Self-tuning Batching with DVFS for Improving Performance and Energy Efficiency in Servers

Dazhao Cheng, Yanfei Guo, Xiaobo Zhou
Department of Computer Science
University of Colorado, Colorado Springs, USA
Email addresses: {dcheng, yguo, xzhou}@uccs.edu

***Abstract*—Performance improvement and energy efficiency are two important goals in provisioning Internet services in data center servers. In this paper, we propose and develop a self-tuning request batching mechanism to simultaneously achieve the two correlated goals. The batching mechanism increases the cache hit rate at the front-tier Web server, which provides the opportunity to improve application's performance and energy efficiency of the server system. The core of the batching mechanism is a novel and practical two-layer control system that adaptively adjusts the batching interval and frequency states of CPUs according to the service level agreement and the workload characteristics. The batching control adopts a self-tuning fuzzy model predictive control approach for application performance improvement. The power control dynamically adjusts the frequency of CPUs with DVFS in response to workload fluctuations for energy efficiency. A coordinator between the two control loops achieves the desired performance and energy efficiency. We implement the mechanism in a testbed and experimental results demonstrate that the new approach significantly improves the application's performance in terms of the system throughput and average response time. The results also illustrate it can reduce the energy consumption of the server system by 13% at the same time.**

I. Introduction

There are two main goals for operating a modern data center: performance guarantee of applications with respect to the service level agreement (SLA) for increasing revenue, and energy efficiency of the server system for reducing the operating cost. A well-known approach to controlling energy consumption is to transition a processor from high-power states to low-power states using Dynamic Voltage and Frequency Scaling (DVFS) technique [23]. However, the transition of CPU power states from high to low for energy saving will increase the response time of applications. Therefore, it is important but challenging to reduce both energy consumption and SLA violations at the same time.

In this paper, we propose and develop a self-tuning request batching mechanism: a middleware approach for performance improvement of Internet applications and energy efficiency in a virtualized server system. Today, popular Internet applications employ a multi-tier architecture with each tier depending on its successor and providing functionality to its preceding tier [8], [24]. Front-tier Web servers process requests in the order of arrivals. Indeed, most of those web requests are independent to each other. Batching requests in a content-aware manner can improve the cache hit rate of Web servers, which in return provides the opportunity to improve the performance of multi-tier applications and energy efficiency of underlying server system at the same time. With batching, we employ DVFS power management technique to minimize the energy consumption while meeting the SLA on application average response time.

However, there are two major challenges in developing the batching approach. Firstly, it is a very hard problem to determine the batching interval length. Intuitively, a longer batching interval can accumulate more requests for the same content for batching and reordering. It can increase the cache hit rate and reduce the application response time. But the batching will also delay the processing of those requests, resulting in longer application response time. The risk is the violation of the SLA. The batching approach must be self-tuning in choosing the batching interval length. Unfortunately, due to the complexities of multi-tier Internet service architecture, high dynamics in workloads and the virtualized server infrastructure, obtaining an accurate model among batching interval, performance and power consumption is a very hard problem.

Second, modern processors have a number of CPU frequency states that are tunable by DVFS. How to synchronize DVFS states with dynamically changing batching intervals will have significant impact on the power control effect and system stability. On one hand, the change in system behaviors due to DVFS-enabled control actions has significant impact on the accuracy of batching control. On the other hand, DVFS-enabled power control decisions are dependent on the system parameters that are affected by batching control.

We design a novel yet practical two-layer control system that is composed of a batching control loop and a power control loop. The batching control is based on a self-tuning fuzzy model predictive control (FMPC). FMPC effectively captures a nonlinear relationship between application performance and batching interval length through fuzzy modeling and predictive control. The power control is designed with expert fuzzy control (EFC) to modify the frequency of CPUs. EFC takes advantage of model-independent fuzzy control techniques to address the issue of lacking an accurate performance-power model due to high workload dynamics. It is a real-time online decision maker based on system conditions and historical experiences. According to the fluctuations of the workload and the usage of CPUs, EFC decides when and which frequency state should be set for CPUs. Furthermore, we design a coordinator between the two-layer controls to achieve the desired performance and energy consumption.

We built a testbed of multi-tier server system based on Xen 3.1 virtualization software. The server is running CentOS 5.8 with Linux kernel 2.6.18. The processor supports three frequency levels: 2 GHz, 2.33 GHz, 2.83 GHz. We implemented the batching approach and evaluated it with the RUBiS benchmark application. Experimental results demonstrate that the new approach can improve the application's system throughput by 13%, average response time by 18%, and reduce the energy consumption of the server by 13%.

The main contributions of our work are:

- We propose to use request batching with DVFS under heavy and dynamic workloads for simultaneously improving energy efficiency and application performance of the server system.
- We design and develop a novel yet practical two-layer control system that is composed of a batching interval control loop and a power control loop. The control system is self-tuning, with a coordinator between the two-layer controls for joint performance and power control.
- We have built a testbed and evaluated the new approach with the RUBiS benchmark application. Experimental results demonstrate that the developed batching mechanism can effectively improve both application performance and energy efficiency.

In the following, Section II discusses related work. Section III gives the batching control system architecture and the batching strategy. Section IV presents the modeling, design, and analysis of the batching interval control. Section V gives the power control design. Section VI presents the integration of batching and power controls. Section VII gives the testbed implementation. Section VIII presents the experimental results and analysis. Section IX concludes the paper.

II. Related Work

Power management in computing systems is an important research area [5], [16], [17], [20], [27]. A few early studies proposed to reduce power consumption in Web servers by applying the DVFS technique [3], [21]. DVFS was applied for maximizing the performance of power constrained high-density servers [16] and improving power efficiency of server farms [4]. Those studies focused on single-tier Web systems.

Horvath *et al.* [10] implemented a coordinated DVFS policy for a three-tier Web system based on distributed feedback control and an optimization model that minimizes total power consumption while meeting end-to-end delay deadline. Wang *et al.* [26] proposed a MIMO controller to accurately regulate the total power consumption of an enclosure by conducting processor frequency scaling for each server while optimizing multi-tier application performance.

