A

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

DAZHAO CHENG, University of Colorado, Colorado Springs, USA
YANFEI GUO, University of Colorado, Colorado Springs, USA
CHANGJUN JIANG, Tongji University, China
XIAOBO ZHOU, University of Colorado, Colorado Springs, USA

Performance improvement and energy efficiency are two important goals in provisioning Internet services in datacenter 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 further extend the self-tuning batching with DVFS approach from a single-server system to a multi-server system. It relies on a MIMO expert fuzzy control to adjust the CPU frequencies of multiple servers and coordinate the frequency states of CPUs at different tiers. We implement the mechanism in a testbed. Experimental results demonstrate that the new approach significantly improves the application performance in terms of the system throughput and average response time. At the same time, the results also illustrate the mechanism can reduce the energy consumption of a single server system by 13% and a multi-server system by 11%, respectively.

Categories and Subject Descriptors: D.4.8 [**Operating Systems**]: Performance-Modeling and Prediction

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Self-tuning Batching, DVFS, Performance Improvement, Energy Efficiency, Fuzzy Model Predictive Control, Internet Applications

1. INTRODUCTION

There are two main goals for operating a modern datacenter: 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 [Addis et al. 2013; Gong and Gu 2010]. A well-known approach to controlling energy consumption is to change a processor from a high-power state to a low-power state using the Dynamic Voltage and Frequency Scaling (DVFS) technique [Unsal and Koren 2003]. However,

This research was supported in part by U.S. NSF CAREER Award CNS-0844983, research grants CNS-1422119, CNS-1320122, CNS-1217979, and NSF of China research grant 61328203.
Authors' address: Dazhao Cheng, Yanfei Guo and Xiaobo Zhou (corresponding author), University of Colorado, Colorado Springs, 1420 Austin Bluffs Pkwy, CO 80918, USA; email: {dcheng, yguo, xzhou}@uccs.edu; Changjun Jiang, Tongji University, 1239 Siping Road, Shanghai, China; email: cjjiang@tongji.edu.cn.

DOI http://doi.acm.org/10.1145/2720023

the transition of CPU power state 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 [Urgaonkar et al. 2008; Lama and Zhou 2013]. Typically, a multi-ter application consists of three tiers; a front-end Web tier that is responsible for HTTP request processing, a middle application tier that implements core application functionality say based on Java Enterprise platform, and a backend database that stores product catalogs and user orders. Front-end Web servers process requests in the order of arrivals. 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 DVFS control benefits from the proposed self-tuning batching mechanism that provides the opportunity to slow down the CPU frequency when the CPU is idle for energy efficiency. On the other hand, dynamic CPU frequency adjustment definitely has significant impact on the computing capability of the server system. It further affects the accuracy of the batching interval determination.

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. We design a coordinator between the two-layer controls to achieve the desired performance and energy consumption.

Furthermore, we extend the self-tuning batching with DVFS approach from a single-server system to a multi-server system. For the multi-server system, we design a power control loop based on the MIMO expert fuzzy control to adjust the CPU frequencies of

servers at each tier. In order to enhance the stability of DVFS control, we further coordinate the CPU states between different tiers to avoid the push-back wave queuing phenomenon in a multi-tier server system [Wang et al. 2013]. We extend the coordinator used in the single server system between the two-layer controls while manipulating the CPU frequencies.

We built a multi-tier system testbed in a single physical server based on virtualization (using Xen 3.1 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 designed self-tuning batching with DVFS approach and evaluated it with the RUBiS benchmark application. Experimental results demonstrate that the new approach can improve the application system throughput by 13%, average response time by 18%, and reduce the energy consumption of the server by 13%.

We further built a multi-tier system testbed in three Dell PowerEdge T110 servers for evaluating the designed self-tuning batching with DVFS approach in a multi-server system with RUBiS benchmark application. We use three frequency levels of the processor: 2.2 GHz, 2.6 GHz, 3.2 GHz. These servers are running CentOS 6.5 with Linux kernel 2.6.32. The experimental results demonstrate that the approach can improve the application system throughput by 21%, average response time by 18%, and reduce the energy consumption of the server system by 11% at the same time.

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 further extend the self-tuning batching with DVFS approach from a single-server system to a multi-server system. It relies on a MIMO expert fuzzy control to adjust the CPU frequencies at different tiers and coordinate the CPU state of the multiple physical servers.
— We have built a testbed and evaluated the new approach with the RUBiS benchmark application. Experimental results demonstrate that the designed batching approach can effectively improve both application performance and energy efficiency.

A preliminary version of this paper appeared in [Cheng et al. 2013]. In this manuscript, we have extended the system design from a single-server system to a multi-server system. We have carried out new experiments and analysis with the extended approach, and studied the impact of different coordination parameters on system performance. Furthermore, we have conducted new experiments to evaluate the performance of the approach based on the proposed fuzzy controller and based on a classic PI controller. We have also analyzed the stability and overhead of the control system.

In the following, Section 2 discusses related work. Section 3 describes the batching control system architecture and the batching strategy. Section 4 presents the modeling, design, and analysis of the batching interval control. Section 5 gives the power control design for a single-server system. Section 6 presents the integration of batching and power controls in a single-server system. Section 8 describes the power control design and its integration with the batching control for a multi-server system. Section 8 gives the testbed implementation. Sections 9 and 10 present the experimental results and analysis for the single-server and multiple-server scenarios, respectively. Section 11 concludes the paper with remarks on future work.

2. RELATED WORK

Power management in computing systems is an important research area. Web server power management utilizes techniques including DVFS, system shut-down, and consolidation. A few early studies proposed to reduce power consumption in Web servers by applying the DVFS technique [Elnozahy et al. 2003; Sharma et al. 2003]. DVFS was applied for maximizing the performance of power constrained high-density servers [Lefurgy et al. 2007] and improving power efficiency of server farms. Those studies focused on single-tier Web systems.

Horvath *et al.* [Horvath et al. 2007] 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.* [Wang et al. 2010] 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. virtualPower [Nathuji et al. 2007] provides coordinated power management in virtualized enterprise systems. pMapper [Verma et al. 2008] tackles power-cost tradeoffs under a fixed performance constraint. vManage [Kumar et al. 2009] performs virtual server placement to save power without degrading performance. Mistral [Jung et al. 2010] is a control architecture for optimizing power consumption, performance benefit, and the transient costs in Cloud environments. SHIP [Wang et al. 2012] is a scalable hierarchical power control architecture for large-scale datacenters. vPnP [Gong and Xu 2010] coordinates power and performance in virtualized datacenters using utility function optimization. It provides the flexibility to choose various tradeoffs between power and performance. PERFUME [Lama and Zhou 2011] is a control system that can meet performance guarantee of multi-tier Internet applications with the power consumption cap of virtualized server. Its follow-up work APPLEware [Lama et al. 2013] is an autonomic and scalable middleware for joint performance and power control of multi-service applications in virtualized datacenters, which features a distributed control structure that provides predictable performance and energy efficiency for large complex systems. NINEPIN [Lama and Zhou 2012] is a non-invasive energy-efficient performance isolation approach for virtualized servers. Those studies did not consider using request batching nor DVFS for achieving performance improvement of multi-tier applications and energy efficiency of servers.

An early study [Elnozahy et al. 2003] 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 [Wang and Wang 2013], 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. In 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 [Gandhi et al. 2011].

Recently, analytical models for estimating the impact of DVFS on multi-tier system performance have been formulated in studies [Wang et al. 2013; Hagimont et al. 2013]. Hagimont *et al.* proposed a power-aware scheduler that addresses the compat-

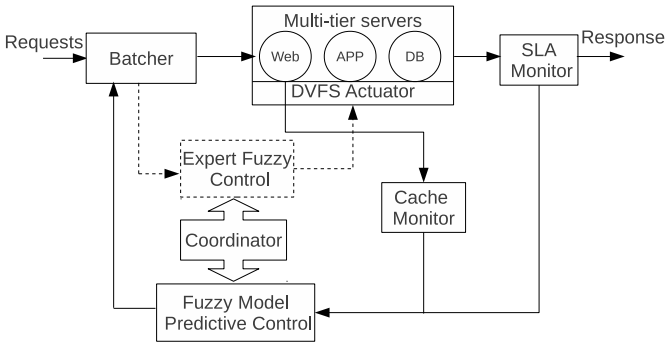


Fig. 1. The System Architecture.

ibility of available virtual machine (VM) schedulers with DVFS management in virtualized environments [Hagimont et al. 2013]. A credit is associated with a VM at its creation time and represents its allocated computing capacity. The proposed scheduler was implemented in the Xen hypervisor and evaluated through different scenarios which demonstrated its advantage over the Credit and SEDF schedulers. The model can get good prediction accuracy when the system is lightly loaded, but not for the system in high utilization. Wang *et al.* investigated the impact of DVFS on multi-tier application performance [Wang et al. 2013]. It reveals two problems: large response time fluctuations due to push-back wave queuing in n-tier systems and throughput loss due to rapidly alternating bottlenecks. The problems arise from anti-synchrony between DVFS adjustment period and workload burst cycles. The experiment demonstrates that a workload-sensitive DVFS adaptive control mechanism can disrupt the anti-synchrony and reduce the performance impact of DVFS at high utilization levels.

