The 14th IEEE International Symposium on High-Performance Computer Architecture (HPCA 2008)

Cluster-level Feedback Power Control for Performance Optimization

Xiaorui Wang and Ming Chen
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{*xwang, mchen11*}*@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 on power management thanks to 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 cluster level where multiple servers are correlated by common workloads and share common power supplies, power cannot be shared to improve application performance. In this paper, we propose a cluster-level power controller that shifts power among servers based on their performance needs, while controlling the total power of the cluster to be lower than a constraint. Our controller features a rigorous design based on an optimal multi-input-multi-output control theory. Empirical results demonstrate that our controller outperforms two state-of-the-art controllers, by having better application performance and more accurate power control.

1 Introduction

In recent years, power has become the most 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 [1]. The total energy cost for this single server would be $338 a year without counting the costs of air conditioning and power delivery subsystems [1]. 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 so-called *blade servers* which pack traditional multi-board server hardware into a single board. Multiple servers are then put into a *chassis* (also called an enclosure) 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 chassis. The increasing high server density may also lead to a greater probability of thermal failure and hence require additional energy cost for cooling [2].

An effective way to reduce energy consumption of blade servers is to transition the hardware components from high-power states to low-power states whenever possible [1]. Most components in a modern blade server such as processors [3, 4], main memory [5, 6] and disks [7] have adjustable power states. Components are fully operational, but consume more power in high-power states while having degraded functionality in low-power states [1]. 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 actual power consumption of the server exceeds the budget [4]. 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 system 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 [3, 8, 9]. For example, recent work [4] 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 [10].

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 a cluster environment. First, we need multi-input-multi-output (MIMO) control algorithms to control the power consumptions of *multiple* servers in a cluster simultaneously by manipulating the performance setting of each server. Second, the servers in a cluster are usually *coupled* together due to the fact that they often run the same application service or share common power supplies at the chassis 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 [11]. 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 servers in a small-scale cluster such as a chassis is usually limited by the capacity of the common power supplies equipped with the chassis [12]. Therefore, servers in a cluster are usually coupled together, and so their power consumption *cannot* be controlled independently from each other. As a result, the power management problem in a cluster 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 a small-scale cluster (*e.g.*, a chassis) of a data center. Specifically, the contributions of this paper are four-fold:

- We analytically model the power consumption of a server cluster using system identification and verify the model with white noise inputs.
- We design and analyze a MIMO power control algorithm based on the MPC control theory to optimize application performance while controlling the total power to be lower than a constraint.
- We present the system architecture of our control loop and the implementation details of each component.
- We present empirical results to demonstrate that our controller outperforms two state-of-the-art power controllers: a heuristic-based MIMO controller [12] and a control-theoretic SISO controller [4].

The rest of the paper is organized as follows. Section 2 introduces the overall architecture of the power control loop. Section 3 describes system modeling. Section 4 discusses the design and analysis of our control algorithm.

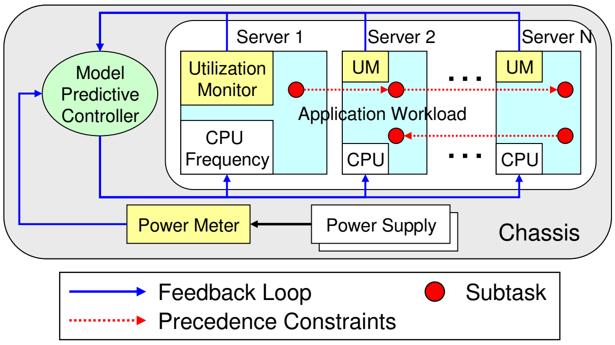

Figure 1. Power control loop for a small-scale cluster

Section 5 gives the implementation details of each component in the control loop. Section 6 presents the results of our empirical experiments conducted on a physical testbed. Section 7 highlights the distinction of our work by discussing the related work. Section 8 concludes the paper.

2 Cluster-level Power Control Loop

In this section, we give a high-level description of our power control loop that adaptively manages the power consumption of a server cluster by conducting *Dynamic Frequency Scaling* (DFS) for the processor of each server.

There are two reasons for us to use processor throttling 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 total power consumption of small form-factor servers [13]. 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 to provide an effective way to support a power budget reduction in the event of power supply failure. 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 cluster 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 [4]. 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 through its feedback lane. The total power consumption is the *controlled variable* of the control loop.