There are recent studies on coordinated power and performance management in virtualized servers [1], [6], [9], [11], [14], [12], [15], [19], [28]. Mistral [11] is a control architecture for optimizing power consumption, performance benefit, and the transient costs in Cloud environments. vPnP [6] coordinates power and performance in virtualized data centers using utility function optimization. It provides the flexibility to choose various tradeoffs between power and performance. PERFUME [14] is a control system that can meet performance guarantee of multi-tier Internet applications with the power consumption cap of virtualized server. Those studies did not consider using request batching for achieving performance improvement of applications and energy efficiency of servers at the same time.

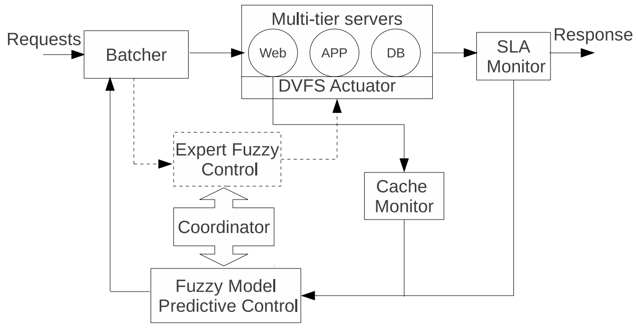


Fig. 1. The System Architecture.

An early study [3] proposed a request batching policy that groups requests during light workloads and executes them in batches, placing the server processor in a low-energy state between batches. In a recent study [27], Wang *et al.* developed a concrete mechanism, virtual batching, which batches requests to a Web server during very light workload scenarios so that the server system can switch between DVFS and the sleep state for energy saving. There are several important issues that need further study. First, the virtual batching approach ignores the SLA on the application average response time. At very light workload scenarios, the batching interval needs to be long enough to accumulate sufficient requests for the CPU state switch that, however, may break the SLA with the application. Second, the batching approach only works in the light workload scenarios. It ignores the request content and thus it does not take advantage of increasing the cache hit rate of Web servers. Third, there is also a practicability problem as there are only few processors that support sleep state modification [5].

In this paper, we tackle the realistic yet challenging problem, that is in heavy and dynamic workload scenarios how to achieve energy efficiency of the servers and to meet the SLA with the applications at the same time.

III. System Architecture and Batching Strategy

A. System Architecture

Figure 1 illustrates the architecture of the batching control system. It is composed of a batching control loop and a power control loop on a virtualized multi-tier server system. The key components in the batching control loop include a fuzzy model predictive control (FMPC), a batcher, a SLA monitor, and a cache hit rate monitor. The control loop relies on FMPC that captures the nonlinear relationship between the application performance and the batching interval length. It outputs the batching interval length based on a dynamic fuzzy model. An

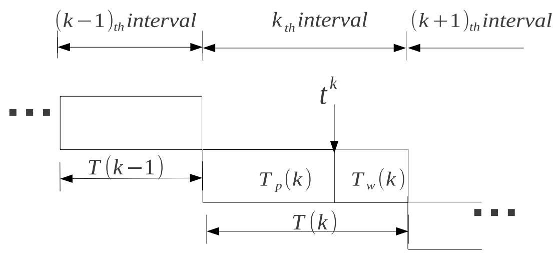


Fig. 2. The batching interval length.

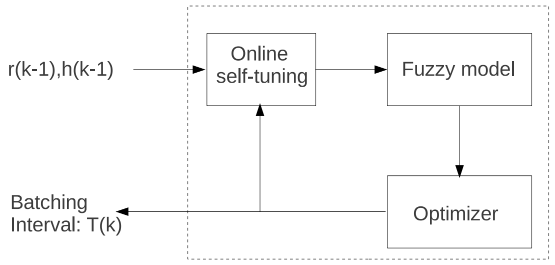


Fig. 3. The design of the fuzzy model predictive control (FMPC).

online self-tuning module is used to update the fuzzy model according to various workloads and CPU frequency changes.

The power control loop is composed of an expert fuzzy control (EFC) and a DVFS actuator. EFC takes advantage of a model-independent fuzzy control technique to address the issue of lacking an accurate performance-power model due to high workload dynamics. It adaptively manages the energy consumption of the server system by manipulating the frequency of CPUs according to the workload fluctuations.

Because of complex interactions between the batching control loop and the DVFS-enabled power control loop, we design a coordinator between the two-layer controls to coordinate application performance improvement and energy efficiency.

B. The Batching Strategy

The request batching approach works in two steps.

1) During each batching interval, requests incoming to the front-tier Web server are classified into groups according to the request content. The request classification is based on its header URL information, which is a good predictor of request content. Note that our work focuses on e-transactional workloads and thus requests are of roughly equivalent processing demand.
2) At the end of each batching interval, the request groups are reordered in a new sequence by the length of each group. The group with the largest number of batched requests will be the first to be sent to the successor application server. The reordering can reduce the average request response time.

The batching interval length changes dynamically. Figure 2 illustrates the process with three continuous intervals.

1) At the end of the $(k-1)_{th}$ batching interval, the requests collected during the interval will be sent to the multi-tier server system. Meanwhile, the k_{th} batching interval starts collecting requests.
2) The multi-tier system completes the processing of the requests collected during the $(k-1)_{th}$ interval at time t^k. Let $T_p(k)$ denote the processing time of those requests. The SLA monitor will measure the average response time of the requests, denoted as $r(k-1)$. The cache monitor will measure the cache hit rate of the requests, denoted as $h(k-1)$.
3) The batching control takes the average response time $r(k-1)$ and the cache hit rate $h(k-1)$ as the inputs. It determines the current batching interval length $T(k)$. After a waiting period of $T_w(k)$, the current batching interval is over and the requests collected during the interval will be sent to the multi-tier server system. At the same time, the $(k+1)_{th}$ batching interval starts.

IV. The Batching Control Design

Due to the SLA requirement and high workload dynamics, the batching interval length should be changed in a self-tuning manner. It is a challenging problem. A longer batching interval can accumulate more requests for the same content, which can increase the cache hit rate and reduce the average response time. However, batching will also delay the processing of those requests, resulting in longer average response time. The SLA requirement might be violated. Unfortunately, there does not exist a linear relationship between the average response time and the batching interval length, due to the high complexities of multi-tier Internet service architecture and high dynamics of workloads.