In this paper, we tackle a realistic yet challenging problem, that is in heavy and dynamic workload scenarios how to achieve energy efficiency of Internet servers and to meet the SLA with multi-tier application at the same time, by designing and developing a self-tuning batching with DVFS approach.

3. SYSTEM ARCHITECTURE AND BATCHING STRATEGY

3.1. 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 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.

Table I. Notation Summary.

Symbol	Description
k	Index of the batching sampling interval
$T(k)$	The length of the k_{th} batching interval
$T_p(k)$	The request processing time in the k_{th} interval
$T_w(k)$	The waiting time in the k_{th} interval
$r(k)$	The average response time in the k_{th} interval
$h(k)$	The average cache hit rate in the k_{th} interval
$y(k)$	The manipulated variable of the average response time $r(k)$
$u(k)$	The controlled variable of the batching interval length $T(k)$
$\Delta u(k)$	The batching interval adjustment in each control period
$e(k)$	The prediction error of the fuzzy model
$\Delta e(k)$	Change in error for the performance prediction
α	The parameter reflects the fluctuation of workload
β	The parameter reflects the CPU usage of the server system
$V(k)$	The average system throughput in the k_{th} interval

3.2. 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.

Sections 4 through 8 describe the system design. Table I summarizes the key symbols used in the design description.

4. 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

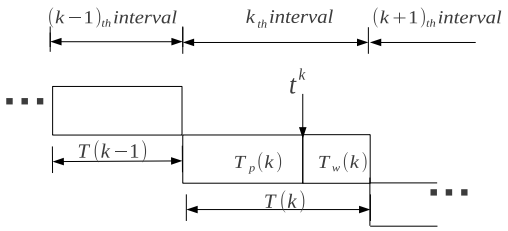


Fig. 2. The batching interval length.

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 $T(k)$.

4.1. 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

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

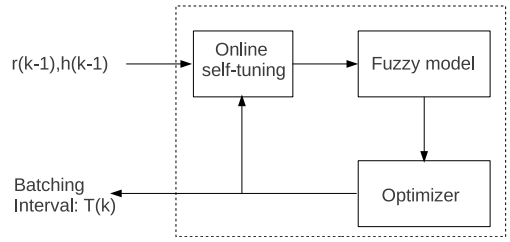


Fig. 3. The design of the fuzzy model predictive control (FMPC).

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 [Chen et al. 2007]. A rule R_i is represented as

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

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

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

Eq. (7) gives the aggregated parameters ζ^*, η^*, ω^* and θ^* that are the weighted sum of vectors ζ_i, η_i, ω_i and θ_i respectively.

$$\begin{aligned} \zeta^* &= \frac{\sum_{i=1}^{K} \gamma_i \zeta_i}{\sum_{i=1}^{K} \gamma_i} \\ \eta^* &= \frac{\sum_{i=1}^{K} \gamma_i \eta_i}{\sum_{i=1}^{K} \gamma_i} \\ \omega^* &= \frac{\sum_{i=1}^{K} \gamma_i \omega_i}{\sum_{i=1}^{K} \gamma_i} \\ \theta^* &= \frac{\sum_{i=1}^{K} \gamma_i \theta_i}{\sum_{i=1}^{K} \gamma_i} \end{aligned} \tag{7}$$

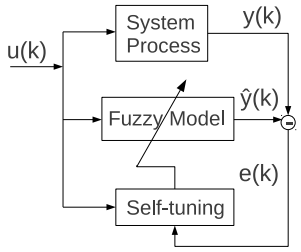


Fig. 4. A self-tuning module of FMPC.

4.2. 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. 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:

$$y(k+1) = \phi(k)X + e(k) \tag{8}$$

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. (9) is minimized. We apply both the current error $e(k)$ and the previous error $e(k-1)$ to estimate the parameter vector.

$$Cost = \sum_{k-1}^{k} (e(k)^2 + \tau e(k-1)^2). \tag{9}$$

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:

$$\begin{aligned} \phi(k) &= \phi(k-1) + Q(k)X(k-1)[y(k) - X(k-1)\phi(k-1)] \\ Q(k) &= \frac{1}{\tau}[Q(k-1) - \frac{Q(k-1)X(k-1)X^T(k-1)Q(k-1)}{\tau + X^T(k-1)Q(k-1)X(k-1)}] \end{aligned} \tag{10}$$

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 $(X^T X)^{-1}$.

4.3. 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:

$$J = a \parallel y(k+1) - y_{ref} \parallel^2 + b \parallel \Delta u(k) \parallel^2 \tag{11}$$

$$\text{where } \Delta u(k) = u(k+1) - u(k) \tag{12}$$

The first part in Eq. (11) 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. (11) 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.

We linearize the fuzzy model and represent it as a state-space linear time variant model in the following form:

$$\begin{aligned} x_{lin}(k+1) &= A(k)x_{lin}(k) + B_u(k)u(k) + B_h(k)h(k). \\ y_{lin}(k) &= C(k)x_{lin}(k). \end{aligned} \tag{13}$$

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

$$x_{lin}(k+1) = [\xi^T(k), 1]^T \tag{14}$$

The matrices $A(k)$, $B_u(k)$, $B_h(k)$ and $C(k)$ are constructed by freezing the parameters of the fuzzy model at current operating point $y(k)$ and $u(k)$. We calculate the degree of fulfillment γ_i and compute the aggregated parameters ζ^*, η^*, ω^* and θ^*. When comparing Eq. (6) and Eq. (13), the state matrices are computed as follows:

$$A = \begin{bmatrix} \zeta_1^* & \zeta_2^* & \cdots & \zeta_j^* & \cdots & \zeta_\rho^* & \theta^* \\ 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{15}$$

$$B_u = \begin{bmatrix} \eta^* \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{16}$$

$$B_h = \begin{bmatrix} \omega^* \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{17}$$

$$C = [1 \ 0 \ 0 \ \cdots] \tag{18}$$

where ζ_j^* $(1 \leq j \leq \rho)$ is the j_{th} element of aggregate parameter vectors ζ^*.

To ensure offset-free reference tracking, the optimization problem is defined with respect to the increment in the control signal, $\Delta u(k)$, rather than the control signal

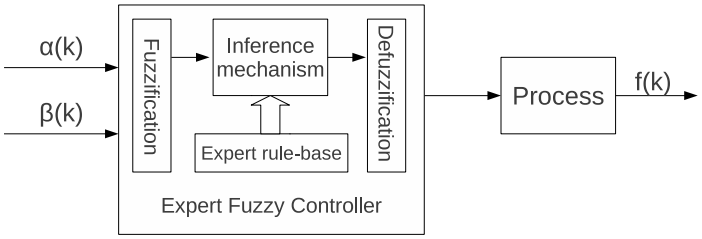


Fig. 5. The expert fuzzy control design.

$u(k)$. The statespace description is extended correspondingly as follows:

$$\begin{aligned} x(k+1) &= \bar{A}(k)x(k) + \bar{B}_u(k)\Delta u(k) + \bar{B}_h(k)h(k). \\ y(k) &= \bar{C}(k)x(k). \end{aligned} \tag{19}$$

$$\begin{aligned} \bar{A}(k) &= \begin{bmatrix} A(k) & B_u(k) \\ 0 & I \end{bmatrix} \\ \bar{B}_u(k) &= \begin{bmatrix} B_u(k) \\ I \end{bmatrix} \\ \bar{B}_h(k) &= \begin{bmatrix} B_h(k) \\ I \end{bmatrix} \\ \bar{C}(k) &= [C(k) \;\; 0] \end{aligned} \tag{20}$$

In the k_{th} control interval, the state vector Eq. (19) is available. The response time and the cache hit rate of previous control intervals are also available from their performance monitors. The current batching interval length $u(k)$ is calculated through successive substitutions, as shown in Eq. (20).

5. POWER CONTROL

A multi-tier server system often exhibits significant variations in the utilization of CPU resources due to the dynamic behaviors of workloads [Padala et al. 2009; Urgaonkar et al. 2008; Wang et al. 2010]. 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.

5.1. 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. (21), 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. (22), where $T_p(k)$ and $T_w(k)$ are the processing 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 $f(-1)$,

α(k) \ β(k)	NL	NM	NS	ZE	PS	PM	PL
NL	f(-1)	f(-1)	f(0)	f(0)	f(0)	f(+1)	f(+1)
NM	f(-1)	f(-1)	f(0)	f(0)	f(0)	f(+1)	f(+1)
NS	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)
ZE	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)
PS	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)	f(0)
PM	f(+1)	f(+1)	f(0)	f(0)	f(0)	f(+1)	f(+1)
PL	f(+1)	f(+1)	f(0)	f(0)	f(0)	f(+1)	f(+1)

Fig. 6. The rule table.

which denote increasing, keeping and decreasing the CPU frequency respectively.

$$\alpha(k) = \frac{V(k) - V(k-1)}{V(k-1)} \tag{21}$$

$$\beta(k) = \frac{T_p(k) - T_w(k)}{T_w(k)} \tag{22}$$

5.2. 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. Figure 6 gives the rule base. 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.

