

MIMO Power Control for High-Density Servers in an Enclosure

Xiaorui Wang, Ming Chen, and Xing Fu

Department of Electrical Engineering and Computer Science

University of Tennessee, Knoxville, TN 37996

{*xwang, mchen11, xful*}@eecs.utk.edu

Abstract—Power control is becoming a key challenge for effectively operating a modern data center. In addition to reducing operating costs, 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. Control-theoretic techniques have recently shown a lot of promise for power management because of their better control performance and theoretical guarantees on control accuracy and system stability. However, existing work oversimplifies the problem by controlling a single server independently from others. As a result, at the enclosure level where multiple high-density servers are correlated by common workloads and share common power supplies, power cannot be shared to improve application performance. In this paper, we propose an enclosure-level power controller that shifts power among servers based on their performance needs, while controlling the total power of the enclosure to be lower than a constraint. Our controller features a rigorous design based on an optimal Multi-Input-Multi-Output (MIMO) control theory. We present detailed control problem formulation and transformation to a standard constrained least-squares problem, as well as stability analysis in the face of significant workload variations. We then conduct extensive experiments on a physical testbed to compare our controller with three state-of-the-art controllers: a heuristic-based MIMO control solution, a single-input-single-output (SISO) control solution, and an improved SISO controller with simple power shifting among servers. Our empirical results demonstrate that our controller outperforms all the three baselines by having more accurate power control and up to 11.8% better benchmark performance.

I. INTRODUCTION

In recent years, power has become an important concern for enterprise data centers that host thousands of computing servers and provide outsourced commercial IT services. For example, running a single high-performance 300 W server for one year could consume 2628 KWh of energy, with an additional 748 KWh in cooling this server [2]. The total energy cost for this single server would be \$338 a year without counting the costs of air conditioning and power delivery subsystems [2]. On the other hand, as modern data centers continue to increase computing capabilities for growing business requirements, high-density servers become more and more desirable due to real-estate considerations and better system management features. Currently, the widely used high-density servers from major vendors (*e.g.*, IBM and HP) are *blade servers* which pack traditional multi-board server hardware into a single board. Multiple servers are then put into an *enclosure* (also called a chassis) which is equipped with common power supplies and various ports. The greatest

immediate concerns about blade servers are their power and cooling requirements, imposed by limited space inside the server enclosure. The increasing high server density may also lead to a greater probability of thermal failure and hence require additional energy cost for cooling [3].

As power is the amount of energy transferred per unit of time, an effective way to reduce energy consumption of blade servers is to transition the hardware components from high-power states to low-power states [2]. Most components in a modern blade server such as processors [4], [5], main memory [6], [7] and disks [8] have adjustable power states. Components are fully operational, but consume more power in high-power states while having degraded functionality in low-power states [2]. An energy-efficient server design is to have run-time measurement and control of power to adapt to a given *power budget* so that we reduce the power (then the performance) of the components when the actual power consumption of the server exceeds the budget [5]. As a result of controlling power consumption, we can have the maximum server performance while not using more power than what power supplies can provide. Even though servers in a data center are usually provisioned to have their peak power consumption lower than the capacity of power supplies, it is particularly important for a server enclosure with multiple power supplies to be able to reduce its power budget at runtime in the case of a partial failure of its supply subsystem.

Traditionally, adaptive power management solutions heavily rely on heuristics. Recently, however, feedback control theory has been successfully applied to power control for a single server [4], [9], [10]. For example, recent work [5] has shown that control-theoretic power management outperforms a commonly used heuristic-based solution by having more accurate power control and better application performance. The benefit of having control theory as a theoretical foundation is that we can have (1) standard approaches to choosing the right control parameters so that exhaustive iterations of tuning and testing are avoided; (2) theoretically guaranteed control performance such as accuracy, stability, short settling time, and small overshoot; and (3) quantitative control analysis when the system is suffering unpredictable workload variations. This rigorous design methodology is in sharp contrast to heuristic-based adaptive solutions that rely on extensive empirical evaluation and manual tuning [11].

While control-theoretic power management for a single server has shown significant promise, its single-input-single-output (SISO) control model may not be sufficient for power management in an enclosure environment. First, we need

multi-input-multi-output (MIMO) control algorithms to control the power consumptions of *multiple* servers in an enclosure simultaneously by manipulating the performance setting of each server. Second, the servers in an enclosure are usually *coupled* together due to the fact that they often run the same application service or share common power supplies at the enclosure level. For example, multi-tier web applications usually have front-end HTTP servers for HTTP parsing and response generation while the second-tier application servers and the third-tier database servers can provide business logic and data service, respectively [12]. Throttling one server independently from others will make it the bottleneck, and so unnecessarily degrade the performance of the whole system. Third, instead of having a power limit for each individual server, the aggregated power consumption of all the servers in an enclosure is usually limited by the capacity of the common power supplies equipped with the enclosure [13]. Therefore, servers in an enclosure are usually coupled together, and so their power consumption *cannot* be controlled independently from each other. As a result, power management for a server enclosure has become a constrained MIMO control problem.

In this paper, we propose a novel MIMO control algorithm, based on the well-established Model Predictive Control (MPC) theory, to provide a control-theoretic solution for managing the power of *multiple* servers in an enclosure of a data center. Specifically, the contributions of this paper are four-fold:

- We analytically model the power consumption of a server enclosure using system identification and validate the model with white noise inputs.
- We design a MIMO power control algorithm based on MPC control theory to optimize application performance while controlling the total power to be lower than a constraint. We present detailed MPC control problem formulation and transformation to a standard constrained least-squares problem.
- Our rigorous analysis proves that the controller can remain stable even when the system power model varies significantly at runtime.
- We implement our control loop in a physical testbed and present extensive empirical results to demonstrate that our controller outperforms three state-of-the-art power controllers. Our results also show that the designed MIMO power controller can provide desired performance differentiation.

The rest of the paper is organized as follows. Section II introduces the overall architecture of the power control loop. Section III describes system modeling. Section IV discusses the design and analysis of our control algorithm. Section V provides the implementation details of each component in the control loop. Section VI presents the results of our experiments conducted on a physical testbed. Section VII highlights the distinction of our work by discussing the related work. Section VIII concludes the paper.

II. ENCLOSURE-LEVEL POWER CONTROL LOOP

In this section, we provide a high-level description of our power control loop that adaptively manages the power

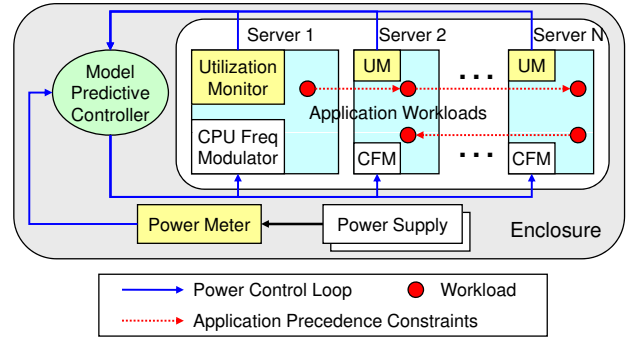


Fig. 1. Power control loop for multiple servers in an enclosure

consumption of a server enclosure by conducting processor frequency scaling (via either Dynamic Voltage and Frequency Scaling (DVFS) or clock modulation) for each server.

There are two reasons for us to use processor frequency scaling as our actuation method in this work. First, processors typically have well-documented interfaces to adjust frequency levels. Second, processors commonly contribute the majority of the total power consumption of small form-factor servers [14]. As a result, the processor power difference between the highest and lowest power/performance states is large enough to compensate for the power variation of other components, and thus provides an effective way to reduce power consumption in the event of a partial failure of the enclosure-level power supplies. We plan to investigate other actuation methods, such as memory and disk throttling, in our future work.

As shown in Figure 1, the key components in the control loop include a centralized *controller* and a *power monitor* at the enclosure level, and a *CPU utilization monitor* and a *CPU frequency modulator* on each server. The control loop is invoked periodically and its period is chosen based on a trade-off between actuation overhead and system settling time [5]. The following feedback control loop is invoked at the end of every control period:

- 1) The power monitor (*e.g.*, a power meter) measures the average value of the total power consumption of all servers in the last control period and sends the value to the controller. The total power consumption is the *controlled variable* of the control loop.
- 2) The utilization monitor on each server sends the CPU utilization of the server in the last control period to the controller. The utilization values can be used by the controller to optimize power allocation in the next control period.
- 3) The controller collects the power value and utilization vector, 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.
- 4) The CPU frequency modulator on each server changes the CPU frequency level of the processors accordingly.

Due to its centralized architecture, our control loop is well suitable for controlling the power consumption of all the servers in a small-scale enclosure. We plan to develop

decentralized power control algorithms for large-scale data centers in our future work. Since the core of the control loop is the model predictive controller, we focus on the system modeling and the controller design and analysis in the next two sections, respectively. The implementation details of other control components are provided in Section V.

III. SYSTEM MODELING

In this section, we analytically model the power consumption of the server enclosure. We first introduce the following notation.

- T : the control period.
- $p_i(k)$: the power consumption of Server i in the k^{th} control period.
- $f_i(k)$: the frequency level of the processor of Server i in the k^{th} control period.
- $d_i(k)$: the difference between $f_i(k+1)$ and $f_i(k)$, *i.e.*, $d_i(k) = f_i(k+1) - f_i(k)$.
- $u_i(k)$: the CPU utilization of Server i in the k^{th} control period.
- N : the total number of servers in the enclosure.
- $tp(k)$: the aggregated power consumption of the whole enclosure.
- P_s : the power set point, *i.e.*, the desired power constraint of the enclosure.