We propose to use fuzzy model predictive control (FMPC). The rationale is that FMPC can effectively capture nonlinear behaviors through fuzzy modeling and quickly adapt to the workload fluctuations through predictive control for meeting the SLA. The strengths of FMPC include the following:

1) It simplifies nonlinear modeling of a complex system behavior by using a set of linear sub-models captured by the fuzzy rules.
2) It performs optimized control over the entire operating space of a nonlinear problem. The optimization can be achieved for each sampling period based on a sub-model.
3) It inherits several benefits of predictive control such as control accuracy and stability.

Figure 3 illustrates the design of the FMPC-based batching control loop. The inputs are the average response time $r(k-1)$ and the cache hit rate $h(k-1)$ of requests in the $(k-1)_{th}$ batching interval. The output is the length of the next batching interval $T(k)$.

We design a fuzzy model that describes the relationship between the controlled variable and the manipulated variable. In the model, the controlled variable $u(k)$ is the batching interval length $T(k)$, and the manipulated variable $y(k)$ is the average response time $r(k)$. The model is updated every control period with an online self-tuning component. The optimizer is used to find the optimal value of the batching interval length $T(k)$.

A. The Fuzzy Model

We adopt a multiple-input-single-output fuzzy model to describe the complex behaviors of a coupled system. The model is of the input-output NARX type (Nonlinear Auto Regressive model with eXogenous inputs) as follows.

$$y(k+1) = R(u(k), h(k), \xi(k)). \tag{1}$$

R is the relationship between the input variables and the output variable. The input variables are the current input $u(k)$, the variable parameter $h(k)$ and the regression vector $\xi(k)$. The regression vector $\xi(k)$ contains a number of lagged outputs and inputs of the previous control periods. It is represented as

$$\begin{aligned}\xi(k) = [&(y(k), y(k-1), \cdots, y(k-n_y)),\\ &(u(k), u(k-1), \cdots, u(k-n_u))]^T\end{aligned} \tag{2}$$

where n_y and n_u are the number of lagged values for outputs and inputs. Let ρ denote the number of elements in the regression vector $\xi(k)$, that is,

$$\rho = n_y + n_u. \tag{3}$$

R is the rule-based fuzzy model that consists of Takagi-Sugeno rules [2]. A rule R_i is represented as

$$\begin{aligned}R_i : &\text{IF } \xi_1(k) \text{ is } \Omega_{i,1}, \xi_2(k) \text{ is } \Omega_{i,2}, \cdots, \text{ and } \xi_\rho(k) \text{ is } \Omega_{i,\rho}\\ &\quad u(k) \text{ is } \Omega_{i,\rho+1} \text{ and } h(k) \text{ is } \Omega_{i,\rho+2}\\ &\text{THEN } y_i(k+1) = \zeta_i \xi(k) + \eta_i u(k) + \omega_i h(k) + \theta_i.\end{aligned} \tag{4}$$

Here, $h(k)$ is the cache hit rate. Ω_i is the antecedent fuzzy set of the ith rule, which is composed of a series of subsets: $\Omega_{i,1}, \Omega_{i,2}, \cdots, \Omega_{i,\rho+2}$. ζ_i, η_i and ω_i are parameters, and θ_i is the offset. Their values are obtained by offline training.

Each fuzzy rule describes an operating space of the nonlinear system model. The spaces have some overlaps. So each output contains several fuzzy rules. For example, the inputs $u(l), h(l)$ may be included in R_f and R_g at the same time. So the output $y(k+1)$ is computed as the weighted average value by the rules. That is,

$$y(k+1) = \frac{\sum_{i=1}^{K} \gamma_i (\zeta_i \xi(k) + \eta_i u(k) + \omega_i h(k) + \theta_i)}{\sum_{i=1}^{K} \gamma_i}. \tag{5}$$

In Eq. (5), K is the number of rules for the output. γ_i is the degree of fulfillment for the i_{th} rule. The value of γ_i 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 in Eq. (5) is expressed in the form of

$$y(k+1) = \zeta^* \xi(k) + \eta^* u(k) + \omega^* h(k) + \theta^*. \tag{6}$$

The aggregated parameters ζ^*, η^*, ω^* and θ^* are the weighted sum of vectors ζ_i, η_i, ω_i and θ_i respectively.

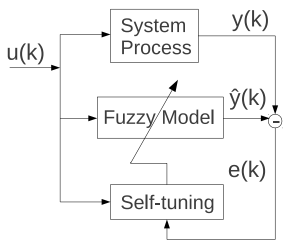


Fig. 4. A self-tuning module of FMPC.

B. Online Self-Tuning

Due to high workload dynamics, we design an online self-tuning module to adapt the fuzzy model. Fig 4 shows the schematic representation of the self-tuning module. It aims to minimize the prediction error of the fuzzy model $e(k)$, $e(k) = y(k) - \hat{y}(k)$. $y(k)$ is the measured output value of the control system and $\hat{y}(k)$ is the model's predicted value for $y(k)$.

The fuzzy model consists of many rules. If $e(k) \neq 0$, we apply a recursive least squares (RLS) method to adapt the parameters of the current fuzzy rule [14]. The technique updates the model parameters as new measurements are sampled from the runtime system. It applies exponentially decaying weights on the sampled data so that higher weights are assigned to more recent observations.

We express the fuzzy model output in Eq. (5) as follow:

$$y(k+1) = \phi(k) X + e(k) \tag{7}$$

where $e(k)$ is the error between the actual output and predicted output. $\phi(k) = [\phi_1^T, \phi_2^T, .., \phi_\rho^T]$ is a vector composed of the model parameters. $X = [\sigma_1 X(k), \sigma_2 X(k), .., \sigma_\rho X(k)]$ where σ_i is the normalized degree of fulfillment or firing strength of i_{th} rule and $X(k) = [\xi(k)^T, u(k)]$ is a vector containing the current and previous outputs and inputs of the control system. The parameter vector $\phi(k)$ is estimated so that the cost function in Eq. (8) is minimized. We apply both the current error $e(k)$ and the previous error $e(k-1)$ to estimate the parameter vector.

$$Cost = \sum_{k-1}^{k} (e(k)^2 + \tau e(k-1)^2). \tag{8}$$

Here τ is a positive number smaller than one. It is called "forgetting factor" as it gives higher weights on more recent samples in the optimization. It determines in what manner the current prediction error and old errors affect the update of parameter estimation. The parameters of the fuzzy model are updated according to the RLS method as follows:

$$\begin{aligned}\phi(k) &= \phi(k-1) + Q(k) X(k-1)[y(k) - X(k-1)\phi(k-1)]\\ Q(k) &= \frac{1}{\tau}[Q(k-1) - \frac{Q(k-1)X(k-1)X^T(k-1)Q(k-1)}{\tau + X^T(k-1)Q(k-1)X(k-1)}]\end{aligned} \tag{9}$$

Here $Q(k)$ is the updating matrix. The initial value of $\phi(0)$ is the value obtained in an offline identification. The initial value of $Q(0)$ is equal to $(X^T X)^{-1}$.

