Virtual Batching for Performance Improvement and Energy Efficiency

Dazhao Cheng
Department of Computer Science
University of Colorado, Colorado Springs, USA
Email addresses: {dcheng}@uccs.edu

Abstract

Performance improvement and energy efficiency are two important goals in provisioning Internet applications in data centers. In this paper, we propose and develop a content-aware self-tuning request batching mechanism that can simultaneously achieve the two correlated goals. The batching mechanism increases the cache hit rate at Web server, which provides the opportunity to improve performance of Internet applications and energy efficiency of servers at the same time. 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 characteristics of workloads. 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 fluctuations of workloads for energy efficiency. A coordinator between the two-layer controls achieves the desired performance and energy consumption. We implement the mechanism in a testbed of virtualized servers hosting RUBiS benchmark applications. Experimental results find that the new approach can improve the application performance by 19% and reduce the energy consumption of the server system by 14%.

Keywords: Batching, Performance Control, Energy Efficiency, Dynamic Voltage and Frequency Scaling, Multi-tier Internet Applications, Fuzzy Model Predictive Control

1 Introduction

There are two main challenges for operating a modern data center: performance guarantee with respect to the service level agreement (SLA) with applications for increasing revenue and energy efficiency of the server system for reducing the operating cost. A well-known approach to controlling energy consumption is to transition a processor from high-power states to low-power states using Dynamic Voltage and Frequency Scaling (DVFS) technique [14, 16, 22]. The transition of CPU power states from high to low for energy saving will increase the response time of applications. Therefore, we need to reduce both energy consumption and SLA violations.

In this paper, we propose and develop a content-aware and self-tuning request batching mechanism for performance improvement of Internet applications and energy efficiency in virtualized servers. Today, popular Internet applications employ a multi-tier architecture with each tier depending on its successor and providing functionality to its preceding tier [3, 10, 11, 18]. Front-tier Web servers process requests in the order of arrival. Indeed, most of those web requests are independent to each other. Batching requests in a content-aware manner can improve the cache hit rate of Web

servers, which in return provides the opportunity to improve the performance of multi-tier applications and energy efficiency of underlying server system at the same time. Together with batching, we employ DVFS power management technique to minimize the energy consumption while meeting the SLA on application 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 and reorder more requests for the same content. Thus it may increase the cache hit rate and reduce the application response time. But batching will also delay the processing of those requests, resulting in longer application response time. It will risk the violation of the SLA. Thus, the batching mechanism must be self-tuning in choosing the batching interval. Unfortunately, due to high complexities of multi-tier Internet service architecture, high dynamics in workloads and the virtualized server infrastructure, obtaining an accurate model among virtual machine (VM) capacity, server configuration, performance and power consumption is a very hard problem even for just one application.

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

We design a novel and 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 technique to address the issue of lacking an accurate performance-power model due to workload dynamics. It is a real-time online decision maker based on system conditions and historical experiences. According to the fluctuations of the workload and the usage of CPUs, EFC decides when and which frequency state should be set for CPUs. Furthermore, we design a coordinator between the two-layer controls to achieve the desired performance and energy consumption.

We build a testbed of multi-tier server system based on Xen 3.1 virtualization software. The servers are 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 evaluate the implemented batching mechanism with RUBiS benchmark applications [7, 15, 18, 23]. Experimental results demonstrate that the new approach can improve the application performance in terms of response time by 19% and reduce the energy consumption of the server system by 14%.