The control goal is to guarantee that $tp(k)$ converges to P_s within a finite settling time.

In order to have an effective controller design, it is important to model the dynamics of the controlled system, namely the relationship between the manipulated variables (*i.e.*, $f_i(k)$, $1 \leq i \leq N$) and the controlled variable (*i.e.*, $tp(k)$). However, a well-established physical equation is usually unavailable for computer systems. Therefore, we use a standard approach to this problem called *system identification* [15]. Instead of trying to build a physical equation between the manipulated variables and controlled variable, we infer their relationship by collecting data on the enclosure and establish a statistical model based on the measured data.

Using the system identification approach, we have observed that the power consumption of a server changes immediately as the CPU frequency changes. This is consistent with the observation presented in [5] that power consumption changes within a millisecond after a processor changes its performance state. Since a power sampling period is usually hundreds of milliseconds or even seconds, power consumption can be regarded to be determined exclusively by the current clock frequency and independent of the power consumption in the previous control periods. Figure 2 plots the average power consumption of the four servers used in our experiments at five available CPU frequency levels, which are relative to the highest level. The workload used to do system identification is Linpack, which is introduced in detail in Section VI. Servers 2 to 4 are almost identical while Server 1 has slightly different components. Although Figure 2 shows that the relationship between the power consumption and the CPU frequency ratio is not perfectly linear, we linearize the models within our operating range of the frequency ratio, *i.e.*, between 0.42

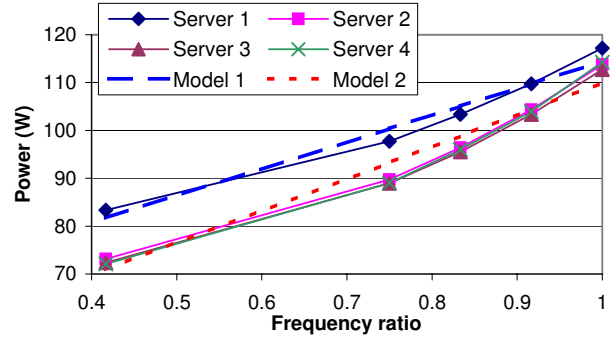


Fig. 2. Power models of four servers

and 1, to simplify the controller design, because the design complexity and computation overhead of a controller increases dramatically as the model order increases. Two linear models fit well ($R^2 > 96\%$) for Server 1 and the other three servers, respectively. In general, our system model of power consumption is:

$$p_i(k) = A_i f_i(k) + C_i \quad (1)$$

where A_i is a generalized parameter that may vary for different servers. The dynamic model of the system as a difference equation is:

$$p_i(k+1) = p_i(k) + A_i d_i(k) \quad (2)$$

To verify the accuracy of our system models, we stimulate the servers with pseudo-random digital white-noise inputs [15] to change the CPU frequency every five seconds in a random fashion. We then compare the actual power consumption with the values predicted by our model. Figure 3 shows that the predicted output by Model 1 is adequately close to the actual power output of Server 1. The predicted output of a second-order model with an additional item $B_i d_i(k-1)$ on the right side of model (2) is just slightly better than that of the first-order model by having only a 3.8% estimation difference. We use the first-order model (2) in this paper to simplify the controller design.

Based on (2), we now consider the total power consumption of all the servers in an enclosure. Their power consumptions can be modeled in the matrix form:

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \mathbf{A}\mathbf{d}(k) \quad (3)$$

$$\text{where, } \mathbf{p}(k) = \begin{bmatrix} p_1(k) \\ \vdots \\ p_N(k) \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A_N \end{bmatrix}, \quad \mathbf{d}(k) = \begin{bmatrix} d_1(k) \\ \vdots \\ d_N(k) \end{bmatrix}.$$

The total power consumption, $tp(k+1)$, is the summation of the power consumed by each individual server.

$$tp(k+1) = tp(k) + [A_1 \quad \dots \quad A_N] \begin{bmatrix} d_1(k) \\ \vdots \\ d_N(k) \end{bmatrix} \quad (4)$$

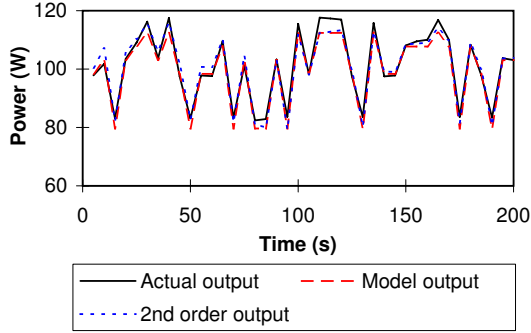


Fig. 3. Comparison between predicted power output and actual power output

Note that each A_i is the *estimated* system parameter resulted from system identification using a typical workload (e.g., Linpack). The *actual* value of A_i in a real system may change for different workloads and is *unknown* at design time. The system stability needs to be reevaluated when the controller designed based on the estimated system parameter is used on a different system with a different workload. In Section IV-C, we prove that a system controlled by the designed controller can remain stable as long as the model variation (i.e., the variation of A_i) is within an allowed range.

IV. CONTROL DESIGN AND ANALYSIS

We apply *Model Predictive Control* (MPC) theory [16] to design the controller based on the system model (4). MPC is an advanced control technique that can deal with coupled MIMO control problems with constraints on the plant and the actuators. This characteristic makes MPC well suited for power control of server enclosures. In this section, we first formulate enclosure-level power control as an MPC control problem. We then present detailed steps to transform the control problem to a standard constrained least-squares problem. Finally, we analyze the stability of the controlled system when the system model may change for different workloads.

A. MPC Controller Design

A model predictive controller optimizes a *cost function* defined over a time interval in the future. The controller uses the 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, $\mathbf{d}(\mathbf{k}), \mathbf{d}(\mathbf{k} + 1|\mathbf{k}), \dots, \mathbf{d}(\mathbf{k} + M - 1|\mathbf{k})$, where M is called the *control horizon*. The notation $x(k + i|k)$ means that the value of variable x at time $(k + i)T$ depends on the conditions at time kT . Once the input trajectory is computed, only the first element $\mathbf{d}(\mathbf{k})$ 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 $tp(k)$ from the power monitor. Note that it is important to re-compute the control input because the original prediction may be incorrect due to uncertainties and inaccuracies in the system model used by the controller. MPC

enables us to combine performance prediction, optimization, constraint satisfaction, and feedback control into a single algorithm.

The controller includes a least squares solver, a cost function, a reference trajectory, and a system model. At the end of every control period, the controller computes the control input $\mathbf{d}(\mathbf{k})$ that minimizes the following cost function under constraints.

$$V(k) = \sum_{i=1}^P \|tp(k + i|k) - ref(k + i|k)\|_{Q(i)}^2 + \sum_{i=0}^{M-1} \|\mathbf{d}(\mathbf{k} + \mathbf{i}|\mathbf{k}) + \mathbf{f}(\mathbf{k} + \mathbf{i}|\mathbf{k}) - \mathbf{F}_{\max}\|_{\mathbf{R}(\mathbf{i})}^2 \quad (5)$$

where P is the prediction horizon, and M is the control horizon. $Q(i)$ is the *tracking error weight*, and $\mathbf{R}(\mathbf{i})$ is the *control penalty weight vector*. The first term in the cost function represents the *tracking error*, i.e., the difference between the total power $tp(k + i|k)$ and a reference trajectory $ref(k + i|k)$. The reference trajectory defines an ideal trajectory along which the total power $tp(k + i|k)$ should change from the current value $tp(k)$ to the set point P_s (i.e., power budget of the enclosure). Our controller is designed to track the following exponential reference trajectory so that the closed-loop system behaves like a linear system.

$$ref(k + i|k) = P_s - e^{-\frac{T}{T_{ref}}i} (P_s - tp(k)) \quad (6)$$

where T_{ref} is the time constant that specifies the speed of system response. A smaller T_{ref} causes the system to converge faster to the set point but may lead to larger overshoot. 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 the cost function (5) represents the *control penalty*. 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, $\mathbf{f}(\mathbf{k} + \mathbf{i} + 1|\mathbf{k}) = \mathbf{d}(\mathbf{k} + \mathbf{i}|\mathbf{k}) + \mathbf{f}(\mathbf{k} + \mathbf{i}|\mathbf{k})$ along the control horizon. The control weight vector, $\mathbf{R}(\mathbf{i})$, can be tuned to represent preference among servers. For example, a higher weight may be assigned to a server if it has heavier or more important workload so that the controller can give preference to increasing its frequency level. As a result, the overall system performance can be optimized. In the experiment presented in Section VI-B.3, we use CPU utilization as an example weight to optimize the system performance.

This control problem is subject to three constraints. First, the CPU frequency of each server should be within an allowed range (e.g., Intel Xeon processor only has eight states). Second, two or more selected servers that run the same application service may need to have the same relative frequency level to avoid having a performance bottleneck. Third, the total power consumption should not be higher than the desired power constraint. The three constraints are modeled as:

$$F_{min,j} \leq d_j(k) + f_j(k) \leq F_{max,j} \quad (1 \leq j \leq N) \quad (7)$$

$$d_i(k) + f_i(k) = d_j(k) + f_j(k) \quad (8)$$

$$tp(k) \leq P_s \quad (9)$$

Based on the above analysis, enclosure-level power control has been modeled as a constrained MIMO optimal control problem. The controller must minimize the cost function (5) under the three constraints. This constrained optimization problem can be transformed to a standard constrained least-squares problem [16]. The controller then uses a standard least-squares solver to solve the optimization problem on-line. In the following section, we present the transformation.

