Energy Conservation Policies for Web Servers *

Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony
Low-Power Computing Research Center
IBM Research, Austin TX 78758, USA.
http://www.research.ibm.com/arl

Abstract

Energy management for servers is now necessary for technical, financial, and environmental reasons. This paper describes three policies designed to reduce energy consumption in web servers. The policies employ two power management mechanisms: dynamic voltage scaling (DVS), an existing mechanism, and request batching, a new mechanism introduced in this paper. The first policy uses DVS in isolation, except that we extend recently introduced task–based DVS policies for use in server environments with many concurrent tasks. The second policy uses request batching to conserve energy during periods of low workload intensity. The third policy uses both DVS and request batching mechanisms to reduce processor energy usage over a wide range of workload intensities. All the policies trade off system responsiveness to save energy. However, the policies employ the mechanisms in a feedback–driven control framework in order to conserve energy while maintaining a specified quality of service level, as defined by a percentile–level response time.

We evaluate the policies using Salsa, a web server simulator that has been extensively validated for both energy and response time against measurements from a commodity web server. Three day-long static web workloads from real web server systems are used to quantify the energy savings: the Nagano Olympics98 web server, a financial services company web site, and a disk intensive web workload. Our results show that when required to maintain a 90th–percentile response time of 50ms, the DVS and request batching policies save from 8.7% to 38% and from 3.1% to 27% respectively of the CPU energy used by the base system. The two policies provide these savings for complementary workload intensities. The combined policy is effective for all three workloads across a broad range of intensities, saving from 17% to 42% of the CPU energy.

*This research has been supported in part by The Defense Advanced Research Projects Agency under contract F33615-00-C-1736.

1 Introduction

There are many technical, financial, and environmental motivations to reduce server energy consumption in Internet data centers. In this paper, we describe three policies for reducing energy consumption of server processors during web serving, a common application in data centers. The policies employ two energy management mechanisms: dynamic voltage scaling, an existing mechanism, and *request batching*, a new mechanism we introduce in this paper. All three policies save energy while maintaining server responsiveness at or better than a specified quality of service level. We evaluate the policies using Salsa, a web server simulator that has been extensively validated for accuracy in both energy consumption and response times against measurements from a commodity server. The evaluation uses three day–long static web serving workloads.

The first policy uses dynamic voltage scaling (DVS), a mechanism that lets the processor frequency and voltage be varied dynamically. DVS–based policies leverage the fact that a given task can be completed for less energy if executed at a lower processor frequency and voltage. We extend recently introduced task–based DVS policies [7, 14] for use in environments with many concurrent tasks, such as in web servers. The second policy is based on a new mechanism called *request batching* that groups requests received by the server and executes them in batches, placing the server processor in a low–energy state between batches. These policies are complementary in two respects: they require different kinds of support from the system hardware and are effective over different ranges of workload intensities. The third policy combines both request batching and dynamic voltage scaling mechanisms to conserve energy across a broader range of workload intensities than the individual policies. All three policies trade off system responsiveness in order to save energy. However, the policies employ a feedback–driven control framework in order to conserve energy while maintaining a given quality of service level, as de-

fined by a percentile–level response time.

The request batching and dynamic voltage scaling policies target complementary ranges of workload intensities: for a given QoS level, request batching works well when the workload is light, while DVS yields more savings as the workload intensifies. The combined policy reduces energy consumption over a broad range of workload intensities. For the range of workloads and intensities examined in this paper, we find that request batching provides from 3.1% to 27% savings in CPU energy consumption while dynamic voltage scaling provides from 8.7% to 38% savings. The combined policy provides savings in CPU energy consumption ranging from 16.6% to 41.9%. All these savings are realized while maintaining the 90th–percentile first–packet response times at or better than 50ms.

This paper makes the following contributions:

- Presents a DVS–based policy for use in server environments with concurrent tasks, an evolution of existing task–based DVS policies [7, 14].
- Introduces a new energy management mechanism, request batching, and a policy that employs it.
- Leverages the DVS and request batching mechanisms into a combined policy that achieves energy savings across a broad range of workload intensities.
- Demonstrates that by using a feedback–driven framework, the policies achieve significant energy savings while maintaining system responsiveness at or better than a desired level.

The rest of this paper is organized as follows. Section 2 explains why energy management is crucial for Internet data centers. Section 3 contains a description of the three policies. Section 4 describes the workloads used in this study, and Salsa, the simulator we use for evaluating the policies. The three policies are evaluated in Section 5. A comparison to related work is presented in Section 6. Finally, Section 7 concludes the paper.

2 Why Manage Energy Usage in Data Centers?

There are many technical, financial, and environmental motivations to reduce server energy consumption. For instance, data centers deploy thousands of servers, densely packed to maximize floor space utilization. Such dense deployment pushes the limits of power delivery and cooling systems. Anecdotal evidence from data center operators points to the intermittent failures of computing nodes in densely packed systems due to insufficient cooling. Furthermore, constraints on the amount of power that can be delivered to server racks makes energy conservation critical for fully utilizing the available space on these racks.

