Coordinating Power Control and Performance Management for Virtualized Server Clusters

Xiaorui Wang, *Member*, *IEEE*, and Yefu Wang, *Student Member*, *IEEE*

Abstract—Today's data centers face two critical challenges. First, various customers need to be assured by meeting their required service-level agreements such as response time and throughput. Second, server power consumption must be controlled in order to avoid failures caused by power capacity overload or system overheating due to increasing high server density. However, existing work controls power and application-level performance separately, and thus, cannot simultaneously provide explicit guarantees on both. In addition, as power and performance control strategies may come from different hardware/software vendors and coexist at different layers, it is more feasible to coordinate various strategies to achieve the desired control objectives than relying on a single centralized control strategy. This paper proposes Co-Con, a novel cluster-level control architecture that coordinates individual power and performance control loops for virtualized server clusters. To emulate the current practice in data centers, the power control loop changes hardware power states with no regard to the application-level performance. The performance control loop is then designed for each virtual machine to achieve the desired performance even when the system model varies significantly due to the impact of power control. Co-Con configures the two control loops rigorously, based on feedback control theory, for theoretically guaranteed control accuracy and system stability. Empirical results on a physical testbed demonstrate that Co-Con can simultaneously provide effective control on both application-level performance and underlying power consumption.

Index Terms—Power control, power management, performance, virtualization, server clusters, data centers, control theory.

1 Introduction

In recent years, power control (also called power capping) has become a serious challenge for data centers. Precisely controlling power consumption is an essential way to avoid system failures caused by power capacity overload or overheating due to increasing high server density (e.g., blade servers). Since a cluster of high-density servers may share the same power supplies when they locate in the same rack enclosure, cluster-level power control is of practical importance because an enclosure may need to reduce its power budget at runtime in the event of thermal emergency or a partial power supply failure. In addition, although power provisioning is commonly used in data centers, strictly enforcing various physical and contractual power limits [2] is important because many data centers are rapidly expanding the number of hosted servers while a capacity upgrade of their power distribution systems has lagged far behind. As a result, it can be anticipated that high-density server enclosures in future data centers may often need to have their power consumption dynamically controlled under tight constraints. Furthermore, power control can also reduce operating costs by having improved performance/power ratio [3], which is critical to today's data centers, as the annual data center energy consumption in the US is estimated to grow to over 100 billion kWh at a cost of $7.4 billion by 2011 [4].

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996-2100. E-mail: {xwang, ywang38}@eecs.utk.edu.*

Manuscript received 25 Apr. 2009; revised 20 Sept. 2009; accepted 25 Sept. 2009; published online 29 Apr. 2010.
Recommended for acceptance by R. Iyer.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-04-0182.
Digital Object Identifier no. 10.1109/TPDS.2010.91.

An effective way to control server power consumption is to dynamically transition the hardware components from high power states to low power states whenever the system power consumption exceeds a given *power budget*. However, the situation is more complicated when we consider the hosted commercial computing services running on the servers. An important goal of data centers is to meet the service-level agreements (SLAs) required by customers, such as response time and throughput. SLAs are important to operators of data centers because they are key performance indicators for customer service and are part of the customer commitments. Degrading the performance of the hardware components solely for the consideration of power may have a negative impact on the SLAs. For example, the transition of processor power states in a server can be used not only to control power consumption, but it also has significant influence on the response time of a hosted web service on the server. Therefore, the power consumption and application-level performance of computing servers must be controlled in a holistic way so that we can have explicit guarantees on both of them.

Existing solutions to power and performance control for enterprise servers approach the problem in two separate ways. Performance-oriented solutions at the system level focus on using power as a knob to meet application-level SLAs while reducing power consumption in a best effort manner [5], [6], [7], [8], [9]. However, those solutions do not have any explicit monitoring and control of power consumption. Consequently, they may violate specified power constraints, and thus, result in undesired server shutdown. On the other hand, power-oriented solutions treat power as the first-class control target by adjusting hardware power states with no regard to the SLAs of the application services running on the servers [10], [3], [11], [12]. As a result, existing solutions *cannot* simultaneously provide explicit guarantees on both application-level performance and underlying power consumption.

Simultaneous power and performance control faces several major challenges. First, in today's data centers, a power control strategy may come directly from a server vendor (e.g., IBM) and is implemented in the service processor firmware [3], without any knowledge of the application software running on the server. On the other side, a performance controller needs to be implemented in the application software in order to monitor and control the desired application-level performance, without direct access to the system hardware. Therefore, it may not be feasible to have a single centralized controller that controls both application-level SLAs and underlying server power consumption [13]. Instead, a coordinated control strategy is more preferable. Second, many existing control strategies in the system were designed with the assumption that the system is controlled exclusively by this strategy. For example, some servers may come with an already implemented power control loop from the vendor. In that case, other control loops (e.g., performance) need to be designed accordingly to achieve the desired overall control functions. Third, multiple high-density servers located within the same rack enclosure share common power supplies and may have different workload intensities. As a result, cluster-level control solutions are needed to allow shifting of power and workload for optimized system performance. Fourth, as many data centers start to adopt virtualization technology for resource sharing, application performance of each virtual machine (instead of the entire server) needs to be effectively controlled. Finally, as both power and performance are critical to data centers, control accuracy and system stability must be analytically assured.

In this paper, we propose Co-Con, a novel coordinated control architecture that provides explicit guarantees on both power and application-level performance for virtualized high-density servers that share the same power supplies in a cluster (e.g., an enclosure). Co-Con is designed based on well-established control theory for theoretically guaranteed control accuracy and system stability. Specifically, the contributions of this paper are fourfold:

- We design a coordinated control architecture that is composed of a cluster-level power control loop and a performance control loop for each virtual machine. We configure different control loops to achieve the desired power and performance control objectives.
- We model the application performance of virtual machines and design the performance controller based on the model. We analyze the impact of power control on the performance model, and prove the control accuracy and system stability of the performance controller even in the face of model variations.
- We implement the cluster-level power control loop and the performance control loops on a physical testbed and provide the implementation details of each component in our control architecture.
- We present empirical results to demonstrate that our control solution can effectively control both the power consumption of a cluster and the application performance of all the virtual machines in the cluster.

The rest of the paper is organized as follows: Section 2 introduces the proposed Co-Con control architecture. Section 3 presents the modeling, design, and analysis of the performance controller. Section 4 introduces the cluster-level power controller. Section 5 provides the implementation details of each component in the control loops. Section 6 presents our empirical results conducted on a physical testbed. Section 7 highlights the distinction of our work by discussing the related work. Finally, Section 8 concludes the paper.

2 Co-Con: Coordinated Control Architecture

In this section, we give a high-level description of the Co-Con coordinated control architecture.

An important feature of Co-Con is that it relies on feedback control theory as a theoretical foundation. In recent years, control theory has been identified as an effective tool for power and performance control due to its analytical assurance of control accuracy and system stability. Control theory also provides well-established controller design approaches, e.g., standard ways to choose the right control parameters, such that exhaustive iterations of tuning and testing can be avoided. Furthermore, control theory can be applied to quantitatively analyze the control performance (e.g., stability, settling time) even when the system model changes significantly due to various system uncertainties such as workload variations. This rigorous design methodology is in sharp contrast to heuristic-based adaptive solutions that heavily rely on extensive manual tuning.

As shown in Fig. 1, Co-Con is a two-layer control solution, which includes a cluster-level power control loop and a performance control loop for each virtual machine.

2.1 Cluster-Level Power Control

The cluster-level power controller dynamically controls the *total* power consumption of all the servers in the cluster by adjusting the CPU frequency of each server with Dynamic Voltage and Frequency Scaling (DVFS). We choose to have cluster-level power control because the total power consumption of a cluster (e.g., an enclosure) needs to stay below the capacity of the shared power supplies. In addition, as shown in the previous work [11], [14], cluster-level *power shifting* among different servers can lead to better system performance. There are several reasons for us to use processor DVFS as our actuation method in this work. First, processors commonly contribute the majority of total power consumption of a server [15]. As a result, the processor power difference between the highest and lowest power states is large enough to compensate for the power variation of other components, and thus, can provide an effective way for server power control. Second, DVFS has small overhead while some other actuation methods like turning on/off servers may lead to long delays and even requires human intervention for security check or service configurations, making them less feasible to be used in a real data center, especially when application response time is a concern. Finally, most today's processors support frequency scaling by DVFS or clock modulation [3] while there are still very few real disks or memory devices that are commercially available and allow power throttling. We plan

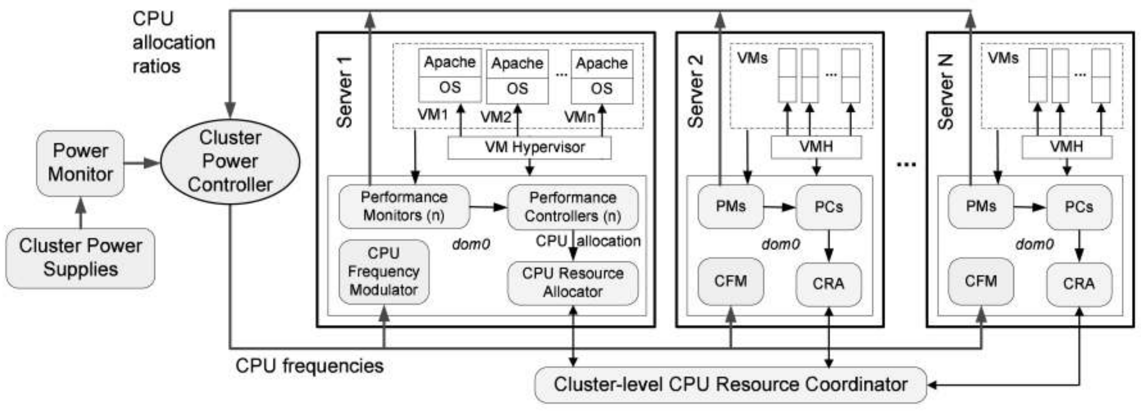

Fig. 1. Coordinated power and performance control architecture for virtualized server clusters. Co-Con controls both power and application-level performance by coordinating a cluster-level power controller and a performance controller for each virtual machine.

to extend our control architecture to include other actuation methods in our future work.