2. The utilization monitor on each processor sends its CPU utilization in the last control period to the con-

troller. 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 processor of each server, and then sends the level to the CPU frequency modulator on each server through its feedback lane. 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 processor accordingly.

Due to its centralized architecture, our control loop is well suitable for controlling the power consumption of a small-scale computer cluster (*e.g.*, a chassis). We plan to develop decentralized power control algorithms for large-scale clusters in our future work. Since the core of the control loop is the model predictive controller, we focus on system modeling and the controller design and analysis in the next two sections. The implementation details of other components are given in Section 5.

3 System Modeling

In this section, we analytically model the power consumption of the server cluster. We first introduce several notations. T is the control period. $p_i(k)$ is the power consumption of Server i in the k^{th} control period. $f_i(k)$ is the frequency level of the processor of Server i in the k^{th} control period. $d_i(k)$ is the difference between $f_i(k+1)$ and $f_i(k)$, *i.e.*, $d_i(k) = f_i(k+1) - f_i(k)$. $u_i(k)$ is the CPU utilization of Server i in the k^{th} control period. N is the total number of servers in the cluster. $tp(k)$ is the total power consumption of the whole cluster, *i.e.*, $tp(k) = \sum_{i=1}^{N} p_i(k)$. P_s is the power set point, *i.e.*, the desired power constraint of the cluster. The control goal is to guarantee that $tp(k)$ converges to P_s within a given settling time.

In order to have an effective controller design, it is crucial 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* [14]. Instead of trying to build a physical equation between the manipulated variables and controlled variable, we infer their relationship by collecting data on the cluster 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 clock frequency changes. This is consistent with the observation presented in [4] 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 represented as a fraction of the highest level. The workload used to do system identification is Linpack, which is introduced in detail in Section 6. Server 2 to 4 are almost identical while Server 1 has slightly different components. 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:

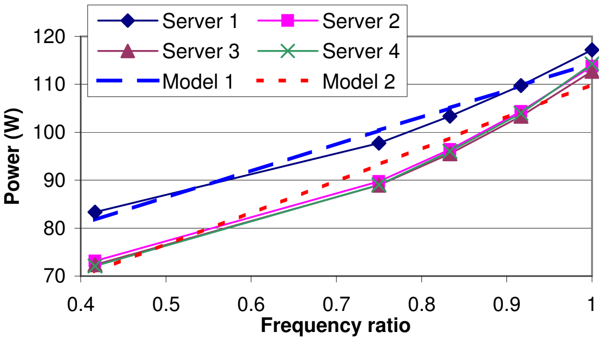

Figure 2. Power models of four servers

$$p_i(k) = A_i f_i(k) + C_i \tag{1}$$

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

$$p_i(k+1) = p_i(k) + A_i d_i(k) \tag{2}$$

To verify the accuracy of our system models, we stimulate the servers with pseudo-random digital white-noise inputs [14] 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 is just slightly better than that of the first-order model by only having a 3.8% variation 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 servers in a cluster. Their power consumptions can be modeled in the matrix form:

$$\mathbf{p(k+1)} = \mathbf{p(k)} + \mathbf{Ad(k)} \tag{3}$$

where, $\mathbf{p(k)} = \begin{bmatrix} p_1(k) \\ \vdots \\ p_N(k) \end{bmatrix}$,

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

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

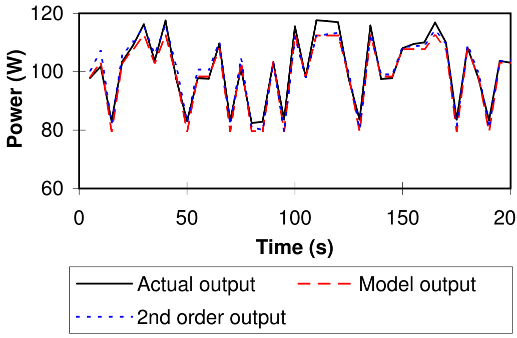

Figure 3. Comparison between predicted power output and actual power output

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

Note that each A_i is the *estimated* system parameter resulted from system identification using a typical workload (*i.e.*, Linpack). The *actual* value of A_i in a real system may change for different workloads and is *unknown* at design time. However, in Section 4.2, we show that a system controlled by the controller designed with the estimated parameters can remain stable as long as the variation of A_i is within an allowed range.

4 Control Design and Analysis

We apply the *Model Predictive Control* (MPC) theory [15] 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 in server clusters.