Beyond these technical motivations for reducing energy usage, there are two compelling financial incentives for data centers in the United States to manage server energy usage. Similar financial arguments may also apply in other parts of the world. First, data centers typically bundle energy costs into the cost they charge consumers for hosting. For instance, the average price to rent a full rack of space (about 3ft × 3ft × 6ft) at a data center is approximately US$1000 per month as of September 2002[1]. For this price, the data center provides physical space, energy, energy backup (UPS), cooling, and support services (offices). Bandwidth is metered and is usually an extra charge. While the amount of power provided per rack varies from one data center to another, all centers include at least 4KW of power per rack, with some delivering up to 7KW. Thus, with energy costs averaging 8 cents per kWh, rack energy usage alone could account for up to 23% to 50% of colocation *revenue*. Similar arguments apply for managed hosting (where the data center controls and owns the IT equipment and provides the service) as well. Consequently, there is a strong incentive to minimize server energy usage in data centers.

Second, faced with the prospect of constructing new facilities in order to meet the electricity demands of data centers, many utilities have instituted rate tariffs or upfront deposits [16, page 56]. For example, Puget Sound Energy's schedule 449 requires the customer to bear costs associated with dedicated generation and/or delivery facilities [6]. Since a data center can always supplement utility-provided power with on-site generation, a center could lower its capital outlay by placing lower demands on the utility and employing on-site generation during periods of

[1] Approximate average colocation price charged by hosting centers such as Mzima (www.mzimahosting.com/prodserv/promotion1.html), XMission (www.xmission.com/business/colocation.html), and Terracom Network Services (www.tns.net/internet/colo.html?gl).

peak power demand. Reducing the average energy usage is critical for this strategy to be successful.

3 Energy Management Policies

Our policies focus on reducing server CPU energy consumption. While CPU energy is only one component of the system energy, our previous research, based on measurements from an actual commodity system that typifies servers currently deployed in Internet Data Centers, determined that the CPU is the dominant consumer of system energy [2]. Furthermore, when considering active system components, the CPU exhibits the most variation in energy consumption. Finally, these policies are applicable to individual web server systems and complement energy management techniques for server clusters [4, 5, 19].

3.1 Feedback–driven Control Framework

All three energy management policies use a feedback–driven control framework to maintain the system responsiveness at a specified level. The system administrator establishes a percentile–based response time goal. A 90th–percentile goal means that 90% of the requests received over a specific interval must have a response time equal to or better than the specified goal. In our policies we compute the 90th–percentile response time over an entire day, but other intervals could be used (see Section 5.1). The server continuously monitors the response time of individual requests as measured by the difference between the time the request was received at the server and the time the first packet of the response was sent to the client. The feedback–driven control framework adjusts policy parameters to increase energy savings when measured response times are lower (better) than the response time goal, or to decrease energy savings when system is not meeting its response time goal. This is done on a best-effort basis and the system may not be able to meet the goal under conditions of extreme load.

When evaluating the policies in this paper, we mostly use a 90th–percentile response time goal of 50ms. Relaxing the server response time to 50ms will have minimal effect on the client perceived response times (CPRT). This is because the CPRT is typically dominated by wide area network overheads [20], enabling a 50ms server response time to be masked by the WAN delays. As a result, the web server appears responsive to the end–user even if it takes up to 50ms to respond to requests. We also show how the energy savings change as both the percentile and response time goal are varied.

While other responsiveness metrics (such as the average response time) could be used, most service level agreements are crafted based on a percentile goal (for instance, see the BS Web Services SLA [22]). The response times for individual requests can be ascertained through a combination of the OS tagging incoming packets with their arrival time, and the web server notifying the OS when requests are serviced. Current versions of Linux already tag incoming packets with their arrival time and support the SIOCGSTAMP ioctl which returns the arrival time of the last packet passed to the application on a specified socket. Web servers commonly generate a time stamp for each request completion, since this information is typically recorded in the web server access log. Thus, computing server response time should require at most one additional ioctl and a simple calculation for each request. Note that the control mechanism only needs to determine the percentage of responses that meet the target, as opposed to actually computing the 90th–percentile response time, which can be computationally expensive.

3.2 Dynamic Voltage Scaling Policy

Dynamic voltage scaling (DVS) policies reduce energy consumption by varying the processor operating point (frequency and voltage) according to the rate at which work must be done. Recent research in DVS policies set the processor operating point using a task–based approach. For instance, Flautner et. al. employ a policy that sets CPU speed on a per–task basis [7]. Lorch and Smith describe an algorithm for improving the performance of task-based DVS policies when task completion times cannot be accurately predicted [14]. These techniques perform well for desktop application workloads but are unsuitable for environments with many concurrent tasks, such as server systems. Consequently, while our DVS policy keeps track of the response time of individual requests ("tasks"), it employs a feedback–driven control framework (explained in Section 3.1) to meet responsiveness requirements in the aggregate, instead of for individual tasks. The

DVS policy adjusts the processor operating point at regular intervals or *quanta* to meet the overall responsiveness requirement.