6. 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 7 shows the interactions.

6.1. 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

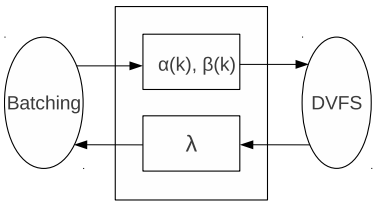


Fig. 7. The coordinator and interactions.

takes a few control 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.

6.2. 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. (21), is a function of throughput that is dependent on the batching control loop. Input $\beta(k)$, given by Eq. (22), 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 [Urgaonkar et al. 2008; Wang and Wang 2013]. 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.

6.3. Stability Analysis of Fuzzy Control Systems

We analyze the stability of the two proposed fuzzy controls based on LaSalles invariant set principle, i.e., global invariant set theorem [Wang et al. 1996]. The premise of the stability criterion is that, if the control output of each rule fulfils same conditions, the overall system will be stable. We focus on the formulation and proof that ensures sufficient conditions for the stability of nonlinear processes controlled by Takagi-Sugeno rule based fuzzy model.

The Lyapunov function candidate $\Gamma : R^n \rightarrow R, \Gamma(x) = x^T P x$ is considered. It is positive and unbounded, where $P \in R^{n \times n}$ is a positive definite matrix. Considering the

state trajectories fulfilling Eq. 13 in order to obtain the closed-loop system dynamics, it results that Γ has continuous partial derivatives and the derivatives of Γ with respect to time are expressed in terms of:

$$\dot{\Gamma}(x) = \dot{x}^T Px + x^T P\dot{x} = F(x) + B(x)u(x), \tag{23}$$

where $F(x) = b(x)^T Px + x^T Pb(x)$. Then the following sets are defined to be used in the stability analysis:

$$B^0 = x \in X \mid B(x) = 0, B^+ = x \in X \mid B(x) > 0, B^- = x \in X \mid B(x) < 0. \tag{24}$$

By the definition of Γ it results that $\Gamma(0) = 0$, $\Gamma(x) > 0, \forall x \neq 0$ and $\Gamma(x) = x^T Px \to \infty$ as $\parallel x \parallel \to \infty$. Further on, it has been proved that $\dot{\Gamma}$ is negative semi-definite with respect to time employing Eq. 23. An arbitrary initial state vector $x_0 \in X$ is accepted. By analyzing the three possible cases it is obtained that: $\dot{\Gamma}(x) < 0, \forall x \in X$. In conclusion, the derivative with respect to time of the Lyapunov function candidate, $\dot{\Gamma}$, is negative semi-definite. It ensures sufficient conditions for the stability of the fuzzy logic control system. So it is proved that the self-tuning batching control system based on the Takagi-Sugeno fuzzy model will be globally asymptotically stable in the sense of Lyapunov.

Furthermore, the Fuzzy Logic Controller (FLC) of self-tuning batching consists of r fuzzy rules. The i_{th} IF-THEN rule in the fuzzy rule base of the FLC, referred to as Takagi-Sugeno fuzzy rule, has the following expression:

$$\begin{aligned} Rule - i : \textbf{IF } & x_i is \tilde{X_{i,1}}, and \ldots, and, x_n is \tilde{X_{i,n}} \\ & \textbf{THEN } u = u_i(x), i = \bar{1,r}, r \in N, r \geq 2, \end{aligned} \tag{25}$$

where $X_{i,1}, X_{i,2} \ldots X_{i,n}$ are fuzzy sets that describe the linguistics terms of input variables, and $u = u_i(x)$ is the control output of rule i. u_i can be a single value or a function of states vector. Each fuzzy rule generates a firing degree $\varepsilon_i \in [0,1], i = 1, 2, \ldots, r$, according to:

$$\varepsilon_i(x) = min(\mu_{X_{i,1}}(x_1), \mu_{X_{i,2}}(x_2), \ldots, \mu_{X_{i,n}}(x_n)). \tag{26}$$

Thus, for any x belonging to the input universe of discourse X, there exists at least one ε_i among all rules that is not equal to zero. In other words, the change of fuzzy rules does not affect the stability of the batching fuzzy control system.

We finally analyze the stability of the power control loop by transferring it to a standard Takagi-Sugeno FLC system which has been demonstrated the stability above. The FLC system of DVFS adjustment consists of a nonlinear process and a fuzzy logic controller as described in Section 5 and Figure 6 respectively. The rules in the Figure 6 can be transferred to the Takagi-Sugeno rules, e.x., IF $\alpha(k)$ is NL and $\beta(k)$ is ZE, THEN the output is $f(0)$. Thus the fuzzy system of the power control is globally asymptotically stable as demonstrated in the batching control system. In summary, we demonstrate that the whole system consisting of the two proposed fuzzy controls is globally asymptotically stable in the sense of Lyapunov.

7. POWER CONTROL IN A MULTI-SERVER SYSTEM

Furthermore, we extend the proposed self-tuning batching with DVFS approach to a multi-tier application architecture built on multiple physical servers. We consider that one tier runs in one physical server. This is due to the fact that the different tiers of an application usually are distributed across different servers in the real system (for load sharing a tier can also be replicated and clustered) [Wang et al. 2013; Lama and Zhou

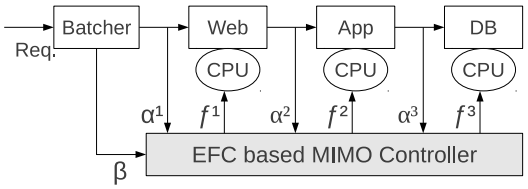


Fig. 8. The power control in multi-server system.

2013; Urgaonkar et al. 2008]. In this section, we describe the power control architecture, the control rules, and the coordination between different tiers while applying the self-tuning batching with DVFS in a multi-server system.

7.1. The Control Architecture

Figure 8 shows the power control architecture of the proposed batching approach when it is applied in a multi-server system. The power control loop is based on the MIMO expert fuzzy control and the DVFS actuator of the physical server at each tier. It adaptively manages the energy consumption of the individual servers by manipulating the frequency of CPUs according to the workload fluctuations at each tier in the system. In a multi-server system, each tier receives partially processed requests from the previous tier and feeds these requests into the next tier after local processing. Because of complex interactions between these DVFS-enabled power control loops at different tiers, we consider the coordination control between the web, application and database DVFS controls to improve application performance and energy efficiency at the overall system level.

Figure 8 shows the architecture of the expert fuzzy control based MIMO power control design. It has four inputs, $\alpha^1(k)$, $\alpha^2(k)$, $\alpha^3(k)$ and $\beta(k)$. Inputs $\alpha^1(k)$, $\alpha^2(k)$ and $\alpha^3(k)$ reflect the fluctuations of the workload at web, application and database tier, respectively. They are given by Eq. (27), where $V^1(k)$, $V^2(k)$ and $V^3(k)$ are the average system throughput at Web, application and database tier in the k_{th} control period, respectively. Recall that input $\beta(k)$ reflects the CPU usage of the server system that is given by Eq. (22). The outputs of MIMO control are the CPU frequencies of the servers, i.e., $f^1_{Web}(k)$, $f^2_{App}(k)$ and $f^3_{DB}(k)$. They have three possible values, i.e., $f(+1)$, $f(0)$, and $f(-1)$, which denote increasing, keeping and decreasing the CPU frequency, respectively.

$$\alpha^x(k) = \frac{V^x(k) - V^x(k-1)}{V^x(k-1)}, x \in \{1, 2, 3\}. \tag{27}$$

7.2. The Control Rules

We employ the DVFS control of a single server designed in Section 5 at each server of the multi-server system. This enables CPU frequency management decisions to be made independently at each server based on local observations. Similarly with the single server scenario, the MIMO control also aims to translate a human expert's knowledge into a set of control rules that manage the CPU frequency of individual servers in a multi-server system. The control rules are defined using linguistic variables corresponding to the two inputs for each tier, $\alpha^x(k)$ and $\beta(k)$. Then various rules for each single server are designed to determine the CPU frequency states based on the fluctuations in the workload and CPU usages at this server. As shown in Table II, the CPU frequency of Web tier server $f^1_{Web}(k)$, application tier server $f^2_{App}(k)$ and database tier server $f^3_{DB}(k)$ are obtained based on the given value of $\alpha^x(k)$ and $\beta(k)$ at each tier. The

Table II. The variables of DVFS control rule for the multi-server system.

inputs	$\alpha^1(k)$	$\alpha^2(k)$	$\alpha^3(k)$
$\beta(k)$	$f^1_{Web}(k)$	$f^2_{App}(k)$	$f^3_{DB}(k)$

details of the control rule at each tier is similar to those of the single server system (described in Section 5).

