

Feedback Control Algorithms for Power Management of Servers

Zhikui Wang, Cliff McCarthy, Xiaoyun Zhu, Partha Ranganathan, Vanish Talwar

Hewlett-Packard Laboratories

Abstract

Power delivery, electricity consumption and heat management are becoming key challenges in data center environments. Solutions have been developed for average and peak power management in the data center. However, these individual solutions are not coordinated resulting in interference and inefficiency. In this paper, we focus on feedback control algorithms for unified power management of a group of servers through frequency scaling knobs. We present individual efficiency and server capping algorithms, as well as their combined deployment through a unified control architecture. We study the dynamic control algorithms with qualitative and quantitative analysis. The overall results through trace-driven simulations show that the servers under integrated control algorithms achieve good tradeoff among power capping, efficiency and application performance.

1. Introduction

Power and cooling are emerging to be key challenges in data center environments. A recent IDC report estimated the worldwide spending on enterprise power and cooling to be more than \$30 billion and likely to even surpass spending on new server hardware in the near future. The increased power also has implications on electricity costs, with many data centers reporting millions of dollars for annual usage. From an environmental point of view, the Department of Energy's 2007 estimate of 59 billion KWhrs spent in U.S. servers and data centers translates to several million tons of coal consumption and greenhouse gas emission per year.

While there has been a lot of progress made on this problem, one of the key challenges has been the fact that the proposed solutions only address individual aspects of the problem in isolation. For example, commercial products already exist that address average power through control of power states (P-states) defined by the ACPI standard [2], at the platform level. An example would be the *HP Power Regulator* [3] that does this through the on-board management processor. Similarly, prior work has discussed power capping solutions that address peak power, at the processor level [1], as well as at the blade server and cluster level [5][7][8]. Power management can also be implemented at the software level, for example, in the OS, in the distributed resource scheduler, and recently through virtual machine consolidation solutions. In the absence of coordination between these various solutions, they are likely to interfere with one another, in unpredictable (and potentially dangerous) ways.

Recent solutions have been proposed to deal with the interaction of the individual power management solutions. A broad *unified* architecture is proposed in [9][6] that integrates power management solutions for tracking, capping, and optimization problems. In [6], architecture and algorithms are developed for coordinating individual power capping controllers to meet power-performance targets. In this paper, we study one architecture for power management that integrates both power efficiency control and power capping. Our solution includes (1)

an adaptive power efficiency controllers using dynamic voltage/frequency scaling, (2) hierarchical controllers for both server-level and group-level power capping through frequency scaling and redistribution of group power budget among individual servers, and (3) an architecture that integrates power efficiency control with power capping. Our work is different from prior work in that we focus on the design of the feedback control algorithms using typical server power and performance models derived from real servers. We provide qualitative analysis on the stability and responsiveness of the algorithms, and validate the results through simulations. The overall evaluation through simulations shows that the proposed control architecture and algorithms can achieve good tradeoff among power capping, efficiency and application performance for various production workload traces.

The remainder of the paper is organized as follows. Section 2 presents our control system architecture for both average and peak power management. Section 3 describes the power and performance models of two typical servers, and introduces our simulator for evaluating power management solutions. Sections 4, 5 and 7 study the three controllers for server power efficiency, server power capping, and group power capping, respectively. Simulation results on the overall performance are presented in Section 6. Section 7 concludes the paper.

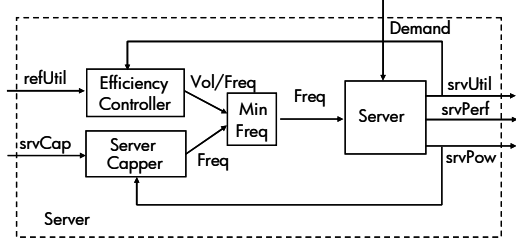
2. Control System Architecture

Figure 1 describes the control system architecture for a group of servers. It is composed of three controllers: an *Efficiency Controller* (EC) that adapts the power consumption of the individual server to track the demand of the workload, a *Server Capper* (SC) that throttles the server power consumption and a *Group Capper* (GC) that throttles the total power consumption of the group of servers.