The cluster-level power control loop is invoked periodically as follows: 1) The cluster-level power monitor (e.g., a power meter) measures the total power consumption of all the servers in the last control period and sends the value to the power controller. The total power consumption is the *controlled variable* of the control loop. 2) Based on the difference between the measured power consumption and the desired power set point, the power controller computes the new CPU frequency level for the processors of each server, and then sends the level to the CPU frequency modulator on each server. The CPU frequency levels are the *manipulated variables* of the control loop. 3) The CPU frequency modulator on each server changes the DVFS level of the processors accordingly. The power controller provides an interface to assign weights to different servers. For example, the CPU allocation ratio of each server (i.e., percentage of CPU resource allocated to all the virtual machines on the server) indicates the CPU utilization of the server in the last control period, and can be provided to the controller as weight to give more power to a server whose ratio is higher than the average.

2.2 Performance Control

In the second layer, for every virtual machine on each server, we have a performance controller that dynamically controls the application performance of the virtual machine by adjusting the CPU resource (i.e., fraction of CPU time) allocated to it. In this paper, as an example SLA metric, we control the response time of the web server installed in each virtual machine, but our control architecture can be extended to control other SLAs. In addition, we control the average response time to reduce the impact of the long delay of any single web request. However, our control architecture can also be applied to control the worst-case or 90-percentile response time. We assume that the response time of a web server is independent from that of another web server, which is usually true because they may belong to different customers. Hence, we choose to have a performance control loop for each virtual machine. Our control solution can be extended to handle multitier web services by modeling the correlations between different tiers, which is part of our future work. A cluster-level resource coordinator is designed to utilize the *live migration* [16] function to move a virtual machine from a server with too much workload to another server for improved performance guarantees.

The performance (i.e., response time) control loop on each server is also invoked periodically. The following steps are executed at the end of every control period:

1. The performance monitor of each virtual machine measures the average response time of all the web requests (i.e., controllable variable) in the last control period, and then sends the value to the corresponding performance controller.
2. The controller of each virtual machine computes the desired amount of CPU resource (i.e., manipulated variable) and sends the value to the CPU resource allocator. Steps 1 and 2 repeat for all the virtual machines on the server.
3. The CPU allocator calculates the total CPU resource requested by the performance controllers of all the virtual machines. If the server can provide the total requested resource, all the requests are granted in their exact amounts. Unallocated resource will not be used by any virtual machines in this control period and can be used to accept virtual machine migration. If the requested resource is more than the available resource, one or more selected virtual machines (running low-priority web services) will be given less resource than requested. If this situation continues for a while, a migration request is sent to the cluster-level CPU resource coordinator to move the selected virtual machines to other servers.
4. The cluster-level coordinator tries to find other servers with enough resource and migrates the virtual machines.

Note that the focus of our paper is the coordination of the performance and power controllers. Therefore, we adopt a simple first-fit algorithm to find the first server with enough resource for each virtual machine that needs to be migrated. More advanced algorithms (e.g., [17], [18]) can be easily integrated into our control architecture to maintain load balancing among different servers while minimizing migration overhead. If the coordinator cannot find any servers with enough resource, the migration request is declined. In that case, admission control can be enforced to reject selected requests for some low-priority virtual machines. As a result, the desired response times can still be guaranteed for high-priority virtual machines.

2.3 Coordination of Control Loops

Clearly, without effective coordination, the two control loops (i.e., power and performance) may conflict with each other. The CPU frequency manipulated by the power controller will have a direct impact on the application performance of all the virtual machines on the server. The CPU resource allocated by the performance control loops may influence the system power consumption as well. To achieve the desired control functions and system stability, one control loop, i.e., the primary loop, needs to be configured with a control period that is longer than the settling time of the other control loop, i.e., the secondary loop. As a result, the secondary loop can always enter its steady state within one control period of the primary control loop. The two control loops are thus decoupled and can be designed independently. The impact of the primary loop on the secondary loop can be modeled as variations in its system model, while the impact of the secondary loop on the primary loop can be treated as system noise. As long as the two control loops are stable individually, the whole system is stable. In our design, we choose the power control loop as the primary loop for three reasons. First, model variations may cause the secondary loop to severely violate its set point, which is less desirable for the power loop because power limit violations may lead to the shutdown of an entire cluster. Second, the impact of CPU frequency on application performance is usually more significant than the impact of CPU resource allocation on power consumption, and thus, is more appropriate to be modeled as model variations than system noise. Finally, the secondary control loop needs to be designed based on the primary loop. In this paper, the response time control loop is designed based on the power control loop. In Co-Con, the control period of the response time control loop is determined based on the estimated execution time of typical web requests such that multiple requests can be processed in a control period. The control period of the power control loop is determined based on the settling time of the response time control loop, as analyzed in Section 3.3. The concrete values of the control periods used in our experiments are provided in Section 5.

Since the core of each control loop is its controller, we introduce the design and analysis of the two controllers in the next two sections, respectively. The implementation details of other components are provided in Section 5.

3 Response Time Controller

In this section, we first introduce the system modeling and design of the response time controller. We then present control analysis to configure the response time control loop to coordinate with the power control loop.

3.1 System Modeling

We first introduce some notations. T_r, the control period, is selected to include multiple web requests. $r(k)$ is the average response time of all the web requests of the virtual machine in the kth control period. R_s is the set point, i.e., the desired response time for the virtual machine. $a(k)$ is the amount of CPU resource allocated to the virtual machine in the kth control period. The virtual machine hypervisor uses $a(k)$ (e.g., the *cap* parameter in Xen [19]) to assign CPU resource to the virtual machine. In the kth control period, given the current average response time $r(k)$, the control goal is to dynamically choose a CPU resource amount $a(k)$ such that $r(k+1)$ can converge to the set point R_s after a finite number of control periods.

The relationship between $r(k)$ and $a(k)$ is normally nonlinear due to the complexity of computer systems. Since nonlinear control can lead to unacceptable runtime overhead, a standard way in control theory to handle such systems is to linearize the system model by considering the deviations of those variables from their respective operating points [20]. Therefore, instead of directly using $r(k)$ and $a(k)$ to model the system, we build a linear model by using their differences with their operating points, r and a, which are defined as the typical values of $r(k)$ and $a(k)$, respectively. Specifically, the controlled variable in the system model is $\Delta r(k) = r(k) - r$. The desired set point is $\Delta R_s = R_s - r$. The control error is $e(k) = \Delta R_s - \Delta r(k)$. The manipulated variable is $\Delta a(k) = a(k) - a$. An example way to choose operating points in the system is to select the middle value of a typical range of CPU resource amount as a, and then measure the resultant average response time as r. To model the dynamics of the controlled system, namely the relationship between the controlled variable (i.e., $\Delta r(k)$) and the manipulated variable (i.e., $\Delta a(k)$), we use a standard approach to this problem called *system identification* [21]. Instead of trying to build a physical equation that is usually unavailable for computer systems, we infer the relationship by collecting data in experiments and establish a statistical model based on the measured data.

Based on control theory, we use the following standard difference equation to model the controlled system:

$$\Delta r(k) = \sum_{i=1}^{m_1} b_i \Delta r(k-i) + \sum_{i=1}^{m_2} c_i \Delta a(k-i), \tag{1}$$

where m_1 and m_2 are the orders of the control output (i.e., $\Delta r(k)$) and control input (i.e., $\Delta a(k)$), respectively. b_i and c_i are control parameters whose values need to be determined by system identification.

For system identification, we need to first determine the right orders for the system, i.e., the values of m_1 and m_2 in the difference equation (1). The order values are normally a compromise between model simplicity and modeling accuracy. In this paper, we test different system orders as listed in Table 1. For each combination of m_1 and m_2, we generate a series of control inputs to stimulate the system and then measure the control output in each control period. Our experiments are conducted on the testbed introduced in detail in Section 5. Based on the collected data, we use the *Least Squares Method (LSM)* to iteratively estimate the values of parameters b_i and c_i. The values in Table 1 are the estimated accuracy of the models in terms of Root Mean Squared Error (RMSE). We choose to have the orders of $m_1 = 1$ and $m_2 = 1$ because this combination has a reasonably small error while keeping the orders low, as shown in Table 1. We then use white noise to generate control inputs

TABLE 1
Estimated Errors for Different Model Orders (Smaller Is Better)

	$m_1 = 0$	$m_1 = 1$	$m_1 = 2$
$m_2 = 1$	127.90	71.70	68.08
$m_2 = 2$	105.59	71.62	71.09
$m_2 = 3$	99.32	71.09	67.99

in a random fashion to validate the results of system identification by comparing the actual system outputs and the estimated outputs based on the model from system identification. Fig. 2 demonstrates that the estimated outputs of the selected model are sufficiently close to the measured actual system outputs. Therefore, the *nominal* system model resulted from our system identification is

$$\Delta r(k) = b_1 \Delta r(k-1) - c_1 \Delta a(k-1), \tag{2}$$

where $b_1 = 0.71$ and $c_1 = 6.57$ are control parameters resulted from our experiments with the relative CPU frequency (i.e., the CPU frequency relative to the highest frequency) as 0.73. Note that the real model of the system may be different from the nominal model at runtime due to CPU frequency changes caused by the power control loop. In Section 3.3, we analyze the impact and prove that the system controlled by the controller designed based on the nominal model can remain stable as long as the CPU frequency change is within a certain range.