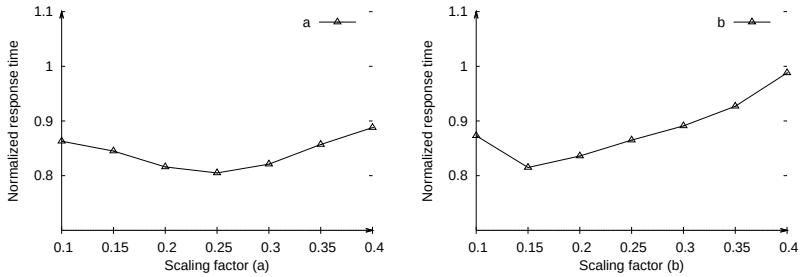
7.3. Coordination of Controls

Coordination between the batching control loop and DVFS control loop. As demonstrated in Section 6, there are complex interactions between the batching control loop and the power control loop. In a multi-server system, such interactions increase quickly when the CPU frequencies are modified frequently or the workload fluctuates significantly. To reduce the impact of interactions on system performance, we extend the designed coordinator for a single-server system to a multi-server system. We apply a scaling factor weight vector $\lambda = [\lambda_1, \lambda_2, \lambda_3]$ in a multi-server system instead of the scaling factor λ in the single-server system. λ_1, λ_2 and λ_3 represent the modeling accuracy impact caused by CPU frequency modification at Web, application and database tier, respectively. The parameters used in the experiments are empirically obtained.

Coordination among servers. Previous studies [Wang et al. 2013; Diao et al. 2006] illustrated there are complex inter-dependencies among different tiers in a multi-tier system due to the resource dependencies. Our experimental observation reveals that DVFS has significant impact on multi-tier application performance, especially at high resource utilization. Study [Wang et al. 2013] also demonstrated that large response time fluctuations happen due to push-back wave queuing for database and application servers in the multi-tier system. We found that the application performance can deteriorate by up to 20% due to rapidly alternating CPU frequency between database and application servers. To address the issue, we coordinate DVFS operations of different tiers while manipulating their CPU frequencies. In this work, we focus on coordinating the CPU frequency between database and application servers due to their prominent push-back wave queuing feature. It mainly relies on the following two rules to eliminate or mitigate performance interference between different tiers.

— When the CPU frequency of application tier server is to be increased, the CPU frequency of the database tier server will correspondingly increase so as to mitigate the push-back wave queuing delay.
— When the CPU frequency of the database tier server is to be reduced, the CPU frequency of the application tier server will correspondingly reduce due to the same rationale as the rule above.

Quantifying CPU frequency modification impact. In order to effectively mitigate the push-back wave queuing feature in a multi-server system, we quantify such CPU frequency modification impact between the application server and the database server. We further explore the performance impact of various scaling factors while applying the proposed coordination technique to avoid the push-back wave queuing feature. We formulate the two rules above as follows: (1) if the CPU frequency of the application server is increased, the CPU frequency of the database server increases as $f_{DB}(k) = a \times \Delta f_{App}(k)$; (2) if the CPU frequency of the database server is decreased, the CPU frequency of the application server decreases as $f_{App}(k) = b \times \Delta f_{DB}(k)$. Here, $\Delta f_{App}(k)$ and $\Delta f_{DB}(k)$ represent the CPU frequency modifications by the control system for the application and database servers, respectively. Parameters a and b repre-



(a) Scaling factor impact by a. (b) Scaling factor impact by b.

Fig. 9. Scaling factor impact on mitigating the push-back wave queuing feature.

sent the scalable factor of the application and database servers, respectively. Then, we empirically obtain suitable scaling factors to coordinate the CPU frequencies of the application and database servers. As shown in Figure 9, we find that the values of a and b should be set to 0.25 and 0.15 respectively in our testbed experiments.

8. SYSTEM IMPLEMENTATION

8.1. Testbeds

The Single-Server Testbed. We built the 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.

As previous studies [Guo et al. 2012; Lama and Zhou 2011; Urgaonkar et al. 2008], we use RUBiS benchmark application for experiments. 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 in the first VM, PHP application server in the second VM, and MYSQL database server in the third VM. Each VM is allocated with 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 a special management privilege that 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.

The Multi-Server Testbed. We built the multi-server system testbed on three Dell PowerEdge T110 physical servers. Each server is equipped with an one-way Intel Xeon E3-1230 quad-core CPU and 16 GB memory. The servers are connected with Gigabit Ethernet. We use three frequency levels of the processor: 2.2 GHz, 2.6 GHz, 3.2 GHz. Each server runs CentOS 6.5 with Linux kernel 2.6.32. We use these three machines for the RUBiS benchmark application, i.e., Apache web server in the first machine, PHP application server in the second machine, and MYSQL database server in the third machine.

8.2. System Components

Batcher. The self-tuning batcher is a daemon program running within the RUBiS Web server. The batcher consists of several modules including requests batching, classification, reordering and a monitor component collecting 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 relatively negligible compared to the average batching interval, which is in the range of 300-500 ms.

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.

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 methods presented in Sections 4 and 5 respectively. The component runs in the RUBiS Web server.

DVFS Actuator. To avoid interference of the default DVFS controller with the designed DVFS actuator, 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`.

Cache Module. We disabled the system cache and enabled `mod_mem_cache` of Apache Web server. For the single server system, `MCacheMaxObjectSize` is set to 1MB and `MCacheRemovalAlgorithm` is set to LRU (Least Recently Used). For the multi-server system, `MCacheMaxObjectSize` is set to 5MB and `MCacheRemovalAlgorithm` is set to LRU.

Coordinator. We empirically 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. We empirically set $\lambda_1 = 0.95$, $\lambda_2 = 0.98$ and $\lambda_3 = 0.93$ in a multi-server system.

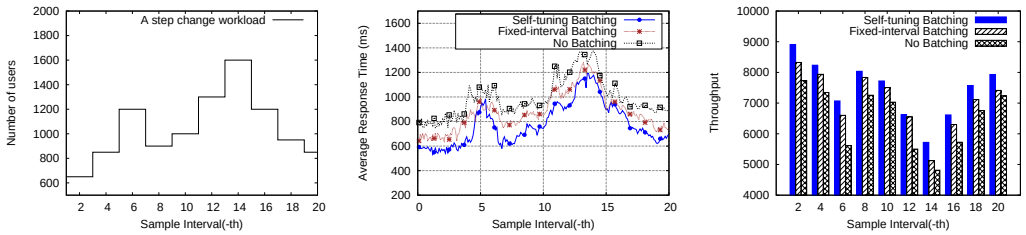
9. PERFORMANCE EVALUATION IN THE SINGLE-SERVER SYSTEM

We first 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. We then present the improvement on energy efficiency due to the self-tuning batching approach. We further show the effectiveness of the approach under different stationary workloads. Finally, we demonstrate the cache size impact on the performance 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 III, we create three different workload scenarios by adjusting the number of concurrent users.

9.1. Performance Improvement of Self-tuning Batching under Dynamic Workloads

Figure 10(a) shows the workload used in the experiment, which is a step-change workload similar to what used in [Guo et al. 2013; Lama and Zhou 2013; Urgaonkar et al. 2008]. The number of concurrent users dynamically changes from 650 to 1600. We compare the self-tuning batching approach with an offline training based fixed-interval batching approach. The cache hit rate, response time, and system throughput are sampled every 30 seconds. As the work in [Stewart et al. 2007; Wang and Wang 2013], our



(a) A highly dynamic workload. (b) Average response time. (c) System throughput.

Fig. 10. Application performance improvement under the highly dynamic workload.

Table III. Performance Improvement of Self-tuning Batching.

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

experiments set the SLA of the system to 1000 ms. A system without batching is used as the baseline. The cache size of `mod_mem_cache` is set to 20 MB.

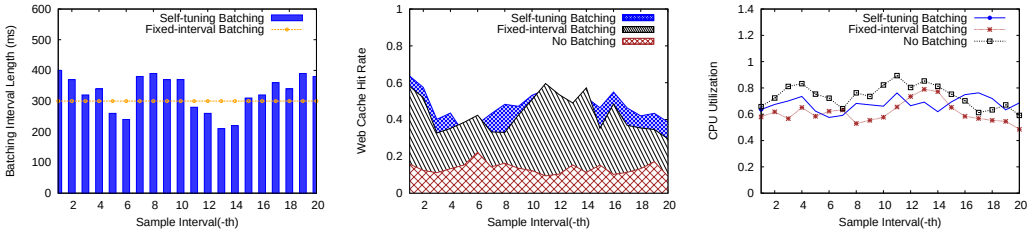
Table IV shows the process of the offline training based fixed-interval batching. We executed 20 experiments with different batching interval lengths using the workload trace. The batching interval length varies from 50 ms to 1000 ms. The results in Table IV show that 300 ms is the optimal batching interval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

Table IV. Offline Training based Fixed-interval Batching.

Batching Interval (ms)	Average Response Time (ms)	Batching Interval (ms)	Average Response 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 10(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 10(c) shows the system 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 10(a)), the average response time increases and the system throughput decreases as shown in Figure 10(b) and 10(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 are replied by the



(a) Self-tuning interval length. (b) Web cache hit rate. (c) CPU consumption.

Fig. 11. Self-tuning batching behaviors under the highly dynamic workloads.