The main contributions of our work are:

- We propose to use request batching with DVFS technique under heavy and dynamic workloads for improving application performance and energy efficiency of server systems at the same time.
- We design and develop a novel and practical two-layer control system that is composed of a batching interval control loop and a power control loop. For self-tuning, the control system is based on fuzzy model predictive control (FMPC) and expert fuzzy control (EFC). We further

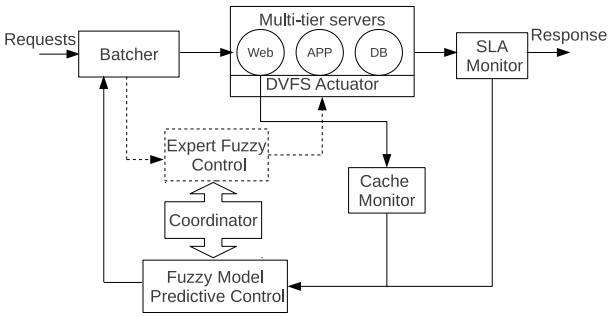


Figure 1: The System Architecture.

design a coordinator between the two-layer controls to integrate FMPC and EFC controls for joint performance and power control.

- We build a testbed and evaluate the new approach with the RUBiS benchmark application. Experimental results demonstrate that the developed batching mechanism can effectively improve both application performance and energy efficiency.

In the following, Section 2 gives the batching control system architecture and the batching strategy. Section 3 presents the modeling, design, and analysis of the batching interval controller. Section 4 gives the power control design and the integration of batching and DVFS. Section 5 introduces the testbed implementation. Section 6 presents the experimental results and analysis. Section 7 concludes the paper.

2 System Architecture and Batching Strategy

2.1 The System Architecture

Figure 1 illustrates the architecture of the batching control system. It is composed of one batching control loop and a power control loop in a virtualized multi-tier server system.

The key components in the batching control loop include a batcher, a SLA monitor, and a cache hit rate monitor. The control loop relies on fuzzy model predictive control (FMPC) that captures the nonlinear relationship between the application performance and the batching interval length. It outputs the batching interval legnth based on a dynamic fuzzy model. An online self-tuning module is applied 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 model-independent fuzzy control technique to address the issue of lacking an accurate performance-power model due to 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 control loop, we design a coordinator between the two-layer controls to achieve the desired application performance while improving energy efficiency.

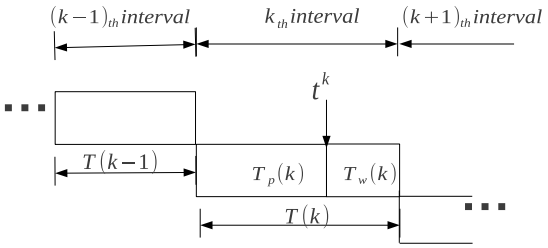


Figure 2: The batching interval length.

2.2 The Batching Strategy

The request classification is done in the following two steps.

1. During each batching interval, requests incoming to the frond-tier Web server are classified into groups according to the requested contents. The request classification is based on its header URL information.

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. Doing so can reduce the average request response time.

The batching interal length is changed dynamically as follows, and Figure 2 illustrates the process.

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 $(k-1)_{th}$ batching interval at time t^k. Let $T_p(k)$ denote the total processing time of the requests. The SLA monitor will measure the average response time of the requests $r(k-1)$. The cache monitor will measure the cache hit rate $h(k-1)$ of the requests.

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 period of $T_w(k)$, the current batching interval is over. The requests collected during the interval will be send to the multi-tier server system. The $(k+1)_{th}$ batching interval starts.

3 The Batching Control Design

Due to the workload dynamics and the SLA, the batching interval length should be changed in a self-tuning manner. It is a challenging problem to determine the batching interval length. Intuitively, a longer batching interval can accumulate and reorder more requests for the same content, and thus it may increase the cache hit rate and reduce the average response time. But batching will also delay

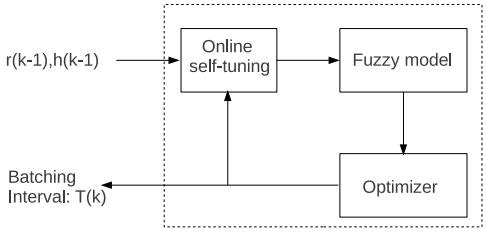


Figure 3: The design of fuzzy model predictive control (FMPC).

the processing of those requests, resulting in longer average response time. It will risk the violation of the SLA. Unfortunately, there does not exist a linear relationship between the average response time and the batching interval length, due to high complexities of multi-tier Internet service architecture and high dynamics in workloads.