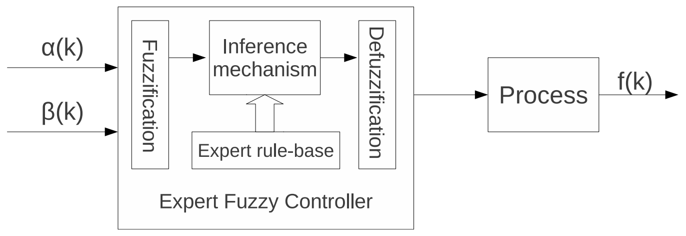


Fig. 5. The expert fuzzy control design.

C. The Optimizer Design

Once the model is established, it is used as a prediction tool by the optimizer to search for the optimal batching length $u(k+1)$. The cost objective function for the optimization is formulated as:

$$J = a \parallel y(k+1) - y_{ref} \parallel^2 + b \parallel \Delta u(k) \parallel^2 \tag{10}$$

$$\text{where } \Delta u(k) = u(k+1) - u(k) \tag{11}$$

The first part in Eq. (10) reflects the predictive error between $y(k+1)$ and y_{ref}. Variable $y(k+1)$ is the predicted average response time of the next batching interval according to the fuzzy model. Parameter y_{ref} is the SLA on the average response time. The second part in Eq. (10) indicates the control effort. The amount of $\Delta u(k)$ is the batching interval adjustment in every control period. Parameters a and b describe the weight of control accuracy and system stability, respectively.

The optimization problem is subject to the constraint that the batching interval length must be bounded by the SLA on the average response time. That is, $\Delta u(k) + u(k) \leq SLA$.

In the presence of a nonlinear model, a nonconvex optimization problem must be solved at each sampling period. The optimization problem is a Quadratic-Programming (QP) problem, which can be effectively solved numerically.

V. Power Control

The multi-tier server system exhibits significant variations in the utilization of CPU resources due to the dynamic behaviors of workloads [18], [24], [25]. This system characteristics can be exploited for CPU power management. Our power control system utilizes DVFS technique to dynamically scale the CPU frequency of the server system in response to time-varying resource demands.

We design a DVFS-enabled power control based on the model-independent expert fuzzy control (EFC) technique to minimize the energy consumption of the server system. The EFC technique allows the DVFS-enabled power control to make online decisions based on server system operations and historical experiences. Its model-independency addresses a practical issue of control design, the lack of an accurate performance-power model in a very complex system.

A. The Control Architecture

Figure 5 shows the architecture of the EFC design. EFC has two inputs, $\alpha(k)$ and $\beta(k)$. Input $\alpha(k)$ reflects the fluctuations in the workload. It is given by Eq. (12), where $V(k)$ is the average system throughput in the k_{th} control period. Input $\beta(k)$ reflects the CPU usage of the server system. It is given by Eq. (13), where $T_p(k)$ and $T_w(k)$ are the process time and waiting time in the batching interval respectively. The output of EFC is the CPU frequency. It has three possible values, i.e., $f(+1)$, $f(0)$, and $f(-1)$, which denote increasing, keeping and decreasing the CPU frequency respectively.

$$\alpha(k) = \frac{V(k) - V(k-1)}{V(k-1)} \tag{12}$$

$$\beta(k) = \frac{T_p(k) - T_w(k)}{T_w(k)} \tag{13}$$

B. The Control Rule

EFC aims to translate a human expert's knowledge into a set of control rules that manage the CPU frequency of the server system. The control rules are defined using linguistic variables corresponding to the two inputs, $\alpha(k)$ and $\beta(k)$. The fuzzification process converts the numeric inputs into linguistic values such as NL(negative large), NM(negative medium), NS(negative small), ZE(zero), PS(positive small), PM(positive medium) and PL(positive large). The rules are in the form of If-Then statements. For example, If $\alpha(k)$ is PL and $\beta(k)$ is PL, the adjustment in CPU frequency is $f(+1)$. The rationale behind this rule is that CPU frequency should increase since the workload and usage of CPUs are growing. Similarly various rules are designed to determine the CPU frequency states based on the fluctuations in the workload and CPU usages. Note that there are three possible output values for the DVFS-enabled power control actions, $f(+1)$, $f(0)$ and $f(-1)$. If a CPU has more than three tunable frequency levels, the rule table will contain more rules and output values for the DVFS-enabled power control actions.

VI. Coordination of Batching and Power Controls

The batching control determines each batching interval length in order to achieve the SLA with respect to the average response time. The power control determines the CPU frequency of the server system in order to reduce energy consumption by the DVFS technique. There are complex interactions between the two control loops. On one hand, the changes in the system behavior due to DVFS-enabled power control actions affect the accuracy of batching control. On the other hand, DVFS-enabled power control decisions are dependent on some parameters that are affected by the batching control. Therefore, we design a coordinator to integrate two controls for system stability. Figure 6 shows the interactions.

A. DVFS Impact on Batching

The change of CPU frequency due to the DVFS technique has a significant impact on the computing capacity of the server system. As the result, the current system model used by the batching control may not be accurate. The batching control has the capability of updating the system model by the self-tuning module at run time. However, it takes a few control

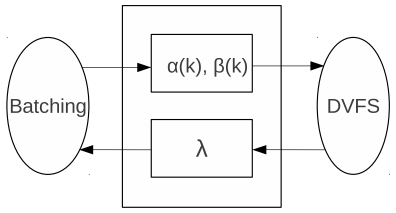


Fig. 6. The coordinator and interactions.

intervals to improve the accuracy of the system model. The coordinator is designed to decrease the impact of the initial modeling inaccuracies with each CPU frequency change. It achieves this goal by using a scaling factor λ to modify the output of the batching control, which is the adjustment in the batching interval length. The scaling factor is heuristically determined according to the difference of computing capacity at different CPU frequency states.