At the beginning of each quantum, the DVS policy selects an operating point (voltage and frequency) for the next quantum based on the response times for all previously serviced requests. If the system is more responsive than required, the processor frequency is decreased by one step (if not already at its minimum) and the voltage is set accordingly. If the response time goal is not being met, the frequency is increased by one step (if not already at the maximum) after the core voltage is sufficiently raised.

Dynamic voltage scaling provides the most energy benefits for moderately intense workloads. Since the processor operating point cannot be lowered below a certain level, DVS provides few benefits for low intensity workloads. Likewise, DVS yields little benefit for very heavy workloads, since the processor must run mostly at full speed to meet the responsiveness requirement. The operating point that yields the most savings depends on the processor parameters and the DVS policy. For instance, with the DVS processor we consider in this paper (see Section 4.2 for the parameters), a DVS policy that keeps the processor fully utilized by adapting the operating point to the available load achieves its maximum energy savings for a load that is approximately 58% of the processor's capacity at its highest frequency setting.

3.3 Request Batching Policy

Policies based on dynamic voltage scaling are not very effective at low workload intensities. However, previous studies have observed that Web servers are relatively idle for large fractions of time [13]. Even when idle, server processors consume significant amounts of power. Unfortunately, since incoming requests arrive asynchronously, web servers cannot afford to use energy conserving states with significant wakeup penalties such as the "hibernation" mode commonly found in laptops.

Request batching is a mechanism that we have developed to conserve energy during periods of low workload intensities. In request batching, the servicing of incoming packets from the network is delayed while the main processor of the web server is kept in a low power state. Incoming packets are accumulated in memory until a packet has been kept pending for longer than a specified *batching timeout*[2]. Request batching saves energy because while requests are being accumulated, the processor can be placed in a lower power state such as Deep Sleep [11, Page 82] instead of just idling it. Furthermore, since web servers are typically idle or at low utilization much of the time, the energy saved during idle periods can result in significant energy savings. For example, if placing the processor in Deep Sleep mode reduces power consumption by 2.5 Watts, a web server that is on average only 25% utilized could save 162 KJ of energy per day.

Request batching provides the most energy saving benefits for light workloads. For a given batching timeout, the savings from request batching decrease with increasing workload intensity (until the processor becomes fully utilized) [3]. Increasing the batching timeout saves more energy at the expense of increased response time. In order to meet the specified quality of service (system responsiveness) level, we use a feedback–driven control framework (explained in Section 3.1) to dynamically adapt the batching timeout.

When a processor is in Deep Sleep, it requires the delivery of a specific set of signals in order to be reactivated. Network adapters that support the Wake-on-LAN feature [10] are already capable of waking up a processor in Deep Sleep, and the same mechanisms could be employed here. Furthermore, most network adapters are capable of DMA-ing incoming packets into pre-specified buffers in memory. Similarly, the disk can process pending commands and DMA data into memory. The experience we gained with our prototype (described in Section 4.2.1) indicates that batching timeouts of up to 100ms will not adversely affect TCP performance.

The request batching policy places the processor into Deep Sleep mode when there are no pending requests to be serviced. The policy adjusts the length of the batching timeout based on the system responsiveness. The batching timeout is varied in steps of

[2]We also considered using a *maximum request backlog*, but discovered that the timeout, by itself, provides the level of control needed.

[3]Consider requests arriving at λ/sec, each taking τ seconds to process. Ignoring the time taken to wake up from Deep Sleep, the energy savings from batching will be $(1 - \lambda\tau)(P_{idle} - P_{DS})$ per unit time. Expressed as a fraction over the base energy, the batching savings are: $(1 - \lambda\tau)(P_{idle} - P_{DS})/[P_{max}\lambda\tau + P_{idle}(1 - \lambda\tau)]$. This decreases sublinearly as the request rate increases.

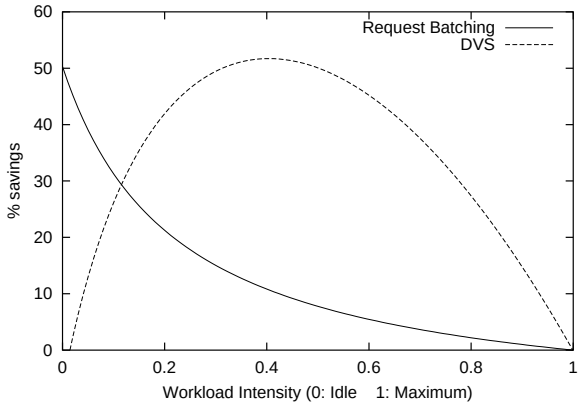


Figure 1: Energy savings (over the base case) from DVS and request batching for a range of workload intensities.

10ms. If the system is more responsive than needed, the timeout is increased one step. If it is not meeting the specified responsiveness target, the timeout is decreased one step. Since the processor operates at full power during the wakeup sequence ($\approx$1.5ms), it is not advantageous to decrease the timeout below a certain limit. If the timeout falls below this limit, the request batching mechanism is disabled until the timeout is again larger than this limit.