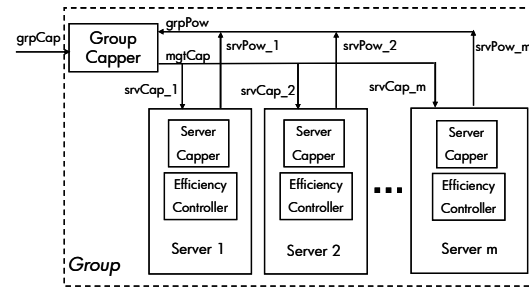
For power efficiency control, we consider the server as a "container" that should be used at a desired fraction of its capacity, or "utilization", notated as the reference (*refUtil*). Regulating resource utilization around its reference drives the *Efficiency Controller* to dynamically "resize the container" by varying the processor voltage/frequency through *P-states*. This allows the power consumed to adapt to the resource demand the workload places on the server in real time. The target utilization is set up for performance purpose. It is usually lower for workloads with bursty demand than that for workload with less variance.

The power capping is enforced through two controllers. At the group level, the *Group Capper* collects the power consumption of the individual servers (*srvPow*'s) and the total power consumption of the group (*grpPow*), based on which it distributes the group power budget (*grpCap*) to the group members (*srvCap*'s). At the server level, the *Server Capper* measures the per-server power consumption (*srvPow*), compares it to its power budget (*srvCap*), and drives the server power to the budget by changing the processor operating frequency.

The two-layered control design for power capping provides architecture for peak power management that is easy to implement, reliable and distributed. The *Server Capper* can be implemented in local servers. The *Group Capper* can be located at different levels, for instance, enclosures of blades, racks, or even data center. They work in different time scales. The control intervals of the *Server Capper* can be in seconds. The *Group Capper* has to communicate with the individual servers. It runs with lower frequency, e.g., in tens of seconds.



(a) A server under control of *Efficiency Controller* and *Server Capper*



(b) A group of servers with power consumption throttled by *Group Capper*

Figure 1: Control system architecture

The *Efficiency Controller* also runs in individual servers. Among the three controllers, it runs in the smallest time scale, e.g., 100ms, so that it can track bursty workload demands. Both *Efficiency Controller* and *Server Capper* tune the frequency. To avoid conflict, the lower frequency is finally actuated. This means that, when the server power is below the server budget, the *Efficiency Controller* tends to be dominant. But if the server power is above the budget, the *Server Capper* can throttle the power consumed even if the utilization is above the reference, which can potentially compromise the application performance due to the high utilization.

The architecture in Figure 1 is based on the following assumptions on the relationships between a number of metrics:

- For a given operating frequency, both the utilization of the server and its power consumption are monotonically increasing functions of the resource demand;
- For a given workload with certain resource demand, the utilization of the server (or its power consumption) is a monotonically decreasing (or increasing) function of the operating frequency.

In the next section, we describe these relationships in detail using data collected from two real servers.

3. Server Models

It is well known that processors these days contribute to a significant portion of the total power consumed by a server, and the processor power consumption varies with the CPU demand of the workload running on the server. In this section, we consider two real servers with different types of processors and different available frequency settings, and approximate the relationships among various metrics using mathematical models.

3.1 Servers with manageable power consumption

We consider two servers: Server A and Server B. They have two types of processors, expose different number of P-states, and show different power and performance characteristics.

- Server A has AMD processors with six P-states, P0, P1, ..., P5, operating at a core voltage/frequency's of 1.35/2.6, 1.30/2.4, 1.25/2.2, 1.2/2.0, 1.15/1.8, 1.10/1.0 V/GHz, respectively.
- Server B has Intel processors supporting two P-states, P0 and P1, operating at a core frequency of 3.0 and 2.0 GHz, respectively. Different from Server A, a set of so called *Q-states* are defined for power management purpose. Q0 and Q1 are the same as P0 and P1. Seven more Q-states, Q2, Q3, ..., Q8 are defined, at which the clock execution time varies between 87.5% and 12.5%, in a step size of 12.5%. In the remaining time, the clock is stopped. This is modulated through the STPCLK pin of the processors.

