\mathbf{r}}(k)$ in the next N control intervals. Additionally, we hope to use as little system resources as possible. Hence the finite horizon cost function corresponding to current state $\Delta\mathbf{r}(0)$ is described as follow:

$$J_1(\Delta\mathbf{r}(0), \Delta\mathbf{u}, \omega) = \sum_{k=1}^{N} \Delta\mathbf{r}(k)^T Q \Delta\mathbf{r}(k) + \sum_{k=0}^{N-1} \Delta\mathbf{u}(k)^T R \Delta\mathbf{u}(k) \tag{2}$$

where Q and R are positive-semidefinite weighting matrices that establish a trade-off between control error and control cost.

In contrast with [12,20,25], we choose the resource allocation solution that performs best for the most pessimistic disturbance ω within a reasonable uncertainty set Θ. Specifically, we search for an optimal control $\Delta\mathbf{u}^*$ such that

$$\min_{\Delta\mathbf{u}\in\mathbb{R}^{N\times n}} \{\max_{\omega\in\Theta} J_1(\Delta r(0), \Delta u, \omega)\} \tag{3}$$

Once the control solution $\Delta\mathbf{u}^*$ is computed, we take the first n components and apply them as the current control.

4.2.2 Probabilistic Response Time Guarantees and Constraint on Resource Allocation

Since the computer resources are limited and subject to the constraints of the hardware, we assume that the CPU resource is the critical resource being shared by each VM. Hence the resource request from each VM needs to satisfy the following capacity constraints:

$$\Delta u_{low} \leq \Delta u_i(k) \leq \Delta u_{up} \tag{4}$$

where Δu_{low} and Δu_{up} mean the minimal and maximal values of weight change, respectively.

Meanwhile, difference with using no constraint or hard constraint on the response time[12,28], we use the probability constraint on the response time (i.e., we do not require constraints on the response time to be satisfied at all times, but only with a predefined probability), since the calculated response time is varying in a certain range due to the variability of the processing times[45]. Hence, we use the soft linear constrains in the form:

$$P[\mathbf{G}^T \Delta\mathbf{r}(k) > g] \leq a \tag{5}$$

where $\mathbf{G}$ is a scaling matrix, $\mathbf{g}$ is the limits on $\Delta\mathbf{r}$ and the scalar $\mathbf{a} \in [0,1]$ (i.e., defined as the probability of constraint violation). Eq.(5) requires that the condition $\mathbf{G}^T\Delta\mathbf{r} > \mathbf{g}$ is fulfilled with probability smaller or equal than $\mathbf{a}$. Such uncertain constraints are called *chance constraints*.

4.2.3 Semidefinite Programming (SDP) Approach

If there is no constraint, the optimization problem under Gaussian disturbance can be solved by linear quadratic regular (LQR). However, with constraints on (4) and (5), this approach involves a huge amount of computation to find the optimal solution. However, we are able to find the optimal solution via the SDP approach.

Without loss of generality, its valid to assume noise ω to be a Gaussian white noise, since Gaussian white noise is thoroughly studied in the control literature, and its widely applied as a good approximation of many real-world situations and generates mathematically tractable models. It's worth mentioning that the Constrained Stochastic Linear-quadratic control scheme could be generalized to the case other than Gaussian noise. For example, as mentioned in [18], it could handle the problem where the noise is assumed to be bounded. For more general noise other than Gaussian white noise, we need more advanced filtering technique to obtain an accurate model that may be our future work.

After the above analysis, the load balancing control can be described by the following robust optimization problem named **Problem1**:

$$\min_{\Delta \mathbf{u} \in \mathbb{R}^{N \times n}} \{\max_{\omega \in \Theta} J_1(\Delta r(0), \Delta u, \omega)\} \quad (6)$$

subject to

$$\Delta u_{low} \leq \Delta u_i(k) \leq \Delta u_{up}, \forall i = 1, \ldots, n, \quad (7)$$

$$P[\mathbf{G}^T \Delta \mathbf{r}(k) > g] \leq a \quad (8)$$

$$\omega \sim \mathcal{N}(0, I) \quad (9)$$

In order to solve this optimization problem with probability constraints, we first transform (8) to the linear constraints. Namely, we now show how to explicitly ensure that linear constraints on the response time will hold with a desirably high probability.

Theorem 1 *Consider a linear system with the states written as*

$$\mathbf{x}_k = \tilde{\mathbf{A}}_{k-1} x_0 + \tilde{\mathbf{B}}_{k-1} \mathbf{u} + \tilde{\mathbf{C}}_{k-1} \omega \quad (10)$$

where $\tilde{\mathbf{A}}_{k-1}, \tilde{\mathbf{B}}_{k-1}, \tilde{\mathbf{C}}_{k-1}$ *are defined the same as in [18]. Then, the constraint*

$$\mathbf{p}^T \mathbf{u} \leq q \quad (11)$$

where $\mathbf{p} = \tilde{\mathbf{B}}^T \mathbf{G}$, $q = g - \mathbf{G}^T \tilde{\mathbf{A}} \mathbf{x}_0 - \parallel \tilde{\mathbf{C}}^T \mathbf{G} \parallel_2 \Phi^{-1}(1-a)$ *implies the following:*

$$P[\mathbf{G}^T \mathbf{x} > g] \leq a \quad (12)$$

Proof :
We have

$$P[\mathbf{G}^T \mathbf{x} > g] = P(\mathbf{G}^T \tilde{\mathbf{C}} \omega > g - \mathbf{G}^T \tilde{\mathbf{A}} \mathbf{x}_0 - \mathbf{G}^T \tilde{\mathbf{B}} \mathbf{u})$$
$$= 1 - \Phi\left(\frac{g - \mathbf{G}^T \tilde{\mathbf{A}} \mathbf{x}_0 - \mathbf{G}^T \tilde{\mathbf{B}} \mathbf{u}}{\parallel \tilde{\mathbf{C}}^T \mathbf{G} \parallel_2}\right) \leq a$$

Hence, **Problem 1** can be rewritten as:

$$\min_{\Delta \mathbf{u} \in \mathbb{R}^{N \times n}} \{\max_{\omega \in \Theta} J_1(\Delta r(0), \Delta u, \omega)\} \quad (13)$$

subject to

$$\Delta u_{low} \leq \Delta u_i(k) \leq \Delta u_{up}, \forall i = 1, \ldots, n \quad (14)$$

$$\mathbf{G}^T \mathbf{B}(k) \Delta \mathbf{u} \leq g - \mathbf{G}^T \mathbf{A}(k) \Delta r(0) - \parallel \mathbf{C}(k)^T \mathbf{G} \parallel_2 \Phi^{-1}(1-a) \quad (15)$$

$$\omega \sim \mathcal{N}(0, I) \quad (16)$$

This stochastic linear-quadratic control problem can be solved by SDP techniques and the stability issue has been analyzed theoretically in [18]. Since our paper does not focus on SDP, for a detailed discussions, interested readers could consult Theorem 3 and Theorem 8 in [18].

One traditional approach to deal with this situation as **Problem1** is to apply the dynamic programming, which requires solving the Bellman equation recursively. The complexity issue claimed in our paper corresponds to the unfavorable property that it becomes, for large-scale problems, impossible to solve optimally when the constraints are included. In contrast to the above approach, SDP method leverages the useful property of convex optimization, whose problem instances are quite robust (in terms of complexity) to perturbations in the constraint structure. It is tractable, even in the presence of control and states pace constraints. For the complexity of our SDP, as mentioned in [18] , its complexity is not much more than the complexity of linear feedback (i.e., the Riccati approach). In particular in this case, optimal policies in this case can be computed from that optimizing a convex function over a scalar, then multiplying the initial state by appropriate matrices.

