# Co-Con: Coordinated Control of Power and Application Performance for Virtualized Server Clusters

Xiaorui Wang and Yefu Wang
*University of Tennessee, Knoxville, TN 37996*
{*xwang, ywang38*}*@utk.edu*

***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. 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 demonstrate that Co-Con can simultaneously provide effective control on both application-level performance and underlying power consumption.**

## I. 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 [1] 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.

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 [2], [3], [4], [5], [6]. 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 [7], [8], [9], [10]. 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 [8], 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 [11]. 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

This work was supported, in part, by 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.

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 four-fold:

- 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 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 II introduces the proposed Co-Con control architecture. Section III presents the modeling, design and analysis of the performance controller. Section IV provides the implementation details of each component in the control loops. Section V presents our empirical results conducted on a physical testbed. Section VI highlights the distinction of our work by discussing the related work. Section VII concludes the paper.

## II. 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 Figure 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.

### A. 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 previous work [9], [12], 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 [13]. 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 can thus 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 [8] while there are still very few real disks or memory devices that are commercially available and allow power throttling. We plan 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) in the last control can be provided to the controller as weight to give more power to a server whose ratio is higher than the average.

In this paper, our power controller is designed based on the control algorithm presented in [9]. The major difference is

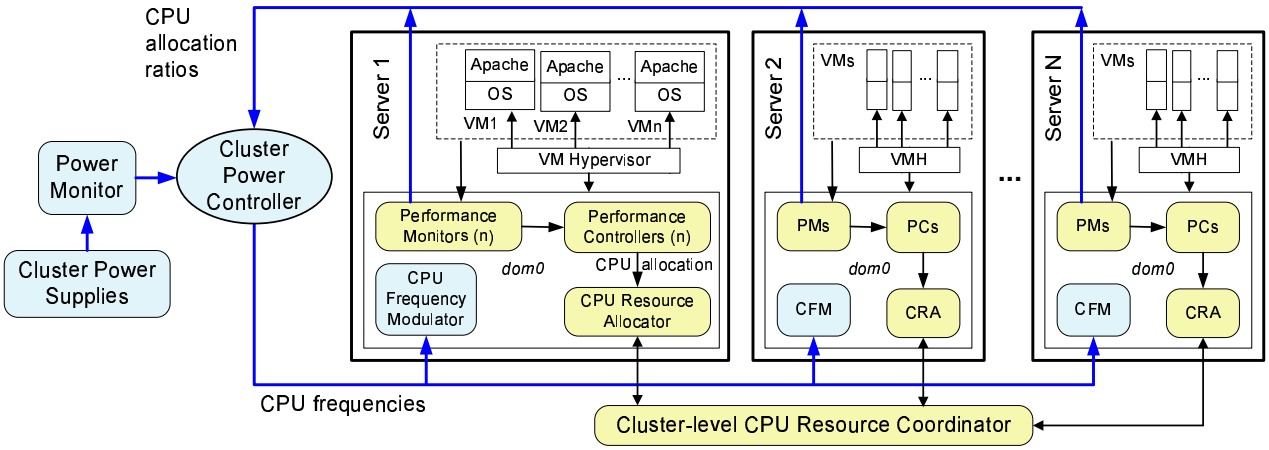



Fig. 1. Coordinated power and performance control architecture for virtualized server clusters. Co-Con controls both power and application-level performance by coordinating a cluster-level power controller and a performance controller for each virtual machine.

that the CPU allocation ratio of each server is used as weight for power allocation in our work, while the CPU utilization of each server is used as weight in [9]. The rationale is that CPU allocation ratio is the total requested CPU resource to achieve the desired response times for all the virtual machines on a server. A large ratio 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.

### *B. 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 multi-tier 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* [14] 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. Please 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.,* [15], [16]) 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 for the low-priority web services.