We illustrate the complex interactions between the batching and DVFS actions with an example. Assume that an increase in the workload causes the average response time of the server system to be longer than the SLA target. As a result, the batching control will reduce the batching interval length to reduce the average response time. However, the DVFS may increase the CPU frequency of the server system at the same time. Thus, the adjustment in batching interval length is scaled down by the coordinator to account for the increase in the computing capacity of the server system.

B. Batching Impact on DVFS

The batching control has a significant impact on two inputs of the DVFS-enabled power control. Input $\alpha(k)$, given by Eq. (12), is a function of throughput that is dependent on the batching control loop. Input $\beta(k)$, given by Eq. (13), reflects the change in CPU usage that is affected by the batching interval length. The coordinator transmits two parameters from the batching control to the power control. The power control uses the two parameters as inputs to make the rule table.

A suitable control metric is necessary to synchronize the two controls and achieve system stability. There are several factors that determine the choice of control period such as the time taken by a DVFS action, the scale of batching interval length, etc. In modern processors, a DVFS frequency change takes effect in about 100 ns. In many popular Internet applications, the SLA on the average response time is about 500 ms to 1000 ms [24], [27]. In our work, the power control period is chosen to be four times of the batching control period. Note that it is also possible to determine the ratio between the two control periods in a self-tuning manner according to the system dynamics. Here, we consider a static ratio of four because the time scale of the batching interval length and the DVFS action time are relatively small.

VII. System Implementation

A. Testbed

We built a testbed on a physical server. The server is equipped with one Intel Q9550 quad-core processor and 8 GB memory. The processor supports three frequency levels: 2 GHz, 2.33 GHz, 2.83 GHz. The server is running CentOS 5.8 with Linux kernel 2.6.18.

We use RUBiS benchmark application for experiments as previous studies [8], [14], [24]. RUBiS is an open source multi-tier Internet benchmark application. It provides a web auction application that is modeled in a similar way to *ebay.com*. It characterizes the workload into three categories, browsing, bidding, and selling. The RUBiS user emulates user requests at different concurrent levels. We create one VM for the RUBiS user emulator.

We create three VMs for the RUBiS benchmark application, i.e., Apache web server at the first VM, PHP application server at the second VM, and MYSQL database server at the third VM. Each VM is allocated 1 VCPU and 512 MB memory. All VMs use Ubuntu Server 10.04 with Linux kernel 2.6.35. The $mem.cache$ module and $cache$ module are enabled in the experiments. The Apache server responds to the HTTP requests from clients with a dynamic webpage written in PHP.

Xen 3.1 is used for server virtualization. In Xen, the hypervisor is the lowest layer with the most privileges. When the physical computer and the hypervisor boot, domain 0 (dom0), is the first guest operating system that is booted automatically. Dom0 has some special management privileges such as it can direct resource access requests to the hardware. In our testbed, dom0 is used to start the three VMs. The request batching controller and the DVFS power controller are configured to run as daemons in dom0 along with a response time monitor.

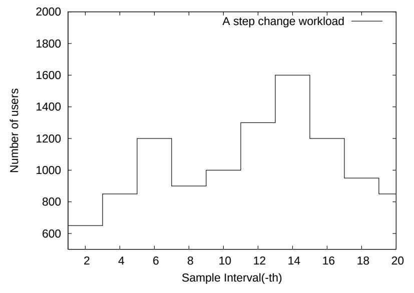
B. System Components

a) Cache Module: We disabled system cache and enabled *mod_mem_cache* of Apache Web server. *MCacheMaxObjectSize* is set to 1MB and *MCacheRemovalAlgorithm* is set to LRU.

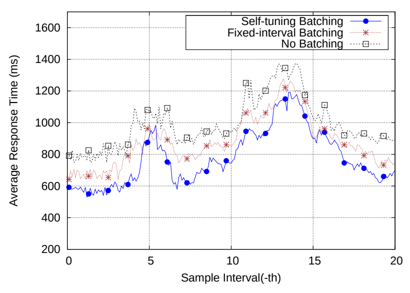
b) Batcher: The self-tuning batcher is a daemon program running with the RUBiS Web server's VM. The batcher consists of several modules including requests batching, classification, reordering and a monitor component to collect parameters for batching interval and power controls. The requests are released to the Web server in a back-to-back manner. In the testbed, the self-tuning batching algorithm takes approximately 10 ms to execute. This overhead is negligible compared to the average batching interval, which is in the range of 300-500 milliseconds.

c) Performance Monitor: We modified the RUBiS user to support online measurement of user-perceived average response time and system throughput. The cache hit rate monitor is implemented as a small daemon program that runs in web tier sever. It periodically collects the information from a log file. The component runs in the RUBiS user emulator's VM.

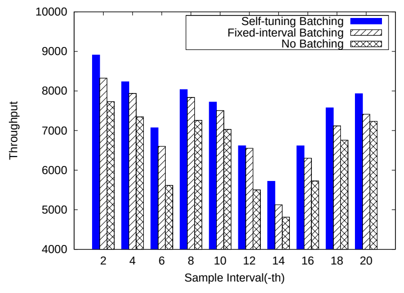
d) FMPC and EFC Controls: These two controls receive the response time, the system throughput and the cache hit rate and other parameters from the performance and power monitors. Accordingly, they run the control algorithms presented in Sections IV and V respectively. The component runs in the RUBiS Web server's VM.



(a) A highly dynamic workload.



(b) Average response time.



(c) Throughput.

Fig. 7. Application performance improvement under the highly dynamic workload.

e) DVFS Actuator: For no interference, we disabled the default DVFS controller in Ubuntu server. We installed the cpufreq package in dom0 to provide the functionality of controlling the CPUs frequency of the physical server. It tunes the CPUs frequency by writing the DVFS states into the system file $/sys/devices/system/cpu/cpu0/cpufreq/scaling.setspeed$.

f) Coordinator: We set the scaling factor λ to be 0.95, which is suitable for the available frequency states of 2 GHz, 2.33 GHz and 2.83 GHz.