4.1 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 sampling 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 sampling periods, $\mathbf{d(k)}$, $\mathbf{d(k+1|k)}$, … $\mathbf{d(k+M-1|k)}$, where M is called the *control horizon*. The notation $x(k+i|k)$ means that the value of variable x at time $(k+i)T$ depends on the conditions at time kT. Once the input trajectory is computed, only the first element $\mathbf{d(k)}$ is applied as the control input to the system. At the end of the next sampling period, the prediction horizon slides one sampling 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 sampling period, the controller computes the control input $\mathbf{d(k)}$ that minimizes the following cost function under constraints.

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

where P is the prediction horizon, and M is the control horizon. $Q(i)$ is the *tracking error weight*, and $\mathbf{R(i)}$ is the *control penalty weight vector*. The first term in the cost function 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 cluster). 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, $\mathbf{f(k+i+1|k)} = \mathbf{d(k+i|k)} + \mathbf{f(k+i|k)}$ along the control horizon. The control weight vector, $\mathbf{R(i)}$, 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.

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 servers that run the same application service may have the same frequency level. Third, the total power consumption should not be higher than the desired power constraint. The three constraints are modeled as:

$$\begin{aligned} &F_{min,j} \le d_j(k) + f_j(k) \le F_{max,j} \quad (1 \le j \le N) \\ &d_i(k) + f_i(k) = d_j(k) + f_j(k) \\ &tp(k) \le P_s \end{aligned}$$

Based on the above analysis, cluster-level power management 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 easily transformed to a standard constrained least-squares problem [15]. The transformation is not shown due to space limitations, but can be found in an extended version of this paper [16]. The controller uses a standard least-squares solver to solve the optimization problem on-line. 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 [6]. The computational complexity of `lsqlin` is polynomial in the number of servers and the control and prediction horizons.

4.2 Stability Analysis

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 for different workloads. We say that a system is *stable* if the total power $tp(k)$ converges to the desired set point P_s, that is, $\lim_{k\to\infty} tp\,(k) = P_s$. Our MPC controller solves a finite horizon optimal tracking problem. Based on optimal control theory [17], the control decision is a linear function of the current power value, the power set point of the cluster, and the previous decisions for CPU frequency levels.

We now outline the general process for analyzing the stability of a server cluster when the actual system model is different from the model resulted from system identification (*i.e.*, with different A_i). First, given a specific system, we derive the control inputs $\mathbf{d(k)}$ that minimize the cost function based on the *estimated* system model (4) with estimated parameters $\mathbf{A}$. The control inputs represent the control decision based on the estimated system model. Second, we construct the *actual* system model by assuming the actual parameter $A'_i = g_i A_i$, where g_i represents the unknown system gain. The stability analysis of the actual system needs to consider a composite system consisting of the dynamics of the original system and the controller. Third, we then derive the closed-loop system model by substituting the control inputs derived in the first step into the actual system model. Finally, we analyze the stability of the closed-loop system by computing all the poles of the closed-loop system. According to control theory, if all poles locate inside the unit circle in the complex space and the DC gain matrix from the control to the state is the identity matrix, the state of the system, *i.e.*, the total power consumption, will converge to the set point. The allowed variation range of g_i can be established by computing the values of g_i that cause the poles to move across the unit circle.

The detailed steps and a complete stability proof for the example cluster used in our experiments can be found in an extended version of this paper [16]. Our results show that the system can remain stable as long as the variation of A_i is within an allowed range. In addition, a Matlab program is developed by us to perform the above stability analysis procedure 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, no control algorithm can guarantee the set point through CPU frequency adaptation. In that case, the system may need to integrate with other adaptation mechanisms (e.g., disk or memory throttling). The integration of multiple adaptation mechanisms is part of our future work.

5 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. All the machines are connected via an internal 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 [18] by plugging the server into the power meter, which is 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 generated 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.

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 note that the same technique is used by a network performance benchmark, NetPerf [19].

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 4, 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 4. 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.

CPU Frequency Modulator: We use AMD's Cool'n'Quiet technology [20] 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 BIOS routine periodically checks this file and resets the CPU frequency accordingly. The average overhead (*i.e.*, transition latency) for the BIOS to change frequency in AMD Athlon processors is about $100\mu s$ according to the AMD white paper report [20].

Since the new CPU frequency level periodically received from the controller is a floating-point (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 [4], which is commonly used in analog-to-digital signal conversion. The detailed algorithm of the first-order delta-sigma modulator can be found in [4]. 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.