In this paper, we assume that the capacity of each Q-state is defined by a certain core frequency. For instance, Q3 of Server B is equivalent to a core "frequency" status of 1.75GHz ($=2.0 \times 87.5\%$) in terms of processing capacity. The difference between P-states and Q-states is twofold. First, Q-states can provide a larger range for power capping since it can tune the operating frequency at a much finer granularity. Second, for the Q-states Q2 to Q8, the processor is operating at the same voltage, which means it has a lower performance/power ratio compared to the P-states. That is why, as in Figure 1(a), the *Efficiency Controller* tunes the P-state only, but the *Server Capper* uses the Q-states as additional knobs for power capping.

3.2 Power and performance models as functions of resource utilization

The models described in this section were derived from data collected in benchmarking experiments, where CPU intensive workloads with different levels of CPU demand were applied to the servers under test. The power consumption of the servers and the application throughput were collected for all the operation frequencies. More detailed description on the approach is available in [4].

The data from the experiments suggest that for a fixed operating frequency, both the power consumption of the server and the application throughput are approximately linear functions of the utilization of the processor. The latter is defined in percentage of the maximum CPU capacity available for a given operating frequency. Based on these observations, we fit the measured data using the following two linear models:

$$srvPow = g_i(srvUtil) = c_i \cdot srvUtil + d_i, \text{ for each frequency } i \quad (1)$$

$$srvPerf = h_i(srvUtil) = a_i \cdot srvUtil, \text{ for each frequency } i \quad (2)$$

We refer to Equation (1) as the *power model*, approximating the relationship between the server power consumption and the

CPU utilization, and Equation (2) as the *performance model*, approximating the relationship between the application performance (throughput in the data) and the CPU utilization. These two models are illustrated in Figure 2(a) and 2(b).

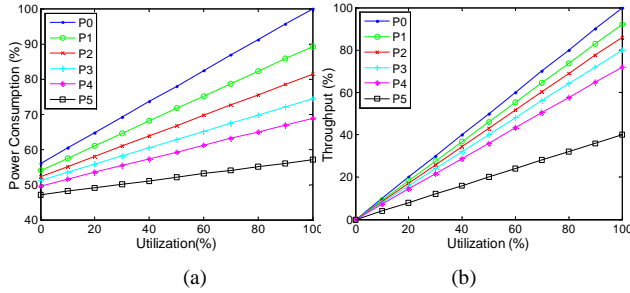


Figure 2. Power and performance models for Server A: (a) Linear function representing the relationship between power consumption and CPU utilization; (b) Linear function representing the relationship between application throughput and CPU utilization.

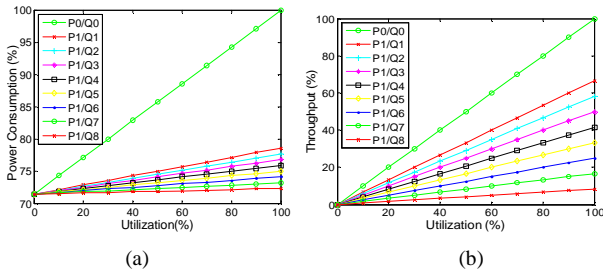


Figure 3: Power model (a) and performance model (b) for Server B in different Q-states, as linear functions of the CPU utilization.

Similar experiments were run on Server B. Again, we found that linear models can be used, as shown in Figure 3. Note that the values of the parameters c_i , d_i and a_i in Equations (1) and (2) are dependent on both the type of the server and the nature of the workload. It is also worth noting that the power model does not explicitly account for power consumptions from other parts of the server, including memory and disk. Instead, it adds a constant offset d_i to the equation that accounts for the basic power consumption even when the processor is idle.