VIII. Performance Evaluation

We demonstrate the performance of the self-tuning batching with DVFS scaling approach in terms of both the average response time and the system throughput under highly dynamic workloads. Then we present the improvement on energy efficiency due to the self-tuning batching with DVFS scaling. We further show the effectiveness of the approach under different stationary workloads. Finally, we demonstrate the cache size impact on the performance improvement by the self-tuning batching approach with different request sizes.

We configure the RUBiS users to generate workloads of different mixes as well as workloads of time-varying intensity. As shown in table I, we create three different workload scenarios by adjusting the number of concurrent users.

TABLE I
Performance Improvement of Self-tuning Batching.

Scenarios	Dynamic Workload	Stationary Workload (Underload)	Stationary Workload (Overload)
Concurrent users	[650, 1600]	800	1200

A. Performance Improvement of Self-tuning Batching

Figure 7(a) shows the workload used in the experiment, which is a step-change workload similar to what used in [7], [13], [24]. The number of concurrent users dynamically changes from 650 to 1600. We compare the self-tuning batching approach with an fixed-interval batching approach. The cache hit rate, response time, and throughput are sampled every 30 seconds. As the work in [22], [27], our experiments set the SLA of the system to be 1000 ms. A system without batching is used as the baseline. The cache size of *mod_mem_cache* is set to 20 MB.

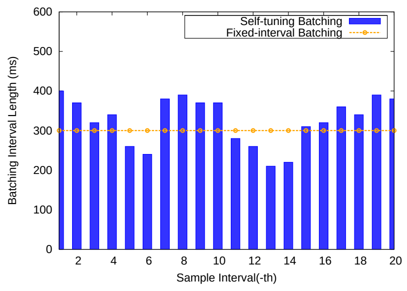
Table II shows the process of the offline training based fixed-interval batching. We executed 20 experiments with different batching interval lengths using the workload trace. The batching interval length varies from 50 ms to 1000 ms. The results in Table II show that 300 ms is the optimal batching interval length that achieves the minimum average response time under the particular workload trace.

TABLE II
Offline Training based Fixed-interval Batching.

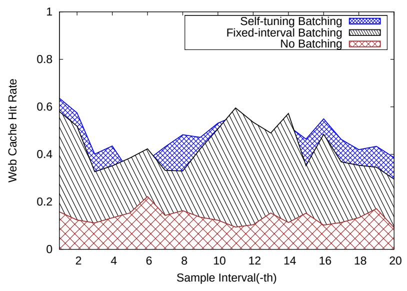
Batching Interval (ms)	Average Resp. Time (ms)	Batching Interval (ms)	Average Resp. Time (ms)
50	993	550	1152
100	975	600	1279
150	1033	650	1335
200	983	700	1476
250	907	750	1533
300	847	800	1764
350	879	850	1930
400	936	900	1985
450	989	950	2257
500	1132	1000	2448

Figure 7(b) shows the comparison of the application's average response time due to the self-tuning batching, the fixed-interval batching (using 300 ms interval length), and the no-batching approach. The self-tuning batching outperforms the fixed-interval batching and no-batching approaches by 7% and 18%, respectively. Figure 7(c) shows the throughput comparison by the three approaches. The self-tuning batching outperforms the fixed-interval batching and no-batching approaches by 6% and 13%, respectively.

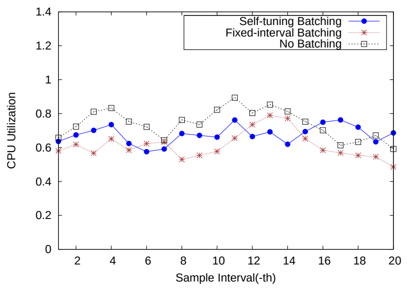
When the workload increases (e.g., the 6_{th} and the 14_{th} sample in Figure 7(a)), the average response time increases and the throughput decreases as shown in Figure 7(b) and 7(c). The reason is that the requests generated by the RUBiS user have temporal dependencies between each other in different sessions. Some requests can only be generated by RUBiS users until the former requests have got feedback from the server system. So when the workload increases, the average response time of the requests increases. Accordingly, the throughput decreases as the RUBiS users generate requests in lower rate.



(a) Self-tuning interval length.

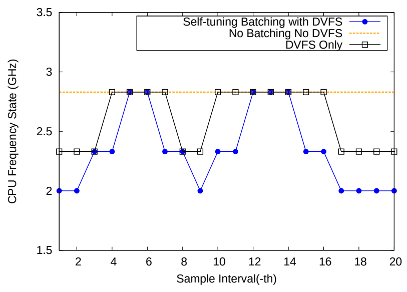


(b) Web cache hit rate.

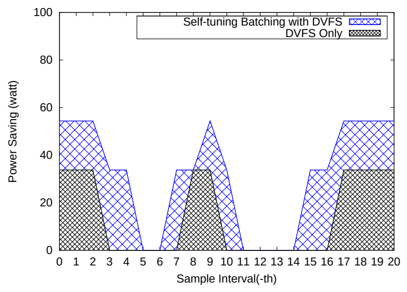


(c) CPU utilization.

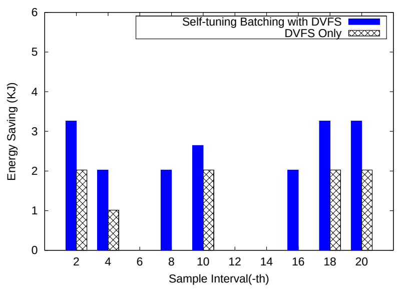
Fig. 8. Self-tuning batching behaviors under the highly dynamic workload.



(a) CPUs frequency adjustments.



(b) Power saving curves.



(c) Energy savings.

Fig. 9. Energy efficiency comparison under the highly dynamic workload.

Figure 8(a) shows how the self-tuning batching interval length varies as the workload dynamics. It shows an opposite trend between the batching interval length and the workload volume. The batching interval length becomes shorter as the workload increases. This is because a long batching interval will increase the average response time when the workload becomes higher. The fixed-interval batching is not able to improve the performance under the highly dynamic workload. In some cases such as the 11_{th} sample, it even cannot maintain the SLA requirement.