4.3 Integration with Proportional Controller

When the system model becomes inaccurate due to the bursty workload that causes a significant disturbance in system output, the performance of cSLQC controller will degrade, or, in some case, the behavior of controller becomes jitter. In the deterministic control approach, the effectiveness of the controller seriously depends on accuracy of the system model and the result of the estimation. However, guaranteeing the accuracy of a model under all conditions may be infeasible. In this section, we integrate our cSLQC controller with a proportional controller to guarantee the desired response time for each VM. The responsibility of the P controller is to compensate for unexpected poor performance of the online estimator. In every control loop, we evaluate the performance of online estimator by calculating the fitting percentage. When the value of fitting percentage is larger than the threshold, the cSLQC controller is invoked while the P controller is disabled. When the fitting percentage is smaller than the threshold, the cSLQC controller is disabled because the accuracy of online estimator is not enough for making a right control decision. In the meantime, the P controller is enabled to make a control decision based on the last good control result. In the following, we briefly introduce how to switch between the cSLQC controller and P controller, and the design of the P controller.

4.3.1 Controller switching condition

The condition of switching between the two controllers is depending on the performance of the online estimator (i.e., the fitting percentage between the training trace and sampled trace during the slide window). If the fitting percentage of the online estimator is larger than the defined threshold δ at time interval k^{th}, the online estimator is deemed good. Otherwise, the online estimator is determined to have an unaccepted performance. In this case, we discard the estimated parameters and switch to the P controller, which is a model-independent control algorithm.

4.3.2 P controller design

An important advantage of P control is that its control performance is not affected by the result of the online estimator [29]. It is robust in the presence of online estimator errors. In each control period of the P controller, it measures the current relative response time $\mathbf{r}(k)$, calculates the difference $\Delta\mathbf{r}(k)$ between $\mathbf{r}(k)$ and the reference $\hat{\mathbf{r}}(k)$, and then computes a new resource allocation $\mathbf{u}(k+1) = \Delta\mathbf{u}(k+1) + \hat{\mathbf{u}}$ accordingly. Based on the control theory, we design the P controller as:

$$\Delta\mathbf{u}(k+1) = \mathbf{K_p}\Delta\mathbf{r}(k) + \Delta\mathbf{u}(k) \tag{17}$$

where the vector $\mathbf{K_p}$ is the proportional control parameter that can be analytically chosen from experimental data statistically. In this paper, we consider the $\mathbf{K_p}$ to be a constant for the convenience.

The control flow of the load balancing loop is shown in Algorithm 1. Specifically, the inputs of the algorithm at interval k^{th} are $\mathbf{u}(k)$ and $\mathbf{r}(k)$, respectively. It returns the optimal resource allocation solution $\mathbf{u}(k+1)$ as the next control input. It first computes the average response time $\hat{r}(k)$ by using the response time of all VMs, which is collected from the performance monitors. Then, the difference $\Delta\mathbf{r}$ and $\Delta\mathbf{u}$ can be calculated. The function *setData()* stores the data $\Delta\mathbf{r}$ and $\Delta\mathbf{u}$ into the array *SampleData* and replaces the oldest data with the latest ones for maintaining constrain the length of sliding window. Next, the function *estimator()* updates the system parameters by using data set *SampleData* and SWRLS algorithm. Then, the switcher makes a control selection based on the fitting percentage $\theta_{\mathbf{i}}(\mathbf{k})$, calculated by the function *calcFP()*. If the $\theta_{\mathbf{i}}(\mathbf{k})$ of all VMs exceeds the performance threshold δ, the optimal control solution is computed by the function *cSLQC(SDP)*, the structure SDP is formulated by using *transtoSDP()*, the details can be found in ([18]). The function *cSLQC()* is implemented by using the Matlab *Linear Matrix Inequalities (LMI) Solver* to solve the robust optimization problem defined in Eq. (13) with constraints Eqs. (14), (15), (16). Otherwise, the P controller is selected to compensate for the degraded performance of the online estimator. The function *Pcontr()* calculates $\Delta\mathbf{u}(k+1)$ according to Eq. (17). Finally, the new resource allocation solution $\mathbf{u}(k+1)$ is computed and sent to the allocator.

Algorithm 1 Pseudo-code of Load balancing control loop

Input:
Output:
$u_i(k+1), 1 \le i \le n$;
1: $[A(k), B(k), C(k)] \leftarrow estimator(SampleData)$;
2: $\theta_i(k) \leftarrow calcFP(SampleData), 1 \le i \le n$;
3: **if** $\forall\theta_i(k) > P_i$ **then**
4: $SDP \leftarrow transtoSDP(A(k), B(k), C(k))$;
5: $\Delta u_i(k+1) \leftarrow cSLQC(SDP),\ 1 \le i \le n$;
6: **else**
7: $\Delta u_i(k+1) \leftarrow Pcontr(\Delta r_i(k), \Delta u_i(k)), 1 \le i \le n$;
8: **end if**
9: $u_i(k+1) \leftarrow \Delta u_i(k+1) + \hat{u}_i,\ 1 \le i \le n$;
10: **return** $u_i(k+1),\ 1 \le i \le n$;

5 Power Saving Control

The power saving control layer controls the average response time of all VMs to the desired level by scaling the physical CPU frequency. Compared with the coupled MIMO control problem in load balancing control, the complexity of power saving control is much simpler than the load balancing control, which is a Single-Input-Single-Output (SISO) control problem. The Linear Quadratic Regulator (LQR) is implemented in this layer, it is a well-known control technique that provides practical feedback gains for control problems [17]. Compared to the cSLQC control, LQR has a smaller runtime computational overhead, which just uses the simple saturation method to constrain the CPU frequency. Together with the frequency adjustment of CPU obtained from LQR gain matrix, the goal of power saving can be achieved. In this section, we first introduce the modeling of power saving control layer, then a detailed design of LQR controller is presented.

5.1 System Modeling

In the k^{th} control interval, the control target is to adaptively manipulate the CPU frequency such that the current average response time can converge to the desired level. A well-established physical equation is unavailable for computer system, therefore we use system identification theory to build the system model. Let us first introduce some notations, $\Delta\hat{r}(k)$ represents the difference between the average relative response time $\hat{r}(k)$ and the set point R, namely, $\Delta\hat{r}(k) = \hat{r}(k) - R$. The frequency change is defined as $\Delta f(k) = f(k) - f$. f is the typical operating point of CPU frequency selected from a set of available frequencies. Here, we choose the midpoint as f and corresponding average response time recorded at frequency f is used as R. Implementing all the aforementioned operations and system identification, we get the following model:

$$\Delta r(k+1) = a(k)\Delta r(k) + b(k)\Delta f(k) + c(k)w(k) \tag{18}$$

where the parameters $a(k)$ and $b(k)$ represent the correlation between $\Delta f(k)$, $\Delta r(k)$ and $\Delta r(k+1)$, respectively. It should be remarked that, in this control layer, we also implement the online estimator with SWRLS algorithm to dynamically tune the system parameters. $w(k)$ is the white noise.

5.2 LQR controller

In order to apply the LQR control approach to this layer, we define the cost function as follow:

$$J_2(\Delta r, \Delta f) = \sum_{k=1}^{\infty} [\Delta r(k)^T Q \Delta r(k) + \Delta f(k)^T R \Delta f(k)] \quad (19)$$

where Q and R are positive-semidefinite matrices, the first term provides a measure of the output energy and the second term provides a measure of the control signal energy. A general rule to determine the values of Q and R is described as: the larger the Q is, the faster the response to average response time change will be; in addition, the larger the R is, the less sensitive to noise the system will be.