3.2 Controller Design

The goal of the controller design is to achieve system stability, zero steady-state error, and short settling time when the nominal system model (2) is accurate. Control performance such as system stability can be quantitatively analyzed when the system model varies due to the impact of CPU frequency changes. The analysis is presented in Section 3.3. Following standard control theory [21], we design a *Proportional-Integral-Derivative (PID)* controller to achieve the desired control performance such as system stability and zero steady-state error. The PID controller function in the Z-domain is

$$F(z) = \frac{K_1 z^2 - K_2 z + K_3}{z(z-1)}, \tag{3}$$

where $K_1 = -0.1312$, $K_2 = -0.0948$, and $K_3 = -0.0139$ are control parameters that are analytically chosen based on the pole placement method [21] to achieve the desired control performance. The time-domain form of the controller (3) is

$$\Delta a(k) = \Delta a(k-1) + K_1 e(k) - K_2 e(k-1) + K_3 e(k-2). \tag{4}$$

As shown in (4), the computational overhead of the designed response time controller is just several multiplications and additions. In a server cluster where each virtual machine has a controller, the overhead of response time control is a linear function of the number of virtual machines.

3.3 Control Analysis for Coordination with Power Control Loop

A major contribution of this work is to coordinate different control loops for achieving the desired power and performance control objectives. As introduced before, we assume that the power control loop is designed with no regard to the application performance. Therefore, the design of performance controller needs to account for the impact of the CPU frequency changes caused by the power control loop.

Although the controlled system is guaranteed to be stable when the system model (2) is accurate, stability has to be guaranteed even when the model varies due to CPU frequency changes. We first quantitatively investigate the impact of CPU frequency on the system model by conducting system identification under different CPU frequencies. In our experiments, we find that parameter c_1 in the nominal model (2) changes significantly when the CPU frequency changes, while parameter b_1 remains almost the same with only negligible variations. This can be explained from the systems perspective. Since $a(k)$ is the amount of CPU resource (i.e., fraction of CPU time) allocated to the virtual machine, its contribution to response time change $\Delta r(k)$ is affected by the current CPU frequency. When the processors are running at a lower CPU frequency, more CPU time is needed to achieve the same application performance. Fig. 3 plots the relationship between the parameter c_1 and the relative CPU frequency. The linear regression fits well (with $R^2 = 0.961$) with the curve. Therefore, the actual system model under different CPU frequencies is as follows:

$$\Delta r(k) = b_1 \Delta r(k-1) - c'_1 \Delta a(k-1), \tag{5}$$

where $c'_1 = 1/(0.155f + 0.041)$ is the actual control parameter and f is the current relative CPU frequency. f can be treated as a constant for the performance control loop because its settling time is designed to be shorter than the control period of the power control loop.

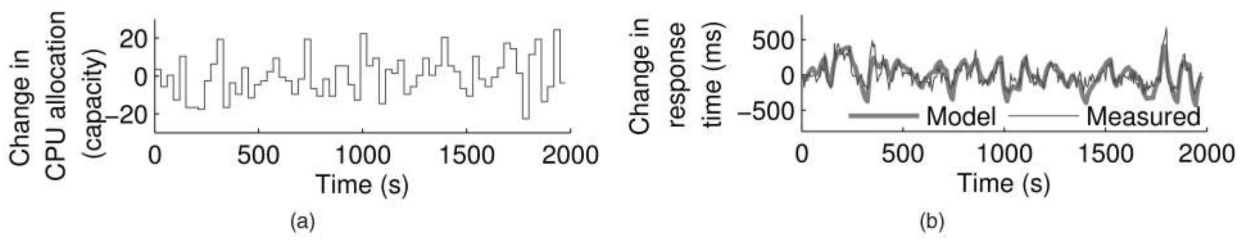

Fig. 2. System model validation with white noise input. (a) Model input. (b) Measured output verifying the estimated model output.

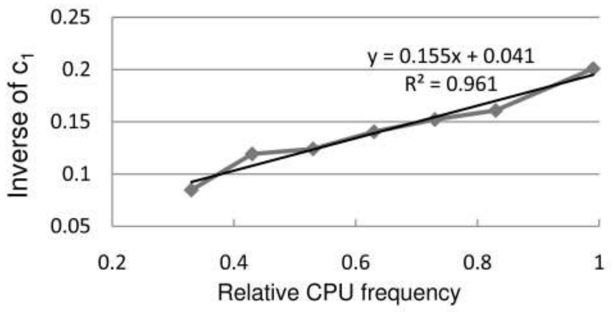

Fig. 3. Parameter c_1 versus CPU frequency.

There are three steps to analyze the stability of the closed-loop system controlled by the response time controller. First, we derive the controller function $F(z)$ presented in (3), which represents the control decision made based on the nominal model (2). Second, we derive the closed-loop system transfer function by plugging the controller $F(z)$ into the *actual* system (5). The closed-loop transfer function represents the system response when the controller is applied to a system whose model is different from the one used to design the controller. Finally, we derive the stability condition of the closed-loop system by computing the poles of the closed-loop transfer function. If all the poles are inside the unit circle, the system is stable when it is controlled by the designed response time controller, even when the real CPU frequency is different from the frequency used to design the controller. We have developed a Matlab program to perform the above analysis automatically. Our results show that the system is guaranteed to be stable as long as the relative CPU frequency is within the range of $[0.19, 1]$. The details are presented in Appendix A.1.2.

We now analyze the variation of system settling time caused by the impact of CPU frequency changes. This analysis is important because we need to select the control period of the power control loop to be longer than the settling time of the performance (i.e., response time) control loop. As a result of control period configuration, the two control loops can be decoupled and designed individually to achieve global system stability. Note that the configuration is a sufficient but not necessary condition for achieving global stability for the coordinated control architecture. The settling time analysis shares the first two steps in the stability analysis presented above. After we derive the closed-loop model, we test the closed-loop system with a step input and compute the output. We then derive the minimum m such that $\forall i > m, |\Delta r(i) - \Delta R_s| < s\Delta R_s$, where s is a threshold used to define the system steady state. In this paper, we use $s = 10\%$. The value of m is the number of control periods for the system to enter the range of ± 10 percent around the set point R_s (i.e., the steady state). The number can then be multiplied with the control period used by the performance control loop (6s in this paper) to get absolute settling time. Fig. 4 plots the resultant settling time when the controlled system is running under different relative CPU frequencies. The controller is designed to have a settling time of 12 s (two control periods) when the relative CPU frequency is 0.73 (the middle vertical dashed line). Fig. 4 shows that the settling time is within 24 s (i.e., four control periods) when the relative CPU frequency is between 0.32 and 1. Therefore, we choose the control period of the power control loop to be 24 s for a trade-off between system response speed and allowed CPU frequency range. To guarantee that the performance control loop always enters the steady state within 24 s, the CPU frequency must be limited within $[0.32, 1]$. The intersection range between this range and the range derived for system stability is used as the frequency constraints for the power control loop. The constraints are used by the power controller to determine the CPU frequency of each server in the cluster.

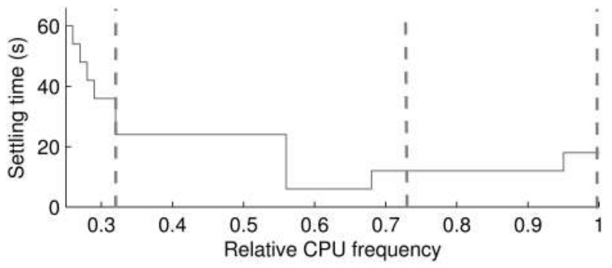

Fig. 4. Settling time versus CPU frequency.

In our analysis, we have also proved that the controlled system has zero steady-state error (i.e., the response time converges to the desired set point) even when the system model varies due to the impact of CPU frequency changes. Similar analysis has also been applied to model the system variations caused by different workload intensities. The analysis is provided in Appendix A.1.1.

4 Cluster-Level Power Controller

In this section, we introduce the cluster-level power controller that is designed without considering application-level performance. The controller provides some configuration interfaces including power set point, control period, control weights, and CPU frequency constraints. Our controller is designed based on the control algorithm presented in [11].

4.1 System Modeling

We first introduce some notations. N is the total number of servers in the cluster. T_p is the control period. $p_i(k)$ is the power consumption of Server i in the kth control period. $cp(k)$ is the total power consumption of all the servers in the cluster, i.e., $cp(k) = \sum_{i=1}^{N} p_i(k)$. P_s is the power set point, i.e., the desired power constraint of the cluster. $f_i(k)$ is the relative CPU frequency of the processors of Server i in the kth control period. $\Delta f_i(k)$ is the frequency change, i.e., $\Delta f_i(k) = f_i(k+1) - f_i(k)$. The control goal is to guarantee that $cp(k)$ converges to P_s within a certain settling time.

Using the system identification approach [21], the power consumption of Server i is modeled as

$$p_i(k+1) = p_i(k) + a_i \Delta f_i(k), \tag{6}$$

where a_i is a generalized parameter whose concrete value may vary for different servers and workloads.

The system model (6) has been validated with pseudorandom digital white noise inputs [21]. The total power consumption, $cp(k+1)$, is the sum of the power consumed by each individual server:

$$cp(k+1) = cp(k) + \mathbf{A}\mathbf{\Delta f}(\mathbf{k}), \tag{7}$$

where $\mathbf{A} = [a_1 \ \dots \ a_N]$ and $\mathbf{\Delta f}(\mathbf{k}) = [\Delta f_1(k) \ \dots \ \Delta f_N(k)]^T$.