Table V. Performance Improvement of Self-tuning Batching under Dynamic Workloads.

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

server system. So when the workload increases, the average response time of the requests increases. Accordingly, the system throughput decreases as the RUBiS users generate requests in lower rate.

Figure 11(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.

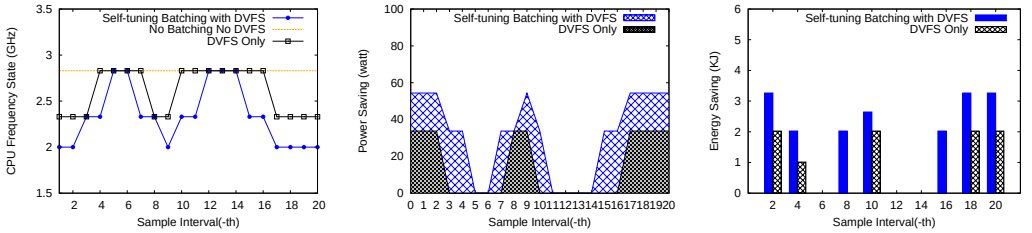
Figure 11(b) shows the comparison of the Web cache hit rate in the three scenarios. Both the self-tuning and fixed-interval batching approaches effectively increase the Web cache hit rate. The self-tuning approach does not always outperform the fixed-interval approach in terms of the cache hit rate. 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 11(c) shows the comparison of CPU utilization by different approaches. It demonstrates that self-tuning batching and fixed-interval batching can effectively reduce consumption. No batching approach has the highest CPU consumption due to its lowest cache hit rate. In return, it has the highest average response time.

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

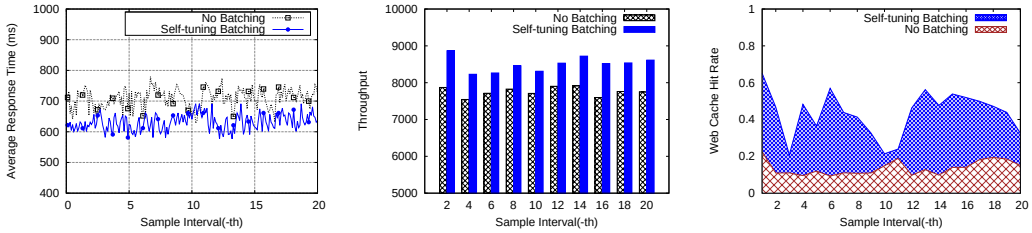
9.2. Energy Efficiency Improvement of Self-tuning Batching Under Dynamic Workloads

Next we 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 CPU to follow the fluctuations in



(a) CPUs frequency adjustments. (b) Power saving curves. (c) Energy savings.

Fig. 12. Energy efficiency comparison under the highly dynamic workload.



(a) Response time. (b) System throughput. (c) Web cache hit rate.

Fig. 13. Performance improvement in an underloaded system.

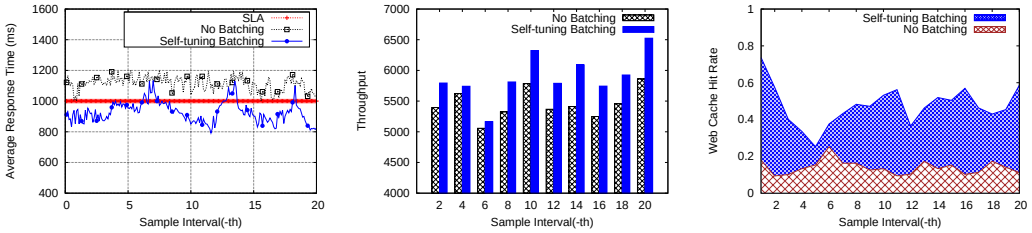
a self-tuning manner. The frequency states of CPU in our experiment start from the lowest level (2 GHz).

Figure 12(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 12(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 12(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 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 CPU frequency.

9.3. Performance Improvement of Self-tuning Batching under Stationary Workloads

We evaluate the average response time and the system 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. Note that under the stationary workload, the offline training based fixed-interval batching approach could find the optimal batching interval length. Thus, we omit its experimental results as the approach performs basically in the same way as the self-tuning batching approach does.

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



(a) Response time. (b) System throughput. (c) Web cache hit rate.

Fig. 14. Performance improvement in a slightly overloaded system.

Table VI. Performance Improvement of Self-tuning Batching under Stationary Workloads.

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

Figure 13(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 13(b) shows the system throughput comparison between the self-tuning batching and no-batching approaches. The batching approach improves the system throughput by 14%. Figure 13(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.

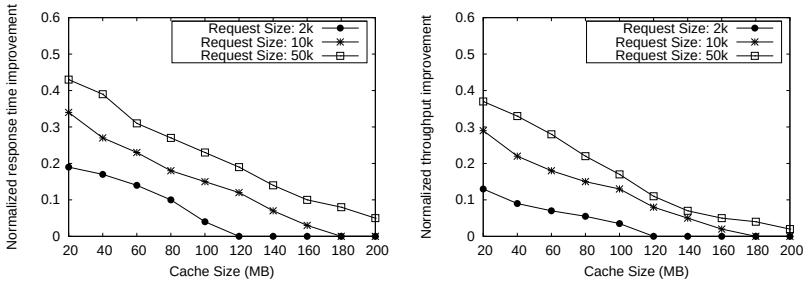
Figure 14(a) plots the average response time. In the no-batching scenario, it is 1091 ms. It indicates the system is slightly overloaded since the SLA on the average response time is 1000 ms. With the self-tuning batching approach, the average response time is significantly reduced to 917 ms. This is 19% improvement compared to that by no-batching.

Figure 14(b) shows that the system throughput due to the self-tuning batching and no-batching approaches. The self-tuning batching improves the system throughput by 17%. Figure 14(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 VI, 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.

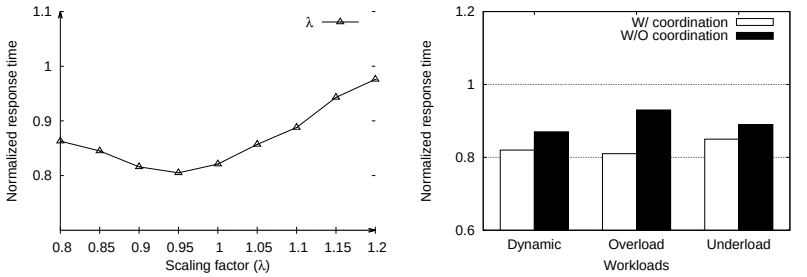
9.4. Cache Size Impact on Performance Improvement

Figure 15 shows the average response time and system throughput improvements 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 due to that the benefit of self-tuning batching is reduced when the cache is large 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 with the same content and reorder them ac-



(a) Average response time. (b) Throughput.

Fig. 15. Cache size impact on performance improvement.



(a) Scaling factor impact. (b) Performance impact.

Fig. 16. The effectiveness of coordinator.

cording to their access frequencies. It thus effectively increases the cache hit rate in a self-tuning manner.

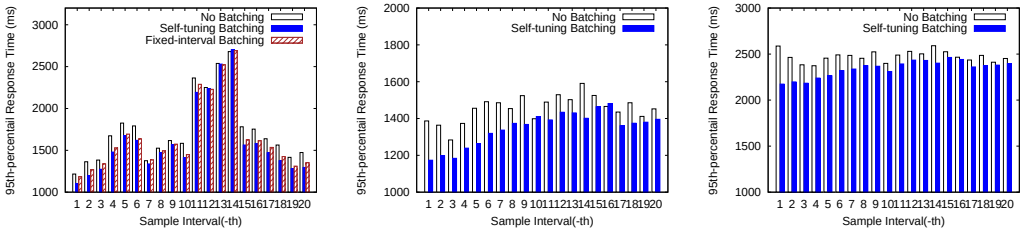
9.5. Effectiveness of the Coordination of Controls

We designed a coordinator to integrate the batching control and the power control for system stability. To identify the scaling factor of the coordinator, we change its value and plot its impact on the application performance in terms of the average response time. Figure 16(a) shows that the average response time initially drops as the scaling factor increases, while further increasing the scaling factor value leads to performance degradation. Thus, we empirically set the scaling factor $\lambda_1 = 0.95$ in our experiments.

Figure 16(b) shows the effectiveness of the coordinator under various workloads. Under three workload scenarios (dynamic, overload, and underload), with the control coodination, the overall application average response time is reduced by 5% when compared to that without the control coordination. The root cause is that the designed coordinator can effectively mitigate the performance interference between the batching control loop and the power control loop.

9.6. Impact on the 95_{th}-percentile Response Time

Percentile-based response time is an important performance metric [Urgaonkar et al. 2008; Lama and Zhou 2013]. Figure 17 illustrates the 95_{th}-percentile response time improvement due to the self-tuning batching approach under the different workloads. Figure 17(a) plots the 95_{th}-percentile response time under the highly dynamic workload (shown in Fgure 10(a)). The self-tuning batching outperforms the offline training based fixed-interval batching approach and the no-batching approach by 3% and 6%, respectively. Figure 17(b) shows the comparison of the 95_{th}-percentile response time



(a) Dynamic workload. (b) Underload. (c) Overload.