3.3 A simulator for evaluation of server power management

We have developed a simulator for evaluation of power management solutions. In this simulator,

- Each server is driven by a workload with time-varying CPU demand. For any given demand, the actual utilization, throughput and power consumption are simulated using the models shown in Figure 2 and Figure 3.
- Each server implements an *Efficiency Controller* and a *Server Capper* as described in Figure 1(a). Each server group implements a *Group Capper*.
- When the frequency is tuned by either controller, the server behavior is simulated based on the fact that, the resource demand, which is proportional to the product of utilization and core frequency, is conserved except when the utilization is bounded at 100% in the new frequency state.
- We don not use queues that carry over the unfinished work from one interval to the next interval. Instead, we assume that demands that cannot be served in one interval are lost.

The statistics on loss provides one way to evaluate the controller performance.

4. Power Efficiency Control and its Stability

For the purpose of *Efficiency Controller* design and analysis, Figure 4 shows the utilization of both Server A and Server B, as a function of the frequency for different levels of workload demand. Note that the frequency is quantized, and especially, only two frequencies are available for Server B since only P0 (Q0) and P1 (Q1) are used for the power efficiency control.

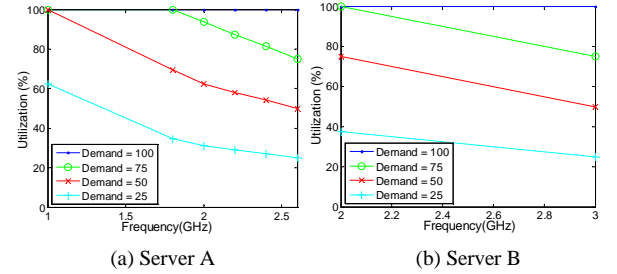


Figure 4. Server utilization as a function of P-state frequency

4.1 Instability and slow response in threshold-based control

A threshold-based controller is usually implemented for efficiency control, that is, the frequency is tuned up or down by one step when the utilization is above or below a reference value or a target region. The algorithm is simple, easy to implement, and requires little information from the processor. However, from dynamic control point of view, it has two issues:

First, the controller may be too aggressive in responding to the tracking error. Take Server B shown in Figure 4(b) as one example. For a workload with a certain demand, the P-state can be oscillating between two consecutive P-states, P0 and P1 in Server B, if the utilization reference falls into the range $[util_0, util_1]$, where $util_0$ and $util_1$ are the actual utilizations at the two P-states. The average utilization is $(util_0 + util_1)/2$. In an even worse case, the server could be overloaded half of the time if $util_1$ hits 100%, causing loss of application throughput and/or longer latency.

Second, the controller may be too sluggish to react to changes in the workload, which could also cause performance loss. Take Server A as another example. For a workload with a demand 25%, the P-state will be set to P5 for a utilization reference of 75%. When the demand changes, for instance, to 75%, the equilibrium P-state should be P0. However, it will take the threshold-based controller 5 steps to push the P-state from P5 to P0 since it can only go to a neighboring P-state in each step. This delayed response could cause, e.g., long response times for transaction based workloads.

4.2 An adaptive efficiency controller

Figure 5 describes an adaptive algorithm for *efficiency controller*. Equation (3) in Figure 5 actually implements an integral control law where the change in the clock frequency is proportional to the observed error in utilization. The integral operation can push the steady-state error to zero, i.e., the utilization can reach its target, in spite of changes in the workload or in the target itself.

Our integral controller has a self-tuning integral gain that is proportional to the resource demand represented using the

utilized capacity. Mathematical analysis shows that the system under control is locally stable under the condition that $\lambda \in (0,2)$. A sufficient condition can also be determined that the system is globally stable if $\lambda \in (0,1/r_{ref})$. Due to space limit, we omit the analysis and simulation study of the controller's stability and responsiveness. A sketch of the proof for global stability can be found in [9].