4.2 Controller Design

We apply the *Model Predictive Control* (MPC) theory [22] to design the controller. MPC is an advanced control technique that can deal with coupled Multi-Input Multi-Output (MIMO) control problems with constraints on the plant and the actuators. This characteristic makes MPC well suited for power control in a cluster.

An MPC controller optimizes a *cost function* defined over a time interval in the future. The controller uses a system model to predict the control behavior over P control periods, called the *prediction horizon*. The control objective is to select an input trajectory that minimizes the cost function while satisfying the constraints. An input trajectory includes the control inputs in the following M control periods, $\mathbf{\Delta f}(\mathbf{k})$, $\mathbf{\Delta f}(\mathbf{k+1|k})$, $\dots \mathbf{\Delta f}(\mathbf{k+M-1|k})$, where M is called the *control horizon*. The notation $x(k+i|k)$ means that the value of variable x at time $(k+i)T_p$ depends on the conditions at time kT_p, where T_p is the control period. Once the input trajectory is computed, only the first element $\mathbf{\Delta f}(\mathbf{k})$ is applied as the control input to the system. At the end of the next control period, the prediction horizon slides one control period and the input is computed again based on the feedback $cp(k)$ from the power monitor. Note that it is important to recompute the control input because the original prediction may be incorrect due to uncertainties and inaccuracies in the system model used by the controller.

At the end of every control period, the controller computes the control input $\mathbf{\Delta f}(\mathbf{k})$ that minimizes the following cost function:

$$\begin{aligned} V(k) = &\sum_{i=1}^{P} \|cp(k+i|k) - ref(k+i|k)\|^2_{Q(i)} \\ &+ \sum_{i=0}^{M-1} \|\mathbf{\Delta f}(\mathbf{k+i|k}) + \mathbf{f}(\mathbf{k+i|k}) - \mathbf{F_{max}}\|^2_{\mathbf{R(i)}}, \end{aligned} \tag{8}$$

where P is the prediction horizon, M is the control horizon, $Q(i)$ is the *tracking error weight*, and $\mathbf{R(i)}$ is the *control penalty weight vector*.

The first term in the cost function (8) represents the *tracking error*, i.e., the difference between the total power $cp(k+i|k)$ and a reference trajectory $ref(k+i|k)$. The reference trajectory defines an ideal trajectory along which the total power $cp(k+i|k)$ should change from the current value $cp(k)$ to the set point P_s (i.e., power budget of the cluster). The controller is designed to track an exponential reference trajectory [22] so that the closed-loop system behaves like a linear system. By minimizing the tracking error, the closed-loop system will converge to the power set point P_s if the system is stable.

The second term in (8), i.e., the control penalty term, causes the controller to optimize system performance by minimizing the difference between the highest frequency levels, $\mathbf{F_{max}}$, and the new frequency levels, $\mathbf{f}(\mathbf{k+i+1|k}) = \mathbf{\Delta f}(\mathbf{k+i|k}) + \mathbf{f}(\mathbf{k+i|k})$, along the control horizon. The control weight vector, $\mathbf{R(i)}$, is commonly configurable and can be tuned to represent preference among servers. A major difference between our work and the algorithm presented in [11] is that we set the weight of Server i as $ar_i(k)$, the CPU allocation ratio of Server i in the kth control period. Specifically, $ar_i(k) = \sum_{j=1}^{N_i} a_j(k)$, where N_i is the number of virtual machines on Server i and $a_j(k)$ is the amount of CPU resource allocated to the jth virtual machine on Server i. The rationale is that $ar_i(k)$ is the total requested CPU resource to achieve the desired response times for all the virtual machines on Server i. A large $ar_i(k)$ means that the server has already allocated most of its CPU resource (i.e., CPU time), and thus, needs a higher power budget such that the server can run at a higher CPU frequency. If the server is already running at the highest CPU frequency and the available CPU resource is still not enough to achieve the desired response times for all the virtual machines on the server, one or more virtual machines will be moved to other servers by the cluster-level CPU resource coordinator, as introduced in Section 2.

This control problem is subject to two constraints. First, the relative CPU frequency of each server should be within an allowed range. This range is a configurable parameter provided to consider the available hardware frequency range of the specific processors used in each server. In this paper, different from [11], we also need to consider the CPU frequency range derived in Section 3.3 to achieve system stability and the desired settling time for the response time controller. The intersection of the two ranges is used as the CPU frequency constraint for each server. Second, the total power consumption should not be higher than the desired power constraint. The two constraints are modeled as follows:

$$\begin{aligned} &F_{min,j} \le \Delta f_j(k) + f_j(k) \le F_{max,j} \qquad (1 \le j \le N_r) \\ &cp(k+1) \le P_s. \end{aligned}$$

This control problem can be transformed to a standard constrained least-squares problem [22]. The transformation is similar to that of [11] and not shown due to the space limitations. The controller can then use a standard least-squares solver to solve the optimization problem online. In our system, we implement the controller based on the `lsqlin` solver in Matlab. The computational complexity of `lsqlin` is polynomial in the number of servers and the control and prediction horizons. The overhead measurement of `lsqlin` can be found in [23].

The controller is designed based on the *nominal* system model (7), where the value of each a_i is the result of system identification using a typical workload. The *actual* value of a_i in a real system may change for different workloads and is *unknown* at design time. Similar to that of the response time controller, system stability needs to be reevaluated when the power controller is used to control a system with a different model. Stability analysis can be conducted by following the steps presented in Section 3.3. As presented in Appendix A.2, our results show that the system can remain stable as long as the variation of a_i is within the range: $(0, 8.8a_i]$.

5 System Implementation

In this section, we introduce our testbed and the implementation details of the two control loops.

5.1 Testbed

Our testbed includes a cluster of four physical computers that are named *Server1* to *Server4*. A fifth computer named *Storage* is used as the storage server for the Network File System (NFS). Storage is not part of the cluster. All the computers run Fedora Core 8 with Linux kernel 2.6.21. Server1 and Server 3 are each equipped with 4 GB RAM and an AMD Opteron 2222SE processor, which supports eight frequency levels from 1 GHz to 3 GHz. Server2 and Server4 are each equipped with 4 GB RAM and an Intel Xeon X5160 processor, which has four frequency levels from 2 GHz to 3 GHz. All the servers are connected via an Ethernet switch.

Xen 3.1 is used as the virtual machine monitor on all the four servers in the cluster. Each virtual machine (VM) is configured with two virtual CPUs and is allocated 512 MB of RAM. An Apache server is installed in each VM and runs as a virtualized web server. The Apache servers respond to the incoming HTTP requests with a dynamic webpage written in PHP. This example PHP file runs a set of mathematical operations. On each server, Xen is configured to allow and accept VM live migration [16], which is used to move VMs from one server to another without stopping the web services. To support live migration, the VMs are configured to have their virtual hard disks on Storage via NFS. This configuration allows a VM to find its virtual hard disk in the same place before and after a live migration. Initially, Server1 is configured to have three VMs. Server2 is idling with no VMs. In our live VM migration experiment, a VM on Server1 is dynamically moved to Server2, when there is no enough CPU resource on Server1 to meet the response time requirements of all the three VMs.

The client-side workload generator is the Apache HTTP server benchmarking tool (ab), which is designed to stress test the capability of a particular Apache installation. This tool allows users to change the concurrency level, which is the number of requests to perform in a very short time to emulate multiple clients. A concurrency level of 60 is used to do system identification while various concurrency levels (detailed in Section 6) are used in our experiments to stress test our system. The workload generator runs on Storage.

5.2 Control Components

We now introduce the implementation details of each component in our two control loops.

Response time monitor. To eliminate the impact of network delays, we focus on controlling the server-side response time in this paper. The response time monitor is implemented as a small daemon program that runs in each VM. The monitor periodically inserts multiple sample requests into the HTTP requests that are received from the client-side workload generator. Two time stamps are used when a sample request is inserted and when the corresponding response is received. The average time difference is used as the server-side response time, which is sent to the corresponding response time controller.

Response time controller. As introduced in Section 2, there is a response time controller for every VM. The controller implements the control algorithm presented in Section 3. A server-level daemon program written in Perl sequentially runs the controller of every VM on the physical server and computes the total CPU resource requested by all the VMs. If the total requested resource keeps exceeding the available resource of the server for a while, a migration request is sent to the cluster-level resource coordinator. Otherwise, the daemon program calls the CPU resource allocator to enforce the desired CPU allocation based on the CPU resource requests. The control period of the response time controller is selected to be 6 seconds such that multiple HTTP requests can be processed. The maximum allowed response time is 800 ms. To give some leeway to the Apache web servers, the set point of the response time controller (i.e., the desired response time) is selected to be 700 ms.

CPU resource allocator. We use Credit Scheduler [19], Xen's proportional share scheduler, to allocate the available CPU resource. In Credit Scheduler, each VM is assigned a *cap* and a *weight*. The value of cap represents the upper limit of CPU time (in percentage) that can be allocated to the VM. A cap of 40 means that the VM is allowed to consume at most 40 percent time of a core of a dual-core CPU, while a cap of 200 means 100 percent time of both the two cores. The value of weight is used to give preference among different VMs. In this paper, we use cap to allocate CPU resource and use the same fixed weight for all the VMs.

Power monitor. The power consumption of the entire cluster is measured with a WattsUp Pro power meter, which has an accuracy of 1.5 percent of the measured value. The power meter samples the power data every second and then sends the reading to the cluster-level power controller through a system file */dev/ttyUSB0*.