3.4 Combined Policy

Dynamic voltage scaling and request batching target different workload intensity ranges on the server. Figure 1 illustrates this behavior graphically. To handle a range of workload intensities beyond what each policy can handle individually, we have devised a combined policy that leverages both request batching and dynamic voltage scaling mechanisms. The combined policy invokes the request batching and voltage scaling mechanisms based on the activity level of the server and reduces energy consumption across a broad range of workload intensities while maintaining system responsiveness at the required level.

When there are no pending requests to be serviced, the processor is placed into DeepSleep mode as in request batching. This policy adjusts the batching timeout as in the request batching policy. When the processor wakes up from Deep Sleep, it is placed at the lowest operating frequency. From this point onwards, the policy adjusts the processor frequency and voltage as in the DVS policy.

4 Methodology

4.1 Workloads

We constructed workloads using web server logs obtained from several production Internet servers. The first workload, Olympics98, is derived from the requests received on Feb 19, 1998 at one geographically replicated facility hosting the 1998 Winter Olympics web site. The second workload, Finance, is derived from the requests recorded on Oct 19, 1999 at the web site of a major financial services company. The third workload, referred to as Disk-Intense, is derived from the May 2, 2001 log of the Silicon Valley (SV) proxy server operated by the Information Resource Caching (IRCache) Project [12]. Strictly speaking, a proxy server is not a web server in that it does not have its own content, but we included this workload to represent a scenario of a web server with extremely low cache locality and thus high levels of disk activity. A complete description of the methodology for generating workloads from server access logs is given elsewhere [2].

The characteristics of the three workloads are summarized in Table 1. The "peak requests per second" is the highest observed rate for a one minute period. The "average requests per connection" is based on our technique of grouping requests into connections [2]. The amount of memory needed to hold the unique data for 97%/98%/99% of all requests is tagged by that moniker.

From the three base workloads, we generate workloads representing a range of client load intensities by scaling the inter-arrival time of connections by a constant amount. A scalefactor of "2.5×" corresponds to reducing the inter-arrival time of connections by a factor of 2.5. While this scaling increases the connection arrival rate by the scalefactor, it maintains the same basic pattern of connection arrivals. We preserve the inter-arrival times of requests within a connection since these represent user think times or network/client overheads in retrieving web page components. Chase et. al. have adopted a similar approach [4].

4.2 Salsa: A Validated Web Server Simulator

In order to evaluate the policies, we have constructed Salsa, a simulator that estimates the en-

Workload	Olympics98	Finance	Disk-Intense
Avg requests (Peak requests) / sec	97 (171)	16 (46)	15 (30)
Avg requests / connection	12	8.5	31
Unique files (Total file size)	61,807 (705MB)	16,872 (171MB)	698,232 (6,205MB)
Distinct HTTP Requests	8,370,093	1,360,886	1,290,196
Total response size (excl HTTP headers)	49,871 MB	2,811 MB	10,172 MB
97%/98%/99% (MB)	24.8 / 50.9 / 141	3.74 / 6.46 / 13.9	2,498 / 2,860 / 3,382

Table 1: Characteristics of three web server workloads over a 24-hour period.

ergy consumption and response time of a web server. Salsa is based on a queuing model built using the CSIM execution engine [15] and determines the CPU time and energy taken to service web requests using a model that is parameterized with energy measurements from a commodity, 600MHz Intel processor–based web server. Among other events, Salsa models process scheduling and file cache hits and misses. At present, Salsa does not model disk delays, but models the extra energy expended by the CPU (due to driver code execution) while making disk accesses.

The processor we simulate with Salsa has maximum power consumption $P_{max} = 27.2$ Watts, idle power consumption $P_{idle} = 4.97$ Watts, and Deep Sleep power consumption $P_{DeepSleep} = 2.47$ Watts. To evaluate the DVS policy, we scaled data obtained from a low–power processor to fit the 600MHz Intel processor's maximum frequency and core operating voltage [18]. The DVS processor has a frequency range of 300MHz - 600MHz variable in steps of 33MHz, a time quantum of 20ms for varying frequency and voltage, and a core operating voltage ranging from 1.5V to 2V. The P_{max} and P_{idle} of the DVS processor are the same as that of the 600MHz non–DVS Intel processor.

Our decision to use a simulator to evaluate the policies was motivated entirely by pragmatic concerns. If we had replayed the traces against a real web server that implemented the policies, it would have taken us over 55 days to create all the data points in this paper [4]. Faster computers could not have speeded up this time. In contrast, the Salsa simulations on a 1GHz Intel machine took under 4 hours of wall clock time. However, as we describe below, we have validated Salsa against a real web server to ensure that its results can be meaningfully interpreted.