Clearly, it is important to coordinate different control loops. Today's high-density servers commonly come with implemented power control loops from vendors. Many of those control loops were designed to change hardware power states with no regard to application-level SLAs. On the other hand, those control loops commonly allow certain degree of configurations such as their power budget and control period. To emulate this common practice in today's data centers, our power control loop is adopted from a control algorithm that has no regard to the application-level SLAs [9]. We then design the response time controller accordingly and configure both the power and response time control loops to work together for achieving the desired control objectives. We introduce the design and analysis of the performance controller in the next section. The

implementation details of other components in the control loop are provided in Section IV.

## III. 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.

### A. System Modeling

We first introduce some notation. $T_r$, 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 $k^{th}$ 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 $k^{th}$ control period. The virtual machine hypervisor uses $a(k)$ (*e.g.,* the *cap* parameter in Xen [17]) to assign CPU resource to the virtual machine. In the $k^{th}$ 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 [18]. 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* [19]. 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:

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

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.

TABLE I
Model orders and corresponding root mean squared errors.

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

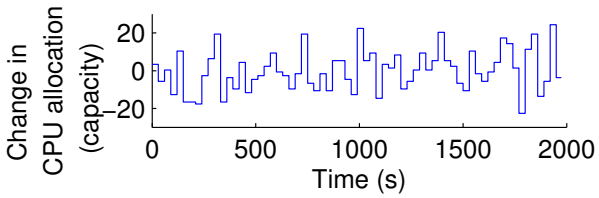
For system identification, we need to first determine the right orders for the system, *i.e.,* the values of $m_1$ and $m_2$ in the difference equation (1). The order values are normally a compromise between model simplicity and modeling accuracy. In this paper, we test different system orders as listed in Table I. 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 IV. 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 I 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 I. We then use white noise to generate control inputs 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. Figure 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:

$$\Delta r(k) = b_1 \Delta r(k-1) - c_1 \Delta a(k-1) \qquad (2)$$

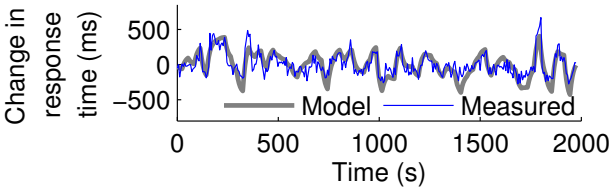
where $b_1 = 0.71$ and $c_1 = 6.57$ are control parameters resulted from our experiments with the relative CPU frequency (*i.e.,* the CPU frequency relative to the highest frequency) as 0.73. Note that the real model of the system may be different from the nominal model at runtime due to CPU frequency changes caused by the power control loop. In Section III-C, 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.

### B. 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 III-C. Following standard control theory [19], we design a *Proportional-Integral-Derivative (PID)* controller to achieve the desired control performance such as system stability and zero steady-




(a) Model input




(b) Measured output verifying the estimated model output

Fig. 2. System model validation with white noise input.

state error. The PID controller function in the Z-domain is:

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

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 [19] to achieve the desired control performance. The time-domain form of the controller (3) is:

$$\Delta a(k) = \Delta a(k-1) + K_1 e(k) - K_2 e(k-1) + K_3 e(k-2) \quad (4)$$

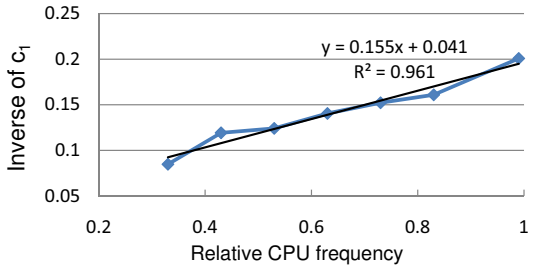



Fig. 3. The relationship between parameter $c_1$ and CPU frequency

### *C. Control Analysis for Coordination with Power Control Loop*

An important contribution of this work is to coordinate different control loops for achieving the desired power and performance control objectives. As introduced before, we assume the power control loop is designed with no regard to the application-level SLAs. 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. Figure 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:

$$\Delta r(k) = b_1 \Delta r(k-1) - c_1' \Delta a(k-1) \quad (5)$$

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.

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 an extended version of this paper [20].

In our analysis, we have also proven that the settling time of the response time control loop is within 24s (*i.e.,* 4 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 24s 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 24s, 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. In addition, we have proven that the controlled system has zero steady state error 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 analyses are provided in the extended version [20].