Power controller. In our experiments, the cluster-level power controller is a C program running on Storage. It receives the total power consumption of the cluster from the power monitor and runs the power control algorithm presented in Section 4. The controller parameters include the prediction horizon as 8 and the control horizon as 2. Based on the analysis in Section 3.3, the longest settling time of the response time control loop is 24 seconds when the CPU frequency varies within a wide range. The control period of the power controller is therefore selected to be 24 seconds. Various power set points are used in our experiments.

CPU frequency modulator. In this paper, we use AMD's Cool'n'Quiet technology and Intel's SpeedStep technology to enforce the desired CPU DVFS level. In Linux systems, to change CPU DVFS level, one needs to install the *cpufreq* package and then use the root privilege to write the new level into the system file */sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed*. However, this is more complicated with the Xen virtual machine monitor because Xen lies between the Linux kernel and the hardware, and thus, prevents the kernel from directly modifying the hardware register. In this work, the source code of Xen 3.1 has been hacked to allow the modification.[1]

1. A recently released version of Xen already has built-in support for DVFS.

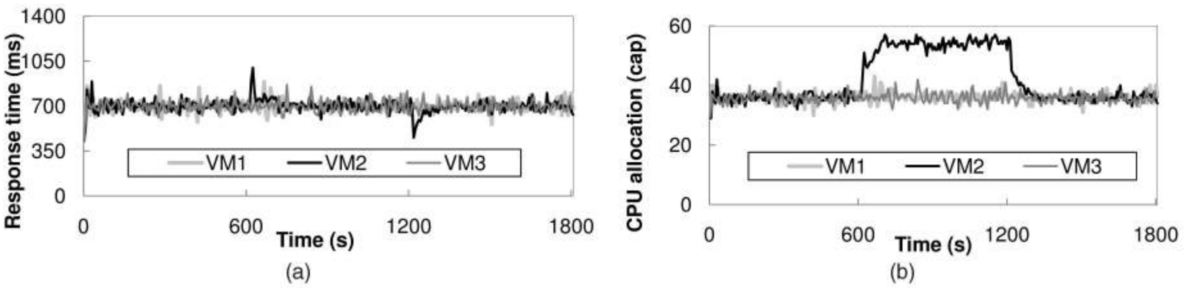

Fig. 5. A typical run of the response time controllers under a workload increase to VM2 from time 600 to 1,200 s. Response time of VM2 is controlled to 700 ms by increasing its CPU resource allocation. (a) Response times. (b) CPU allocation.

The Intel and AMD CPUs used in our experiments support only several discrete frequency levels. However, the new CPU frequency level periodically received from the power controller could be any value that is not exactly one of the supported frequency levels. Therefore, the modulator code must resolve the output value of the controller to a series of supported frequency levels to approximate the desired value. For example, to approximate 2.89 GHz during a control period, the modulator would output a sequence of supported levels: 2.67, 3, 3, 2.67, 3, 3, etc., on a smaller timescale. The detailed modulator algorithm can be found in [3].

6 Empirical Results

In this section, we present our empirical results. We first evaluate the response time controller. We then examine the simultaneous power and response time control provided by Co-Con. Finally, we stress test Co-Con in two important scenarios: power budget reduction at runtime (e.g., due to thermal emergency) and live VM migration.

6.1 Response Time Control

In this experiment, we disable the power controller to evaluate the response time controllers of the three VMs on Server1. Fig. 5 shows a typical run of the response time control loops. At the beginning of the run, the controllers of the three VMs all achieve the desired response time set point, i.e., 700 ms, after a short settling time. At time 600 s, the workload of VM2 increases significantly. This is common in many web applications. For example, breaking news on a major newspaper website may incur a huge number of accesses in a short time. To stress test the performance of our controller in such an important scenario, we increase the concurrency level of VM2 from 60 to 90 between time 600 s and time 1,200 s to emulate the workload increase. The suddenly increased workload causes VM2 to violate its response time limit at time 600 s. The response time controller of VM2 responds to the violation by allocating more CPU resource to VM2. As shown in Fig. 5b, the CPU resource allocated to VM2 is increased from around 35 (i.e., 35 percent CPU time) to around 55. As a result, the response time of VM2 converges to 700 ms again. Without the controller, the response time of VM2 would increase to approximately 1,000 ms, which is significantly longer than the maximum allowed response time (i.e., 800 ms). At time 1,200 s, the concurrency level of VM2 returns to 60, leading to an unnecessarily short response time. The controller of VM2 then reduces the CPU resource allocated to VM2. Note that reducing VM2's resource is necessary because the CPU resource could be allocated to other VMs if they have increased workloads. In Fig. 5, we can see that the response times of VM1 and VM3 are not influenced by the workload variations occurred to VM2, which validates our design choice of having a response time controller for each virtual machine.

As discussed in Section 3.3, our response time model is the result of system identification with a CPU frequency of 0.73 and a concurrency level of 60. To test the robustness of the response time controller when it is applied to a system that is different from the one used to do system identification, we conduct two sets of experiments with wide ranges of CPU frequency and concurrency level, respectively. Fig. 6a shows the average response times (with standard deviations) achieved by the controllers when the CPU frequency varies from 0.63 to 0.93. Fig. 6b shows that less CPU resource is needed to achieve the same response times when CPU frequency is higher. Fig. 7 is the result when the concurrency

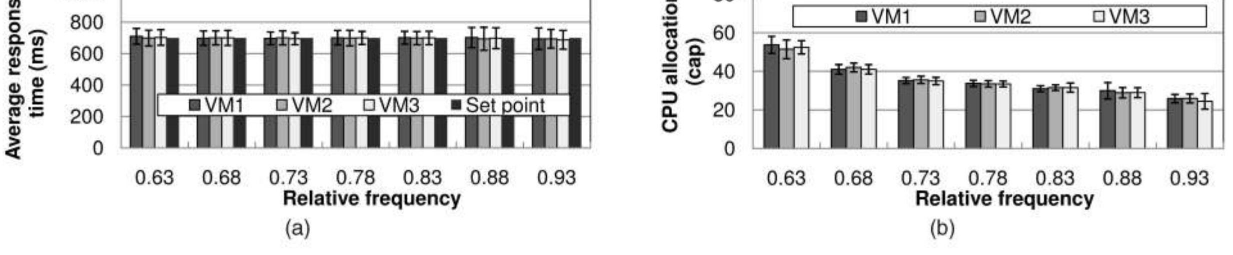

Fig. 6. (a) Response times and (b) CPU allocation of the VMs under different CPU frequencies. The controllers can effectively achieve the set point (700 ms) even when the CPU frequency is changed for power control.

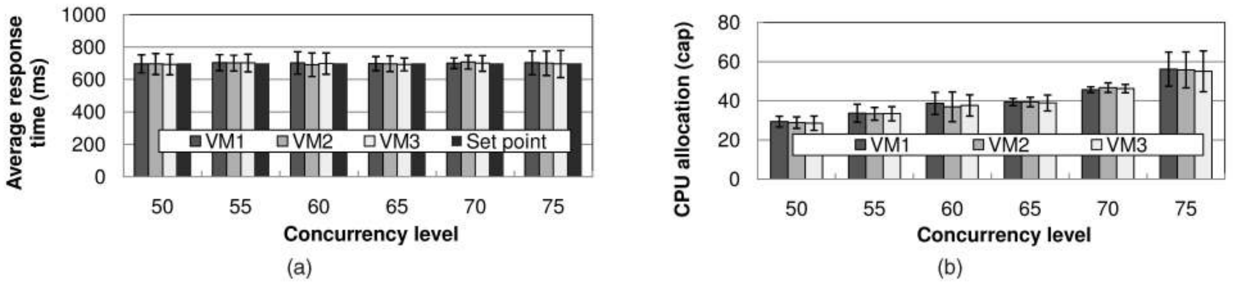

Fig. 7. (a) Response times and (b) CPU allocation of the VMs under different workloads. The controllers can effectively achieve the desired set point (700 ms) even when the workload concurrency level changes.

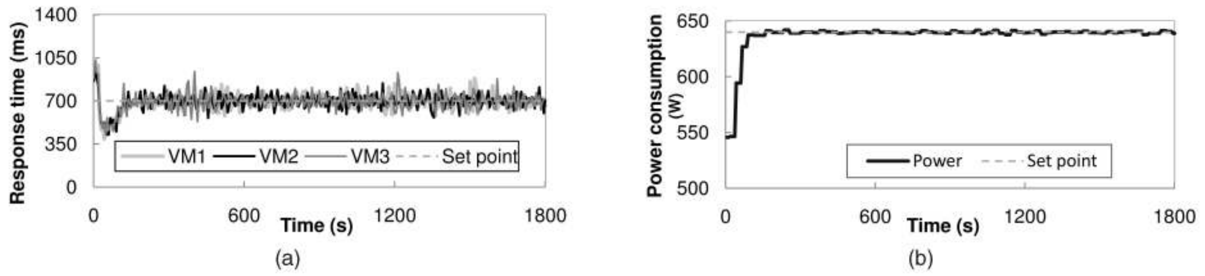

Fig. 8. A typical run of Co-Con. Both power and response times have been successfully controlled to their respective desired set points (700 ms for response times and 640 W for power consumption). (a) Response times. (b) Power consumption of the cluster.