B. Transformation to Least-Squares Problem

To transform our power control problem to a least-squares problem, we need to rewrite our cost function in (5) and constraints (7), (8) and (9) in the form of a standard constrained least-squares problem, as follows:

$$\begin{aligned} \min_{\mathbf{s}(\mathbf{k})} & (\|\Theta\mathbf{s}(\mathbf{k}) - \mathbf{E}(\mathbf{k})\|_{Q(i)}^2 + \mathbf{s}(\mathbf{k})_{\mathbf{R}(i)}^2) \\ \text{subject to constraints} & \quad \Omega\mathbf{s}(\mathbf{k}) \leq \omega. \end{aligned} \quad (10)$$

where $\mathbf{s}(\mathbf{k})$ denotes the vector whose values need to be determined for the minimization of (10) in the control horizon. Θ is a constant matrix and $\mathbf{E}(\mathbf{k})$ is the vector whose values are known at time kT . The concrete values of Θ and $\mathbf{E}(\mathbf{k})$ depend on the system models of a given control problem.

In our enclosure-level MPC power controller, we use the control penalty term in (5) as $\mathbf{s}(\mathbf{k})$ in (10), namely,

$$\mathbf{s}(\mathbf{k}) = \begin{bmatrix} \mathbf{d}(\mathbf{k}|\mathbf{k}) + \mathbf{f}(\mathbf{k}|\mathbf{k}) - \mathbf{F}_{\max} \\ \vdots \\ \mathbf{d}(\mathbf{k} + \mathbf{M} - 1|\mathbf{k}) + \mathbf{f}(\mathbf{k} + \mathbf{M} - 1|\mathbf{k}) - \mathbf{F}_{\max} \end{bmatrix}.$$

We now only need to transform the tracking error term (*i.e.*, the first item) in (5) and constraints (7), (8) and (9) to formulations in terms of $\mathbf{s}(\mathbf{k})$.

We first work on the tracking error term in (5). From the plant model (3), the predicted power consumption for the given prediction horizon can be written as:

$$\begin{aligned} \begin{bmatrix} \mathbf{p}(\mathbf{k} + 1|\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k} + \mathbf{M}|\mathbf{k}) \\ \mathbf{p}(\mathbf{k} + \mathbf{M} + 1|\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k} + \mathbf{P}|\mathbf{k}) \end{bmatrix} &= \begin{bmatrix} \mathbf{p}(\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k}) \\ \mathbf{p}(\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k}) \end{bmatrix} - \begin{bmatrix} \mathbf{A} \\ \vdots \\ \mathbf{A} \\ \mathbf{A} \\ \vdots \\ \mathbf{A} \end{bmatrix} \mathbf{f}(\mathbf{k}) + \begin{bmatrix} \mathbf{A} \\ \vdots \\ \mathbf{A} \\ \mathbf{A} \\ \vdots \\ \mathbf{A} \end{bmatrix} \mathbf{F}_{\max} \\ &+ \begin{bmatrix} \mathbf{A} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A} \\ \mathbf{0} & \mathbf{0} & \cdots & -\mathbf{A} & 2\mathbf{A} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -(P-M)\mathbf{A} & (P-M+1)\mathbf{A} \end{bmatrix} \mathbf{s}(\mathbf{k}). \end{aligned} \quad (11)$$

We rewrite the above equation as:

$$\mathbf{p}'(\mathbf{k}) = \mathbf{p}(\mathbf{k}) - \Gamma\mathbf{f}(\mathbf{k}) + \Gamma\mathbf{F}_{\max} + \Psi\mathbf{s}(\mathbf{k}), \quad (12)$$

$$\text{where } \mathbf{p}'(\mathbf{k}) = \begin{bmatrix} \mathbf{p}(\mathbf{k} + 1|\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k} + \mathbf{M}|\mathbf{k}) \\ \mathbf{p}(\mathbf{k} + \mathbf{M} + 1|\mathbf{k}) \\ \vdots \\ \mathbf{p}(\mathbf{k} + \mathbf{P}|\mathbf{k}) \end{bmatrix}, \quad \Gamma = \begin{bmatrix} \mathbf{A} \\ \vdots \\ \mathbf{A} \\ \mathbf{A} \\ \vdots \\ \mathbf{A} \end{bmatrix}, \text{ and}$$

$$\Psi = \begin{bmatrix} \mathbf{A} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{A} \\ \mathbf{0} & \mathbf{0} & \cdots & -\mathbf{A} & 2\mathbf{A} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & -(P-M)\mathbf{A} & (P-M+1)\mathbf{A} \end{bmatrix}.$$

Based on the plant model (4), we define a $P \times NM$ matrix

$$\Phi = \begin{bmatrix} \phi & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \phi \end{bmatrix}, \text{ where } N \text{ is the number of servers in the}$$

enclosure, and ϕ and $\mathbf{0}$ are two N -dimensional row vectors. Specifically, $\phi = [1 \ \cdots \ 1]$, $\mathbf{0} = [0 \ \cdots \ 0]$. We then get the following equation based on (12):

$$\begin{aligned} \mathbf{tp}'(\mathbf{k}) &= \Phi\mathbf{p}'(\mathbf{k}) \\ &= \mathbf{tp}(\mathbf{k}) - \Phi\Gamma\mathbf{f}(\mathbf{k}) + \Phi\Gamma\mathbf{F}_{\max} + \Theta\mathbf{s}(\mathbf{k}). \end{aligned} \quad (13)$$

$$\text{where } \Theta = \Phi\Psi, \text{ and } \mathbf{tp}'(\mathbf{k}) = \begin{bmatrix} \mathbf{tp}(k+1|k) \\ \vdots \\ \mathbf{tp}(k+P|k) \end{bmatrix}.$$

In addition, we define

$$\mathbf{E}(\mathbf{k}) = \mathbf{ref}'(\mathbf{k}) - \mathbf{tp}(\mathbf{k}) + \Phi\Gamma\mathbf{f}(\mathbf{k}) - \Phi\Gamma\mathbf{F}_{\max} \quad (14)$$

where $\mathbf{ref}'(\mathbf{k})$ represents the reference trajectory for specified

$$\text{prediction horizon: } \mathbf{ref}'(\mathbf{k}) = \begin{bmatrix} \mathbf{ref}(k+1|k) \\ \vdots \\ \mathbf{ref}(k+P|k) \end{bmatrix}.$$

Given Θ in (13) and $\mathbf{E}(\mathbf{k})$ in (14), the tracking error term (*i.e.*, the first term) in (5) is now in the following formulation, which is equivalent to the one in the least-squares problem (10).

$$\|\Theta\mathbf{s}(\mathbf{k}) - \mathbf{E}(\mathbf{k})\|_{Q(i)}^2 \quad (15)$$

We now transform constraints (7), (8) and (9) to the linear inequality constraint form as $\Omega\mathbf{s}(\mathbf{k}) \leq \omega$.

First, constraint (7) can be transformed as follows:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \\ -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{bmatrix} \mathbf{s}(\mathbf{k}) \leq \begin{bmatrix} 0 \\ \vdots \\ 0 \\ F_{\max} - F_{\min} \\ \vdots \\ F_{\max} - F_{\min} \end{bmatrix}. \quad (16)$$

Second, based on (14), constraint (9) can be directly transformed to the following linear inequality:

$$\Theta\mathbf{s}(\mathbf{k}) \leq -\mathbf{tp}(\mathbf{k}) + \Phi\Gamma\mathbf{f}(\mathbf{k}) - \Phi\Gamma\mathbf{F}_{\max} + \mathbf{P}\mathbf{s}. \quad (17)$$

Finally, we transform constraint (8). Let us suppose the m^{th} and n^{th} servers in the system need to have the same frequency level. We have:

$$\mathbf{T}(\mathbf{s}(\mathbf{k}))_i = 0, \quad (18)$$

where $(\mathbf{s}(\mathbf{k}))_i = \mathbf{d}(\mathbf{k} + \mathbf{i}|\mathbf{k}) + \mathbf{f}(\mathbf{k} + \mathbf{i}|\mathbf{k}) - \mathbf{F}_{\max}$ ($0 \leq i \leq$

$M - 1$) and \mathbf{T} is an N -dimensional row vector with the m^{th} element as 1 and the n^{th} element as -1. All other elements in \mathbf{T} are 0. For the whole control horizon M , we have

$$\begin{bmatrix} \mathbf{T} & \cdots & \mathbf{T} \end{bmatrix} \mathbf{s}(\mathbf{k}) = 0. \quad (19)$$

If the system has more than two servers that need to have the same relative frequency level, the above transformation can be applied to each pair of them.

We have now transformed our MPC formulation to a constrained least-square formulation described by (10), (14), (16), (17) and (19). We can use any standard *least-squares* solver to solve this problem. In our system, we implement the controller based on the `lsqlin` solver in Matlab. `lsqlin` uses an active set method similar to that described in [17]. The computational complexity of `lsqlin` is polynomial in the number of servers and the control and prediction horizons. A preliminary overhead measurement of `lsqlin` is presented in [18]. In the controller design discussions in Section IV-D, we also discuss possible alternative (*e.g.*, optimized) implementations of the MPC controller with less overhead.