## IV. System Implementation

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

### A. 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 4GB RAM and an AMD Opteron 2222SE processor, which supports 8 frequency levels from 1GHz to 3GHz. Server2 and Server4 are each equipped with 4G RAM and an Intel Xeon X5160 processor, which has 4 frequency levels from 2GHz to 3GHz. 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 2 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 web page 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 [14], 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 3 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 V) are used in our experiments to stress test our system. The workload generator runs on Storage.

### B. 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 II, there is a response time controller for every VM. The controller implements the control algorithm presented in Section III. 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 800ms. 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 700ms.

**CPU Resource Allocator.** We use Credit Scheduler [17], 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 the VM is allowed to consume at most 40% time of a core of a dual-core CPU, while a cap of 200 means 100% 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% 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 [9]. Based on the analysis in Section III-C, 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.

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 2.89GHz 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 [8].

## V. 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 in an important scenario: power budget reduction at runtime (*e.g.,* due to thermal emergency). More empirical results for Co-Con's performance under different power budgets and in the scenario of live VM migration can be found in the extended version [20].

### A. Response Time Control

In this experiment, we disable the power controller to evaluate the response time controllers of the three VMs on Server1. Figure 4 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.,* 700ms, after a short settling time. At time 600s, 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 600s and time 1200s to emulate the workload increase. The suddenly increased workload causes VM2 to violate its response time limit at time 600s. The response time controller of VM2 responds to the violation by allocating more CPU resource to VM2. As shown in Figure 4(b), the CPU resource allocated to VM2 is increased from around 35 (*i.e.,* 35% CPU time) to around 55. As a result, the response time of VM2 converges to 700ms again. Without the controller, the response time of VM2 would increase to approximately 1000ms, which is signficantly longer than the maximum allowed response time (*i.e.,* 800ms). At time 1200s, 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. From Figure 4, 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.

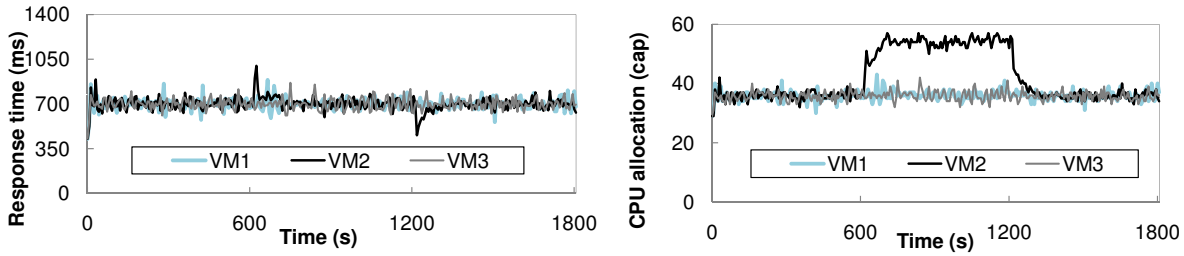
As discussed in Section III-C, our response time model is the result of system identification with a CPU frequency of 0.73 and a concurrency level of 60. To test the robustness of the response time controller when it is applied to a system that is different from the one used to do system identification, we conduct two sets of experiments with wide ranges of CPU frequency and concurrency level, respectively. Figure 5(a) shows the average response times (with standard deviations) achieved by the controllers when the CPU frequency varies from 0.63 to 0.93. Figure 5(b) shows that less CPU resource is needed to achieve the same response times when CPU frequency is higher. Figure 6 is the result when the concurrency 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.

### B. 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 stress test Co-Con's control functions in a scenario that is important to data centers. In this scenario, the power 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.