Then the optimal CPU frequency can be calculated by dynamic programming as follows: [17]:

$$\Delta f(k) = -K(k)\Delta r(k) \quad (20)$$

where the control gain matrix $K(k)$ is

$$K(k) = (R + b^T(k)P(k)b(k))^{-1} b^T(k)P(k)a(k) \quad (21)$$

and $P(k)$ is calculated iteratively backwards in time by the dynamic Riccati equation from initial condition P_0:

$$P(k+1) = Q + a^T(k)(P(k) - P(k)b^T(k)(R + b^T(k)P(k)b(k))^{-1} b^T(k)P(k))a(k)$$

Since the CPU frequency needs to satisfy the following capacity constraints:

$$f_{min} \leq f(k) \leq f_{max} \quad (22)$$

we employ a simple saturation method to constrain the control input:

$$f(k) = \begin{cases} f^*(k) & \text{if } \mathrm{f_{min}} \leq \mathrm{f^*(k)} \leq \mathrm{f_{max}} \\ f_{min} & \text{if } \mathrm{f^*(k)} \leq \mathrm{f_{min}} \\ f_{max} & \text{if } \mathrm{f^*(k)} \geq \mathrm{f_{max}} \end{cases} \quad (23)$$

If the optimal CPU frequency computed by LQR exceeds the range of available physical frequencies, then the control input will be cut off to be the maximum or minimum value.

6 System Implementation

In this section, we introduce the details of our testbed and each component in stochastic control architecture.

6.1 Testbed

We built two testbeds using Xen to evaluate the PAPMSC. The testbed1 consists of two physical computers and connected via an Ethernet switch. One is *Server* that hosts three virtual machines, and the other one is *Client* that generates the HTTP requests to the Server. Both of them are equipped with an Intel Xeon X5365 processor, 4GB RAM and is running Fedora Core 8 with Linux Kernel 2.6.31. The processor has four cores and supports four physical frequency levels: 2 GHz, 2.33 GHz, 2.67 GHz and 3 GHz. In order to evaluate the scalability of the stochastic controller, testbed2 was built. It also includes two physical computers, but the computer used as *Server* that hosts seven virtual machines. *Server* is equipped with two Intel Xeon E5405 processors, 12MB cache and 8GB RAM. The host operating system is Ubuntu 12.04.5LTS with Linux Kernel 3.13.0.

Xen 3.4 is running on the server-side as the virtualization tool. The Xen hypervisor module controls the CPU clock, memory, and disk space allocated to each virtual machine. Each virtual machine is allocated 1024 MB of RAM, 10 GB of hard disk space. Each of them is also configured with the same number virtual CPUs as the physical computer has (e.g., each VM has 4 VCPUs and 8 VCPUs in testbed1 and testbed2, respectively). They are running on the domU, the guest operating system is Debian GNU/Linux 5.0 and the Apache server is running on each VM as the virtualized web server. The web application is a dynamical webpage with written in PHP, which can serve large-scale numerical calculations. In detail the working principle of the webpage is that when it receives a HTTP request from the client, it solves a mathematical formula automatically by executing a finite iteration process. It should be remarked that applications in the VMs are independent of each other.

On the client-side, as many related studies [12,25,31], the Apache HTTP server benchmarking tool (ab) is used, which is designed to test the capability of a particular Apache installation. It allows users to manually set the concurrency level, which is the number of requests to perform in a very short time to emulate multiple clients. Without loss of generality, we assign different maximum allowed response times to each VMs based on experiment results. Specially, d1=300ms for VM1, d2=350ms for VM2, d3=400ms for VM3. Meanwhile, we use the relative response time as the normalized measure of response time, which has defined in Section 4.1. In a real system, the system administrator can manually set the maximum response times according to the requirements of the web applications running in the VMs.

The WattsUp Pro power meter [30] is used as the measurement tool for power consumption of the physical server. It has an accuracy of 1.5% of the measured value with a 1HZ sampling frequency. The measured data are sent to the controllers through a system file */dev/ttyUSB0*.

Table 1 Calculation overhead

Components	Average time(sec)
Online estimator	0.22
SDP Controller	0.78
CPU allocator	0.11
Total time	1.11

The online calculation overhead of the SDP controller is mainly affected by the four factors: 1). Time taken to collect response time statistics from the applications, 2). Time required to update the estimated model, 3). Time required to compute a control decision, 4). Actuation time. Table1 shows the average time taken for each of these factors. In the paper we set the collection performance statistics time is fixed (i.e. running time of Performance monitor is 3s), sufficiently longer than any VM response time. Table1 shows the average time taken for each of these factors. Note that control overheads play an important role in determining the control period. In here, we set the control interval of load balancing control layer to be 6s, which is sufficiently large to overcome the control overheads and also get the stable response time from VMs. Since the longest settling time of the load balancing control is 12s , the control period of power saving control is chosen to 24s .

6.2 Control Components

We now introduce the implementation details of components in control system.

6.2.1 Performance Monitor

To eliminate the impact of network delays, we focus on controlling the server-side response time in this paper. A small daemon program runs on each VM as a response time monitor. It periodically inserts multiple sample requests into the request queue, which comes from the client-side workload generator to the Apache server. Two time stamps are used, one when a sample request is inserted and the other when the response is received. The difference between the two stamps is used as the server-side response time and sent to the controller periodically.

6.2.2 CPU Resource Allocator

The Credit Scheduler [15], Xen's proportional share scheduler, is used to allocate available CPU resource. The scheduler allocates the CPU resource by assigning a *weight* and a *cap* to each VM. The *cap* limits the maximum amount of CPU resource a VM can use. The *weight* specifies the CPU resource allocation for each VM, the scheduler allocates CPU in proportion to the *weight* assigned to the VMs. For example, a VM with a *weight* of 512 will get twice as much CPU as a domain with a *weight* of 256 on the same physical server. In this paper, we use the *weight* to implement this allocation and set the *cap* value to 0 for each VM, which means that each VM can use all of the CPUs as the physical computer has.

6.2.3 CPU Frequency Modulator

The Intel's SpeedStep technology is used to enforce the desired CPU frequency. Xen 3.4 has built-in support for Dynamic Voltage and Frequency Scaling (DVFS). We use the *xenpm* tool to modify the CPU frequency to the desired frequency and select different scaling governors. Four scaling governors are available in our system, the on-demand governor, the power-save governor, the performance governor and the user-space governor. Specifically, the on-demand governor dynamically switches between CPU(s) available if at 95% CPU load but it has a low efficiency; the performance governor runs the CPU at max frequency while the power-save governor runs the CPU at a minimum frequency; the user-space governor runs the CPU at user specified frequencies. The user-space governor is the default choice.

The Intel CPU used in our experiments only supports four discrete physical frequency level, but the new CPU frequency level periodically calculated by Eq.(22) and (23) could be any value in not exactly one of the four supported frequency levels. Hence, in order to approximate the desired value, the modulator code must locally resolve the output value of LQR controller to a series of supported frequency levels. To do this, we implement a first-order delta-sigma modulator, which is commonly used in analog-to digital signal conversion. For example, to approximate 2.89GHz during a control period, the modulator would output a sequence of supported level: 2.67, 3, 3, 2.67, 3, 3, etc., on a smaller timescale. The detailed algorithm of the firs-order delta-sigma modulator can be found in [46]. The modulator is a daemon program written in C. In every control period, the LQR controller sends a new desired CPU frequency level to the modulator through a named pipe, then the modulator changes the CPU frequency according to the output sequence resulted from the firs-order delta-sigma modulator based on the desired CPU frequency.

6.2.4 Controllers

All the controllers are implemented in dom0 to receive response times from the monitors and run the control algorithms presented in Section 4 and 5. They are all implemented in a daemon program written in C.