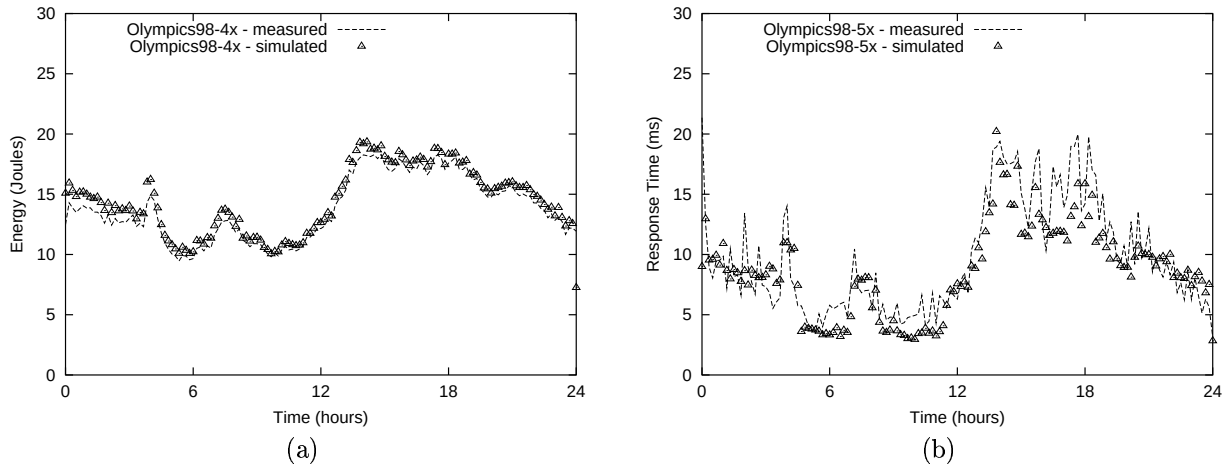
[4]Figure 4(a) alone would have taken 2×(10 points × 12 hrs + 10 points × 2 hrs + 9 points × 6 hrs) = 388 hours (over 16 days).

The energy consumption reported by Salsa has been validated against actual measurements for all three workloads (none of these workloads were used to calibrate Salsa). We used a modified version of `httperf` [17] to replay the workload against the server. Figure 2(a) shows the measured CPU energy consumed by the 600MHz system during the execution, overlaid with the simulator output. The Finance and Disk-Intense workloads exhibit similar behavior. Table 2 summarizes the validation results over a broad range of intensities, from light (Disk-Intense-2×, Finance-12×) to heavy (Olympics98-4×). For all three workloads, the error in predicted energy is less than 6%.

We have also validated the response times predicted by Salsa for the Olympics98 and Finance workloads against those measured using the `httperf` tool. Figure 2(b) shows the response time predicted by the simulator overlaid with the measured response times for the Olympics98 workload at a 5× intensity (we show data for the 5× intensity because the response time shows little variation at lower intensities). For both workloads, the average response time predicted by the simulator has an error of at most 13.2% compared to response times measured during execution. This error arises because the measured response time data involves three components: the server, the network, and the client, of which Salsa is modeling only the server. Since Salsa does not yet model disk delays, we have not attempted to validate the response times predicted by Salsa for the disk–intensive workload. However, when response time is dominiated by disk access time, any increase in response time due to CPU power management will be marginal.

Workload	Olympics98-4×	Finance-12×	Disk-Intense-2×
Measured CPU Energy (Joules)	1,232,710	711,415	627,977
Simulator CPU Energy (Joules)	1,253,652	739,200	663,648
Error in Total Energy	1.70%	3.91%	5.68%
Correlation Coefficient	0.9846	0.9960	0.8485

Table 2: Comparison of Measured to Simulated CPU energy for three workloads. Correlation coefficients were computed based on the energy used in 30 second intervals over the length of the run.



(a) (b)

Figure 2: (a) Measured vs Simulated Energy Consumption for Olympics98-4× workload. (b) Measured vs Simulated Response times for Olympics98-5× workload.

4.2.1 Salsa Validation For Request Batching

Before simulating the request batching policy in Salsa, we wanted to gain a measure of confidence in the simulation technique we used. We achieve this confidence by validating Salsa's predictions against a prototype implementation of request batching.

The request batching prototype is implemented on a commodity 600MHz server running the Apache web server over a Linux 2.4.3 kernel. We modified the web server to batch requests whenever all the Apache child processes were in an idle state: either waiting for a new connection or waiting for new requests on an existing connection. Batching is disabled when new work arrives until all child processes become idle again. We batch packets by leveraging the interrupt coalescing feature in the Alteon ACEnic Gigabit Ethernet card [1] on the web server. Interrupt coalescing groups the interrupts for a set of packet events (reception or transmission) into one single interrupt to the host processor and was introduced to improve network throughput by reducing interrupt processing overheads [1]. We modified the ACEnic device driver to allow the packet count and timeout for interrupt coalescence to be modified dynamically using an `ioctl` call.

To validate Salsa, we conducted an experiment where the prototype batches requests with a timeout of 100 milliseconds or a maximum backlog of 100 packets. There are two key differences between the prototype and the Salsa implementation we discuss in the later sections. First, the prototype does not place the processor in Deep Sleep. Instead, we keep track of the time that the processor *would have been* in Deep Sleep mode, since this is a direct indicator of the energy savings from batching. Second, the prototype does not incorporate the response time based feedback control explained in Section 3.1. Consequently, for validation purposes, we execute Salsa in "open–loop" mode without the feedback control, and compare the predicted batching and response times against the measured values.