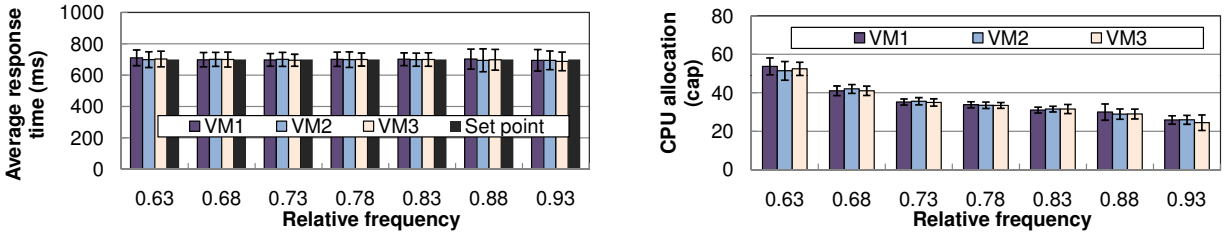
For comparison, we first run a baseline called PowerOnly, which has only the power controller used in Co-Con, without the response time controllers. In PowerOnly, each VM has a fixed amount of CPU resource that is configured to achieve the desired response time initially. PowerOnly represents a typical power-oriented solution that is designed with no regard to application-level performance. Figure 7(a) shows that PowerOnly can effectively reduce power by lowering the CPU frequencies of the servers, when the budget is reduced from 640W to 560W at time 600s. However, without explicit performance control, lowering CPU frequency has a negative impact on the application-level response times. As shown in Figure 7(b), PowerOnly has unnecessary short response times before the power budget reduction and violates the maximum allowed limit (*i.e.,* 800ms) afterward. Please 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. Figure 7(c) 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.,* 700ms) on average and stay below the maximum allowed limit most of the time, as shown in Figure 7(d). This experiment demonstrates that Co-Con can simultaneously provide robust power and response time guarantees despite power budget reduction at runtime.



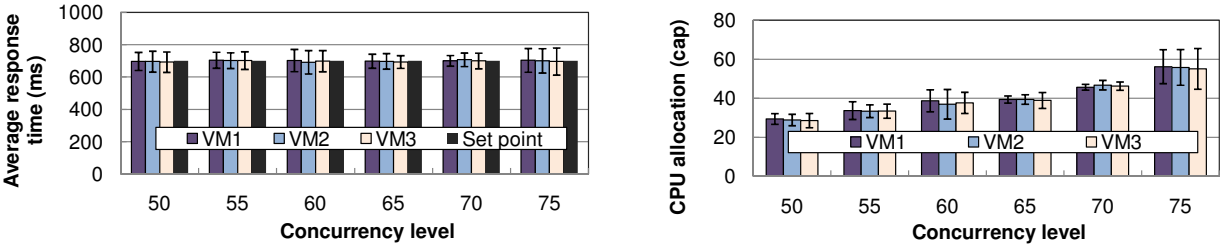

(a) Response times (b) CPU allocation

Fig. 4. A typical run of the response time controllers under a workload increase to VM2 from time 600s to 1200s. Response time of VM2 is controlled to 700ms by increasing its CPU resource allocation.



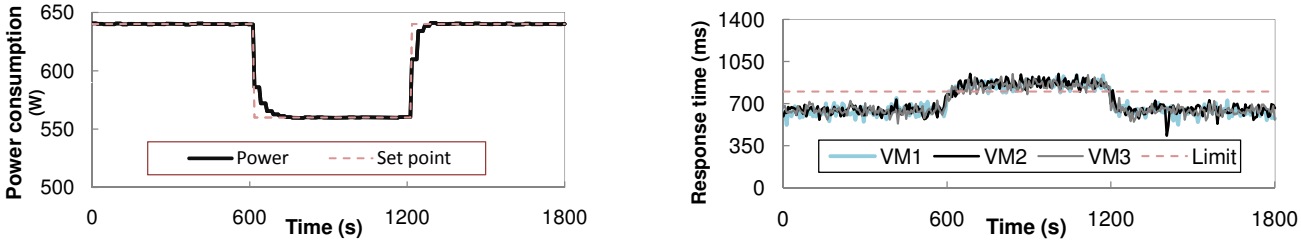

(a) Response times (b) CPU allocation

Fig. 5. Response times and CPU allocation of the VMs under different CPU frequencies. The controllers can effectively achieve the set point (700ms) even when the CPU frequency is changed for power control.