We propose to use fuzzy model predictive control (FMPC). 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 the control includes the following aspects:

1. It simplifies nonlinear modeling of a complex 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 the traditional predictive control such as control accuracy and stability.

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

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 to find the optimal value of the batching interval length $T(k)$.

3.1 The Fuzzy Model

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

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

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

represented as

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

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

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

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

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

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

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

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

In Eq. (5), K is the number of rules for the output. γ_i is the degree of fulfillment for the i_{th} rule. The value of γ_i is the product of the membership degrees of the antecedent variables in that rule. Membership degrees are determined by fuzzy membership functions associated with the antecedent variables. The model output in Eq. (5) is expressed in the form of

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

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

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

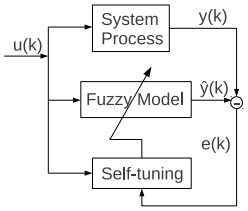


Figure 4: A self-tuning module of FMPC.

3.2 Online Self-Tuning of the Model

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 is 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.

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

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

where $e(k)$ is the error between the actual output and predicted output. $\phi(k)$ is a vector composed of the model parameters. $X(k) = [\xi(k)^T, u(k)]$ 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.

$$Cost = \sum_{k-1}^{k} (e(k)^2 + 0.5e(k-1)^2). \tag{9}$$

3.3 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:

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

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

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

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

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

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

$$\begin{aligned} x_{lin}(k+1) &= A(k)x_{lin}(k) + B_u(k)u(k) + B_h(k)h(k). \\ y_{lin}(k) &= C(k)x_{lin}(k). \end{aligned} \tag{12}$$

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

$$x_{lin}(k+1) = [\xi^T(k), 1]^T \tag{13}$$

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. (12), the state matrices are computed as follows:

$$A = \begin{bmatrix} \zeta_1^* & \zeta_2^* & \cdots & \zeta_j^* & \cdots & \zeta_\rho^* & \theta^* \\ 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{14}$$

$$B_u = \begin{bmatrix} \eta^* \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{15}$$

$$B_h = \begin{bmatrix} \omega^* \\ 0 \\ 0 \\ \vdots \end{bmatrix} \tag{16}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & \cdots \end{bmatrix} \tag{17}$$

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

$$\begin{aligned} x(k+1) &= \bar{A}(k)x(k) + \bar{B}_u(k)\Delta u(k) + \bar{B}_h(k)h(k). \\ y(k) &= \bar{C}(k)x(k). \end{aligned} \tag{18}$$

$$\begin{aligned} \bar{A}(k) &= \begin{bmatrix} A(k) & B_u(k) \\ 0 & I \end{bmatrix} \\ \bar{B}_u(k) &= \begin{bmatrix} B_u(k) \\ I \end{bmatrix} \\ \bar{B}_h(k) &= \begin{bmatrix} B_h(k) \\ I \end{bmatrix} \\ \bar{C}(k) &= \begin{bmatrix} C(k) & 0 \end{bmatrix} \end{aligned} \tag{19}$$

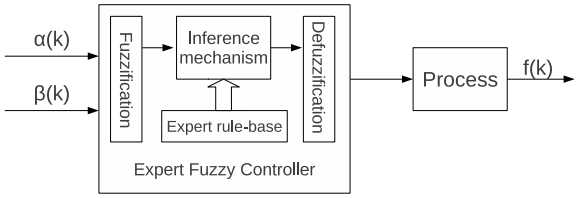


Figure 5: The expert fuzzy control design.

At the k_{th} control interval, the state vector Eq. (18) 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 calcualted through successive substitutions.