Figure 3(a) shows the percentage of time for which request batching was enabled averaged over 90–

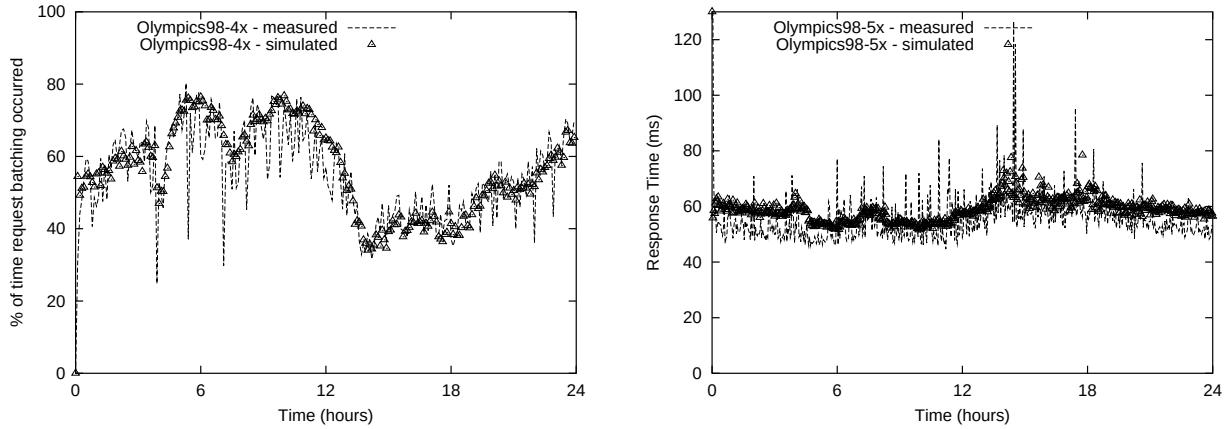


Figure 3: Salsa validation with batching (a) Measured vs Simulated Batch Time for Olympics98-4× workload. (b) Measured vs Simulated Response times for Olympics98-5× workload.

second intervals in both the prototype implementation and Salsa for the Olympics98-4× workload. The Salsa predictions correlate well with measurements of the prototype, with a correlation coefficient of 0.828. Over the course of the run, the prototype batched requests for a total of 11,953 seconds while Salsa predicted 12,373 seconds, an error of just over 3.5%. Figure 3(b) shows the client response time, averaged over 30–second intervals, for both the prototype and Salsa executing the Olympics98-5× workload. The correlation coefficient in response times is 0.754. The average response time predicted by the simulator has an error of 4.7% compared to that measured using the prototype. The same difficulties for validating response times that were outlined in Section 4.2 apply here as well. Finally, even in a LAN environment, we observed that the prototype experienced extra TCP retransmissions of less than 0.1% for a batching timeout of 100ms.

5 Evaluation

We begin by evaluating the DVS and request batching policies, explaining first how much energy each policy is able to save when the 90th–percentile response time goal is varied. Next, we set the 90th–percentile response time goal to be 50ms and evaluate how both policies behave in terms of energy savings as the workload intensity is changed. Finally, we evaluate the combined policy showing how it is able to sustain energy savings across different workload intensities for a 90th–percentile response time goal of 50ms. We also show how the combined policy behaves as the percentile cutoff is relaxed to 80% and tightened to 95% for a response time goal of 50ms.

5.1 Dynamic Voltage Scaling Policy

Figure 4(a) shows the energy savings from the DVS policy over the base system for a range of 90th–percentile response time goals. The three workloads exhibit different energy savings because of differing workload intensities. Relaxing the response time goal increases the amount of energy savings, up to a point. For example, the energy savings for the Disk-Intense-2× and Finance-12× workloads level off at response time goals higher than 20ms. In fact, the 20ms response time goals for these two workloads can be satisfied by running the DVS processor at no more than the minimum frequency of 300MHz. In contrast, for a heavier workload such as Olympics98-4×, the energy savings diminish only after the 90th–percentile response time goal is relaxed to about 100ms. This heavier workload benefits from DVS to a greater extent. Table 3 shows the energy consumption with and without DVS for the three workloads when the 90th–percentile response time goal is 50ms.