Figure 8(b) shows the comparison of the Web cache hit rate in the three scenarios. Both the self-tuning and fixed-interval batching approaches effectively increase the Web cache hit rate. The self-tuning approach does not always outperform the fixed-interval approach in terms of the cache hit rate. For example, between the 11_{th} and the 14_{th} sample intervals, the fixed-interval approach achieves higher Web cache hit rate than the self-tuning approach does. This is due to the fact that the fixed-interval approach uses a longer batching interval length than the self-tuning approach. However, please note that the gain due to the higher cache hit rate does not compensate the increased delay due to the longer batching interval.

Figure 8(c) shows the comparison of CPU utilization by different approaches. It demonstrates that self-tuning batching and fixed-interval batching can effectively reduce CPU utilization. The fixed-interval batching has lower CPU utilization than self-tuning batching since it always has the highest CPU frequency state. No batching approach has the highest CPU utilization due to its lowest cache hit rate.

The results in Table III show that the self-tuning batching approach is able to significantly improve the average response time and the throughput. The reason is it is able to find the favorable batching interval length under the highly dynamic workload due to its batching control design.

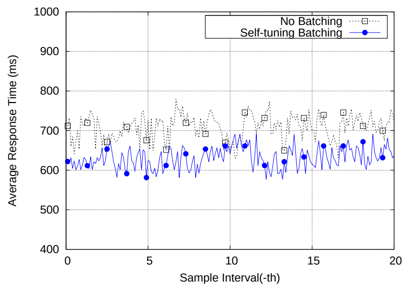
TABLE III
PERFORMANCE IMPROVEMENT OF SELF-TUNING BATCHING.

Batching Strategy	Average Resp. Time	Throughput	Energy Efficiency
Fixed-interval	10%	6%	13%
Self-tuning	18%	13%	13%

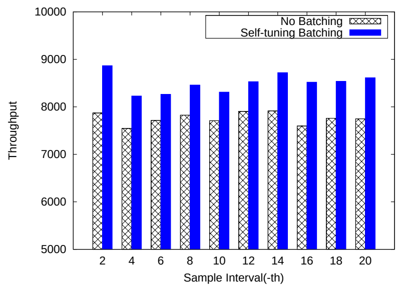
B. Energy Efficiency Improvement of Self-tuning Batching

We next demonstrate the effectiveness of integrating power control with batching control for energy efficiency under the highly dynamic workload. When the workload fluctuates, the power control modifies the frequency of CPUs to follow the fluctuations in a self-tuning manner. The frequency states of CPUs in our experiment start from the lowest level (2 GHz).

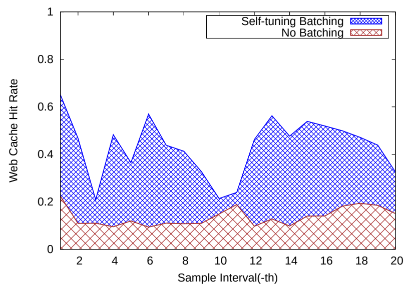
Figure 9(a) shows how the frequency state of the CPU is dynamically adjusted as the workload varies due to two approaches, DVFS only and self-tuning batching with DVFS. Figure 9(b) depicts the power saving of the two approaches compared to the baseline approach that has no batching and no DVFS. The size of the area below a power state curve represents the energy saving of the server. Figure 9(c) shows the energy saving of two approaches in each control interval. Note that in some intervals, i.e., the 6_{th}, 12_{th}, and 14_{th}, there is no energy saving because two approaches both run the CPU at the highest frequency. Overall, compared to the baseline



(a) Response time.

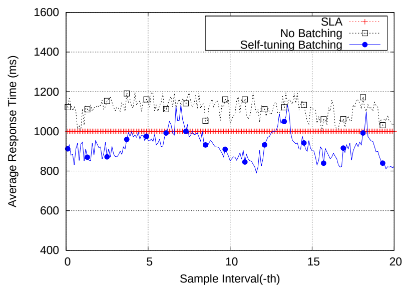


(b) Throughput.

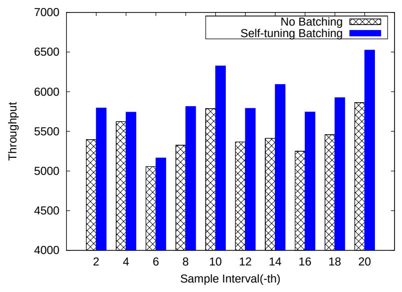


(c) Web cache hit rate.

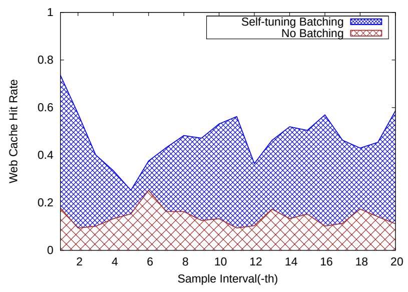
Fig. 10. Performance improvement in an underloaded system.



(a) Response time.



(b) Throughput.



(c) Web cache hit rate.

Fig. 11. Performance improvement of slightly overloaded system.

TABLE IV
PERFORMANCE IMPROVEMENT UNDER STATIONARY WORKLOADS.

Workload Scenarios	Average Resp. Time	Throughput	Energy Efficiency
Underload	15%	14%	7%
Overload	19%	17%	3%

approach, the DVFS only approach saves 7% total energy usage and the self-tuning batching with DVFS approach saves 13% total energy usage. The energy consumption results are estimated based on the configured frequency.

C. *Self-tuning Batching with Stationary Workloads*

We evaluate the average response time and the throughput due to the self-tuning batching in two scenarios under the stationary workloads. The first is an underloaded scenario in which the average response time can satisfy the SLA requirement. The second is a slightly overloaded scenario in which the average response time may violate the SLA requirement. We demonstrate the effectiveness of the batching mechanism by comparing the performance metrics due to the self-tuning batching and no batching.

For the underloaded scenario, we set the RUBiS user with browsing workload mix and 800 concurrent users. It is a moderate workload for our testbed with respect to the SLA.