4 Power Control

4.1 DVFS Control

A virtualized web server system exhibits significant variations in the utilization of underlying CPU resources due to the dynamic behaviors of workloads. This system characteristic can be exploited for CPU power management. Our control system dynamically scales the CPU frequency of server in response to time-varying resource demands. Modern processors allow such frequency scaling for energy savings via DVFS technique .

We design a DVFS control based on model-independent expert fuzzy control technique to minimize the energy consumption of the server system. The expert fuzzy control (EFC) technique allows the DVFS controller to make online control decisions based on site conditions and historical experiences. EFC is applied in measurement combined fuzzy control theory with expert control theory. Its model-independence addresses a practical issue of control design, which is the lack of an accurate performance model for a complex web system in the face of dynamic workload variations.

$$\alpha^k = \frac{V(k) - V(k-1)}{V(k-1)} \tag{20}$$

$$\beta^k = \frac{T_p(k) - T_w(k)}{T_w(k)} \tag{21}$$

Figure 5 shows the expert fuzzy control (EFC) architecture of the DVFS control. EFC has two inputs, α^k and β^k. The α^k reflects the fluctuations in the workload. It is given by Eq. (20), where $V(k)$ is the average throughput of web requests processed in the k_{th} control period. The second input, β^k, reflects the CPU usage of the server system. It is given by Eq. (21), where $T_p(k)$ and $T_w(k)$ are the process time and waiting time in the batching interval respectively. The output of EFC, $f(.)$, is the necessary change in the CPU frequency. It has three possible values, $(f(+1); f(0); f(-1))$, which denote increasing, keeping and decreasing the CPU frequency respectively. The transition of the CPU frequency is designed to occur in discrete steps.

The design objective of EFC is to translate a human expert's knowledge into a set of control rules that manage the CPU frequency of the server system. Figure 6 shows the rule base of EFC. The control rules are defined using linguistic variables corresponding to the two inputs, α^k and β^k. The

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)

Figure 6: The rule table.

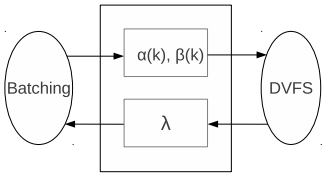


Figure 7: Coordinator.

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) [24]. The rules are in the form of If-Then statements. For example, If α^k is PL and β^k is PL, then 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 grown. Similarly various rules are designed to determine the CPU frequency states based on the fluctuations in workload and CPU usage. Note that the rule base table is divided into three zones corresponding to the control actions $f(+1)$, $f(0)$ and $f(-1)$. Because the discrete CPU frequency states available in our testbed are 2 GHz, 2.33 GHz and 2.83 GHz. However, it can be further split into more number of smaller zones for precise control if more frequency states are available in the CPUs.

4.2 Integration of Batching and DVFS

An important aspect of the proposed control framework is to integrate two different control loops for improving performance and energy efficiency. The system uses both batching and DVFS controls, dynamically and simultaneously. The batching control determines each batching interval length in order to achieve the average response time target. Meanwhile, the DVFS control determines the CPU frequency of the server in order to reduce energy consumption.

There are some complex interactions between these two control loops. On one hand, the changes in system behavior due DVFS control action have an impact on the accuracy of batching control. On the other hand, DVFS control decisions are dependent on some parameters, which are affected by the batching control. Therefore, a coordinator as shown in Figure 7 is designed to integrate the two controllers and provide system stability. It involves the following interactions between the controllers.

5 System Implementation

5.1 Testbed

We build a testbed consisting of one server and one client. 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 servers are running CentOS 5.8 with Linux kernel 2.6.18.

We use RUBiS [1] as the benchmark application for experiment. RUBiS is an open source multi-tier Internet benchmark application. RUBiS provides a web auction application that modeled in a similar way of *ebay.com*. It characterizes the workload into three categories, browsing, bidding, and selling. RUBiS client emulates user requests at different concurrent levels.