C. Stability Analysis for Model Variations

A fundamental benefit of the control-theoretic approach is that it gives us theoretical confidence for system stability, even when the system model (*i.e.*, system parameter A_i) may change due to several reasons. First, the workload at runtime can be different from the one used to build the system model at design time. Second, many systems may run different workloads at different times. Finally, the designed controller can also be used to control rack enclosures that have different hardware configurations and thus different power behaviors than the one used to design the controller.

We say that a system is *stable* if the total power $tp(k)$ converges to the desired set point P_s , that is, $\lim_{k \rightarrow \infty} tp(k) = P_s$. Our MPC controller solves a finite horizon optimal tracking problem. Based on optimal control theory [19], the control decision is a linear function of the current power value, the power set point of the enclosure, and the previous decisions for CPU frequency levels.

An important observation from our experiments (*e.g.*, Figure 2) is that the workloads always exhibit an approximately linear relationship between performance state and power consumption, even running on different servers. This has confirmed the same observation reported in [5]. 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

$$tp(k+1) = tp(k) + \begin{bmatrix} g_1 A_1 & \cdots & g_N A_N \end{bmatrix} \begin{bmatrix} d_1(k) \\ \vdots \\ d_N(k) \end{bmatrix}, \quad (20)$$

where g_1, g_2, \dots, g_N are system gains and are used to model the variations between the actual system model (20) and the nominal model (4). We investigate system stability when a controller designed based on the nominal model (4) is used to control the real system (20). We now outline the general

process for analyzing the stability of the power consumption of a server enclosure controlled by MPC.

- 1) We compute the feedback and feedforward matrices for the controller by solving the control input $\mathbf{d}(\mathbf{k})$ based on the system model (4) of a specific system and the reference trajectory (6). The solution is in the following form:

$$\begin{aligned} \mathbf{d}(\mathbf{k}) &= (\mathbf{K}_{tp} + \mathbf{K}_v + \mathbf{K}_w) \mathbf{tp}(\mathbf{k}) + \mathbf{K}_d \mathbf{d}(\mathbf{k} - 1) \\ &+ \mathbf{K}_f \mathbf{f}(\mathbf{k} - 1) + \mathbf{H} + \mathbf{J} \end{aligned} \quad (21)$$

where \mathbf{K}_{tp} , \mathbf{K}_v , \mathbf{K}_w , \mathbf{K}_d , and \mathbf{K}_f are parameter matrices. \mathbf{H} and \mathbf{J} are constant matrices that are independent of $\mathbf{tp}(\mathbf{k})$, $\mathbf{d}(\mathbf{k} - 1)$ and $\mathbf{f}(\mathbf{k} - 1)$. The designed MPC controller is a dynamic controller. Therefore, the stability analysis needs to consider the composite system consisting of the dynamics of the original system and the controller.

- 2) We then derive the closed-loop model of the composite system by substituting the control inputs derived in Step 1 into the actual system model (20). The closed-loop composite system is in the following from:

$$\begin{bmatrix} \mathbf{tp}(\mathbf{k} + 1) \\ \mathbf{d}(\mathbf{k}) \\ \mathbf{f}(\mathbf{k}) \end{bmatrix} = \begin{bmatrix} \mathbf{I} + \mathbf{GA}(\mathbf{K}_{tp} + \mathbf{K}_v + \mathbf{K}_w) & \mathbf{GAK}_d & \mathbf{GAK}_f \\ \mathbf{K}_{tp} + \mathbf{K}_v + \mathbf{K}_w & \mathbf{K}_d & \mathbf{K}_f \\ \mathbf{0} & \mathbf{I} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{tp}(\mathbf{k}) \\ \mathbf{d}(\mathbf{k} - 1) \\ \mathbf{f}(\mathbf{k} - 1) \end{bmatrix} + \begin{bmatrix} \mathbf{GA} & \mathbf{GA} \\ \mathbf{I} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{H} \\ \mathbf{J} \end{bmatrix}, \quad (22)$$

where $\mathbf{G} = \text{diag}[g_1, g_2, \dots, g_N]$ and \mathbf{I} is the identity matrix.

- 3) Derive the stability condition of the closed-loop system described by (22). According to control theory, if all poles are located inside the unit circle in the complex space, the system is stable. Solving this stability condition will give the range of g_i ($1 \leq i \leq N$) where the system will guarantee stability.

Example. We now apply the stability analysis approach to the server enclosure used in our experiments, which is emulated by a server cluster composed of four servers. We design the MPC controller by using the nominal system matrix $\mathbf{A} = \text{diag}[56.12, 66.87, 66.87, 66.87]$. 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 $0 < g \leq 8.8$. That means a system controlled by the MPC controller designed in our experiments can remain stable as long as its system parameters (*i.e.*, the slopes of the model lines in Figure 2) are smaller than 8.8 times of the values used to design the controller. In real systems, to ensure stability for different workloads, the system model used for controller design should be chosen to include those workloads within the wide stability range

around its model parameter. Note that the wide stability range is important because it allows the MPC control approach to be a general solution that can be applied directly, without parameter tuning, to different rack enclosures with different workloads and still achieve the desired system stability.

In the case that servers in an enclosure commonly 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:

$$0 < g_1 \leq 42.1 \quad (23)$$

$$0 < g_2, g_3, g_4 \leq 29.9 \quad (24)$$

Therefore, we have proven that a system controlled by our designed controller can remain stable even when it has two kinds of workload variations. The system stability with other workload variation patterns can be proven in a similar way. A Matlab program is developed by us to perform the above stability analysis automatically. In our stability analysis, we assume 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 (e.g., the set point is too low), no control algorithm can guarantee the set point through CPU frequency adaptation alone. In that case, the system may need to integrate with other adaptation mechanisms (e.g., disk or memory throttling). The integration of multiple adaptation mechanisms is part of our future work.

D. Discussions

We first discuss the selection of control period. Since the shortest period for our power meter to sample power is 1 second, the control period T for the MPC and baseline controllers is set to 5 seconds to eliminate instantaneous reading errors by having an averaged value. While 5 seconds may seem to be a long time for a power controller to respond to power budget violations, our control algorithm can achieve much faster response time when it runs on high-end server enclosures equipped with high-precision power monitors that can sample power in a much shorter period. In such an enclosure where all servers share a common power supply, the control period should be determined based on the manufacturer-specified time interval for the power supply to sustain a power overload. Most today's power supplies are built to survive spikes in power draw up to 20-30% of the maximum load on the order of 100ms to a few seconds. Therefore, the control period in a real system should be derived to ensure that the settling time of the MPC controller is shorter than the manufacturer-specified time interval. As a result, in the case of a budget violation, power can be lowered to the desired budget within the time interval and so undesired server shutdown can be avoided. In our analysis, the worst-case settling time of our controller is 16 control periods. Therefore, if we assume the manufacturer-specified time interval for our power supplies is 1 second, the desired control period should be shorter than $1/16 = 0.625s$. This control period can be supported by many power monitors used in real servers. For example, in [5], a sampling period

of 1ms and a control period of 64ms have been used in IBM blade servers.

The control period is also related to the computational complexity of the MPC controller. In our prototype system, we implement the controller based on the `lsqlin` solver in Matlab. While a preliminary overhead measurement of `lsqlin` in Matlab can be found in [18], we have also measured the overhead of the `lsqlin` solver when it is implemented in native C++ code as a dynamically linked library generated from Matlab. In our experiments, each invocation of the MPC controller with 50 and 100 servers takes 0.09s and 0.39s on average, respectively. Note that the computational complexity and runtime overhead of an MPC controller can be significantly reduced by using a multi-parametric approach proposed in a recent study [20]. This approach can divide the MPC control problem into an offline part and an online part. The offline part is complex but can be solved before the controller is implemented in the service processor firmware. At runtime, the controller only needs to solve the online part incrementally, which is a piecewise linear function. Therefore, the multi-parametric approach allows the MPC controller to run with small control periods (e.g., 64ms) in practice for fast response to sudden power budget violations.

We then discuss the trade-off between power oscillation and the speed of convergence (*i.e.*, settling time) involved in the power controller design for a stable system. Severe oscillation in power consumption is undesirable even if the average power remains close to the set point. In practice, this may lead to the failure of the enclosure power supplies if the oscillation degree is greater than the allowed threshold. The convergence speed is also important because it represents how quickly a system can recover from power violations. If the gains used in the controller (1 in our controller design) is lower than the actual one (g_i), the real effect of the control input is going to be larger than what the controller has predicted and the system may oscillate. Using pessimistic estimation on model parameters A_i will reduce system oscillation because the system gains are less than 1 when parameters are overestimated. However, more pessimistic estimation leads to smaller gains, which may cause slower convergence to the set points.

Our stability analysis gives us theoretical confidence in the performance of our controller and provides a guideline to choose the parameters in the nominal system model (1). For example, given the possible minimum and maximum values of A_i for typical workloads and a stability range of $0 < g \leq 8.8$, we can choose the nominal A_i to be a value slightly greater than $\frac{A_{i,min} + A_{i,max}}{8.8}$ such that the system is guaranteed to be stable even when the real model is unknown at design time. Similarly, based on our discussion regarding settling time, we can use a greater A_i for faster reaction to variations or a smaller A_i to reduce the system oscillation. This kind of theoretical guidance is in sharp contrast to commonly used heuristic solutions that heavily rely on extensive empirical evaluation and manual tuning for desired system response.