7 Experimental Results

In this section, we evaluate the modeling accuracy of the PAPMSC, and examine the performance of load balancing control under the dynamic workload scenario. In order to compare the performance, we adopt an existing control solution PARTIC from [12] as the baseline, which is the two-layer control theory based approach. Furthermore, we demonstrate the effectiveness of integrating cSLQC controller with a P controller under the bursty workload scenario. To evaluate the power efficiency, we use the performance governor as the baseline. It works as the performance-oriented solution. Finally, we evaluate the scalability of the load balancing control (i.e., the combination of cSLQC controller and P controller) using the real-world Internet trace as the workload.

7.1 Model Validation

The accuracy of the ARMAX model has a significant impact on the effective control of performance and power. We first validate system models using a sequence of pseudo-random digital white noise to simulate the system input, and then measure the control output. For the load balancing control loop, we set the CPU frequency at a fixed level, and allocated the randomly varying VCPU weight to each VM from 100 to 300 in every control period. For the power saving control layer, we assigned the fixed weight value to each VM and set the randomly varying CPU frequency in every control period. Each VM faces a workload of 60 concurrent users. Fig.2a and Fig.2b show the performance of ARMAX models in the different control layer. It demonstrates that the ARMAX models can accurately predict the response time changes of VM2 in the presence of interference between co-located VM1 and VM3.

To quantify the prediction accuracy of ARMAX models, we used the mean absolute percentage error (MAPE), defined as $MAPE = \frac{1}{N}\sum_{k=1}^{N}|\frac{y_{pre}(k)-y(k)}{y(k)}|$, where N is the total number of samples, $y_{pre}(k)$ and y are the model-predicted value and the measured value of response time. We compare our results with a popular and recently used technique for modeling web services, ARMA. Fig.2c shows that ARMAX model outperforms ARMA model in terms of predicting accuracy and stability. On average, the improvement in prediction accuracy for response time of each VM and the average response time are $33.98\%, 36.59\%, 31.67\%$ and 34.23% respectively. Compared to the ARMA modeling, ARMAX modeling has an additional benefit, it incorporates the white noise items to describe the unknown system disturbance, which adds the flexibility to the model.

To validate the performance of estimation model with different workload, a set of experiments is conducted by using the actual traces from Fig.3. Fig.3 shows an example of time varying incoming workloads to each of the three virtual machines. The strength of concurrency level exhibits time-of-day variations typical of many enterprise workloads and the concurrency level changes significantly within a very short time period. The synthetic web workloads used in our experiment, in part, referring to the log files are from the Soccer World Cup 1998 Web site [40]. As introduced before, we allocate the varying VCPU weight and CPU frequency in different layer respectively. For the actual traces, Fig.4 plots the mean and variance of ARMAX model and ARMA model in terms of MAPE. Compared with the results using fixed workload case (i.e., Fig.2c), the mean of MAPE using ARMA model increases significant in the load balancing layer, while the mean MAPE of ARMAX model still keep under 25% with a little increase. For the power saving layer, both of ARMAX model and ARMA model can predicate the behavior of average relative response time as well as in Fig.2c and all of them under 15%. That is because that the estimation model in seconder layer is a SISO system, which

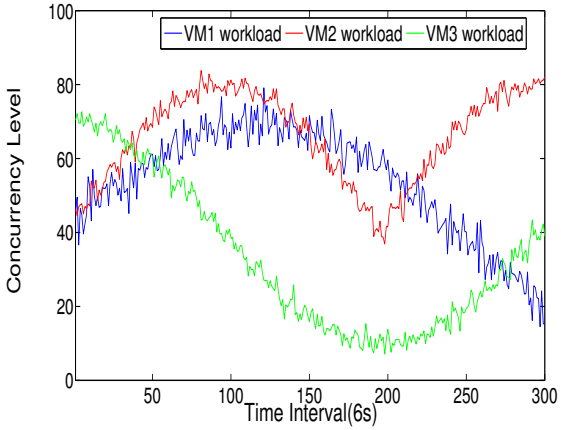


Fig. 3 High dynamical workloads

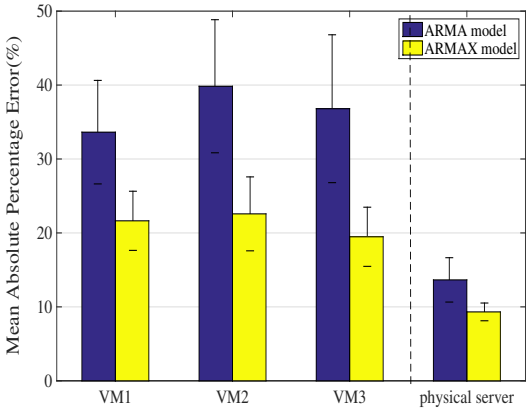
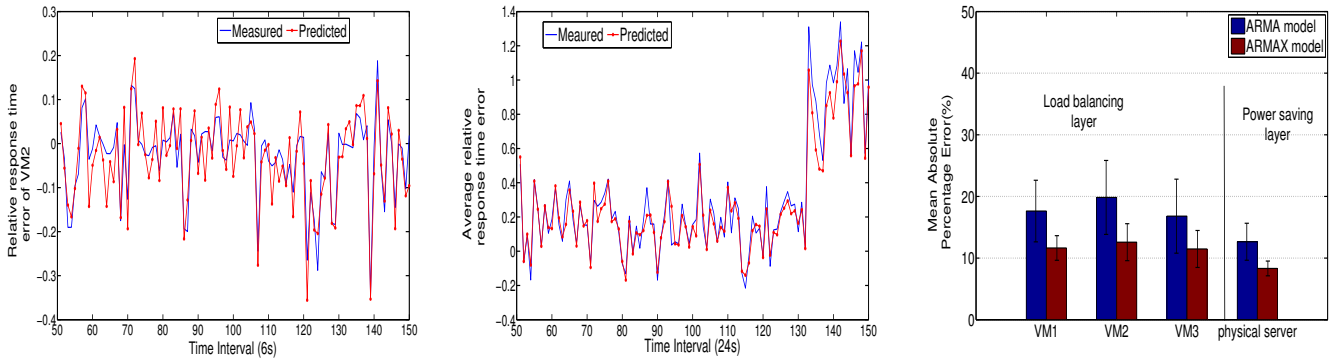


Fig. 4 The pefromance of AMRAX and ARMA under dynamical workload



(a) Load balancing control layer (b) Power saving control layer (c) Response time

Fig. 2 The prediction accuracy of ARMAX models and comparison with ARMA model

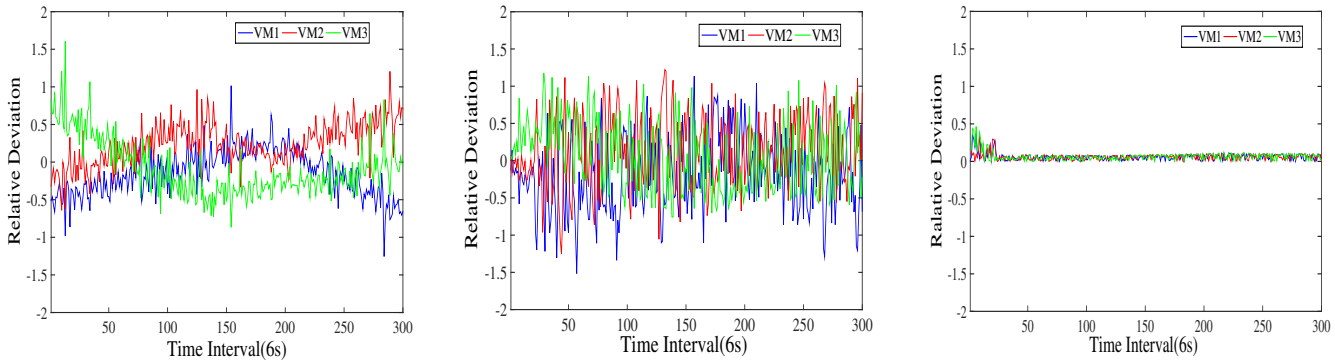
the complexity is much simpler than coupled MIMO model in the load-balancing layer. Throughout the experiments, the online estimator using the ARMAX model can still adapt gracefully to workload change under the actually traces and shows more stability than ARMA model. The essential reason is that due to the response time is affected by many factors (e.g., varying process delay and workload dynamic as well as the control over resource allocation, etc.), hence the estimation model should have the flexibility in terms of system error and the ARMAX model can describe the system disturbances by using a moving average of white noise.