Fig. 17. 95_{th}-percentile response time improvement.

Table VII. 95_{th}-percentile Response Time Improvement by Self-tuning Batching.

Workloads	Dynamical workload	Underload	Overload
Batching Improvement	6%	9%	4%

Table VIII. Performance Improvement of Self-tuning Batching.

Scenarios	Dynamic Workload	Stationary Workload (Underload)	Stationary Workload (Overload)
Concurrent users	[3200, 9000]	4000	6000

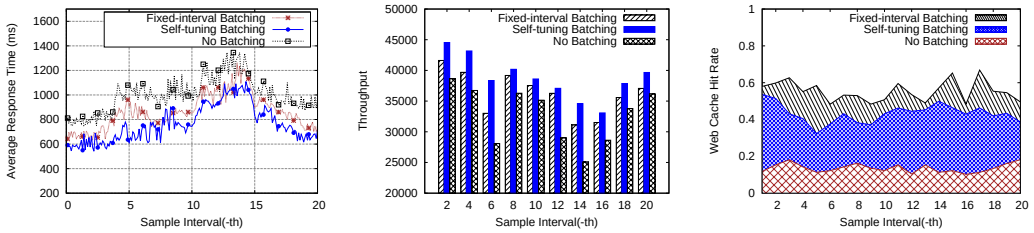
due to the self-tuning batching and no-batching approach for the stationary underload scenario. The self-tuning batching approach improves the 95_{th}-percentile response time by 9%. For the stationary overloaded scenario, we compare the self-tuning batching approach with no-batching approach. Figure 17(c) shows the self-tuning batching approach improves the 95_{th}-percentile response time by 4%.

The results in Table VII demonstrate that the self-tuning batching approach is able to improve the 95_{th}-percentile response time under the varied workloads. But the 95_{th}-percentile response time improvement is not as significant as the average response time improvement. The reason is that the self-tuning batching approach focuses to improving the performance of requests with high frequency.

10. PERFORMANCE EVALUATION IN THE MULTI-SERVER SYSTEM

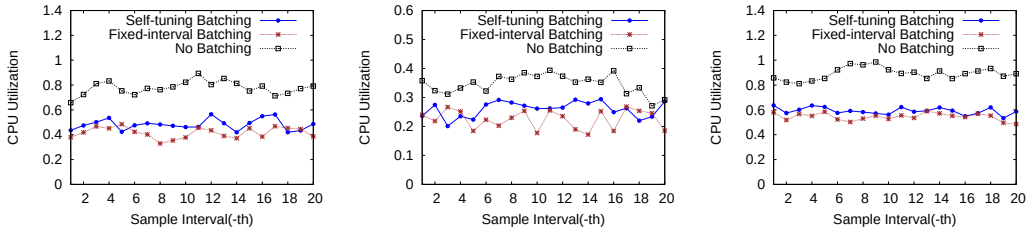
In the multi-server system, we demonstrate the application performance and energy efficiency improvements achieved by the self-tuning batching with DVFS scaling approach under the highly dynamic workload. We further show the effectiveness of the approach under two stationary workloads. Finally, we study the impact of the control coordinator on the performance improvement by the self-tuning batching approach.

As shown in table VIII, we create three different workload scenarios by adjusting the number of concurrent users in the multi-tier server system. Compared to the single-server scenario, we use the same dynamic workload trace as shown in Figure 10(a) but scale up the number of concurrent users due to the expansion of the experimental testbed. For performance reference, we obtained the fixed-interval batching aproach by the offline training process. The training results show that 250 ms is the optimal batching interval length that achieves the minimum average response time under the particular workload trace.



(a) Response time. (b) System throughput. (c) Cache hit rate.

Fig. 18. The performance improvement under the highly dynamic workload in the multi-server system.



(a) Web tier. (b) Application tier. (c) Database tier.

Fig. 19. CPU utilization improvement at each tier in the multi-server system.

10.1. Performance Improvement of Self-tuning Batching under Dynamic Workloads

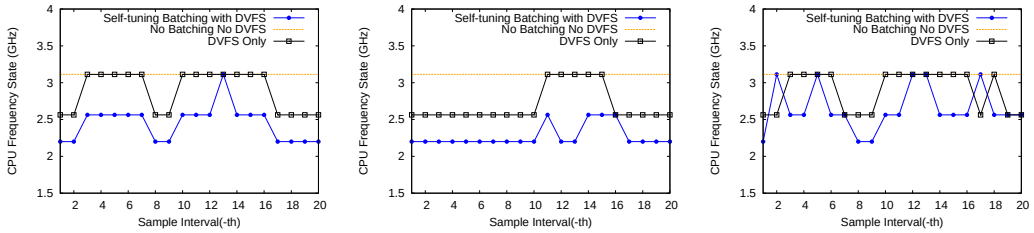
Figure 18(a) shows the comparison of the application's average response time under the highly dynamic workload due to the self-tuning batching approach, the fixed-interval batching approach (using 250 ms interval length), and the no-batching approach. The self-tuning batching outperforms the fixed-interval batching and no-batching approaches by 7% and 18%, respectively. Figure 18(b) shows the system throughput comparison of the three approaches. The self-tuning batching outperforms the fixed-interval batching and no-batching approaches by 9% and 21%, respectively.

Figure 18(b) shows the Web tier cache hit rates achieved by the self-tuning batching approach and the fixed-interval batching approach are about 2.5 to 3 times higher than that achieved by the no-batching approach, respectively. This is due to the fact that the batching approaches increase the Web cache hit rate by classifying and reordering the requests. The fixed-interval approach outperforms the self-tuning approach in terms of the cache hit rate in this scenario because that the fixed-interval approach uses a longer batching interval than the self-tuning approach does.

Figure 19 shows the comparison of CPU utilization by the different approaches. It demonstrates that both self-tuning batching and fixed-interval batching can effectively reduce CPU consumption. No batching approach has the highest CPU consumption due to its lowest cache hit rate. Figure 19(b) illustrates that the PHP application tier server has the lowest CPU utilization compared to the other two tier servers. This is due to the fact that the PHP application server has the lightest workload while the MYSQL database server has much heavier workloads in the experiment.

10.2. Energy Efficiency Improvement of Self-tuning Batching under Dynamic Workloads

We illustrate the effectiveness of integrating MIMO power control with self-tuning batching control for improving energy efficiency in the multi-server system under vari-



(a) Web tier. (b) Application tier. (c) Database tier.

Fig. 20. CPUs frequency adjustments under the highly dynamic workload in the multi-server system.

ous workloads. When the workload fluctuates, the power control modifies the frequency of CPU at each tier server to follow the fluctuations in a self-tuning manner.

Figure 20 shows how the frequency state of the CPU is dynamically adjusted at the Web, application and database tiers as the workload varies due to two approaches, i.e., DVFS only and self-tuning batching with DVFS. Figure 20(a) shows that the CPU frequency of the Web tier server is dynamically scaled up or down by the two approaches. The self-tuning batching with DVFS approach works at the lower CPU frequency state during the experiment. This is due to the fact that the approach benefits from the performance improvement by the request batching while the DVFS only approach does not. Figure 20(b) reveals that the application tier server has the lowest CPU frequency for both the DVFS only and the self-tuning batching with DVFS approaches. The lowest CPU consumption at this tier provides an opportunity to scale down CPU frequency for more energy savings. On the other side, the database tier server has the highest CPU frequency, due to its highest CPU consumption among all three tiers as shown in Figure 19(c).

Figure 21 depicts the energy savings achieved by DVFS only and self-tuning batching with DVFS approaches under the highly dynamic workloads. Their results are compared to that of the baseline approach, i.e., no batching and no DVFS. It shows that the self-tuning batching with DVFS approach reduces energy consumption of the multi-server system by 11%. Furthermore, the results provide three important insights of energy efficiency in the multi-tier server system. Firstly, both DVFS only and self-tuning batching with DVFS approaches effectively reduce the energy consumption at each tier in the system. The energy savings are due to the dynamic CPU frequency scaling operations when the workload fluctuates. Secondly, the energy saving achieved by the self-tuning batching with DVFS approach outperforms what is achieved by the DVFS only approach. This is due to the fact that the self-tuning batching approach can scale down the CPU frequency aggressively while avoiding the performance degradation. Finally, Web and application servers obtain more energy savings than the database server does, due to the highest CPU consumption at the database tier.