V. SYSTEM IMPLEMENTATION

Our testbed includes a cluster composed of 4 Linux servers to run workloads and a Linux desktop machine to run the

controller. The four servers are equipped with 2.4GHz AMD Athlon 64 3800+ processors with 1GB RAM and 512KB L2 Cache. The controller machine is a Dell OptiPlex GX520 with 3.00GHz Intel Pentium D Processor and 1GB RAM. The cluster is used to emulate a server enclosure in a data center. All the machines are connected via an internal Ethernet switch. The 4 servers run openSUSE Linux 10.2 with kernel 2.6.18 while the controller machine runs SUSE Linux 10.1 with kernel 2.6.16.

We now introduce the implementation details of each component in our power control loop.

Power Monitor: The power consumption of each server in the cluster is measured with a WattsUp Pro power meter [21] by plugging the server into the power meter and then connected to a standard 120-volt AC wall outlet. The WattsUp power meter has an accuracy of $\pm 1.5\%$ of the measured value. To access power data, the data port of each power meter is connected to a serial port of the controller machine. A system file is then used for power reading in Linux systems. The power meter samples power data every 1 second and responds to requests by writing all new readings after last request to the system file. The controller then reads the power data from the system file and conducts the control computation. The aggregate power consumption of the 4 servers is used as the total power consumption of the server enclosure.

Utilization Monitor: The utilization monitor uses the `/proc/stat` file in Linux to estimate the CPU utilization in each control period. The `/proc/stat` file records the number of jiffies (usually 10ms in Linux) when the CPU is in user mode, user mode with low priority (`nice`), system mode, and when used by the idle task, since the system starts. At the end of each sampling period, the utilization monitor reads the counters, and estimates the CPU utilization as 1 minus the number of jiffies used by the idle task in the last sampling period divided by the total number of jiffies in the same period. We notice that the same technique is used by a network performance benchmark, NetPerf [22].

Controller: The controller is implemented as a multi-thread process. The main thread uses a timer to periodically invoke the control algorithm presented in Section IV, while the child thread employs the `select` function to get CPU utilizations from all the servers in the cluster. Every time the periodic timer fires, the controller requests a new power reading and the utilizations of all the servers in the last control period, and then invokes a Matlab program to execute the control algorithm presented in Section IV. As the outputs of the control algorithm, new CPU frequency levels are calculated and sent to the CPU frequency modulator on each server to enforce in the next control period. The MPC controller parameters used in all experiments include the prediction horizon as 8 and the control horizon as 2. The time constant T_{ref}/T_s used in (6) is set as 2 to avoid overshoot while having a relatively short settling time.

CPU Frequency Modulator: We use AMD's Cool'n'Quiet technology [23] to enforce the new CPU frequency. AMD Athlon 64 3800+ microprocessor has 5 discrete CPU frequency levels and can be extended to have 8 levels. We use 5 levels in this paper. To change CPU frequency, one needs to

install the `cpufreq` package and then use root privilege to write the new frequency level into the system file `/sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed`. A routine periodically checks this file and resets the CPU frequency accordingly. The average overhead (*i.e.*, transition latency) of changing frequency in AMD Athlon processors is about $100\mu s$ according to the AMD white paper report [23].

Since the new CPU frequency level periodically received from the controller is a fractional value, the modulator code must locally resolve this to a series of discrete frequency values to approximate the fractional value. For example, to approximate 3.2 during a control period, the modulator would output the sequence 3, 3, 3, 3, 4, 3, 3, 3, 3, 4, etc on a smaller timescale. To do this, we implement a first-order delta-sigma modulator [5], which is commonly used in analog-to-digital signal conversion. The detailed algorithm of the first-order delta-sigma modulator can be found in [5]. Clearly, when the sequence has more numbers during a control period, the approximation will be better but the actuation overhead may become higher. In this paper, we choose to use 50 discrete values to approximate the fractional frequency level, which leads to a subinterval of 100ms during an example control period of 5s. As a result, the effect of actuation overhead on system performance is no more than 0.1% ($100\mu s/100ms$) even in the worst case when the frequency needs to be changed in every subinterval. This amount of overhead is acceptable to most computer systems. In addition, recent studies [24] have shown that the overhead of DVFS in future processors can be in nanoseconds. Therefore, the overhead of DVFS is small enough to be used in real systems even when a much smaller control period is adopted.

VI. EMPIRICAL RESULTS

In this section, we present the experimental results conducted on the testbed introduced in Section V. We first introduce three state-of-the-art baselines and discuss the benchmarks used in our experiments and the experimental set-up. We then compare our MPC MIMO controller against the three baselines, in terms of control accuracy and application performance, using two standard benchmarks.

A. Baselines, Benchmark and Set-up

We use three state-of-the-art controllers, referred to as Ad Hoc, SISO, and Improved SISO, as baselines in our experiments. Ad Hoc is a heuristic-based controller designed for enclosure-level power control, which is adapted (with minor changes) from the preemptive control algorithm presented in a recent paper [13]. Ad Hoc represents a typical industry solution to power control of a server enclosure. We compare our controller against Ad Hoc to show that a well-designed ad hoc controller may still fail to have accurate power control and thus lead to degraded application performance. The control scheme of Ad Hoc is briefly summarized as follows.

- 1) Start with the processors of all servers throttled to the lowest frequency level;
- 2) In each control period, (i) if the total power consumption is lower than the set point, choose the server with the

highest CPU utilization to increase its frequency level by one; or (ii) if the power reading is above the set point, choose the server with the lowest CPU utilization to decrease its frequency level by one; (iii) in steps i and ii, if all servers have the same CPU utilization, choose a server in a round-robin fashion.

3) Repeat step 2 until the system stops.

A fundamental difference between Ad Hoc and MPC is that Ad Hoc simply raises or lowers the clock frequency level by one step, depending on whether the measured power is lower or higher than the power set point. In contrast, MPC computes a fractional frequency level based on well-established control theory and uses the frequency modulator to approximate this output with a series of discrete frequency levels.

The second baseline, SISO, is a control-theoretic controller designed to control the power consumption of a *single* server, which is also presented in a recent paper [5]. In contrast to MPC, SISO is a proportional (P) controller designed based on the system model of a single server (2). With SISO, a separate controller is used on each server to control its power *independently* from other servers. The power budget of each server is calculated by evenly dividing the total budget of the enclosure by the number of servers in the enclosure, because it is impossible to predict which server would have more workload and thus need more power at runtime. Compared with SISO, a fundamental advantage of our MPC controller is that MPC explicitly incorporates the interprocessor coupling in an enclosure into its MIMO model and controller design, so that power can be shifted among servers to improve overall system performance.

The third baseline, Improved SISO, is SISO with an enclosure-level supervisor to periodically shift power among different servers based on their CPU utilizations. Improved SISO is similar to the group power manager proposed in [25]. In each control period of the supervisor, if the current power consumption of the enclosure is lower than the set point, the supervisor chooses the server with the highest CPU utilization to increase the power budget of its SISO controller by one step. In order to enforce the enclosure-level power set point, the supervisor then chooses the server with the lowest CPU utilization to decrease the its budget by one step. If the enclosure power is higher than the set point, the supervisor decreases the budget of the server with the highest utilization by one step, and increase the budget of the server with the lowest utilization by one step to reduce power consumption and enforce the set point. Servers with the same utilization are selected in a round-robin fashion. Since Improved SISO is a two-layer control solution, after each invocation of the supervisor, the SISO controllers enter the transient state to achieve the new local budgets determined by the supervisor. Therefore, to guarantee the stability of the SISO controllers, the control period of the supervisor should be configured to be longer than the settling time of the SISO controllers [26]. Based on the analysis in [5], the SISO controller has a settling time of 9 control periods. Therefore, the control period of the supervisor is set to be 50s, which is longer than the settling time of SISO, *i.e.*, $5s \times 9 = 45s$. Clearly, it is not easy to determine a step size for Improved SISO that works well for

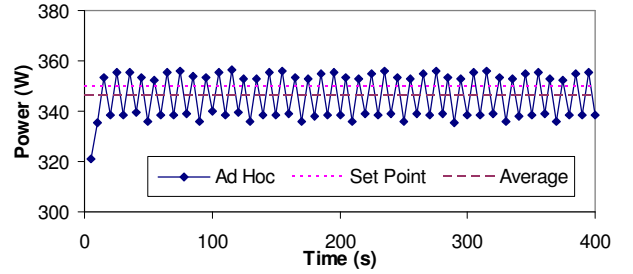


Fig. 4. A typical run of Ad Hoc

all workloads. In our experiments, we test two step sizes, 1W and 5W, to show that a small step size may lead to slow convergence while a large size may lead to oscillations.

In our experiments, we first use the High Performance Computing Linpack Benchmark (HPL) (V1.0a) [27] which is the workload used for system identification. To demonstrate that our control algorithm can effectively control a system running a different workload, we then run the experiments using SPEC CPU2006 (V1.0). HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic. The problem size of HPL is configured to be $10,000 \times 10,000$ and the block size is set as 64 in all experiments unless otherwise noted. SPEC CPU2006 is configured with one user thread and recorded as performance ratio, *i.e.*, the relative speed of the server to finish each benchmark (compared to a reference Sun UltraSparc II machine at 296MHz). CPU2006 includes CINT2006 and CFP2006 which consist of integer and floating-point benchmarks, respectively. The reported result is the average of all benchmarks in each category. Please note that we use HPL and SPEC CPU2006 in our experiments because they are standard benchmarks that can quantify the performance of computer systems. Our control algorithm is not limited to the two benchmarks and can be used to achieve similar performance improvement for other workloads in data centers.

B. Comparison to Ad Hoc

In this subsection, we compare our MPC controller against the first baseline, Ad Hoc.