7.2 Load Balancing

To evaluate the performance of cSLQC controller under the high dynamical workload cases (i.e., Fig.3) in the load-balancing layer, the controller in the second layer and P controller are temporarily disabled. To quantify and compare the performance of cSLQC controller and PARTIC, we define the relative deviation for the relative response time as $RD_i(k) = \frac{[r_i(k)-\bar{r}]}{\bar{r}}$. $\bar{r}$ is the SLAs target of the VMs in terms of relative response time and $\bar{r} = 0.85$ in here.

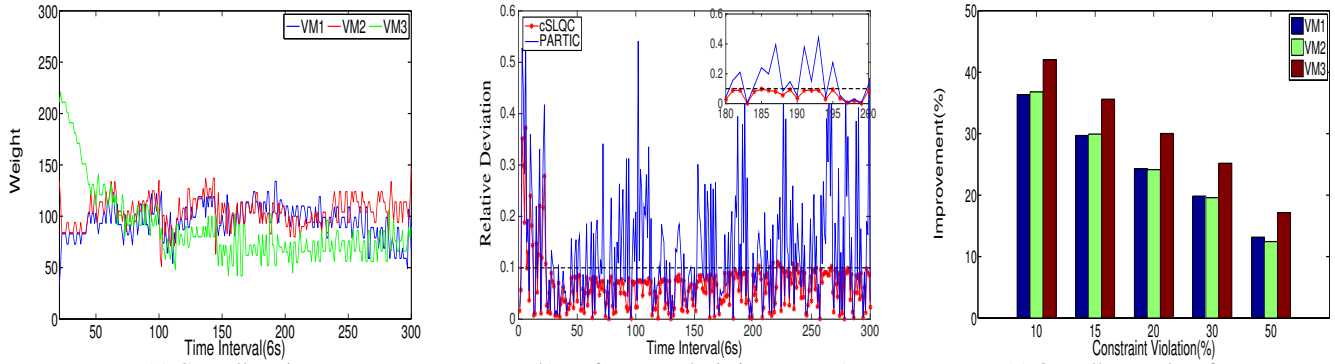
Fig.5a is the performance of OPEN solution, it is an open-loop solution that assigns the fixed CPU resource to each VM in the system. As a result, OPEN solution cannot guarantee the same relative response time for different VMs when they have nonuniform workloads. Fig.5b is the performance of PARTIC. The AMRA(2,2) is used in PARTIC and the control gain of LQR controller is get by using the Matlab command *dlqry* with Q=100, R=1. It shows that the response time of each VM is unstable and exhibiting large oscillation during the whole process, that is because the PARTIC adopts a deterministic control strategy, the control performance heavily depends on the accurate of estimated model. When high dynamical workloads change beyond a certain range, the accuracy of estimated model will become degrades and directly leads to a poor controller. Furthermore, the model PARTIC used is a time-invariant model and build on the offline method, when the working condition is changed, the PARTIC needs to retrain the model in order to keeping the control performance. It is not suitable for the time-varying workload case. Fig.5c shows the performance of cSLQC controller. cSLQC achieves the performance target by dynamically adjusting the CPU resource allocation among the three virtual machines.

From Fig.6a, the controller gives different portions of CPU resource to the three virtual machines according to the dynamical changes in workloads, such that each VM can converge to the average response time of all VMs and the target is met. Fig.6b compares the performance assurance capability of cSLQC controller on $VM1$ with that of PARTIC. For cSLQC controller, we set the limit on the relative response time deviation to 0.1 (i.e., $g = 0.1$), and the probability of constraint violation is 10% (i.e., $a = 0.1$), the prediction horizon $N = 2$. We can see that cSLQC could achieve small relative response time after several step. It implies that it can provide guarantee the performance in terms of response time when the workload varies significantly. cSLQC significantly outperformed the PARTIC in terms of control accuracy and stability, since most of the control error is below 0.1 (i.e., defined limit level). This is mainly due to three reasons: First, cSLQC is able to obtain a better accuracy system model based on the ARMAX model and the online estimator. Second, the cSLQC controller provides more accurate control and system stability due to its defined soft constraint on the response time. The most important reason is that cSLQC controller chooses the resource allocation solution based on minimizing the cost function for the most pessimistic unknown disturbance in system, while the cost function in PARTIC just minimizes the cost function without consider the disturbance. Fig.6c illustrates the improvement in performance efficiency by cSLQC controller for various constraint violations, compared with PARTIC. It



(a) OPEN method (b) PARTIC method (c) cSLQC method

Fig. 5 Comparison with different method in term of performance



(a) CPU allocation (b) Performance deviation on VM1 (c) Overall control performance

Fig. 6 Performance analysis of cSLQC controller

demonstrates that the performance of cSLQC controller degrades when allowing more constraint violations.

7.3 Integration with P Controller

In this experiment, we demonstrate the effectiveness of integrating cSLQC controller with a P controller to handle the degraded performance of the online estimator. We test the cSLQC controller in a scenario where the performance of the online estimator degrades sharply because the bursty workload of VM2. As shown in the Fig.7, we add burstiness into the workload of VM2 by doubling the concurrency level from 60 to 120 randomly. This is common in many websites. For example, the CNN.com website experiences a huge number of accesses suddenly after the terrorist attacks on the United States on September 11, 2001. Furthermore, we add white Gaussian noise to the workload to simulate the unknown and unpredictable incoming requests.

In order to evaluate the effectiveness of PAPMSC for the load-balancing control, we test the performance of P controller and cSLQC with P controller under the bursty workload case respectively. The results are shown in Fig.7b and

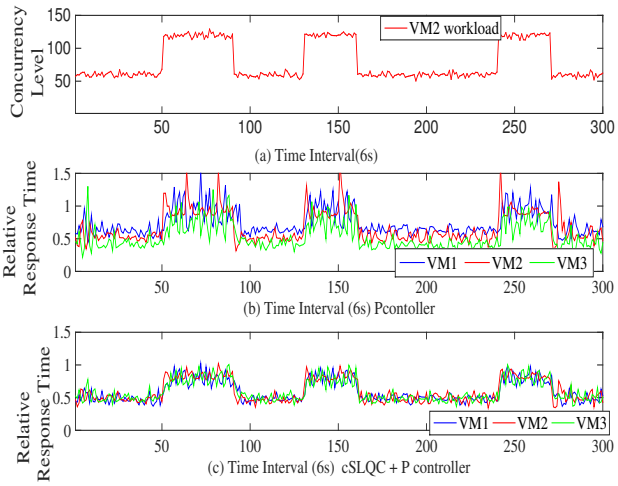


Fig. 7 Performance of load balancing controller under the bursty workload

Fig.7c. For the P controller, there still has a large steady-state error for VMs in terms of relative response time during the whole experiment. Furthermore, the relative response