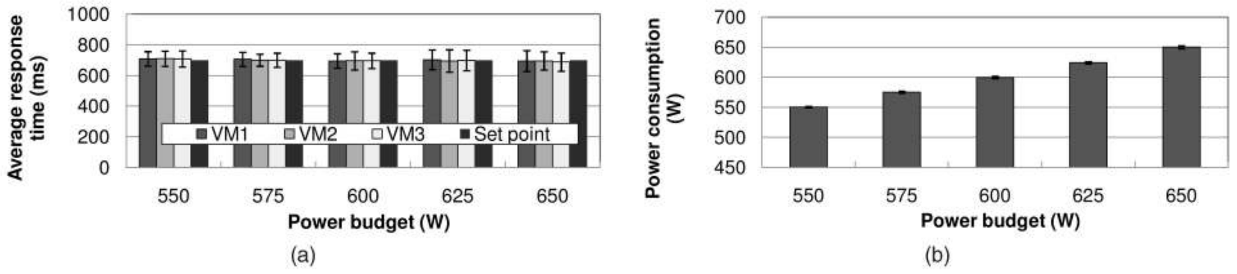

Fig. 9. Response times and power consumption of Co-Con under different power budgets. Co-Con successfully controls both power and response times to the desired set points. (a) Response times. (b) Power consumption of the cluster.

level varies from 50 to 75. The controllers also achieve the desired response times and use more CPU resource when the concurrency level is higher. The two sets of experiments demonstrate that the response time controllers work effectively to achieve the desired response times for all the VMs even when the actual system model is different from the nominal model used to design the controller.

6.2 Coordinated Power and Response Time Control

An important contribution of Co-Con is that it can provide simultaneous control of power and application-level response time for virtualized server clusters. In this experiment, we enable both the response time controllers and the power controller to examine Co-Con's control functions. Fig. 8 shows a typical run of Co-Con. At the beginning of the run, power is lower than the 640 W set point while response times are longer than the 700 ms set point. Co-Con increases the power consumption of the cluster by running the servers at higher CPU frequencies for improved performance. As a result, the response times are reduced significantly and become lower than the set point. Co-Con then adjusts CPU resource allocation of the VMs to achieve the desired response times. Fig. 8a shows the response times of the three VMs on Server1. VMs on other servers have similar results and are not shown due to the space limitations. After the transient state, both the power consumption of the cluster and the response times of all the VMs are precisely controlled to their respective set points.

In a data center, a server cluster may be given different power budgets at different times. For example, a cluster with multiple power supplies may need to reduce its power budget in the case of a partial failure of its power supply subsystem. Therefore, it is important for Co-Con to precisely control power for different power set points and achieve the desired application-level performance at the same time. We test Co-Con under a wide range of power set points (from 550 to 650 W). Fig. 9a plots the average response times of the three VMs on Server1 with standard deviations, while Fig. 9b shows the average power consumption of the cluster with standard deviations. This experiment demonstrates that Co-Con can simultaneously provide explicit guarantees on both power (with very small deviations) and application-level response times.

6.3 Power Budget Reduction at Runtime

In this experiment, we stress test Co-Con in a scenario that is important to data centers. In this scenario, the power

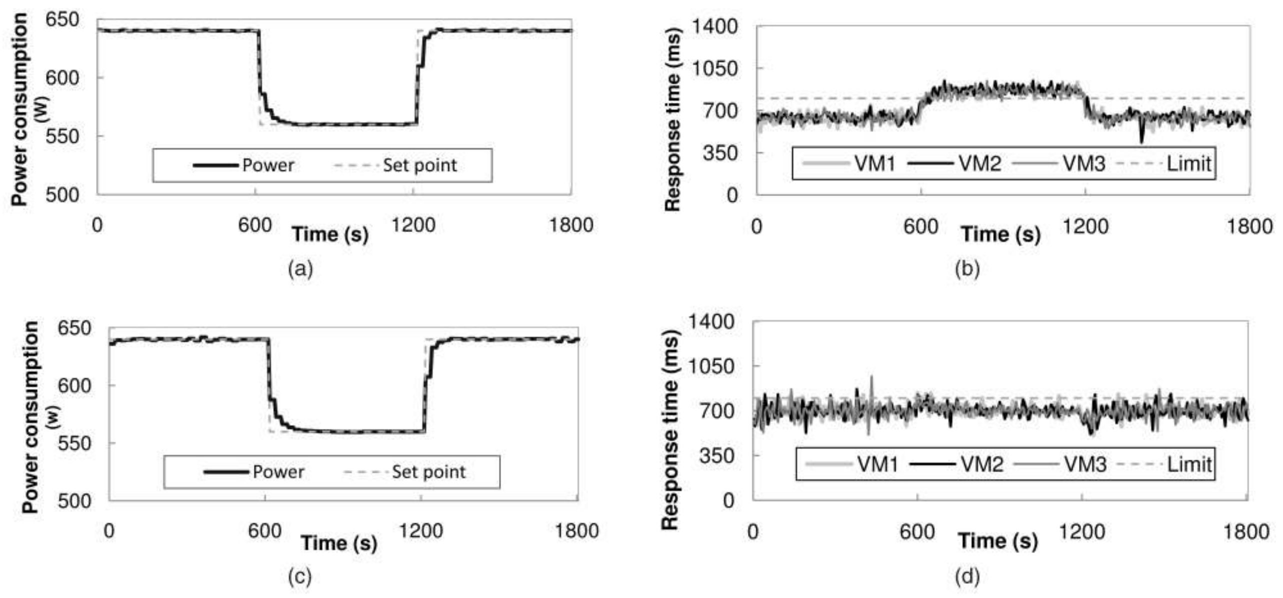

Fig. 10. Comparison between PowerOnly and Co-Con in terms of response times and power consumption, when power budget is reduced at runtime (e.g., due to failures) between time 600 and 1,200 s. PowerOnly violates the response time limit (800 ms) while Co-Con successfully controls both power and response times. (a) Power consumption of the cluster under PowerOnly. (b) Response times under PowerOnly. (c) Power consumption of the cluster under Co-Con. (d) Response times under Co-Con.

budget of the cluster needs to be reduced at runtime due to various reasons such as failures of its cooling systems or its power supply systems. The power budget is then raised back after the problem is fixed.

For comparison, we first run a baseline called PowerOnly, which has only the power controller used in Co-Con, without the response time controllers. In PowerOnly, each VM has a fixed amount of CPU resource that is configured to achieve the desired response time initially. PowerOnly represents a typical power-oriented solution that is designed with no regard to application-level performance. Fig. 10a shows that PowerOnly can effectively reduce power by lowering the CPU frequencies of the servers, when the budget is reduced from 640 to 560 W at time 600 s. However, without explicit performance control, lowering CPU frequency has a negative impact on the application-level response times. As shown in Fig. 10b, PowerOnly has unnecessary short response times before the power budget reduction and violates the maximum allowed limit (i.e., 800 ms) afterward. Note that a solution with only performance control would also fail because it could not effectively reduce power during the budget reduction and so would result in undesired server shutdown.

We then test Co-Con in the same scenario. Fig. 10c shows that Co-Con can effectively control power even though the budget is reduced and raised back. In the meantime, Co-Con dynamically adjusts CPU resource allocated to the VMs to control the application-level response times. As a result, all the VMs achieve the desired response time (i.e., 700 ms), on average, and stay below the maximum allowed limit most of the time, as shown in Fig. 10d. This experiment demonstrates that Co-Con can simultaneously provide robust power and response time guarantees despite power budget reduction at runtime.

6.4 Power and Performance Control during Live Migration of Virtual Machine

As introduced in Section 2, if the total requested CPU resource from all the VMs on a server keeps exceeding the available resource of the server for a certain period of time, the cluster-level resource coordinator will move selected VMs to other servers that have enough CPU resource. In this experiment, we test Co-Con in this scenario. At the beginning of the run, Server1 has three VMs while Server2 has no VMs. At time 600 s, the power budget of the cluster is reduced from 620 to 550 W, which causes all the servers to have lower CPU frequencies, and thus, longer response times. The response time controllers on Server1 try to allocate more CPU resource to their respective VMs to achieve the desired set point. However, due to the lowered CPU frequency, the available CPU resource is no longer enough for all the three VMs. VM3 has the lowest priority web services and is thus selected to receive less resource than requested. Fig. 11a shows that VM3's response time becomes much longer than the set point (700 ms) and also violates the maximum allowed limit (800 ms) after the budget reduction at time 600 s. After the violation continues for 600 seconds, a migration request is sent to the cluster-level CPU resource coordinator at time 1,200 s. The coordinator then moves VM3 to Server2. Fig. 11b shows that VM3 achieves its desired set point after the migration. This experiment demonstrates that Co-Con, as a cluster-level control solution, can distribute workload among virtualized servers in the cluster to effectively control the response times of the VMs even when the CPU resource of a single server is not enough to provide the desired performance guarantee. Previous work [11] has also demonstrated that cluster-level power control can lead to improved application performance by shifting power among servers.

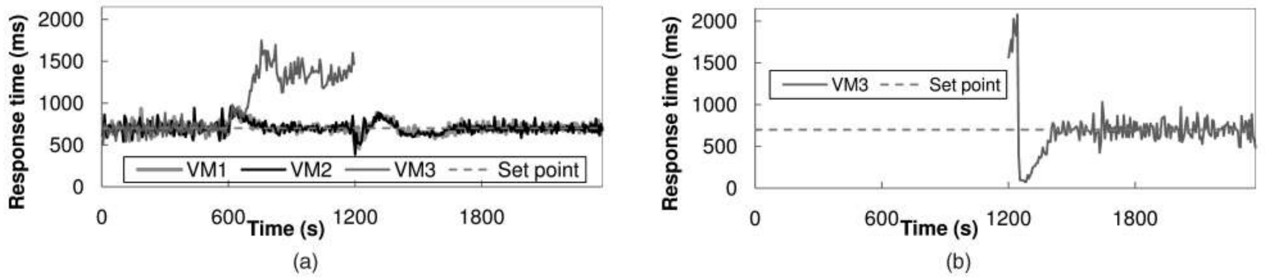

Fig. 11. Response times of the VMs during the live migration of VM3. Co-Con migrates the lowest priority VM (VM3) from Server1 to Server2, when a runtime power budget reduction makes it infeasible to achieve the desired response times for all the VMs on Server1 solely by CPU resource allocation. (a) Response times of the VMs on Server1. (b) Response times of the VMs on Server2.