1) *Control Accuracy*: In this experiment, we run the HPL benchmark on all the four servers. The power set point is 350 W. Since all servers have a CPU utilization of 100% when running HPL, Ad Hoc chooses servers to change frequency level in a round-robin fashion. Figure 4 shows that Ad Hoc starts with all processors throttled to the lowest frequency level. Since the power is lower than the set point at the beginning of the run, Ad Hoc responds by stepping up the frequency level of one server at a time, until the power is higher than the set point at time 15s. Afterwards, Ad Hoc oscillates between two frequency levels once for each server in a round-robin way, because the set point power is between the two power consumption levels at two adjacent frequency levels for every server. As a result, the average power consumption never settles to the set point and has a steady-state error of -3.8 W.

Figure 5 shows a typical run of our MPC controller. In contrast to Ad Hoc, MPC accurately achieves the desired

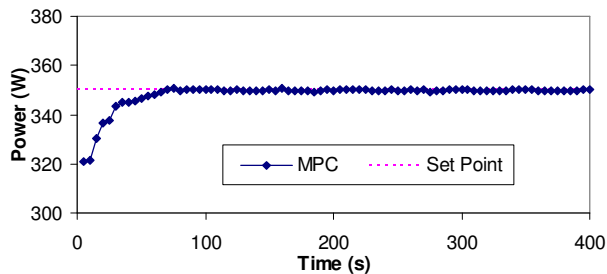


Fig. 5. A typical run of the MPC controller

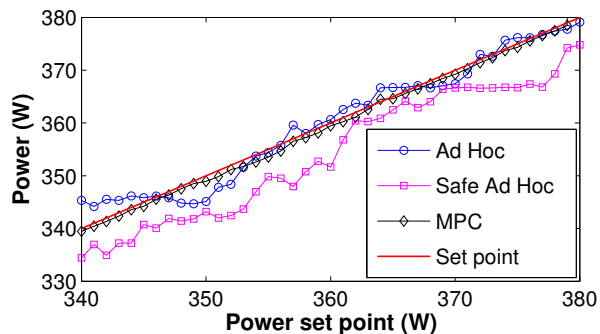


Fig. 6. Comparison of steady state errors

power set point by having a fractional frequency level from the MPC controller, and then using the frequency modulator to generate a series of discrete frequency levels on a finer timescale to approximate the fractional level. One may think that Ad Hoc could be improved by also using a series of discrete levels to change the CPU frequency every 100ms. However, Ad Hoc would still have the same steady-state error because, without a fractional frequency level based on control theory, it can only oscillate between two frequency levels of each server. In addition, it is actually infeasible to run Ad Hoc every 100ms in practice, because Ad Hoc needs to measure the *actual* power consumption and then step up or down by one frequency level. We acknowledge that MPC controller may have slightly higher actuation overhead than Ad Hoc by having a finer actuation timescale. However, as discussed in Section V, the overhead is usually negligible in most systems.

Figure 6 shows the result of running both MPC and Ad Hoc under a series of power set points from 340 W to 380 W. Each data point is the average of the steady-state power levels of three independent runs. The steady-state power level of each run is the averaged power level in the steady state of the controller, which is calculated by eliminating the transient power values at the beginning of the run. The MPC controller is able to meet the set point with a precision less than 1 W. However, Ad Hoc shows steady-state error that is often above the set point. For example, when set point is 340 W, Ad Hoc has the maximum positive steady-state error as 5.3 W above the set point. Note that an increased number of frequency levels may allow the control accuracy of Ad Hoc to get closer to that of MPC. However, since Ad Hoc only steps up/down one frequency level for one server in each control period, it will have an extremely long settling time with a large number of frequency levels. Such a long settling time is

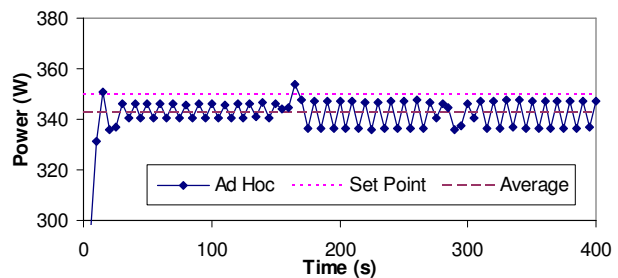


Fig. 7. A typical run of Safe Ad Hoc

undesired because it may lead to undesired server shutdown, as discussed in Section V. While it is possible for Ad Hoc to have a different step size other than one level at a time, it is hard to determine the optimal step size for all systems. In contrast, the step size of MPC is selected based on well-established control theory for the desired control accuracy and settling time.

Since Ad Hoc has steady state error, it may be inappropriate to use Ad Hoc in a real system because a positive steady-state error (*i.e.*, average power is above the set point) may cause the power supply to have overload and then likely failure. One may think that Ad Hoc could be easily modified to eliminate its positive steady-state error by having a safety margin. To do this, we can get the steady-state error of each single run of Ad Hoc. We then get the maximum positive steady-state errors of the three runs for each set point and then the maximum errors for all set points from 340 W to 380 W. By doing that, we get a safety margin of 5.545 W and we re-run the experiments for Ad Hoc with its power budget deducted this margin. This modified Ad Hoc policy is referred to as Safe Ad Hoc. We note that a similar baseline called *Improved Ad Hoc* has been used in [5]. Figure 7 shows Safe Ad Hoc runs at or below the set point most of the time. Note that at time 165s, Safe Ad Hoc still violates the power constraint once. This is because the safety margin is calculated based on the steady-state errors which are averaged values. Please note that Safe Ad Hoc is actually *infeasible* in practice because it is hard to have such a *priori* knowledge about the safety margin before spending a lot of time measuring this margin at runtime. However, we use Safe Ad Hoc as a baseline that can achieve the best possible performance in an ad hoc way and yet does not violate the power constraint.

2) *Application Performance*: In this subsection, we investigate the impact of enclosure-level power control on the performance of the HPL benchmark. We use Safe Ad Hoc (with the safety margin of 5.545 W) instead of Ad Hoc because Ad Hoc would violate the power constraint and thus may not be suitable for a real system. Figure 8 plots the benchmark performance (aggregated value of all the four servers) of Safe Ad Hoc and MPC. MPC has better performance than Safe Ad Hoc for all five set points from 340 W to 380 W, with a maximum performance improvement of 6.1% at 370 W. This is because MPC can accurately achieve the set-point power while Safe Ad Hoc stays below the set point most of the time. As a result, the performance of Safe Ad Hoc is worse than that of MPC. Please note again that it is actually *infeasible* in practice for Safe Ad Hoc to have such a tight safety margin

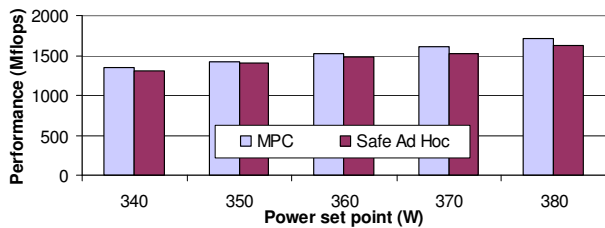


Fig. 8. Application performance comparison

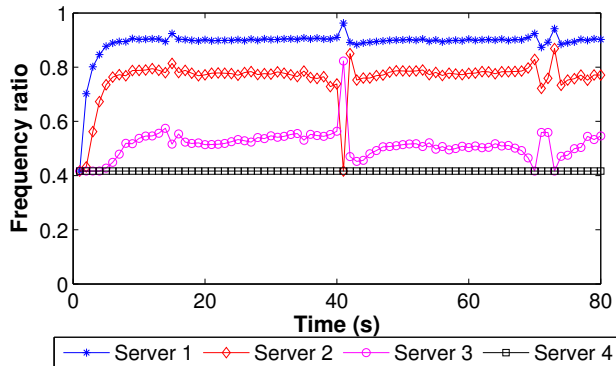


Fig. 9. Differentiated CPU frequency ratio under MPC control

resulted from extensive experiments in this paper. In a real system, Ad Hoc is commonly configured with a large safety margin and thus would result in much worse performance.

3) *Performance Differentiation*: An interesting property of Ad Hoc is that it uses CPU utilization to decide which server to change CPU frequency level. In that way, Ad Hoc can give more power to servers that have more workload. In this subsection, we show that MPC can also utilize similar performance metrics to improve the overall system performance. To compare MPC against Safe Ad Hoc, we assign weight to each server in the control weight vector, $\mathbf{R}(\mathbf{i})$, of the MPC controller proportionally to its CPU utilization. As a result, MPC control theory guarantees that servers with higher utilization can have higher CPU frequency level by minimizing the cost function (5).