6 Empirical Results

In this section, we present the experimental results conducted on the testbed introduced in Section 5. We first introduce two state-of-the-art baselines: a MIMO ad hoc controller and a single-input-single-output (SISO) controller. We then discuss the benchmark used in our experiments and the experimental set-up. We then compare our MPC MIMO controller against the two baselines, in terms of control accuracy and application performance.

6.1 Baselines, Benchmark and Set-up

We use two state-of-the-art controllers, referred to as Ad Hoc and SISO, as baselines in our experiments. Ad Hoc is a heuristic-based controller designed for cluster-level power control, which is adapted (with minor changes) from the preemptive control algorithm presented in a recent paper [12]. Ad Hoc represents a typical industry solution to power control of a server cluster. 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 system stops.

A fundamental difference between Ad Hoc and our MPC controller 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 [4]. In contrast to our MPC controller, 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 cluster by the number of servers in the cluster, because it is impossible to predict which server would have more workload and thus need more power at runtime. We use SISO as our baseline to show that a controller designed for a single server *cannot* be easily extended to control a server cluster where multiple servers are coupled together due to the common power budget. A fundamental advantage of our MPC controller is that MPC explicitly incorporates the interprocessor coupling in a cluster into its MIMO model and controller design, so that power can be shifted among servers to improve overall system performance.

In our experiments, we first use the High Performance Computing Linpack Benchmark (HPL) (V1.0a) [21] 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. Other specific configuration parameters for the HPL benchmarks are available in an extended version of this paper [16]. 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 is divided into CINT2006 and CFP2006 which consist of integer and floating-point benchmarks, respectively. The reported result is the average of all benchmarks in each category.

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. Since the shortest period for our power meter to sample power is 1 second, the control period T for all 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 violation, our control algorithm can achieve much faster response time when it runs on high-end server clusters equipped with high-precision power monitor that can sample power in a much shorter period. In such a cluster where all servers share a common power supply, the control period should be derived to make sure that the settling time of the controller is shorter than the designed time interval that the power supply can sustain a power overload [4].

6.2 Comparison to Ad Hoc

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

6.2.1 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 power consumption never settles to the set point and has a steady-state error of -3.8 W.

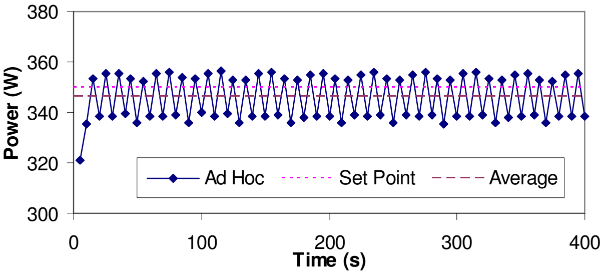

Figure 4. A typical run of Ad Hoc

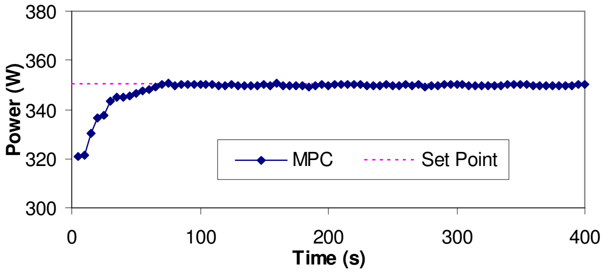

Figure 5. A typical run of the MPC controller

Figure 5 shows a typical run of our MPC controller. In contrast to Ad Hoc, MPC accurately achieves the desired power set point by having a floating-point (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 5, the overhead is usually ignorable 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

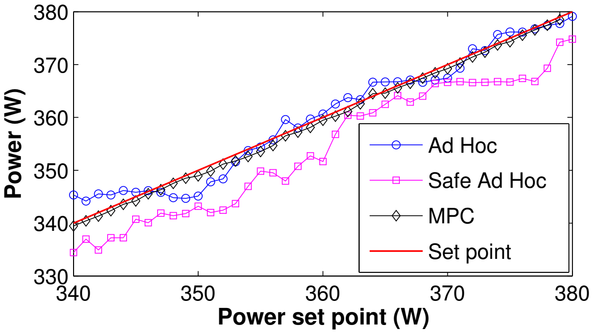

Figure 6. Comparison of steady state errors

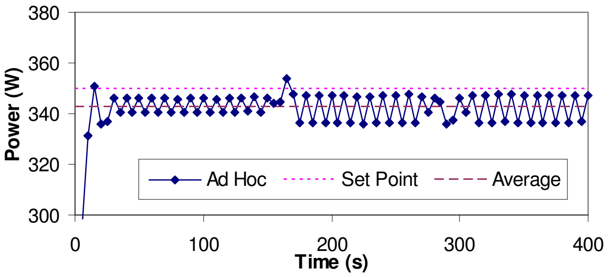

Figure 7. A typical run of Safe Ad Hoc

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.