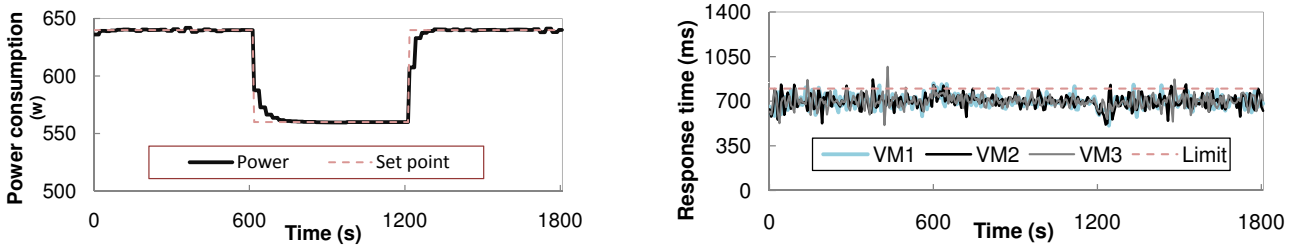



(a) Response times (b) CPU allocation

Fig. 6. Response times and CPU allocation of the VMs under different workloads. The controllers can effectively achieve the desired set point (700ms) even when the workload concurrency level changes.




(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

Fig. 7. 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 600s and 1200s. PowerOnly violates the response time limit (800ms) while Co-Con successfully controls both power and response times.

## VI. Related Work

Control theory has been successfully applied to control power or cooling for enterprise servers [7], [21]. For example, Lefurgy et al. [8] 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. [10] manage power by controlling the synchronizing queues in multi-clock-domain processors. Wang et al. [9] develop a MIMO control algorithm for cluster-level power control. 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. [22] use dynamic voltage scaling (DVS) to control end-to-end delay in multi-tier web servers. Sharma et al. [5] apply control theory to control application-level quality of service requirements. Chen et al. [3] also present a feedback controller to manage the response time in a server cluster. Wang et al. [6] 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 work [23], [24], [25] presents heuristic solutions to manage power in virtualized environments. Li et al. also propose optimization algorithms to manage energy for disks and memory [26]. 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 non-virtualized server [11]. Kusic et al. present a power and performance management strategy based on lookahead control [27]. 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. [28] present a multi-level 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.

## VII. 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. Empirical results demonstrate that Co-Con can simultaneously provide effective control on both application-level performance and underlying power consumption.

## References


[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA*, 2007.

[2] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *SOSP*, 2001.

[3] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *SIGMETRICS*, 2005.

[4] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *PACS*, 2002.

[5] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *RTSS*, 2003.

[6] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in *RTSS*, 2008.

[7] R. J. Minerick, V. W. Freeh, and P. M. Kogge, "Dynamic power management using feedback," in *COLP*, Sep. 2002.

[8] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, 2008.

[9] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *HPCA*, 2008.

[10] 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, 2005.

[11] 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," in *ICAC*, 2007.

[12] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation." in *ICAC*, 2005.

[13] 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*, 2002.

[14] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI*, 2005.

[15] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Automatic virtual machine configuration for database workloads," in *SIGMOD*, 2008.

[16] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos, "On controllability and feasibility of utilization control in distributed real-time systems," in *ECRTS*, 2007.

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

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

[19] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[20] X. Wang and Y. Wang, "Co-con: Coordinated control of power and application performance for virtualized server clusters, Tech Report, UTK," http://pacs.ece.utk.edu/techreports/tech0908.pdf, 2008.

[21] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *HPCA*, 2002.

[22] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multi-tier web servers with end-to-end delay control," *IEEE Transactions on Computers*, vol. 56, no. 4, 2007.

[23] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," in *SOSP*, 2007.

[24] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *USENIX*, 2007.

[25] J. Choi, S. Govindan, B. Urgaonkar, and A. Sivasubramaniam, "Profiling, prediction, and capping of power consumption in consolidated environments," in *MASCOTS*, Sep 2008.

[26] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar, "Performance-directed energy management for main memory and disks," in *ASPLOS*, 2004.

[27] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *ICAC*, 2008.

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