We create three VMs for the RUBiS benchmark application, Apache web server in the first, PHP application server in the second, and MYSQL database server in the third. Each VM is allocated 1 VCPU and 512 MB memory. All VMs use Ubuntu server 10.04 with Linux kernel 2.6.35. The *mem.cache* module and *cache* module are enabled for the experiment.

Each VM is allocated 512 MB of RAM and each of them is also configured with four virtual CPUs, as the physical computer has four CPU cores. An Apache server is installed on the first VM and runs as a virtualized web server. The module *mem.cache* and *cache* is enabled and use as cache to increase access speed. The Apache servers respond to the HTTP requests from Client with a dynamic webpage written in PHP. The PHP and Mysql Database server are installed on the second and third VM respectively.

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

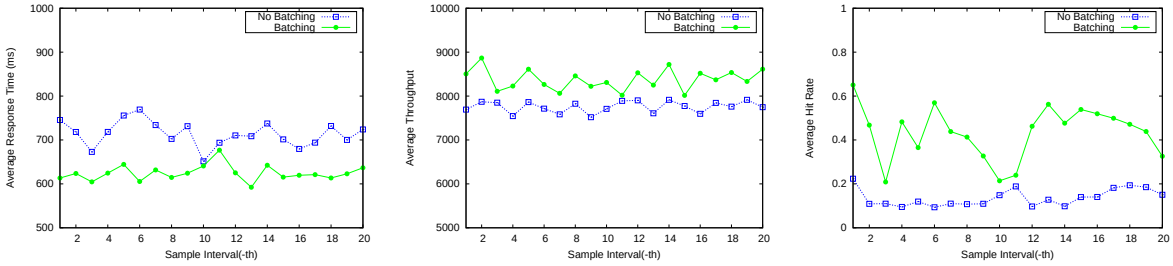
5.2 System Components

Batcher The content-aware batcher is a daemon program running with the web tier server. The batcher is consists of serval modules including requests batching, classification, reordering and a monitor component to collect parameters for controllers.

Performance Monitor We modify the RUBiS client to support online measurement of user-perceived response time and system throughput over a period of time. The performance monitor collecting the performance data from the RUBiS client.

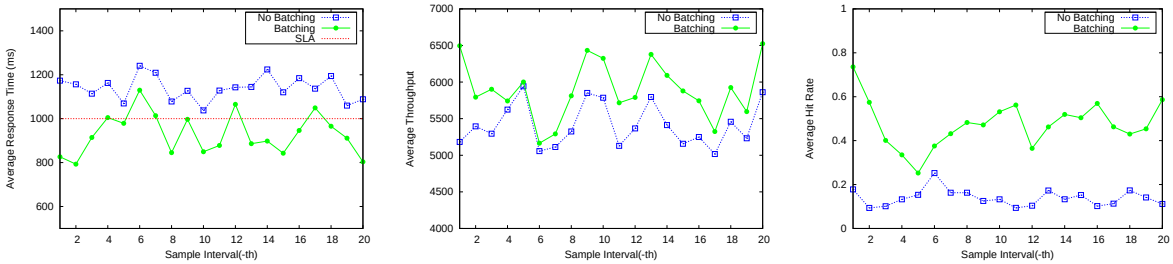
FMPC and EFC Controller These two controllers receives the response time, the system throughput and the cache hit rate and other parameters from the monitors. Accordingly, they run the control algorithms presented in Sections 3 and 4 respectively.

DVFS actuator We install the cpufreq package in dom0 to provide the functionality of controlling the CPU frequency of physical server. It tunes the CPU frequency by writing the DVFS state into



(a) Response time. (b) Throughput. (c) Hit rate.

Figure 8: Average response time, throughput and cache hit rate of underloaded system.



(a) Response time. (b) Throughput. (c) Hit rate.

Figure 9: Average response time, throughput and cache hit rate of slightly overloaded system.

the system file */sys/devices/system/cpu/cpu*0*/cpufreq/scaling.setspeed*.