At the beginning of each interval k ,

- 1) Polls the average CPU utilization in the previous interval $util(j-1)$;
- 2) Based on the core frequency in the previous interval $f_Q(j-1)$, get

$$f(j) = f(j-1) - \lambda \frac{f_Q(j-1)util(j-1)}{refUtil} (refUtil - util(j-1));$$
 (3) If $f(j) > f_0$, $f(j) = f_0$, the core frequency at P0;
 Else if $f(j) < f_{min}$, $f(j) = f_{min}$, the minimum core frequency ;
- 3) Quantize $f(j)$ to $f_Q(j)$, the lowest P- state frequency that is above $f(j)$;
- 4) Enforce the new P- state if $f_Q(j) \neq f_Q(j-1)$.

Figure 5: An adaptive power efficiency controller algorithm

5. Server Power Capping

Before we describe a detailed *Server Capper* algorithm, we will first analyze the models that describe the nonlinear relationships between the operating frequency and the power consumption of the servers for a given workload demand.

5.1 Models for server power capping

Figure 6 shows the power consumption of the two servers as a function of the frequency, for different demand levels. Note that the demand may not always be satisfied when the frequency is reduced, which explains why the lines could converge.

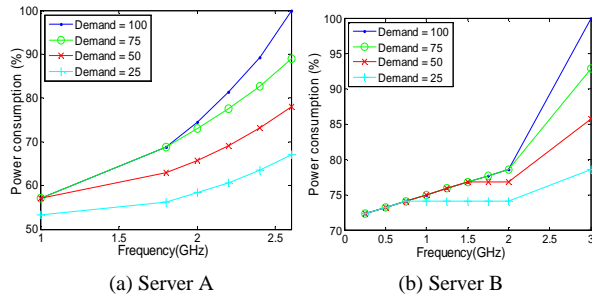


Figure 6: Power models as a function of frequency

First, for a given workload demand, the power consumption is not a linear function of the frequency. Figure 6(a) shows that the power consumption of Server A is a convex function of the frequency. This behavior is due to the different voltage levels at the different P-states.

Second, for Server B, the power consumption can be approximated by a piecewise linear function of the frequency. In particular, for those performance states where only “frequency” is throttled while voltage stays unchanged (using Q-states), for instance, when the demand is 25% and the frequency varies between 0.75GHz and 2GHz, the power consumption remains a constant. At those points, the demand can be satisfied.

Third, the gains from frequency to power consumption of Server B are fairly different at the two sets of performance states (P-states and Q-states). In fact, those two sets of states also have significantly different throughput/power (e.g., requests/watt) performance. Figure 7 shows the relationships between the

maximum throughputs and the peak power consumptions of the two servers. For both servers, states with higher frequencies have better throughput/power performance, that is, less throughput loss for a given percentage of power reduction. For example, for Server B, the throughput/power performance of the states Q2 to Q8 is much worse than that of Q0 and Q1.

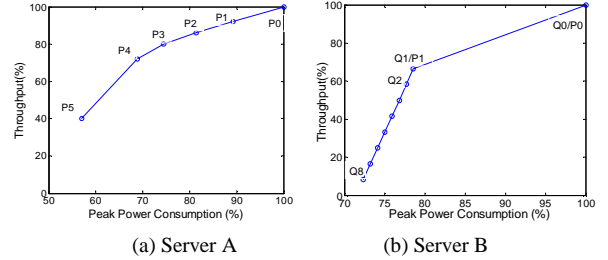


Figure 7: Different throughput/power performance at different performance states (P-states or Q-states)

5.2 An adaptive server capping algorithm

The nonlinear relationship between frequency and power consumption imposes challenges for design of the server power capping algorithms. We first describe one such example.