Fortunately, the figure shows that the extent to which the response time goal must be relaxed to capture the major fraction of the energy savings is well within acceptable QoS norms. Relaxing the

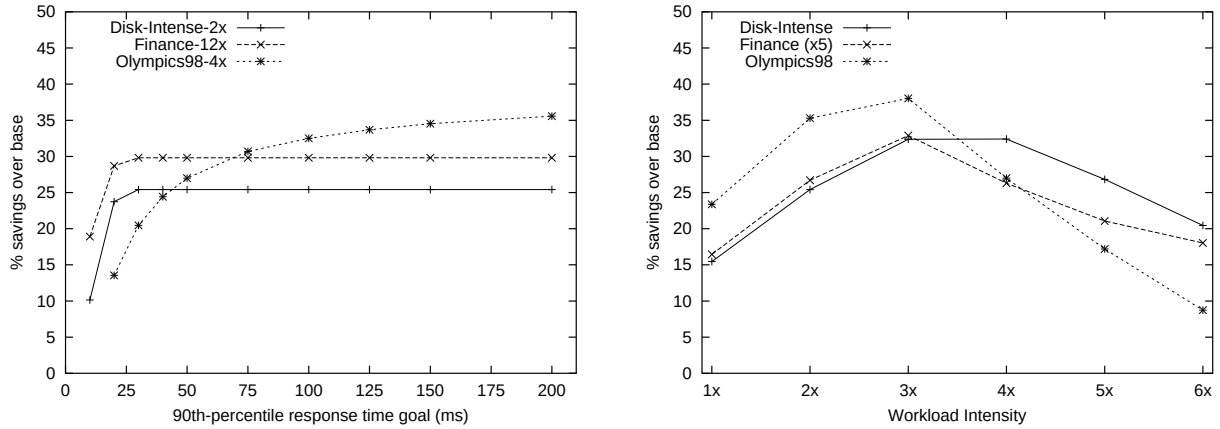


Figure 4: (a) Dependence of DVS energy savings on the response time goal. Note: Olympics98-4× does not have a 10ms data point, since the baseline 90th–percentile response time is itself 12.3ms. (b) Dependence of DVS energy savings on workload intensity for a 50ms 90th–percentile response time goal.

goal beyond the knee of the curve provides little improvement in energy savings at the expense of worsened responsiveness.

Figure 4(b) depicts the effect of workload intensity on the energy savings obtained from the DVS policy for a 90th–percentile response time goal of 50ms. Here, the x-axis indicates the scalefactor of the Olympics98 and Disk-Intense workloads, and the scalefactor divided by 5 for the Finance workload. Consistent with the analysis above, the results show that the energy savings from DVS improve with increasing workload intensity – but only up to a point. The largest savings are obtained when the intensity is between the light and heavy extremes. Changing the response time goal affects the magnitude of the savings, but not the intensity at which the gains are maximized.

Despite measuring 90th–percentile response time over an entire day, our control framework is surprisingly robust. Even for the most intense workloads we consider here, the 90th–percentile goal was attained for 84%, 86%, and 89% respectively of all 5-minute intervals of the Olympics98, Finance, and Disk-Intense workloads. A more complete discussion of the effect of the control system parameters on the system responsiveness is beyond the scope of this paper.

The results make two important points. First, for moderately intense workloads, dynamic voltage scaling can provide significant energy savings. Second, dynamic voltage scaling is less effective for workloads that are either light or intense.

5.2 Request Batching Policy

Figure 5(a) illustrates the energy savings due to request batching as the 90th–percentile response time goal is varied from 10ms to 200ms. The response time goal has a significant effect on the energy savings from request batching. Consistent with the analysis presented in Section 3.3, our simulation results show that increasing the response time goal yields diminishing returns in energy savings. Table 3 shows the energy consumption with and without request batching for a 90th–percentile response time goal of 50ms. The Finance and Disk-Intense workloads exhibit higher savings because of the prolonged periods of very low request rates in these workloads. For example, the Finance workload has a very low request rate until about 9:30am (see Figure 6). Request batching achieves high energy savings by placing the processor in the Deep Sleep state for a significant fraction of this time. In contrast, the Olympics98 workload has relatively fewer periods where requests can be batched. Compared to the 90th–percentile response time without request batching, the 50ms goal is several factors larger. However, as explained in Section 3.1, the effect on the client–perceived response time will be minimal due to the CPRT's dependence on the WAN delay between the client and the server.