Since Ad Hoc has steady state error, it is 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 [4]. 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.

6.2.2 Application Performance

In this subsection, we investigate the impact of cluster-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 *impossible* in practice for Safe Ad Hoc to have such a tight safety margin 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.

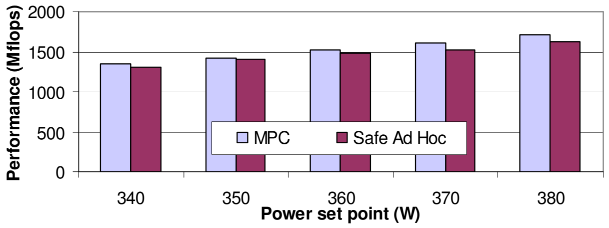

Figure 8. Application performance comparison

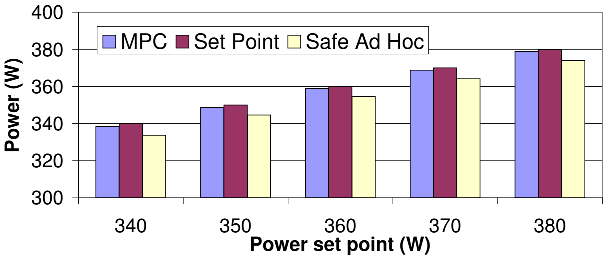

Figure 9. Power consumption using SPEC

6.2.3 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 9 shows that the power consumption of the cluster 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 better application performance (*i.e.*, the relative CPU speed compared to the reference machine used by SPEC) than Safe Ad Hoc for both CINT and CFP, as shown in Figure 10. More results using SPEC CPU2006 are available in an extended version of this paper [16].

6.3 Comparison to SISO

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

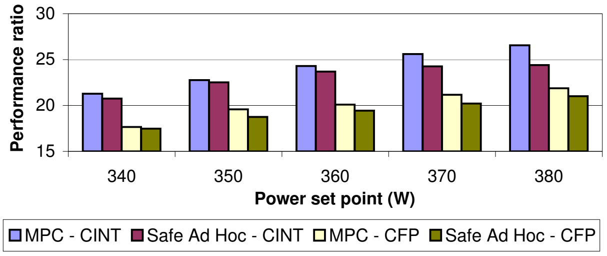

Figure 10. Application performance using SPEC

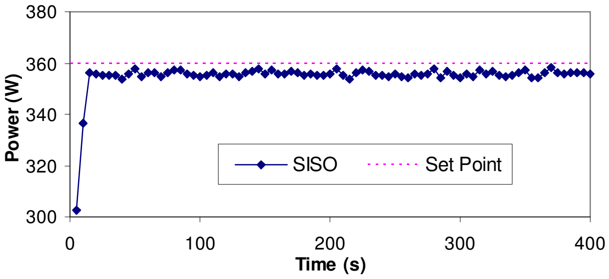

Figure 11. A typical run of SISO

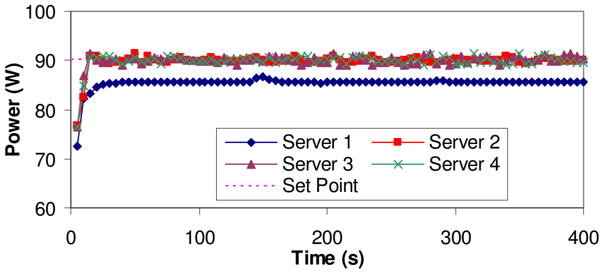

Figure 12. Power consumption of each server under SISO

The 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 11 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 12. On the other hand, the three busy servers (Server 2 to 4) cannot get enough power so that they can only run at degraded frequency levels.

Figure 13 shows the HPL performance data of MPC and SISO in this experiment. MPC has better performance because its total power consumption can exactly reach the desired power budget. In contrast, SISO policy has worse performance because some power budget is wasted by the idle server due to the even distribution of total budget. The maximum performance improvement of MPC is 15.3% at 360 W. Hence, it is clearly important for different servers in a cluster to share power resource because they commonly have non-uniform workloads. Even though it might be possible for the SISO control policy to have a cluster-level supervisor to dynamically adjust power budget among different SISO controllers at runtime, doing so in practice usually involves negotiation between the supervisor and the local controllers and thus cause undesired long response time. In addition, it may also cause each local controller to have constantly varying set point and thus may affect the stability of the whole cluster.