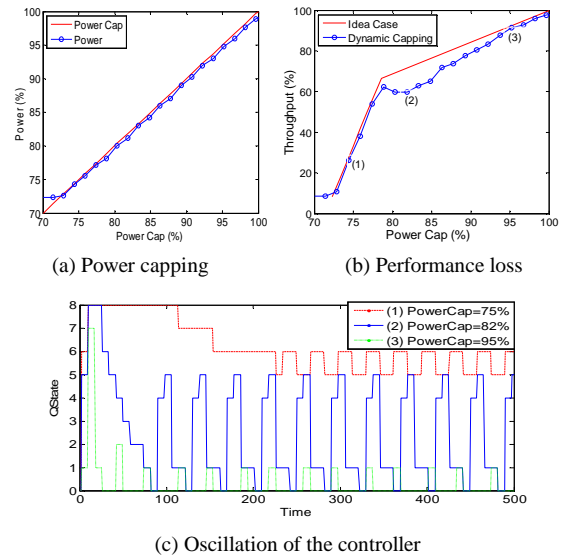


Figure 8: Power and application performance of Server B under control of a PID controller

Figure 8 shows the power consumption and throughput for Server B under control of a well-tuned PID power capping algorithm. We used a demand that was higher than the capacity of the server so that only the power capping controller, not the efficiency controller, was affected. We varied the power budget (or cap) between 70% and 100% of the maximum power. From Figure 8(a), we can see that the mean power consumption can be kept below the budget. However, around point (2) in Figure 8(b), the actual throughput (blue line) was much lower than the ideal case (red line). This deteriorated efficiency was caused by the piecewise linearity as illustrated in Figure 6(b). Figure 8(c) provides more details for the reasons behind. For power caps of 95% and 75%, corresponding to points (1) and (3) in Figure 8(b), the power state was tuned between Q0 and Q1, or Q5 and

Q7, respectively. However, when the budget was set to 82%, the power state was oscillating between Q0 and Q6, although the equilibrium should be between Q0 and Q1. That means the controller was too aggressive at those points. Pushing the power states to those with lower performance, i.e., Q4 and Q5 compromised the processing capability of the server, although the mean power could be kept below the budget.

Is it possible to reduce the controller gains to damp the oscillation? From Figure 8(c), we can see that the chosen gains are appropriate for a budget of 95%, but the controller is sluggish for a budget of 75% as evident by the long settling time. This means the gain values cannot be reduced further.

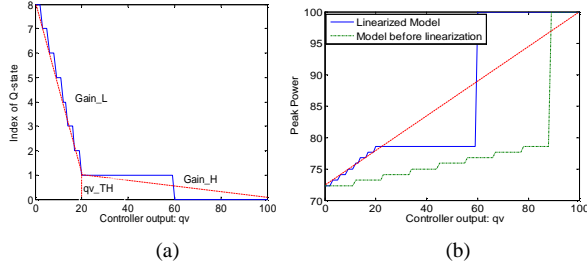
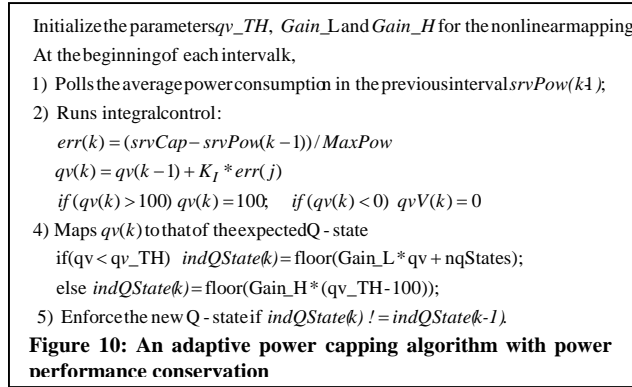


Figure 9: Nonlinear mapping and linearization for Server B. (a) Nonlinear mapping from controller output to the frequency; (b) Linearized relationship between controller output and power consumption.

To deal with the nonlinearity from frequency to power consumption, we introduce a piecewise linear transformation from the controller output to the operating frequency as in Figure 9. The controller output is no longer mapped uniformly to the frequency. As in Figure 9(a), the controller outputs are multiplied by a larger gain $Gain_L$ when it is below a threshold qv_TH . Otherwise, the controller outputs are multiplied by a smaller gain $Gain_H$. After this transformation, the relationship between the controller output and the peak power consumption is linearized as shown in Figure 9(b). The detailed algorithm is described in Figure 10. As in the *Efficiency Controller*, an integral controller is used in Step 2), where the error between the power budget and the actual consumption is normalized by the maximum power consumption. The output of the controller is bounded within $[0, 100]$, and then mapped to the frequency non-uniformly.