10.3. Performance Improvement of Self-tuning Batching under Stationary Workloads

We evaluate the average response time and the system throughput due to the self-tuning batching in two stationary workload scenarios. 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. Again, note that under the stationary workload, the offline training based fixed-interval batching approach could find the optimal batching interval length. Thus, we

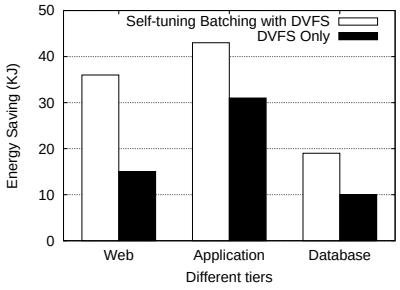
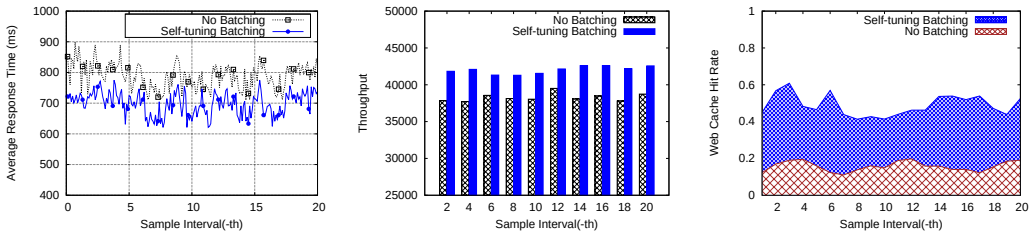


Fig. 21. Energy savings at each tier under the highly dynamic workloads.



(a) Response time. (b) Throughput. (c) Web cache hit rate.

Fig. 22. Performance improvement in an underloaded system.

omit its experimental results as the approach performs basically in the same way as the self-tuning batching approach does.

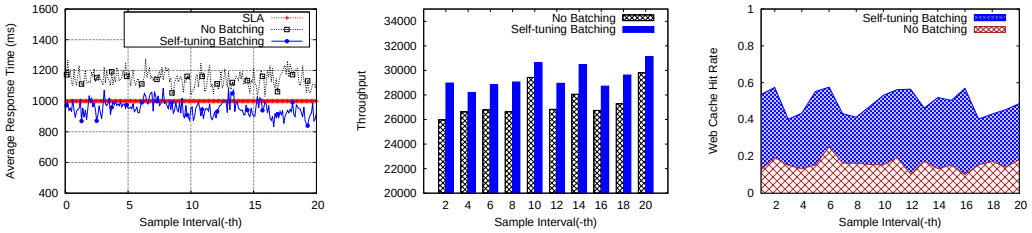
For the underloaded scenario, we set the RUBiS user with browsing workload mix and 4000 concurrent users. It is a moderate workload for the multi-server testbed with respect to the SLA.

Figure 22(a) plots the average response time. The self-tuning batching approach improves the average response time by 11%, reducing the average response time from 791 ms to 702 ms. Figure 22(b) shows the system throughput comparison between the self-tuning batching and no-batching approaches. The self-tuning batching approach improves the system throughput by 12%. Figure 22(c) shows that the Web cache hit rate increases significantly due to the self-tuning batching approach. These results demonstrate that the self-tuning batching can significantly improve the application performance in the underloaded scenario.

For the slightly overloaded scenario, we set the RUBiS user with browsing workload mix and 6000 concurrent users. Figure 23(a) plots the average response time by the two approaches. In the no-batching approach, it is 1174 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 972 ms. This is 17% improvement compared to that by no-batching approach.

Figure 23(b) shows that the system throughput due to the self-tuning batching and no-batching approaches. The self-tuning batching improves the system throughput by 15%. Figure 23(c) shows that the Web cache hit rate due to the self-tuning batching is 3 times higher than that of no-batching approach.

As shown in Table IX, 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 in the multi-server system.



(a) Response time. (b) Throughput. (c) Web cache hit rate.

Fig. 23. Performance improvement in a slightly overloaded system.

Table IX. Performance Improvement of Self-tuning Batching under Stationary Workloads.

Workload Scenarios	Average Response Time	System Throughput	Energy Efficiency
Underload	11%	12%	9%
Overload	17%	15%	7%

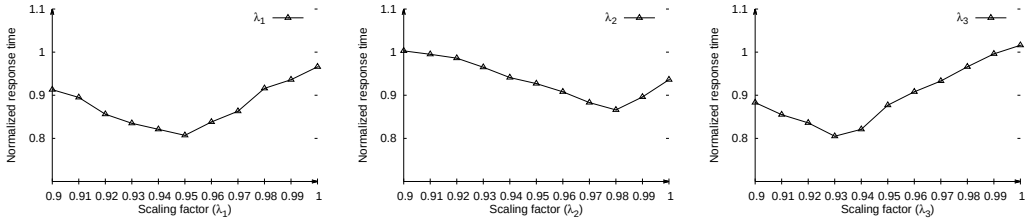
10.4. Effectiveness of the Coordination of Controls

In the multi-server system, there are complex interactions between the batching control loop and the power control loop. To reduce the impact of interactions on system performance, we designed a coordinator between the controls. The coordinator applies a scaling factor weight vector $\lambda = [\lambda_1, \lambda_2, \lambda_3]$ that represent the modeling accuracy impact caused by CPU frequency modification at Web, application and database tier, respectively. We change the values of these three parameters and plot their impact on application performance in terms of average response time in Figure 24. The results show that the average response time initially drops as the value of scaling parameters increases. However, further increasing the value of parameters leads to performances degradation for all of the tiers. Thus, we empirically set $\lambda_1 = 0.95$, $\lambda_2 = 0.98$ and $\lambda_3 = 0.93$ in all experiments. This parameter setting represents the system performance sensitivity of the DVFS operation at different tiers. The results illustrate that the CPU frequency of the server at the database tier has the highest impact on the system performance modeling.

In Figure 25, we compare the average response time achieved by the self-tuning batching approach with and without the control coordination. The results illustrate that the coordinator reduces the application average response time by 8% compared with that due to the no-coordination approach under the three different workload scenarios. The results also demonstrate the self-tuning batching approach gains much more performance improvement under the dynamic workload and overload scenarios than the underload scenario. This is due to the fact that the self-tuning batching approach focuses on the heavy workload, which provides more opportunities to improve Web cache hit rate in the multi-server system.

10.5. Comparison with a PI Controller under Varying Workloads

We evaluate the performance of the proposed fuzzy model based DVFS control in case of varying workload characteristics. As the classical proportional integral (PI) controller has been used for DVFS control and performance assurance in Internet servers, we use it for comparison with the fuzzy controller. The control interval is set to 30 seconds in both controllers.



(a) λ_1. (b) λ_2. (c) λ_3.

Fig. 24. The response time improvement impact by various scaling factors of coordinator.

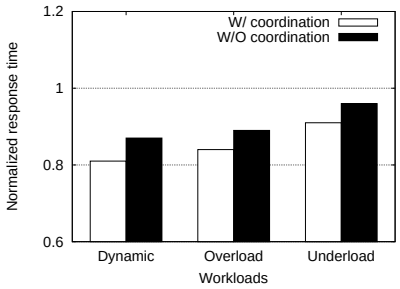


Fig. 25. The effectiveness of coordinator.

PI control is model based. As others [Wang and Wang 2009; Wang et al. 2012], we first obtain the system performance model of the multi-tier server system by using a standard system identification technique, the Least Squares Method (LSM). Based on data collected from the system offline, we use LSM to estimate the parameters of the PI controlled system performance model. The system model and obtained parameters are given by Eq. 28,

$$V'(k) = b_1 V'(k-1) + C_1 f'(k-1), \tag{28}$$

where $b_1 = -0.0453$ and $c_1 = -0.7261$. The controlled variable in the PI model is $V'(k) = V(k) - V$ where $V(k)$ is the average system throughput in the k_{th} control period. The manipulated variable is $f'(k) = f(k) - f$ where $f(k)$ is the CPU frequency adjustment of the system in the k_{th} control period. V and f are the average system throughput and the CPU frequency of the system corresponding to a chosen operating point that is used to linearize the system model. In our experiment, f is set to be the medium value of a typical CPU frequency range of the system. We then measure the average system throughput V. Based on the system model, the PI controller is designed to achieve the desired control performance in system stability and zero steady-state error. The transfer function of the designed PI controller is:

$$F(z) = \frac{-0.2738(z+1)}{(z-1)}. \tag{29}$$

We compare the performance of DVFS power control approach achieved by our fuzzy controller and by the PI controller under the varying workloads, i.e., stationary workloads and dynamic workloads. Figure 26 shows the energy savings for the three workload scenarios, i.e., dynamic, overload and underload. In the underload scenario, our fuzzy controller shows similar performance of the PI controller. However, it significantly outperforms the PI controller in the overload scenario. The PI controller is de-

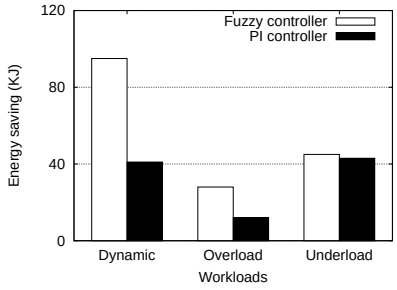
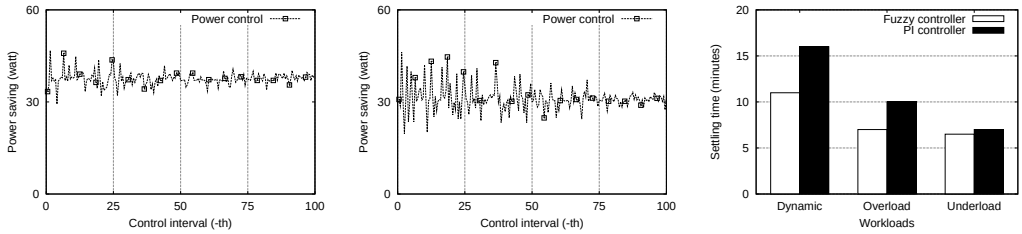


Fig. 26. Energy saving comparison of two controllers under three various workloads.