7 Related Work

Control theory has been successfully applied to control power or cooling for enterprise servers [10], [24]. For example, Lefurgy et al. [3] have shown that a control-theoretic solution outperforms a commonly used heuristic-based solution by having more accurate power control and better performance. Wu et al. [12] manage power by controlling the synchronizing queues in multiclock-domain processors. Wang and Chen [11] develop a MIMO control algorithm for cluster-level power control. Several research projects have also proposed algorithms to control power consumption of main memory [25] and chip multiprocessors [26]. Those projects are different from our work because they control power consumption only, and thus, cannot provide guarantees for application-level performance.

Some prior work has proposed control-theoretic approaches to controlling application-level SLAs by using power as a knob. For example, Horvath et al. [27] use dynamic voltage scaling (DVS) to control end-to-end delay in multitier web servers. Sharma et al. [8] apply control theory to control application-level quality of service requirements. Chen et al. [6] also present a feedback controller to manage the response time in a server cluster. Wang et al. [9] control response times for virtualized servers. Although they all use control theory to manage system performance and reduce power consumption, they do not provide any absolute guarantee on power consumption. Some recent works [28], [29], [30] present heuristic solutions to manage power in virtualized environments. In contrast, we develop control strategies based on rigorous control theory.

Recently, Kephart et al. have proposed a coordinated management strategy to achieve trade-offs between power and performance for a single nonvirtualized server [13]. Kusic et al. present a power and performance management strategy based on lookahead control [31]. In contrast, our coordinated architecture is a cluster-level solution that provides explicit guarantees on both power and performance for virtualized server clusters. Raghavendra et al. [32] present a multilevel coordinated power management framework at the cluster level. In contrast to their work that focuses only on power and system-level resource utilizations, we explicitly control *application-level* SLAs, i.e., the response time of web requests. In addition, while their work is evaluated *only* based on simulations, we present extensive empirical results to demonstrate the efficacy of our control architecture.

Feedback control theory has also been applied to other computing systems. A survey of feedback performance control in various computing systems is presented in [33]. Control techniques have been applied to operating systems [34], real-time systems [35], storage systems [36], networks [37], Internet servers [20], and virtualized computing systems [38]. While all the aforementioned works are designed to control certain performance metrics, our coordinated control architecture can simultaneously provide explicit guarantees on both application-level performance and power consumption.

8 Conclusions

Existing solutions to power and performance control for enterprise servers approach the problem in two separate ways. Performance-oriented solutions at the system level focus on meeting application-level performance requirements while reducing power consumption in a best effort manner. On the other hand, power-oriented solutions treat power as the first-class control target while trying to maximize the system performance. As a result, these solutions *cannot* simultaneously provide explicit guarantees on both application-level performance and underlying power consumption. In this paper, we have presented Co-Con, a cluster-level control architecture that coordinates individual power and performance control loops to explicitly control both power and application-level performance for virtualized server clusters. To emulate the current practice in today's data centers, the power control loop changes hardware power states with no regard to the application-level performance. The performance control loop is then designed for each virtual machine to achieve the desired performance even when the system model varies significantly due to the impact of power control. The two control loops are configured rigorously, based on feedback control theory, for theoretically guaranteed control accuracy and system stability. Empirical results demonstrate that Co-Con can simultaneously provide effective control on both application-level performance and underlying power consumption.

Appendix

A.1 Stability Analysis for Response Time Control Loop

A fundamental benefit of the control-theoretic approach is that it gives us theoretical confidence for system stability, even when the system model may change at runtime. In this appendix, we evaluate the system stability of the response time controller when the nominal system model (i.e., (2)) varies due to two reasons: 1) workload variations and 2) CPU frequency change caused by the power control loop.

A.1.1 Stability Analysis for Workload Variations

We now outline the detailed steps to analyze the stability when the system model changes due to workload variations (e.g., varying concurrency levels).

1. We first get the actual system model by conducting automated system identification on the web server in the target virtual machine. The actual system model is in the following format:

$$\Delta r(k) = b'_1 \Delta r(k-1) - c'_1 \Delta a(k-1), \tag{9}$$

where b'_1 and c'_1 are the actual parameters that may be different from b_1 and c_1 in the nominal model (2). The Z-domain form of the actual system model is

$$G(z) = \frac{\Delta R(z)}{\Delta A(z)} = \frac{-c'_1}{z - b'_1}, \tag{10}$$

where $\Delta R(z)$ and $\Delta A(z)$ are the Z-transform of $\Delta r(k)$ and $\Delta a(k)$, respectively.

2. The controller function $F(z)$ presented in (3) in Section 3.3 represents the control decision made based on the nominal model (2). We then derive the closed-loop system transfer function by plugging the controller into the actual system. The closed-loop transfer function represents the system response when the controller is applied to a system whose model is different from the one used to design the controller. The closed-loop transfer function is

$$\begin{aligned} H(z) &= \frac{F(z)G(z)}{1 + F(z)G(z)} \\ &= \frac{-c'_1 K_1 z^2 + c'_1 K_2 z - c'_1 K_3}{z^3 - (c'_1 K_1 + b'_1 + 1)z^2 + (c'_1 K_2 + b'_1)z - c'_1 K_3}. \end{aligned} \tag{11}$$

3. Finally, we derive the stability condition of the closed-loop system (11). According to control theory, the closed-loop system is stable if all the poles of (11) locate inside the unit circle in the complex space. The poles are calculated as the roots of the denominator in (11), i.e., the following equation:

$$z^3 - (c'_1 K_1 + b'_1 + 1)z^2 + (c'_1 K_2 + b'_1)z - c'_1 K_3 = 0. \tag{12}$$

The stability condition of applying the controller designed based on the nominal model (2) to the actual system with a different system model can be stated as: if the roots of (12) all locate inside the unit circle in the complex space, the controlled system is stable.

Example. In our experiments, our controller is designed based on a nominal workload with a concurrency level of 60. Therefore, our controller function is (3) with parameters as $K_1 = -0.1312$, $K_2 = -0.0948$, and $K_3 = -0.0139$. We now analyze the system stability when the controller is used to control a different workload with a 50 percent higher concurrency level. By following Step 1 given above, we get the actual system model with the real workload as (9) with parameters as $b'_1 = 0.3124$ and $c'_1 = 14.0992$, which are significantly different from b_1 and c_1 (in (2)) used to design the controller. By following Step 2, we derive the closed-loop transfer function of the actual system as

$$G(z) = \frac{-14.0992}{z - 0.3124}. \tag{13}$$

By following Step 3, we substitute b'_1, c'_1, K_1, K_2, and K_3 into (12) and get the closed-loop poles as -0.2736, $0.1882 \pm 0.0920i$. Since all the poles locate inside the unit circle, the closed-loop system has been proved to be stable even the real workload has a 50 percent higher concurrency level. We have developed a script to analyze system stability automatically using numerical methods.

A.1.2 Stability Analysis for CPU Frequency Variations

In Section 3.3, we have outlined the three general steps used to evaluate the system stability when the CPU frequency varies due to power control. The detailed steps are similar to the three steps in Appendix A.1.1, except that CPU frequency variation leads to a different actual system model. As analyzed in Section 3.3, the actual system model under CPU frequency variation is in the following format:

$$\Delta r(k) = b_1 \Delta r(k-1) - c'_1 \Delta a(k-1), \tag{14}$$

where $c'_1 = 1/(0.155f + 0.041)$ is the actual control parameter and f is the current relative CPU frequency. f can be treated as a constant for the performance control loop because its settling time is designed to be shorter than the control period of the power control loop. By following the three steps in Appendix A.1.1, we derive the closed-loop system poles as the roots of the following equation:

$$z^3 - (c'_1 K_1 + b_1 + 1)z^2 + (c'_1 K_2 + b_1)z - c'_1 K_3 = 0. \tag{15}$$

We then substitute c'_1 into (15) and get the following equation:

$$\begin{aligned} z^3 &- \left(\frac{K_1}{0.155f + 0.041} + b_1 + 1\right)z^2 \\ &+ \left(\frac{K_2}{0.155f + 0.041} + b_1\right)z - \frac{K_3}{0.155f + 0.041} = 0. \end{aligned} \tag{16}$$

Since b_1, K_1, K_2, and K_3 all are constants, we can derive the roots of the equation as a function of the relative CPU frequency f. As long as all the roots locate inside the unit circle in the complex space, the controlled system is stable. Therefore, the stability condition of the controlled system is now a function of f. Considering that we have $0 < f \leq 1$ in real computer systems, we use a numerical method as a script to automatically examine all the possible frequencies in the range of $(0, 1]$ with a small step. In each step, the script uses the frequency value to compute the roots of (16) and then checks whether they locate inside the unit circle. By doing that, we have derived the stability range as $0.19 \leq f \leq 1$.

A.2 Stability Analysis for Power Control Loop

In this section, we analyze the stability of the cluster-level power controller when the system model (i.e., system parameter a_i) may change due to different workloads and server configurations. In model predictive control, we say that a system is *stable* if the total power $cp(k)$ converges to the desired set point P_s, that is, $\lim_{k\to\infty} cp(k) = P_s$. Our MPC controller solves a finite horizon optimal tracking problem. Based on optimal control theory, the control decision is a linear function of the current power value, the power set point of the cluster, and the previous decisions for CPU frequency levels.