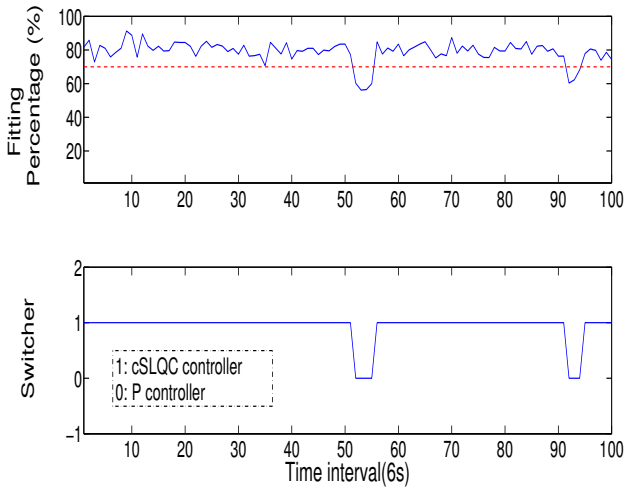


Fig. 8 The behaviors of switcher

time of VM2 higher than 1 partly during the workload of VM2 increases suddenly, which means that VM2 violates response time deadline. The same behavior is also happened on the VM1. Fig. 7c shows that the PAPMSC provides the same response time for all three VMs in the face of the bursty workload. Obviously, VM1 and VM3 experience a longer response time during the burstiness periods, since PAPMSC dynamically assigns less resource to VM1 and VM3, and correspondingly assigns more resource to VM2 for coping with the abrupt changes in the workload of VM2. As a result, the relative response time of all three VMs can converge to the average response time and keep under 1, which means the desired response time have been reached.

To better understand the mechanism of the load balancing control layer, we use the coefficient of determination (i.e., R^2) as the fitting percentage to evaluate the performance of the online estimator. It provides a measure of how well observed outcomes are replicated by the model, which ranges from 0 to 1 (e.g., 1 means the output of model fits the measured data in 100%), $R^2 = 1 - \frac{\sum_{j=1}^{W_c}(\Delta r(j) - \Delta r_m(j))^2}{\sum_{j=1}^{W_c}(\Delta r(j) - \Delta r_{avg})^2}$, where W_c is the size of sampling window, $\Delta r(j)$ and $\Delta r_m(j)$ are the model-predicted value and the measured value by response time monitor, Δr_{avg} is the sample mean of Δr_m in the sample window. Without loss of generality, we define the prediction threshold $P_{thre} = 0.75$ experimentally. Fig. 8 shows that the fitting percentage of VM2 during the first 100 time intervals and the behavior of switcher. we see that the cSLQC controller works as the default controller in the load balancing control loop, since the online estimator has a good performance most of time. However, at the time interval 50^{th}, the workload of VM2 suddenly increases double, causing a serious disturbance in the system output and directly affecting the performance of the online estimator. As shown in the Fig.8, the fitting percentage of VM2 drops sharply under the defined threshold P_{thre}. As a result, for maintaining the desired response time, the system disables the cSLQC controller and switches to the P controller at time interval 53^{th}. When the sliding window collects enough latest data about changes in workloads, the performance of the online estimator becomes good. At time interval 57^{th}, the fitting percentage of VM2 is higher than P_{thre}, for getting a better control performance, the system switches to cSLQC controller and disables the P controller. Note that the desired response time can be maintained even when the online estimator has a degraded performance, because the P controller is a model-independent control method and makes a control decision based on the control error and the last optimal result. Hence, it enhances the robustness of the controller.

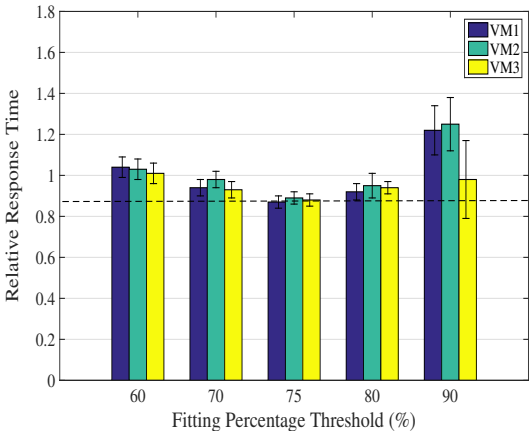


Fig. 9 the performance with variation fitting threshold

To select the best fitting percentage threshold to trigger the P controller, we test the performance of PAPMSC with different fitting percentage threshold. The results are shown in Fig.9. When the fitting percentage threshold is set to 60%, the relative response time of VM1 and VM2 is little higher than 1, which means the VM1 and VM2 violate the response time deadline. The reason can be attributed to the switcher always chooses the cSLQC controller for the load balancing control, since the fitting percentage of the online estimator has remained above 60% during the whole process. However the predication accuracy of online estimator degrades sharply when the workload of VM2 increases suddenly, which directly affects the cSLQC control quality and system stability. The performance of PAPMSC is improved when increasing the value of fitting percentage threshold. The relative response time of all VMs is closer to the QoS target (i.e. 0.85) at the $p_{res} = 75\%$. However, the performance of PAPMSC degrades significantly when setting the fitting percentage threshold to 90%. That's because the switcher exactly chooses the P controller as the load-

balancing controller during the whole process and the P controller cannot guarantee the SLAs target of VMs, which has shown in Fig.7b.

7.4 Power Efficiency

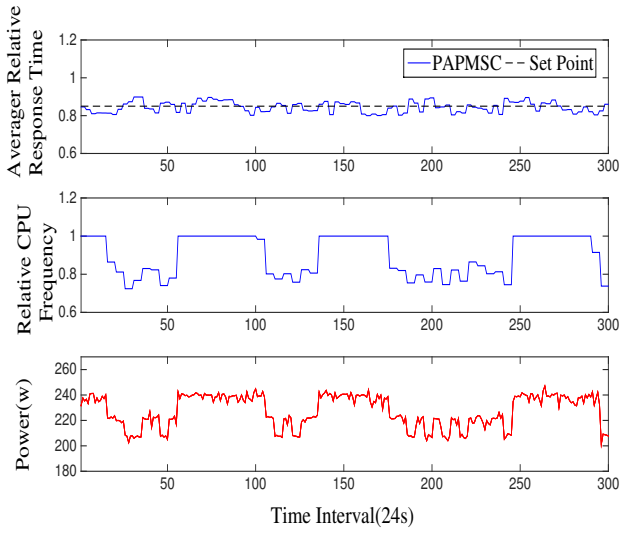


Fig. 10 Performance of PAPMSC under the bursty workload case

In this experiment, we enable the load balancing control and power saving control to examine the power efficiency. We conduct the experiments in the same scenario discussed in the last section and as depicted in Fig.7. Fig.7 and Fig.10 show the results of our PAPMSC. The experiment starts with the physical CPU of the *Server* running at the highest frequency. As discussed before, the load balancing controller works effectively to achieve the same response time for all the VMs. As shown in Fig.10, at the beginning, the power saving controller lowers the relative CPU frequency from 1 to 0.67 for power efficiency, since the response time is initially unnecessarily shorter than the desired set point(i.e.,0.85). Then the power saving controller detects the workload change at the time interval $50th$, and adjusts the relative CPU frequency to 1 so that the system can run faster to meet the desired response time. When the workload of VM2 decrease to the normal level, the power saving controller throttles the relative CPU frequency again to achieve power saving and maintain the desired response time. Throughout this experiment, the average relative response time has been successfully around the set point and kept to be lower than 1, which mean the SLAs (i.e., the desired response time) have been guaranteed. Fig.10 also shows the power consumption of PAPMSC.