To evaluate the algorithm in Figure 10, we repeated the same experiments as for Figure 8. The results are shown in Figure 11. Again, the power consumption can be maintained below the budget. Moreover, the performance in terms of the throughput is

close to the ideal case, as shown in Figure 11(b). The time series in Figure 11(c) show that, the adaptive algorithm not only eliminates the oscillation we saw in Figure 8(c) for a power cap of 82%, but also allows the performance state to approach the equilibrium much more quickly for all the three power caps .

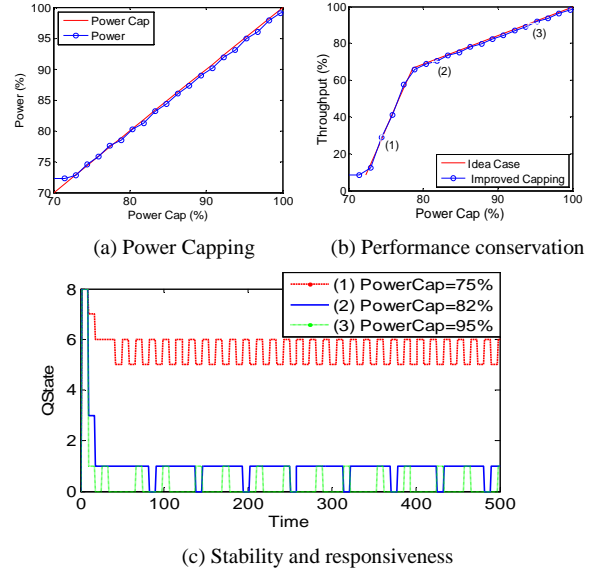
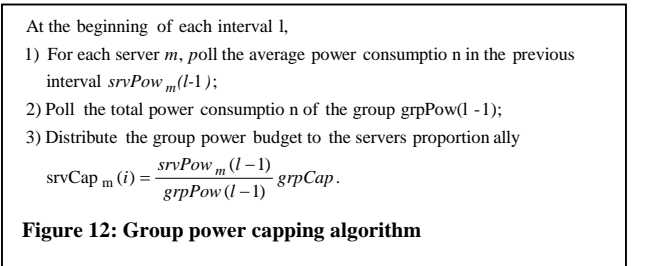


Figure 11. Performance and dynamic properties of the adaptive capping algorithm

6. Group Power Capper

The goal of the *Group Capper* is to distribute the group power budget to the individual servers by setting up their power budgets, which may then be enforced by the *Server Cappers*. Many policies are available for redistribution of the budget. Figure 12 describes one proportional sharing policy, i.e., the group power budget is allocated to the individual servers proportionally to their power consumptions.



Although very simple, the algorithm in Figure 12 still holds reasonable properties.

- It is easy to find that, at equilibrium, the budgets of the servers are equally utilized.
- The budget of a server is increased when it consumes more power, in the case that the power consumptions of the other servers are unchanged.
- In the special case when the budget of a server is used up, greater budget is allocated to the server than that in the previous interval, if there were extra budget available.

However, it is not always guaranteed that the budgets in the individual servers are enforced by the *Server Cappers*. For instance, overshoots could happen since the power capping in

the servers is defined as a tracking problem, and implemented as an integral controller. Implementation of the policies needs to take care of these exceptional cases.

7. Overall Evaluation through Simulations

The integrated architecture was evaluated through simulations driven by workload traces. This approach enables the time-varying workload behavior and system characteristics to be modeled expediently while allowing detailed evaluation. The traces we used were collected from production environments such as e-commerce applications, SAP applications, enterprise web sites, and servers in a pharmacy. The average demand of the sets of the workloads is between 4% and 27%. We run simulations with configurations of different types of servers, multiple workload sets, a range of budget levels and different controller combinations. The same models as described in previous sections were used in the simulation. We are not able to enumerate all the results here. But overall, we found that the integrated architecture shows reasonable tradeoff among power saving, application performance loss and budget enforcement.