Running HPL on a server always leads to 100% CPU utilization. Hence, to test performance differentiation for the two controllers, we slightly modify the HPL workload to achieve different utilizations as 25%, 50% and 75% for different servers, by inserting a sleep function at the end of each iteration in its computation loop. Note that the modified HPL benchmark is used *only* in this experiment. The HPL problem size is $4,000 \times 4,000$ and the block size is 1 for all

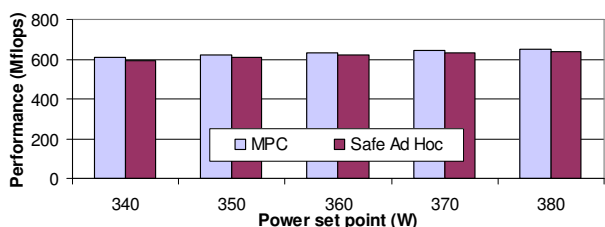


Fig. 10. Performance comparison when servers have non-uniform workload

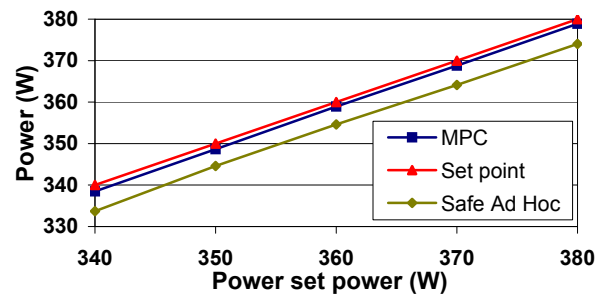


Fig. 11. Power consumption comparison using SPEC CPU2006

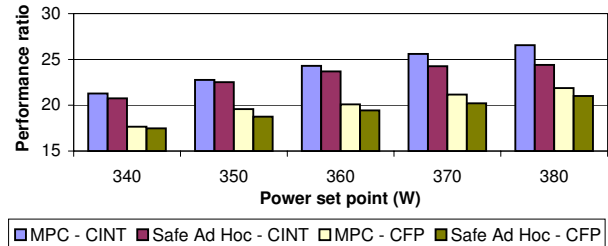


Fig. 12. Application performance comparison using SPEC CPU2006

the servers. We run the original HPL and the modified HPL on the four servers so that they have utilizations approximately as 100% (Server 1), 75% (Server 2), 50% (Server 3) and 25% (Server 4), respectively. Figure 9 shows that MPC can assign higher CPU frequency levels to the servers with higher CPU utilization. For example, Server 4 has the lowest frequency level (0.42 of the highest, *i.e.*, full frequency level) because it has the lowest CPU utilization (about 25%). Figure 10 shows that the overall performance of the system is better under MPC than under Safe Ad Hoc, with a maximum performance improvement of 3.4% at 340 W. Note that our MPC controller provides a general framework where other performance metrics such as application-level performance measurements can be dynamically fed into the control weight vector to improve overall system performance.

4) *Results using SPEC CPU2006*: To demonstrate the effectiveness of MPC with different workloads, we compare MPC with Safe Ad Hoc using SPEC CPU2006. All parameters for both MPC and Safe Ad Hoc remain the same. Figure 11 shows that the power consumption of the enclosure under MPC is very close to the set point. The small gaps are caused by the short idle intervals between the runs of different benchmarks in CPU2006. In contrast, Safe Ad Hoc wastes the power budget because it uses the safety margin to ensure that power consumption always stays below the budget. As a result, MPC achieves higher application performance ratio than Safe Ad Hoc for both CINT and CFP, as shown in Figure 12. The performance ratio is defined as the relative CPU speed compared to the reference machine used by SPEC.

C. Comparison to SISO and Improved SISO

In this subsection, we compare our MPC controller against the second and third baselines, SISO and Improved SISO.

In this set of experiments, we leave the first server idling and run the HPL benchmark only on the other three servers. The

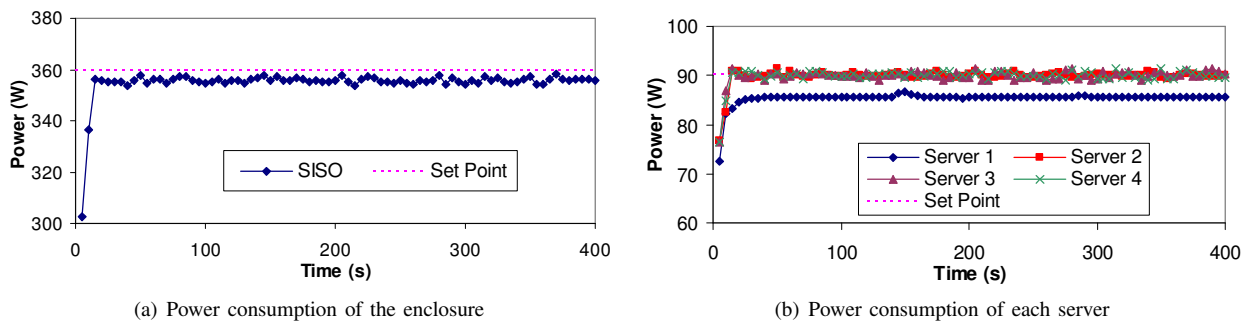


Fig. 13. A typical run of SISO

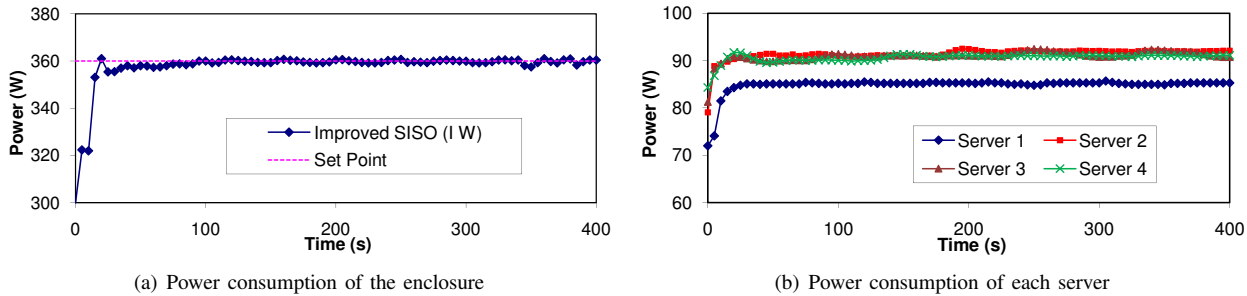


Fig. 14. A typical run of Improved SISO with 1W step size

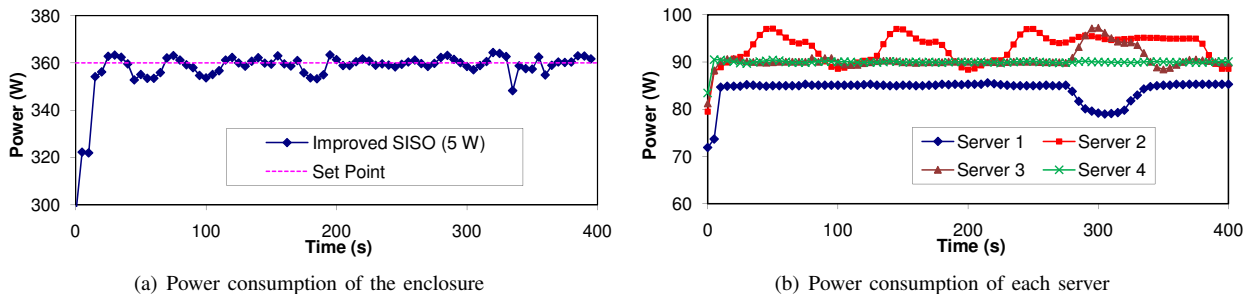


Fig. 15. A typical run of Improved SISO with 5W step size

power set point is set to 360 W. Under the SISO control policy, each server evenly gets 90 W as its local power budget. Figure 13(a) shows that SISO has steady-state errors. This is because the idle server (Server 1) cannot use up its power budget of 90 W even when it is running at the highest CPU frequency level, as shown in Figure 13(b). On the other hand, the three busy servers (Servers 2 to 4) cannot get enough power so that they can only run at degraded frequency levels.

Figure 14(a) shows that Improved SISO with a 1W step size has smaller steady-state errors than SISO. This is because that the enclosure-level supervisor dynamically shifts the unused power budget from Server 1 to the other three servers, as shown in Figure 14(b). However, due to the small step size and long control period of the supervisor for global stability, the total power of the enclosure takes a long time to converge to the set point. Figure 15(a) shows that Improved SISO with a 5W step size has a shorter convergence time. However, under Improved SISO (5W), though the average power consumption of the enclosure converges to the set point, the increased step size leads to greater power oscillations, which could cause the enclosure power supplies to fail and are thus undesirable. The reason of the oscillates is that the 5W step size amplifies the

small power oscillations caused by the workload noise in the system. Both the two Improved SISO baselines have a longer response time to power violations than MPC due to the long period of the supervisor. Figure 16 shows that different power set points can be achieved precisely by MPC with the largest standard deviation smaller than 0.1W. The set points can also be approximately achieved by Improved SISO with the largest standard deviation as 4.76W (for 5W step size) and 2.73W (for 1W step size) under the set point of 380W.