An important observation from our experiments is that the system has an approximately linear relationship between the power consumption and the CPU frequency within a feasible range, even when the system is running different workloads or on different servers. This has confirmed the same observation reported in [3]. Based on this observation, we mathematically analyze the impact of model variations on system stability. Without loss of generality, we model the real system as

$$cp(k+1) = cp(k) + [g_1 a_1 \quad \dots \quad g_N a_N] \begin{bmatrix} \Delta f_1(k) \\ \vdots \\ \Delta f_N(k) \end{bmatrix}, \qquad (17)$$

where $g_1, g_2, \dots, g_N$ are system gains and are used to model the variations between the actual system model (17) and the nominal model (7). We investigate system stability when a controller designed based on the nominal model (7) is used to control the real system (17). The steps are similar to the three steps in Appendix A.1.1 and skipped due to the space limitations.

Example. We now apply the stability analysis approach to the server cluster used in our experiments, which is composed of four servers. We design the MPC controller by using the nominal parameters $\mathbf{A} = diag[56.1, 66.9, 66.9, 66.9]$. To analyze the system stability when the designed controller is used to control a different workload with a different system model, we derive the range of $\mathbf{G}$ in which all of the poles of the composite system are within the unit circle.

We use two example ways to conduct this proof. First, we can assume that all servers in the cluster have a uniform workload variation, i.e., $\mathbf{G} = g\mathbf{I}$. By following the steps stated above, we derive the range of g as $0 < g \leq 8.8$. It means that a system controlled by the MPC controller designed in our experiments can remain stable as long as its system parameters (i.e., $a_i, \dots, a_N$) are smaller than 8.8 times of the values used to design the controller.

In the case that servers in a cluster have different workload variations, our second way of analyzing the system stability is to assume that only one server has workload variations at a certain time. We conduct the analysis for each server and compute the range of g_i as follows:

$$0 < g_1 \leq 42.1, \qquad (18)$$

$$0 < g_2, g_3, g_4 \leq 29.9. \qquad (19)$$

Therefore, we have proved that a system controlled by our designed controller can remain stable even it has two kinds of workload variations. The system stability with other workload variation patterns can be proved in a similar way. In our stability analysis, we assume that the constrained optimization problem is feasible, i.e., there exists a set of CPU frequency levels within the acceptable ranges that can make the total power consumption equal to its set point. If the problem is infeasible, no control algorithm can guarantee the set point through CPU frequency adaptation. In that case, the system may need to integrate with other adaptation mechanisms (e.g., disk or memory throttling), which is part of our future work.

Acknowledgments

This work is a significantly extended version of a conference paper [1]. This work was supported in part by the US National Science Foundation (NSF) CSR Grant CNS-0720663, NSF CAREER Award CNS-0845390, ONR Grant N00014-09-1-0750, and a Power-Aware Computing Award from Microsoft Research.

References

[1] X. Wang and Y. Wang, "Co-Con: Coordinated Control of Power and Application Performance for Virtualized Server Clusters," *Proc. 17th IEEE Int'l Workshop Quality of Service (IWQoS),* 2009.

[2] X. Fan, W.-D. Weber, and L.A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2007.

[3] C. Lefurgy, X. Wang, and M. Ware, "Power Capping: A Prelude to Power Shifting," *Cluster Computing,* vol. 11, no. 2, pp. 183-195, 2008.

[4] United States Environmental Protection Agency, "Report to Congress on Server and Data Center Energy Efficiency," 2007.

[5] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *Proc. Symp. Operating System Principles (SOSP),* 2001.

[6] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *Proc. ACM SIGMETRICS,* 2005.

[7] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. Second Workshop Power-Aware Computing Systems,* 2002.

[8] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-Aware QoS Management in Web Servers," *Proc. Int'l Real-Time Systems Symp. (RTSS),* 2003.

[9] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-Efficient Response Time Guarantees for Virtualized Enterprise Servers," *Proc. Int'l Real-Time Systems Symp. (RTSS),* 2008.

[10] R.J. Minerick, V.W. Freeh, and P.M. Kogge, "Dynamic Power Management Using Feedback," *Proc. Workshop Compilers and Operating Systems for Low Power,* Sept. 2002.

[11] X. Wang and M. Chen, "Cluster-Level Feedback Power Control for Performance Optimization," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* 2008.

[12] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D.W. Clark, "Formal Control Techniques for Power-Performance Management," *IEEE Micro,* vol. 25, no. 5, pp. 52-62, Sept./Oct. 2005.

[13] J.O. Kephart, H. Chan, R. Das, D.W. Levine, G. Tesauro, F. Rawson, and C. Lefurgy, "Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs," *Proc. Int'l Conf. Autonomic Computing (ICAC),* 2007.

[14] M.E. Femal and V.W. Freeh, "Boosting Data Center Performance through Non-Uniform Power Allocation," *Proc. Int'l Conf. Autonomic Computing (ICAC),* 2005.

[15] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The Case for Power Management in Web Servers," *Power Aware Computing,* Kluwer Academic Publishers, 2002.

[16] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," *Proc. Symp. Networked Systems Design and Implementation (NSDI),* 2005.

[17] A.A. Soror, U.F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Automatic Virtual Machine Configuration for Database Workloads," *Proc. ACM Int'l Conf. Management of Data,* 2008.

[18] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos, "On Controllability and Feasibility of Utilization Control in Distributed Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems (ECRTS),* 2007.

[19] "Credit Scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler, 2010.

[20] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury, *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[21] G.F. Franklin, J.D. Powell, and M. Workman, *Digital Control of Dynamic Systems,* third ed. Addison-Wesley, 1997.

[22] J.M. Maciejowski, *Predictive Control with Constraints.* Prentice Hall, 2002.

[23] C. Lu, X. Wang, and X. Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 6, pp. 550-561, June 2005.

[24] K. Skadron, T. Abdelzaher, and M.R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *Proc. Int'l Symp. High-Performance Computer Architecture (HPCA),* 2002.

[25] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini, "Limiting the Power Consumption of Main Memory," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2007.

[26] Y. Wang, K. Ma, and X. Wang, "Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation," *Proc. Int'l Symp. Computer Architecture (ISCA),* 2009.

[27] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic Voltage Scaling in Multi-Tier Web Servers with End-to-End Delay Control," *IEEE Trans. Computers,* vol. 56, no. 4, pp. 444-458, Apr. 2007.

[28] R. Nathuji and K. Schwan, "VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems," *Proc. Symp. Operating System Principles (SOSP),* 2007.

[29] J. Stoess, C. Lang, and F. Bellosa, "Energy Management for Hypervisor-Based Virtual Machines," *Proc. USENIX Ann. Technical Conf.,* 2007.

[30] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam, "Profiling, Prediction, and Capping of Power Consumption in Consolidated Environments," *Proc. IEEE Int'l Symp. Modeling, Analysis and Simulation of Computers and Telecomm. Systems (MASCOTS),* Sept. 2008.

[31] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and Performance Management of Virtualized Computing Environments via Lookahead Control," *Proc. Int'l Conf. Autonomic Computing (ICAC),* 2008.

[32] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No Power Struggles: Coordinated Multi-Level Power Management for the Data Center," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* 2008.

[33] T.F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE Control Systems,* vol. 23, no. 3, pp. 74-90, June 2003.

[34] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportion Allocator for Real-Rate Scheduling," *Proc. Symp. Operating Systems Design and Implementation (OSDI),* 1999.

[35] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 7, pp. 996-1009, July 2007.

[36] M. Karlsson, C.T. Karamanolis, and X. Zhu, "Triage: Performance Differentiation for Storage Systems Using Adaptive Control," *ACM Trans. Storage,* vol. 1, no. 4, pp. 457-480, 2005.

[37] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. ACM SIGCOMM,* 1991.

[38] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," *Proc. European Conf. Computer Systems (EuroSys),* 2007.

Xiaorui Wang received the BS degree from Southeast University, China, in 1995, the MS degree from the University of Louisville in 2002, and the PhD degree from Washington University in St. Louis in 2006, all in computer science. He is currently an assistant professor in the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville. In 2005, he worked at the IBM Austin Research Laboratory, designing power control algorithms for high-performance computing servers. One of the power control algorithms codesigned by him has been directly adopted in more than 10 IBM System x and System p server models and is the basis of the power capping feature now shipped in more than a billion dollars' worth of IBM servers annually. From 1998 to 2001, he was a senior software engineer and then a project manager at Huawei Technologies Co. Ltd, China, developing distributed management systems for optical networks. He is an author or coauthor of more than 40 refereed publications. He is the recipient of the US National Science Foundation (NSF) CAREER Award in January 2009, the Chancellor's Award for Professional Promise in Research and Creative Achievement from the University of Tennessee in 2009, the Power-Aware Computing Award from Microsoft Research in 2008, and the IBM Real-Time Innovation Award in 2007. He also received the Best Paper Award at the 29th IEEE Real-Time Systems Symposium (RTSS) in 2008. His research interests include power-aware computer systems, real-time embedded systems, and cyber-physical systems. He is a member of the IEEE and the IEEE Computer Society.

Yefu Wang received the BS and MS degrees from Harbin Institute of Technology, China, in 2003 and 2006, respectively. He is currently working toward the PhD degree in computer engineering in the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville. His current research focuses on system-level power management of computing systems, with applications to virtualized enterprise server environments. He is a student member of the IEEE.