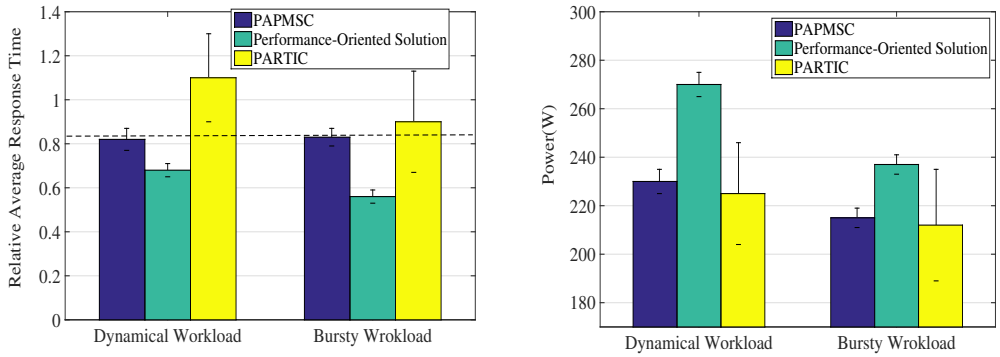
To examine the performance and power efficiency of PAPMSC with different workloads, a set of experiments is conducted by using the performance-oriented solution, the PARTIC solution and the PAPMSC, respectively. The statistical results are given in Fig.11. Performance-oriented solution is a standard Linux baseline policy. For any system, the performance-oriented solution provides no energy saving benefits and reproduces the behavior of a typical commercial data center. In performance-oriented solution, we just use the load balancing controller without the power saving controller, and set the fixed CPU frequency at the highest level. For the performance-oriented solution, it often had unnecessarily short response times in terms of QoS assurance. As a result, the performance-oriented solution had to produce extra power to satisfy an unnecessarily short response time as shown in Fig.11.

For the PARTIC, the power consumption of PARTIC is slightly lower than the PAPMSC, but the PAPMSC achieves the desired relative response time (i.e. 0.85) while the relative average response time of PARTIC is higher than 1, either in the dynamical workload case or the bursty workload case. It demonstrates that PARTIC violates the response time deadline for all VMs. Hence, although the PARTIC has a better power saving effect in both the dynamical workload and robust workload cases, it suffers a heavy price in terms of SLAs assurance. Most important is that the PAPMSC shows the stability and robustness than the PARTIC in terms of QoS assurance and power consumption.

7.5 Scalability Experiment

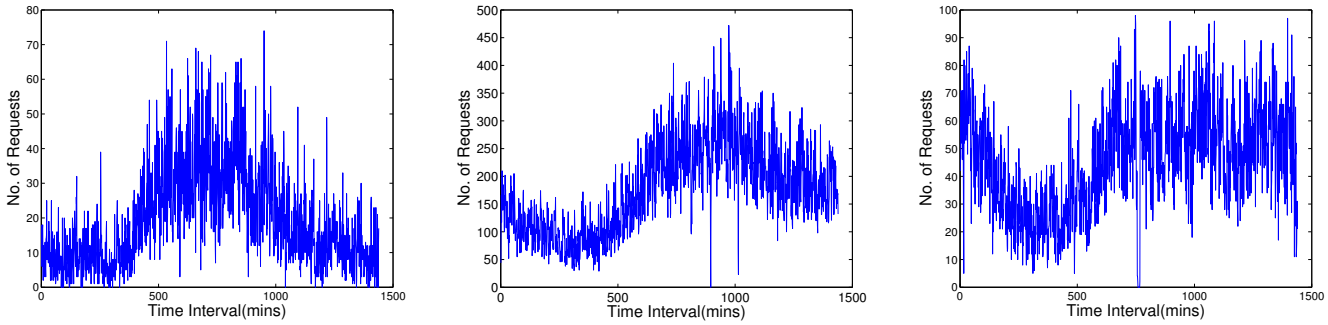
In this section, we disable the power saving controller to evaluate the scalability of load balancing controller (i.e., the combination of cSLQC controller and P controller) running in the testbed2. The *Server* runs 7 web applications on 7 individual virtual machines. The workloads used in our experiment are generated based on real-world Internet traces. The first workload is generated from the WWW server located at the San Diego Supercomputer Center (SDSC) in San Diego, California. It contains a day's worth of all HTTP requests to the SDSC WWW server and 28,228 requests in total. The second workload used in this experiment is obtained from the ClarkNet web server. ClarkNet is a full Internet access provider for the Metro Baltimore-Washington DC area. The trace contains 3,328,587 requests spanning 2 weeks. The third workload has been collected from the NASA Kennedy Space Center WWW server in Florida. This trace contains 3,461,612 requests issued to the server during two months. As shown in Fig.12, one day's traffic from each of these traces is chosen.

The response time target of each application is set to 1000ms and the maximum allowed response time of each application is 1200ms. Fig.13 plots the means and the standard deviations of the response times of the applications in each VM. Specifically, the workload of VM1, VM6 and



(a) Performance (b) Power consumption

Fig. 11 Performance and power analysis under different workloads



(a) Workload-SDSC (b) Workload-Clarknet (c) Workload-NASA

Fig. 12 Workload traces for seven applications

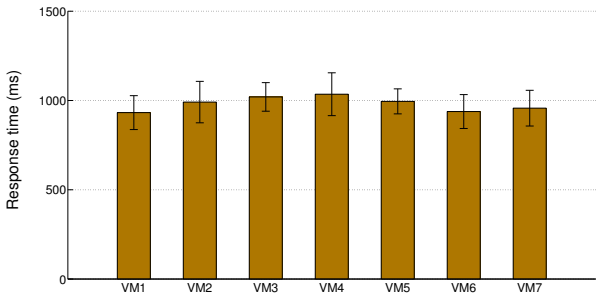


Fig. 13 Performance of load balancing controller in 7 applications

VM7 use the SDSC trace, VM2 and VM5 use the Clarknet trace, and the NASA trace is used as the workload of VM3 and VM4. Fig.13 demonstrates that load balancing controller works effectively to achieve the desired response time for all the applications.

8 Conclusion

In this paper, we presented a software service to achieve performance guarantee with power consumption in Virtualized Servers. As demonstrated by modeling, analysis and experiments on the testbed, the key contribution of the paper is a stochastic control approach that supports robust performance of web applications under high dynamical and bursty workloads, power efficiency and optimal resource utilization. The main technical novelty of this paper is to formulate the limited resource allocation among co-located VMs as a semidefinite optimization problem, and the constraint on SLAs index of web application as a chance constraint in the probabilistic sense. The constraint stochastic linear quadratic control approach is employed to solve this problem. Furthermore, a proportional controller is integrated to enhance performance robustness. In the future, we intend to investigate the use of the proposed control approach to deal with multi-tier web applications on cluster-level servers. In the future, we intend to investigate the use of the state-space model [42–44] and distributed control to deal with multi-tier web applications on datacenter-level servers.