As one example, Table 1 shows the statistics of two groups, composed of 20 Server A and 20 Server B respectively, under control of different combinations of the *Efficiency Controller*(EC), *Server Capper*(SC) and *Group Capper*(GC). The simulations for the data were driven by traces from a set of enterprise web servers, with 22.5% of average demand. The control intervals of EC, SC and GC, if any, were set to 1, 8 and 80 time units respectively. The budgets were increased in series of simulations. The data in the table were from the two cases when the servers were under control of only EC and budget violations started to show up. The budget of the first group was 65%, and that of the second group was 80%. Both were aggressive enough so that violation happened when the group is under control of only EC.

Table 1. Statistics on performance of two groups under different combinations of the controllers

Controllers	Group of Server A's with 65% of budget level			Group of Server B's with 80% of budget level		
	Power Cons.	Perf. Loss	Budget Viol.	Power Cons.	Perf. Loss	Budget Viol.
None	69	0	86	80	0	40
EC	58	2	16	77	0.4	20
EC+SC	54	20	0	75	7.8	0
SC+GC	60	19	4	79	5.6	12
EC+SC+GC	55	15	0	76	3.9	0

Three performance metrics are shown in Table 1: the average power consumption of the group, as a percentage of the maximum one when the servers were fully utilized, the performance loss due to overloading, as a percentage of the total demand, and the group budget violation levels, defined as the *Group Capper* control intervals during which the group power consumptions went to above the budgets, as a percentage of the total number of the GC intervals. A few observations can be made from the data in the table.

- The contribution of EC to power saving is dominant.

- The performance of the integrated control, "EC+SC+GC", is similar as that of "EC+SC", with minor improvement of application performance but increment of power consumption. The effect of GC can be seen in cases with less aggressive budget levels where the *Group Capper* helps redistribute the group budget to the individual servers with different resource demands.
- In the case SC+GC where EC was disabled, least power is saved. Moreover, the group power budget can not be always enforced. The main reason behind is that, the power consumptions of the servers can be bounded. The error between the power and budget could cause "windup" of the SC, and so that budget violation.

8. Conclusions

In this paper, we study an integrated average and peak power management solution for a group of servers, using frequency scaling as the knob. Power and performance models created from experimental data on real servers help us on design of the feedback control algorithms and analysis of stability and responsiveness of the servers under control. Especially, we develop one power capping algorithm that can deal with the nonlinearity of the servers due to different performance/power ratio of the knobs. We evaluate the integrated architecture through simulations driven by production traces with time-varying resource demand. Results show that the architecture and the algorithms are able to provide reasonable tradeoff among power efficiency, power budget enforcement and application performance guarantee.

9. References

- [1] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *7th Int. Symposium on High-Performance Computer Architecture (HPCA)*, 2001.
- [2] I. Corporation, M. Corporation, and T. Corporation. Advanced configuration and power interface specification. Online, December 1996. <http://www.teleport.com/acpi>.
- [3] Hewlett Packard. HP Power Regulator for Proliant. Online. <http://h18013.www1.hp.com/products/servers/management/ilo/power-regulator.html>.
- [4] P. Ranganathan and P. Leech. Simulating complex enterprise workloads using utilization traces. In *10th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, February 2007.
- [5] P. Ranganathan, P. Leech, *et al.* Ensemble-level power management for dense blade servers. In *33rd International Symposium on Computer Architecture (ISCA)*, 2006.
- [6] J. Kephart, H. Chan, R. Das, D. Levine, G. Tesauro, F. Rawson, and C. Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *4th Int. Conference on Autonomic Computing (ICAC)*, June 2007.
- [7] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *4th Int. Conference on Autonomic Computing (ICAC)*, June 2007.
- [8] X. Wang and M. Chen. Cluster-level Feedback Power Control for Performance Optimization. In *14th IEEE Int. Symposium on High-Performance Computer Architecture (HPCA)*, February 2008.
- [9] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: A unified multi-level power management architecture for the data center. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.