Figure 17 shows the HPL performance data in this experiment. MPC has the best performance because its total power consumption can exactly reach the desired power budget. In contrast, SISO has the worst performance because some power budget is wasted by the idle server due to the even distribution of the total budget. The maximum performance improvement of MPC over SISO is 11.8% at 380 W. Hence, it is clearly important for different servers in an enclosure to share power resource because they commonly have non-uniform workloads. The performance of Improved SISO is worse than that of MPC because the supervisor does not *directly* throttle the CPU frequency of the servers based on their utilizations. Instead, the supervisor reallocates power

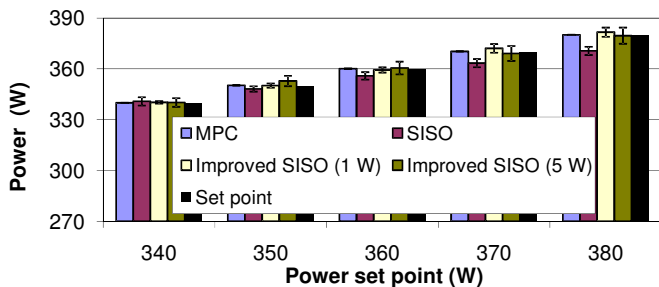


Fig. 16. Power consumption comparison with SISO and Improved SISO

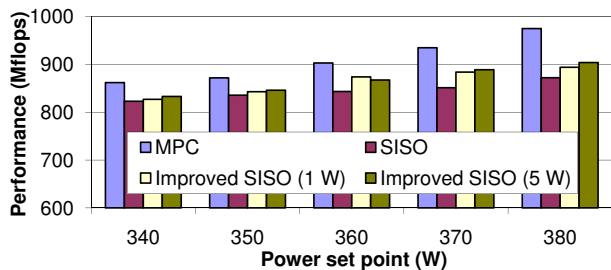


Fig. 17. Application performance comparison with SISO and Improved SISO

budgets only when the total power has deviations from the set point. As a result, the idle server (Server 1) can run at a high frequency level most of the time under Improved SISO. In contrast, Server 1 always has the lowest frequency level under MPC, as demonstrated in Section VI-B.3. This experiment shows that Improved SISO, as a two-layer solution that needs delicate coordination, cannot use power budget as efficiently as MPC, resulting in degraded performance.

We have tested the four control solutions using SPEC CPU2006 and achieved similar results. The performance improvement of MPC over SISO, Improved SISO (1W), and Improved SISO (5W) is 6.7%, 6.4%, and 4.4%, respectively.

VII. RELATED WORK

Power consumption is one of the most important design constraints for high-density servers. Much of the prior work has attempted to reduce power consumption by improving the energy-efficiency of individual server components [28]. There has been some work on system-level power and thermal management. Zeng et al. [29] and Lu et al. [30] have developed strategies to reduce server power consumption. Fan et al. [31] investigate the aggregate power usage characteristics of a large number of servers at the data center level. Felter et al. [7] use open-loop control to shift power between processor and memory to maintain a server power budget. Brooks et al. [32] use ad-hoc control to limit processor temperature so cheaper heat-sinks can be used. Diniz et al. [33] have proposed dynamic approaches to limit the power consumption of main memory. Meisner et al [34] have proposed an energy-conservation approach to eliminate server idle power. In sharp contrast to their work, which relies on heuristic-based control schemes, we adopt a rigorous design methodology that features a *control-theoretic* framework for systematically developing control strategies with analytic assurance of control accuracy and system stability [11].

Control-theoretic approaches have been applied to a number of computing systems. A survey of feedback performance control in various computing systems is presented in [35]. Feedback control scheduling algorithms have been developed for operating systems [36] and real-time systems [37]. Control techniques have also been applied to data service and storage systems [38], networks [39], and Internet servers [11]. Decentralized control algorithms have also been developed to control large-scale distributed systems [37].

Several research projects [4], [9] have successfully applied control theory to explicitly control power or thermal of computing servers. Lefurgy et al. [5] have shown that control-theoretic solution outperforms a commonly used heuristic-based solution by having more accurate power control and less overhead. However, most current work focuses on controlling the power consumption of a single server independently. Much less work has been done at the enclosure level where *multiple* blade servers run the same software service and share common power supplies. Ranganathan et al. [13] propose a negotiation-based algorithm to allocate power budget to different servers in an enclosure for better performance. Femal et al. [40] present another algorithm based on linear programming. In contrast to their work, we propose a control-theoretic solution based on the MPC MIMO control theory for optimal system performance at the cluster level by sharing power budget. Wang et al. [41] propose SISO control algorithms for power efficiency control and power capping at both group and server levels and then evaluate those algorithms using simulations. In this paper, we design a MIMO control algorithm based on MPC theory and evaluate our algorithm on a physical testbed. Raghavendra et al. [25] recently present a multi-level coordinated power management architecture. Different from their work, we focus on power control at the cluster level to optimize application performance.

Some prior work has been proposed to use power as a tool for application-level performance requirements. For example, Zhu et al. [42] have developed a feedback control scheduling algorithm using dynamic voltage scaling. Sharma et al. [43] effectively apply control theory to control application-level quality of service requirements. Chen et al. [44] also present a feedback controller to manage the response time in a server cluster. Although they all use control theory to manage power consumption, power is only used as a knob to control application-level service metrics. As a result, they do not provide any absolute guarantee to the power consumption of a computing system. In this project, we explicitly control the power consumption to adhere to a given constraint.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an enclosure-level power controller that shifts power among servers based on their performance needs, while controlling the total power of the enclosure to stay at or below a constraint imposed by the capacity of its power supplies. Our controller features a multi-input-multi-output system model and a rigorous controller design based on the model predictive control theory. Empirical results using standard benchmarks demonstrate that our controller

outperforms two state-of-the-art controllers, by having better application performance and more accurate power control. Our results also show that the designed power controller can provide desired performance differentiation. Rigorous analysis based on control theory proves that the controller can remain stable even when the system power model varies significantly at runtime.

In our future work, we plan to integrate other actuation methods, such as memory and disk throttling, as well as workload distribution. We will also design coordinated control frameworks that can provide guarantees on both application performance and power. As data centers start to adopt server virtualization strategies for resource sharing to reduce hardware and operating costs, we also plan to extend our power control algorithms for virtualized server environments.

ACKNOWLEDGEMENTS

We would like to thank Charles Lefurgy and Malcolm Ware at the IBM Austin Research Laboratory for their insightful discussions. This work was supported by NSF under a CSR grant CNS-0720663 and an NSF CAREER Award CNS-0845390, and by Microsoft Research under a Power-Aware Computing Award in 2008.

REFERENCES

- [1] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *14th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Salt Lake City, UT, 2008.
- [2] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *IEEE Computer*, vol. 37, no. 11, pp. 68–74, 2004.
- [3] C. Patel, C. Bash, R. Sharma, M. Beitelmal, and R. Friedrich, "Smart cooling of data centers," in *Proceedings of the ASME Interpack*, Maui, Hawaii, July 2003.
- [4] R. J. Minerick, V. W. Freeh, and P. M. Kogge, "Dynamic power management using feedback," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP)*, Sep. 2002.
- [5] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, 2008.
- [6] V. Delaluz, M. T. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM energy management using software and hardware directed power mode control," in *HPCA*, 2001.
- [7] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in *ICS*, 2005.
- [8] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: dynamic speed control for power management in server class disks," in *ISCA*, 2003.
- [9] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *HPCA*, Washington, DC, USA, 2002.
- [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, 2005.
- [11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.
- [12] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *ICDCS*, 2006.
- [13] P. Ranganathan, P. Leech, D. Irwin, and J. S. Chase, "Ensemble-level power management for dense blade servers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006.
- [14] 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.
- [15] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems*, 3rd edition. Addison-Wesley, 1997.
- [16] J. M. Maciejowski, *Predictive Control with Constraints*. Prentice Hall, 2002.
- [17] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Academic Press, London, UK, 1981.
- [18] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, Jun. 2005.
- [19] F. L. Lewis and V. L. Syrmos, *Optimal Control, Second Edition*. John Wiley & Sons, Inc., 1995.
- [20] P. Tondel, T. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit mpc solutions," in *IEEE Conference on Decision and Control*, 2001.
- [21] Electronic Educational Devices Inc., "Watts Up Pro Power Meter," <http://www.wattsupmeters.com>.
- [22] *Netperf: A Network Performance Benchmark*, Information Networks Division, Hewlett-Packard, Cupertino, CA, Mar. 1993, edition B.
- [23] AMD, *White Paper Publication 26094: BIOS and Kernel Developer's Guide for AMD Athlon 64 and AMD Opteron Processors, Revision 3.30*, Advanced Micro Devices, Inc., Feb. 2006.
- [24] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *HPCA*, 2008.
- [25] 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.
- [26] X. Wang, M. Chen, C. Lefurgy, and T. Keller, "Ship: Scalable hierarchical power control for large-scale data centers," in *PACT*, 2009.
- [27] Innovative Computing Laboratory, University of Tennessee, "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers," <http://www.netlib.org/benchmark/hpl/>.
- [28] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *IEEE Computer*, vol. 36, no. 12, 2003.
- [29] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: managing energy as a first class operating system resource," in *ASPLOS X*, 2002.
- [30] Y.-H. Lu, L. Benini, and G. D. Micheli, "Operating-system directed power reduction," in *ISLPED*, 2000.
- [31] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA*, 2007.
- [32] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *HPCA*, 2001.
- [33] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *ISCA*, 2007.
- [34] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ASPLOS*, 2009.
- [35] 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, Jun. 2003.
- [36] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *OSDI*, 1999.
- [37] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, 2007.
- [38] M. Karlsson, C. T. Karamanolis, and X. Zhu, "Triage: Performance differentiation for storage systems using adaptive control," *ACM Transactions on Storage*, vol. 1, no. 4, pp. 457–480, 2005.
- [39] S. Keshav, "A control-theoretic approach to flow control," in *Proceedings of ACM SIGCOMM*, 1991.
- [40] M. E. Fernald and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *ICAC*, 2005.
- [41] Z. Wang, C. McCarthy, X. Zhu, P. Ranganathan, and V. Talwar, "Feedback control algorithms for power management of servers," in *FeBLID*, 2008.
- [42] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *IEEE RTAS*, 2004.
- [43] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *IEEE Real-Time Systems Symposium (RTSS)*, 2003.
- [44] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautham, "Managing server energy and operational costs in hosting centers," in *ACM SIGMETRICS*, 2005.