(a) Fuzzy controller oscillation. (b) PI controller oscillation. (c) Settling time comparison.

Fig. 27. Transient behaviors comparison between the fuzzy controller and the PI controller.

signed for a system model that is identified by using the medium workload. Hence, it shows poor performance when the workload intensity changes significantly. On the other hand, our fuzzy controller is adaptive to dynamic workloads. It outperforms the PI controller significantly due to its adaptiveness to variations in the workload intensity. Compared to the use of the PI controller, the use of the fuzzy controller achieves overall energy saving improvement of 52% under the three workload scenarios.

The proposed fuzzy controller is more accurate than the PI controller to capture the complex workload characteristics since the PI controller is designed based on the medium workload. The inaccurate model of PI controller leads to worse transient properties during the control adjustment process, such as greater oscillation and longer settling time. Figure 27 compares the transient behaviors between the fuzzy controller and the PI controller. Figure 27(a) and 27(b) show that the fuzzy controller effectively reduces the oscillation of control adjustment compared to the PI controller. Figure 27(c) demonstrates that the fuzzy controller significantly improves the control settling time under the dynamic and overload workload scenarios. This is due to the fact that the fuzzy controller has more accurate system model compared to the PI controller.

10.6. Overhead of the Self-tuning Batching Mechanism

The overhead of the self-tuning batching mechanism is due to three major components: (1) the time required to activate control adjustments according to its control algorithm; (2) the time required to perform two control algorithms in each control interval; and (3) the time required to read system statistics. Under different workload scenarios, we measure the overhead of each component. Table X shows the results. We find that the total overhead is negligible compared to the control interval of 30 seconds.

Table X. The overhead of the self-tuning control mechanism.

	Dynamic	Underload	Overload
Batching activation	226 ms	142 ms	255 ms
DVFS adjustment	48 ms	28 ms	15 ms
Interval control algorithm	65 ms		
DVFS control algorithm	20 ms		
System statistics reading	30 ms		

11. CONCLUSIONS

In this paper, we proposed and developed a self-tuning batching approach integrated with DVFS to simultaneously improve the application performance and energy efficiency of the underlying 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 the batching interval length to avoid SLA violations and integration of two controls for improving the system stability. The approach can improve the application average response time by 18%, system throughput by 13%, and reduce the energy consumption of the server system by 13% at the same time.

We further extended the self-tuning batching with DVFS approach from a single-server system to a multi-server system. We designed a novel power control loop based on the MIMO expert fuzzy control to adjust the CPU frequencies of multiple servers at the same time. In order to enhance the performance of DVFS controls, we developed a coordinator between the power control and the batching control. The experimental results demonstrate that the self-tuning batching with DVFS approach effectively improves the application average response time by 18%, system throughput by 21%, and reduces the energy consumption of the multi-server system by 11% at the same time.

In future work, we plan to extend the self-tuning batching with DVFS approach for heterogeneous workloads in virtualized server clusters.

REFERENCES

B. Addis, Dr. Ardagna, B. Panicucci, M. Squillante, and L. Zhang. 2013. A Hierarchical Approach for the Resource Management of Very Large Cloud Platforms. *IEEE Trans. on Dependable and Secure Computing* 10, 5 (2013), 253–272.

Y. Chen, B. Yang, A. Abraham, and L. Peng. 2007. Automatic Design of Hierarchical Takagi-Sugeno Type Fuzzy Systems Using Evolutionary Algorithms. *IEEE Transactions on Fuzzy Systems* 15 (2007), 385–397. Issue 3.

D. Cheng, Y. Guo, and X. Zhou. 2013. Self-tuning Batching with DVFS for Improving Performance and Energy Efficiency in Servers. In *Proc. IEEE/ACM Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, and M. Surendra. 2006. Controlling quality of service in multi-tier Web applications. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*.

M. Elnozahy, M. Kistler, and R. Rajamony. 2003. Energy conservation policies for web servers. In *Proc. the USENIX Symp. on Internet Technologies and Systems (USITS)*.

A. Gandhi, M. Harchol-Balter, and M. Kozuch. 2011. The case for sleep states in servers. In *the USENIX Workshop on Power Aware Computing and Systems (HotPower)*.

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

Z. Gong and X. Gu. 2010. PAC: Pattern-Driven Application Consolidation for Efficient Cloud Computing. In *Proc. IEEE/ACM Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecomm. Systems (MASCOTS)*.

Y. Guo, P. Lama, Jia Rao, and X. Zhou. 2013. 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)*.

Y. Guo, P. Lama, and X. Zhou. 2012. Automated and Agile Server Parameter Tuning with Learning and Control. In *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS)*.

D. Hagimont, C. M. Kamga, L. Broto, A. Tchana, and N. D. Palma. 2013. DVFS Aware CPU Credit Enforcement in a Virtualized System. In *Proc. ACM/IFIP/USENIX Int'l Conf. on Middleware (Middleware)*.

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

G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. 2010. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*.

S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. 2009. vManage: Loosely Coupled Platform and Virtualization Management in Data Centers. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*.

P. Lama, Y. Guo, and X. Zhou. 2013. Autonomic Performance and Power Control for Co-located Web Applications on Virtualized Servers. In *Proc. ACM/IEEE Int'l Workshop on Quality of Service (IWQoS)*. 1–10.

P. Lama and X. Zhou. 2011. PERFUME: Power and Performance Guarantee with Fuzzy MIMO Control in Virtualized Servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*.

P. Lama and X. Zhou. 2012. NINEPIN: Non-Invasive and Energy Efficient Performance Isolation in Virtualized Servers. In *Proc. IEEE/IFIP Int'lConference on Dependable Systems and Networks (DSN)*. 1–12.

P. Lama and X. Zhou. 2013. Autonomic Provisioning with Self-adaptive Neural Fuzzy Control for Percentile-based Delay Guarantee. *ACM Transactions on Autonomous and Adaptive Systems* 8, 2 (2013), 1–31.

C. Lefurgy, X. Wang, and M. Ware. 2007. Server-level power control. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*.

R. Nathuji, C. Isci, and E. Gorbatov. 2007. Exploiting Platform Heterogeneity for Power Efficient Data Centers. In *Proc. IEEE Int'l Conf. on Autonomic Computing (ICAC)*.

P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. 2009. Automated control of multiple virtualized resources. In *Proc. of the EuroSys Conference (EuroSys)*. 13–26.

V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. 2003. Power-aware QoS management in web servers. In *Proc. the IEEE Real-Time Systems Symp. (RTSS)*.

C. Stewart, T. Kelly, and A. Zhang. 2007. Exploiting nonstationarity for performance prediction. In *Proc. of the EuroSys Conference (EuroSys)*. 31–44.

O.S. Unsal and I. Koren. 2003. System-level power-aware design techniques in real-time systems. *Proc. of the IEEE* 91, 7 (2003), 1–15.

B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. 2008. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems* 3, 1 (2008), 1–39.

A. Verma, P. Ahuja, and A. Neogi. 2008. pMapper: power and migration cost aware application placement in virtualized systems. In *Proc. ACM/IFIP/USENIX Int'l Middleware Conference*.

H. Wang, Q. Teng, X. Zhong, and P.F. Sweeney. 2010. Using the middle tier to understand cross-tier delay in a multi-tier application. In *Proc. IEEE Int'l Parallel Distributed Processing Symposium (IPDPS)*.

H. O. Wang, K. Tanaka, and M. F. Griffin. 1996. An Approach to Fuzzy Control of Nonlinear Systems: Stability and Design Issues. 4 (1996), 14–23. Issue 1.

Q. Wang, Y. Kanemasa, J. Li, C. A. Lai, M. Matsubara, and C. Pu. 2013. Impact of DVFS on n-Tier Application Performance. In *Proc. ACM Conference on Timely Results in Operating Systems (TRIOS)*.

X. Wang, M. Chen, and X. Fu. 2010. Mimo power control for high-density servers in an enclosure. *IEEE Trans. on Parallel and Distributed Systems* 21, 10 (2010), 1412–1426.

X. Wang, M. Chen, C. Lefurgy, and T.W. Keller. 2012. SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers. *IEEE Trans. on Parallel and Distributed Systems* 23 (2012), 168–176. Issue 1.

X. Wang and Y. Wang. 2009. Co-Con: Coordinated control of power and application performance for virtualized server clusters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*.

Y. Wang and X. Wang. 2013. Virtual Batching: Request Batching for Server Energy Conservation in Virtualized Data Centers. *IEEE Trans. on Parallel and Distributed Systems* 24 (2013), 1695–1705. Issue 8.