References

1. U.S.Department of Energy, Secretart Chu Announces $47 Million to Improve Efficiency in Information Tech-

nology and Communication Sectors, Available from: http://www.energy.gov/articles/secretary-chu-announces-47-million-improve-efficiency-information-technology-and, (2010)
2. G. Group, Gartner urges it and business leaders to wake up to its energy crisis, http://www.gartner.com/it/page.jsp?id=496819, (2007)
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.,: A view of clouding computing, Communication of the ACM, 53(4), 50-58, (2010)
4. Amazon elastic compute cloud, http://aws.amazone.com/ec2, (2006)
5. Google App Engine, http://appengine.google.com, (2008)
6. Huebscher, M.C., and McCann, J.A.,: A survey of autonomic computing: Degrees, models, and applications, ACM Computing Surveys, 40(3), 7, (2008)
7. Zhuravlev, S., Blagodurov, S., Fedorova, A.,: Addressing shared resource contention in multicore processors via scheduling, ACM SIGARCH Comupter Architecture News, 38(1), 129-142,(2010)
8. Singh, R., Sharma, U., Cecchet, E., and Shenoy,P.,: Autonomic mix-aware provisioning for non-stationary data center workloads, In Proc. of 7th ACM Int'l Conference on Autonomic Computing, pp. 21-30, (2010)
9. Gmach, D., Rolia, J., Cherkasova, L.: Resource and virtualization costs up in the cloud: models and design choices. In Proceedings of 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), pp.395-402 (2011)
10. Lu, C., Lu, Y., Abdelzaher, T.F., Stankovic, J.A., Son, S.H.,: Feedback control architecture and design methodology for service delay guarantees in web servers. IEEE transactions on Parallel and Distributed Systems, 17(9), 1014-1027, (2006)
11. Leite, J. C., Kusic, D. M., Mossé, D., Bertini, L.,: Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In Proceedings of the 7th international conference on Autonomic computing, pp. 41-50, (2010)
12. Wang, Y., Wang, X., Chen, M., Zhu, X.,: PARTIC: power-aware response time control for virtualized web servers, IEEE Trans. on Parallel and Distributed Systems, 22(2), 323-336, (2011)
13. Bohrer, P., Elnozahy, E.N., Keller, T., Kistler, M. , Lefurgy, C., McDowell, C., and Rajamony, R.,:The case for power management in web servers, Power Aware Computing, Springer US, pp. 261-289, (2002)
14. Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., and Gautam, N.,: Managing server energy and operational costs in hosting centers ACM SIGMETRICS Performance Evaluation Review, 33(1), 303-314, (2005)
15. Credit Scheduler, http://wiki.xen.org/wiki/Credit_Scheduler.
16. Niedzwiecki, M.,: Identification of time-varying processes, New York: Wiley, (2000)
17. Hespanha, J.P.,: Linear systems theory, Princeton university press, (2009)
18. Bertsimas, D., and Brown, D.,: Constrained stochastic LQC: A tractable approach, IEEE Trans. on Automatic Control, 52(10), 1826-1841, (2007)
19. Jin, Z., Li, F., Ma, X., and Djouadi, S.M.,: Semi-Definite Programming for Power Output Control in Wind Energy Conversion System, IEEE Trans. on Sustainable Energy, 5(2), 466-575, (2014)
20. Padala, P., Zhu, X., Uysal, M., Wang, Z.,: Automated control of multiple virtualized resources, In Proc. of the 4th ACM European conference on Computer systems, pp. 13-26, (2009)
21. Lama, P., Guo, Y., and Zhou, X.,: Autonomic performance and power control for co-located web applications on virtualized servers, In Proc. of 21th IEEE/ACM Int'l Workshop on Quality of Service, pp.1-10, (2013)
22. Nathuji, R., and Schwan, K.,: VirtualPower: coordinated power management in virtualized enterprise systems, ACM SIGOPS Operating Systems Review, 41(6), 265-278, (2007)
23. Lama, P., and Zhou, X.,: Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee, In Proc. of IEEE Int'l Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication System, pp. 151-160, (2010)
24. Lama, P., and Zhou, X.,: PERFUME: power and performance guarantee with fuzzy MIMO control in virtualized servers, In Proc. of 19th IEEE Int'l Wrokshop on Quality of Service, pp. 1-9, 2(2011)
25. Wang, X., and Wang, Y.,: Coordinating power control and performance management for virtualized server clusters, IEEE Trans. on Parallel and Distributed Systems, 22(2), 2, 245-259, (2011)
26. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., and Jiang, G.,: Power and performance management of virtualized computing environments via lookahead control, Cluster Computing, 12(1), 1-15, (2009)
27. Nathuji, R., England, P., Sharma, P., and Singh, A.,: Feedback driven qos-aware power budgeting for virtualized servers, In 4th Intl. Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID), (2009)
28. Gong, J., and Xu, C.,: vPnP: Automated coordination of power and performance in virtualized datacenters, In Proc. of 18th IEEE Intl. Workshop on Quality of Service, pp. 1-9, (2010)
29. Franklin, G.F., Powell, J.D., and Workman, M.,: Digital Control of Dynamic Systems, 3rd edition, Addition-Wesley, (1997)
30. Electronic Educational Devices, Inc., http://www.wattsupmeters.com, (2010)
31. Wang, Y., and Wang, X.,: Performance-controlled server consolidation for virtualized data centers with multi-tier applications, Sustainable Computing: Information and Systems, 4(1), 52-65, (2014)
32. Wang, X., Du, Z. , Chen, Y., Li, S.,: Virtualization-based autonomic resource management for multi-tier Web applications in shared data center, Journal of Systems and Software, 81(9), 1591-1608, (2008)
33. Kundu, S. , Rangaswami, R., Dutta, K., Zhao, M.,: Application performance modeling in a virtualized environment. In Proc. of 16th International Symposium on High Performance Computer Architecture (HPCA), pp.1-10, (2010)
34. Zhu, Q., Agrawal, G.,: Resource provisioning with budget constraints for adaptive applications in cloud environments, In Proceedings of the 19th International Symposium on High Performance Distributed Computing, pp.304-307,(2010)
35. Urgaonkar, B., Shenoy, P., Chandra, A.,: Agile dynamic provisioning of multi-tier internet applications, ACM Transactions on Autonomous and Adaptive Systems 3(1), 1,(2008)
36. Rao, J., Bu, X., Xu, C., W, L., and Yin, G.,: VCONF: a reinforcement learning approach to virtual machines auto-configuration, In Proc. of 6th Intl.Conf.on Autonomic computing, pp.137-146, (2009)
37. Gandhi, A., Balter, M., Das, R., and Lefurge, C.,: Optimal power allocation in server farms, in Proc. of the 11th conference on Measurement and modeling of computer systems, pp.157-168, (2009)
38. Lama, P., and Zhou, X.,: NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers, In Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks, pp. 1-12, (2011)
39. Beloglazov, A., and Buyya, R.,: Energy efficient resource management in virtualized cloud data centers, In Proc. IEEE/ACM Int'l Conference on Cluster, Cloud and Grid Computing, pp. 826-831, (2010).
40. Arlitt, M., and Jin, T., A workload characterization study of the 1998 world cup web site, IEEE Network, 14(3), 30-37, (2000)
41. Rui, W., Kusic, D.M., Kandasamy, N.,: A Distributed Control Framework for Performance Management of Virtualized Computing Environments, In Proc. of 7th ACM International conference on Autonomic Computing, pp. 89-98, (2010)
42. Ma, X., Li, H., and Djouadi, S. M.,: Stochastic modeling of short-term power consumption for smart grid: A state space approach and real measurement demonstration, In Proc. of 45th Annual Conference on Information Sciences and Systems, pp. 1-5, (2011)

43. Dong, J., Ma, X., Djouadi, S.M., Li, H., and Kuruganti, P.T.,: Real-time prediction of power system frequency in FNET: A state space approach, In Proc. IEEE Int. Conf. Smart Grid Communications, pp. 109-114, (2013)
44. Dong, J., Ma, X., Djouadi, S.M., Li, H., and Liu, Y.,: Frequency Prediction of Power Systems in FNET Based on State-Space Approach and Uncertain Basis Functions, IEEE Transactions on Power System, 29(6), 2602-2612, (2014)
45. Dean, J., and Barroso, L.A.,: The tail at scale. Communications of the ACM, 56(2), 74-80,(2013)
46. Lefurgr, C., Wang, X., and Ware, M.,: Power Capping: A Prelude to Power Shifting. Cluster Computing, 11(2),183-195, (2008)