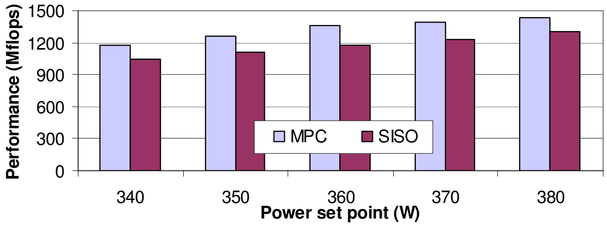

Figure 13. Application performance comparison with SISO

7 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 [22]. There has been some work on system-level power and thermal management. Zeng et al. [23] and Lu et al. [24] have developed strategies to reduce server power consumption. Fan et al. [25] investigate the aggregate power usage characteristics of a large number of servers at the datacenter level. Felter et al. [6] use open-loop control to shift power between processor and memory to maintain a server power budget. Brooks et al. [26] use ad-hoc control to limit processor temperature so cheaper heat-sinks can be used. 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 [10].

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 [27]. Feedback control scheduling algorithms have been developed for operating systems [28] and real-time systems [29]. Control techniques have also been applied to data service and storage systems [30], networks [31], and Internet servers [10]. Decentralized control algorithms have also been developed to control large-scale distributed systems [29].

Several research projects [3, 8] have successfully applied control theory to explicitly control power or thermal of computing servers. Lefurgy et al. [4] 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 chassis level where *multiple* blade servers run the same software service and share common power supplies. Ranganathan et al. [12] propose a negotiation-based algorithm to allocate power budget to different servers in a chassis for better performance. Femal et al. [32] 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 chassis level by sharing power budget.

Some prior work has been proposed to use power as a tool for application-level performance requirements. For example, Zhu et al. [33] have developed a feedback control scheduling algorithm using dynamic voltage scaling (DVS). Sharma et al. [34] effectively apply control theory to control application-level quality of service requirements. Chen et al. [35] 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.

8 Conclusions

In this paper, we presented a cluster-level power controller that shifts power among servers based on their performance needs, while controlling the total power of the cluster 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.

Acknowledgements

We would like to thank Charles Lefurgy and Malcolm Ware at the IBM Austin Research Laboratory for their insightful discussions, and anonymous reviewers for their valuable comments. This work is funded in part by NSF under Grant No. CNS-0720663.

References

[1] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *IEEE Computer*, vol. 37, no. 11, pp. 68–74, 2004.

[2] 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.

[3] 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.

[4] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC)*, 2007.

[5] 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.

[6] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," in *ICS*, 2005.

[7] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: dynamic speed control for power management in server class disks," in *ISCA*, 2003.

[8] 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.

[9] 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.

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

[11] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *ICDCS*, 2006.

[12] 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.

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

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

[16] X. Wang and M. Chen, "Feedback Multi-Server Power Control in a Computing Cluster, Tech Report, Electrical Engineering and Computer Science, University of Tennessee," http://www.ece.utk.edu/∼xwang/papers/power-tr.pdf, 2007.

[17] F. L. Lewis and V. L. Syrmos, *Optimal Control, Second Edition*. John Wiley & Sons, Inc., 1995.

[18] Electronic Educational Devices Inc., "Watts Up Pro Power Meter," http://www.wattsupmeters.com.

[19] *Netperf: A Network Performance Benchmark*, Information Networks Division, Hewlett-Packard, Cupertino, CA, Mar. 1993, edition B.

[20] 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.

[21] 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/.

[22] 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.

[23] 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.

[24] Y.-H. Lu, L. Benini, and G. D. Micheli, "Operating-system directed power reduction," in *ISLPED*, 2000.

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

[26] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors." in *HPCA*, 2001.

[27] 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, June 2003.

[28] 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.

[29] 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.

[30] 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.

[31] S. Keshav, "A control-theoretic approach to flow control." in *Proceedings of ACM SIGCOMM*, 1991.

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

[33] Y. Zhu and F. Mueller, "Feedback EDF scheduling exploiting dynamic voltage scaling," in *IEEE RTAS*, 2004.

[34] 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.

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