6 Evaluation

We evaluate the average response time and throughput due to content-aware batching in two scenarios. The first scenario is a underloaded system. The second scenario is slightly overloaded. We demonstrate the effectiveness of content-aware batching by comparing the performance metrics due to batching with those without batching.

For the scenario of underloaded system, we set the RUBiS client with browsing workload mix and 800 concurrent users. It is a moderate workload for our testbed.

Figure 8(a) plots the average response time. The content-aware batching approach improves the average response time by 15%. It significantly reduces the average response time form 687ms to 584ms. Figure 8(b) shows that throughput comparison between Regular and Batching approach. Batching improves the throughput by 14%. These results demonstrate that using content-aware batching can significantly improve the application performance. Figure 8(c) shows cache hit rate comparison. The cache hit rate due to the batching approach is 3.5 times higher than no batching scenario.

The SLA of the system is set to 1000 millisecond. For the scenario of slightly overloaded system, we use the 1200 concurrent users.

Figure 9(a) plots the average response time. The average response time of no batching scenario is 1128ms. It indicates the system is overloaded. By using content-aware batching, the average

response time is significantly reduced form $1128ms$ to $917ms$. The content-aware batching approach improves the average response time by 19%.

Figure 9(b) shows that throughput comparison between batching and no batching scenarios. The content-aware batching approach improves the throughput by 17%. Figure 8(c) shows cache hit rate comparison. The cache hit rate due to the batching approach is 3 times higher than no batching scenario.

These two experiments demonstrate the effectiveness of the content-aware batching approach. The content-aware batching is able to improve both average response time and throughput for both underloaded and slightly overloaded systems.

7 Conclusion

Most existing works neglect the content characteristics of workload, which actually play an important role in application performance. In this paper, we have proposed and developed a novel approach: content-aware and self-tuning batching integrated with DVFS to improve the performance of applications and energy efficiency of servers simultaneously. As demonstrated by experimental results based on the testbed implementation, its main contributions are the precise control of batching interval length to improve performance and avoid SLA violations. Furthermore, we have proved that the batching system controlled by FMPC and EFC works well when there are stationary and dynamic workloads. Our future work will integrate some heterogeneous workloads within content-aware and self-tuning batching technique in a virtualized server cluster.

Acknowledgement

This research was supported in part by U.S. National Science Foundation CAREER Award CNS-0844983.

References

[1] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. In *Proc. IEEE Int'l Workshop on Workload Characterization (WWC)*, pages 3 – 13, 2002.

[2] Y. Chen, B. Yang, A. Abraham, and L. Peng. Automatic design of hierarchical takagisugeno type fuzzy systems using evolutionary algorithms. In *IEEE Transactions on Fuzzy Systems*, 2007.

[3] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaihk, M. Surendra, and A. Tantawi. Modeling differentiated services of multi-tier web applications. In *Proc. IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2006.

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

[5] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proc. ACM SIGMETRICS*, 2009.

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

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

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

[9] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2010.

[10] A. Kochut and K. Beaty. On strategies for dynamic resource management in virtualized server environments. In *Proc. IEEE Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2007.

[11] P. Lama and X. Zhou. Efficient server provisioning for end-to-end delay guarantee on multi-tier clusters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2009.

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

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

[14] J. C. B. Leite, D. M. Kusic, D. Mossé, and L. Bertini. Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. In *Proc. IEEE Int'l Conf. on Autonomic computing (ICAC)*, 2010.

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

[16] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *Proc. ASPLOS*, 2008.

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

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

[19] R. Wang, Y. Deaver and X. Wang. Virtual batching: Request batching for energy conservation in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality-of-Service (IWQoS)*, 2010.

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

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

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

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

[24] J. Wei and C. Xu. eqos: Provisioning of client-perceived end-to-end qos guarantees in web servers. *IEEE Trans. Comput.*, 55, 2006.