Figure 10(a) plots the average response time. The self-tuning batching approach improves the average response time by 15%. It significantly reduces the average response time from 712 ms to 619 ms. Figure 10(b) shows the throughput comparison between the self-tuning batching and no-batching approaches. The batching approach improves the throughput by 14%. Figure 10(c) shows that the Web cache hit rate increases by the batching approach. These results demonstrate that the self-tuning batching can significantly improve the application performance for the underloaded scenario.

For the slightly overloaded scenario, we set the RUBiS user with browsing workload mix and 1200 concurrent users.

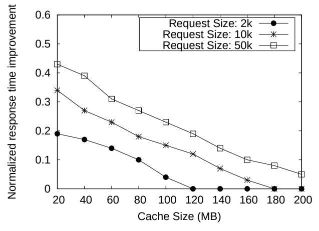
Figure 11(a) plots the average response time. In the no-batching scenario, it is 1091 ms. It indicates the system is slightly overloaded since the SLA on the average response time is 1000 ms. With the self-tuning batching approach, the average response time is significantly reduced to 917 ms. This is 19% improvement compared to that by no-batching.

Figure 11(b) shows that the system throughput due to the self-tuning batching and no-batching approaches. The self-tuning batching improves the throughput by 17%. Figure 11(c) shows that the Web cache hit rate due to the self-tuning batching is 3 times higher than that of no batching.

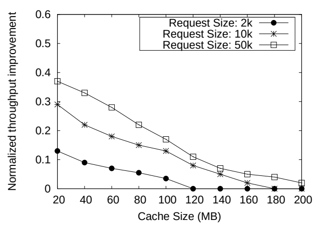
As shown in Table IV, these two experiments demonstrate the effectiveness of the self-tuning batching approach for improving the system performance and server energy efficiency in both the underloaded and the overloaded scenarios.

D. *Cache Size Impact on Performance Improvement*

Figure 12 shows the average response time and throughput improvement due to the self-tuning batching approach with different cache sizes and average request sizes. It illustrates that the performance improvement decreases as the cache size increases from 20 MB to 200 MB. This is because the benefit of self-tuning batching is reduced when the cache is large



(a) Average response time.



(b) Throughput.

Fig. 12. Cache size impact on performance improvement.

enough to hold frequently accessed data items. When the cache size is fixed, the performance improvement increases as the average request size increases from 2 KB to 50 KB. The self-tuning batching can improve the locality of request reference by delaying requests for the same content and reorder them according to their access frequencies. It thus effectively increases the cache hit rate in a self-tuning manner.

IX. CONCLUSION

We proposed and developed a self-tuning batching approach integrated with DVFS to simultaneously improve the performance of applications and energy efficiency of the server system. Its core is a novel and practical two-layer control system that adaptively adjusts the batching interval and frequency states of CPUs according to the service level agreement and the workload characteristics. As demonstrated by experimental results based on the testbed implementation, its main contributions are the precise control of batching interval length to avoid SLA violations and integration of two controls for system stability. The new approach can improve the application's average response time by 18%, system throughput by 13%, and reduce the energy consumption of the server system by 13% at the same time.

In future work, we will extend the proposed approach for heterogeneous workloads in virtualized server clusters.

Acknowledgement

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

REFERENCES

[1] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *Proc. USENIX Operating Systems Design and Implementation (OSDI)*, 2004.
[2] Y. Chen, B. Yang, A. Abraham, and L. Peng. Automatic design of hierarchical takagisugeno type fuzzy systems using evolutionary algorithms. In *IEEE Transactions on Fuzzy Systems*, 2007.
[3] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Proc. the USENIX Symp. on Internet Technologies and Systems (USITS)*, 2003.
[4] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proc. ACM SIGMETRICS*, 2009.
[5] A. Gandhi, M. Harchol-Balter, and M. Kozuch. The case for sleep states in servers. In *the USENIX Workshop on Power Aware Computing and Systems (HotPower)*, 2011.
[6] 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.
[7] Y. Guo, P. Lama, J. Rao, and X. Zhou. V-cache: Towards flexible resource provisioning for multi-tier applications in iaas clouds. In *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2013.
[8] Y. Guo, P. Lama, and X. Zhou. Automated and agile server parameter tuning with learning and control. In *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2012.
[9] Y. Guo and X. Zhou. Coordinated vm resizing and server tuning: Throughput, power efficiency and scalability. In *Proc. IEEE Int'lSymposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 289–297, 2012.
[10] 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.
[11] 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 Conf. on Distributed Computing Systems (ICDCS)*, 2010.
[12] P. Lama, Y. Guo, and X. Zhou. Autonomic performance and power control for co-located web applications on virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 20013.
[13] 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 Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 151–160, 2010.
[14] 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)*, 2011.
[15] P. Lama and X. Zhou. NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers. In *Proc. IEEE/IFIP Int'lConference on Dependable Systems and Networks (DSN)*, pages 1–12, 2012.
[16] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2007.
[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 Conf. on Autonomic computing (ICAC)*, 2010.
[18] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of the EuroSys Conference (EuroSys)*, pages 13–26, 2009.
[19] V. Patil and V. Chaudhary. Rack aware scheduling in hpc data centers: An energy conservation strategy. In *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2011.
[20] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *Proc. ASPLOS*, 2008.
[21] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware qos management in web servers. In *Proc. the IEEE Real-Time Systems Symp. (RTSS)*, 2003.
[22] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proc. of the EuroSys Conference (EuroSys)*, pages 31–44, 2007.
[23] O. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proc. of the IEEE*, 91(7):1–15, 2003.
[24] 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.
[25] H. Wang, Q. Teng, X. Zhong, and P. Sweeney. Using the middle tier to understand cross-tier delay in a multi-tier application. In *Proc. IEEE Int'l Parallel Distributed Processing Symposium (IPDPS)*, 2010.
[26] X. Wang, M. Chen, and X. Fu. Mimo power control for high-density servers in an enclosure. *IEEE Trans. on Parallel and Distributed Systems*, 21(10):1412–1426, 2010.
[27] Y. Wang and X. Wang. Virtual batching: Request batching for server energy conservation in virtualized data centers. *IEEE Trans. on Parallel and Distributed Systems, to appear in 2013, doi: 10.1109/TPDS.2012.237*.
[28] Q. Zhang, F. Mohamed, S. Zhang, Q. Zhu, B. Raouf, and L. Joseph. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*, 2012.