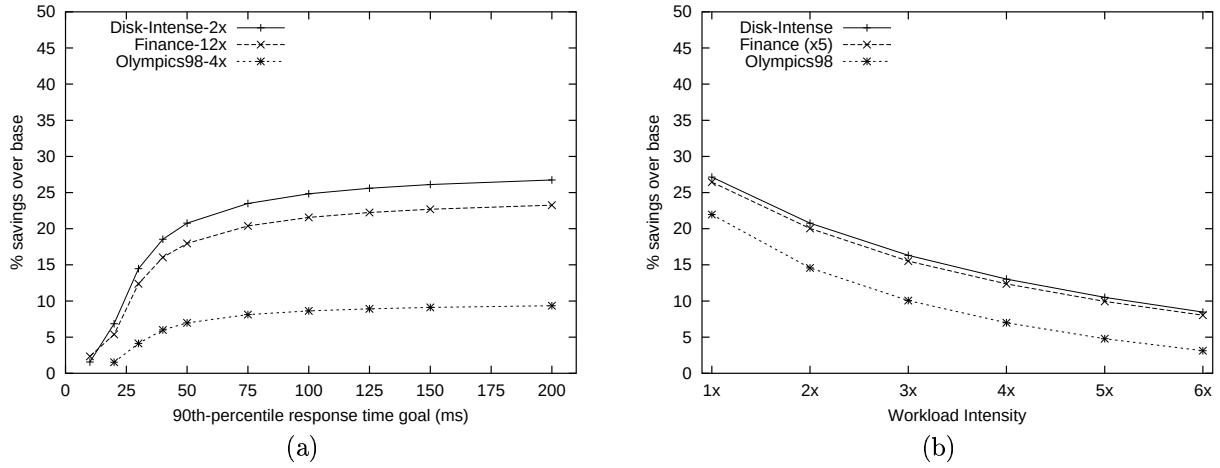


Figure 5: (a) Dependence of request batching energy savings on the response time goal (b) Dependence of request batching energy savings on workload intensity for a 50ms 90th–percentile response time goal.

Workload	Olympics98-4×	Finance-12×	Disk-Intense-2×
Base energy – no DVS or Batching (J)	1,253,672	739,212	663,648
Base 90th–percentile response time (ms)	12.3	6.4	3.0
DVS Joules (% savings)	915,204 (27%)	518,844 (30%)	494,982 (25%)
Request batching Joules (% savings)	1,166,128 (7.0%)	606,468 (18.0%)	525,836 (20.8%)

Table 3: Energy savings from DVS and Request Batching for a 90th–percentile response time goal of 50ms.

Figure 5(b) illustrates the energy savings with a 90th–percentile response time goal of 50ms compared to the base case where requests are not batched. The x-axis represents the workload intensity for the Olympics98 and Disk-Intense workloads, and the intensity divided by 5 for the Finance workload. Note that the savings decrease with intensity, as explained in Section 3.3. As in the case of DVS, changing the response time goal affects the resultant energy savings, but does not alter where the gains from request batching are maximized.

In summary, request batching is an extremely effective strategy for saving energy when the workload has significant periods of low activity. While increasing the response time goal has a positive effect on the energy savings, most of the savings can be captured using a response time goal of around 50ms, without adversely affecting either the client-perceived response time or TCP/IP performance.

5.3 Combined Policy

Figure 6 shows the savings from DVS and request batching superimposed over the request rate during the course of the Finance-12× workload for a 90th–percentile response time goal of 50ms The figure clearly shows the effectiveness of request batching at low request rates or workload intensities, e.g., until 9:30am. After a steep climb at 9:30am, the request rate remains relatively high until about 6pm. During this interval, request batching provides reduced energy savings. In contrast, it is precisely during this period that DVS provides maximum savings. This complementary behavior clearly illustrates the potential benefits of a policy that employs both mechanisms for improved energy savings across a broad range of workload intensities.

Figure 7 compares the energy savings from the DVS, request-batching, and combined policies for the Disk-Intense workload at intensities from 1× to 6×. We can see the DVS and request-batching mechanisms complementing each other: when the workload is light, request batching provides the most savings. As the workload becomes more and

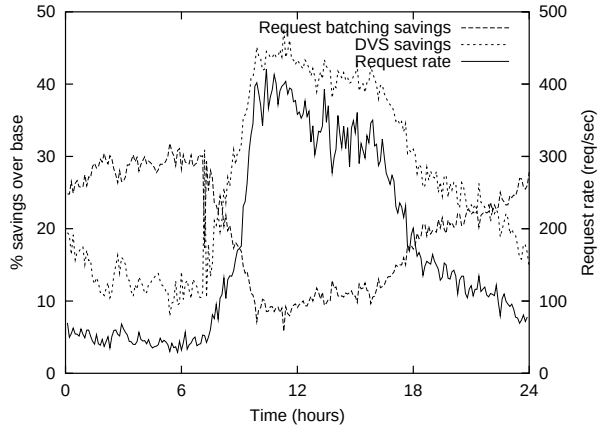


Figure 6: Energy savings from DVS and request batching with a 90th–percentile response time goal of 50ms for Finance-12×, superposed with the request rate (intensity). All values averaged over 6–minute windows.

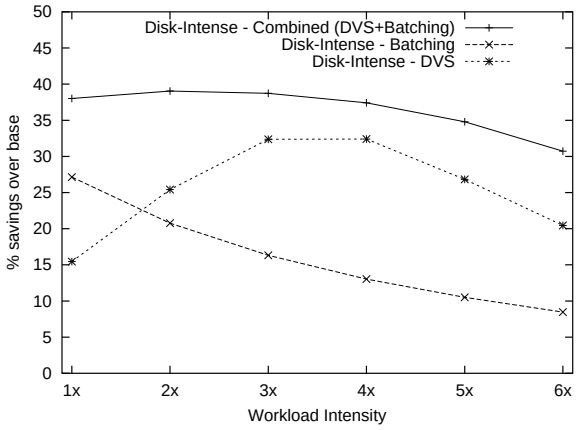


Figure 7: Energy savings from the DVS, request batching, and combined policies with a 90th–percentile response time goal of 50ms for Disk-Intense over a range of intensities.

more intense, requests are batched less and less frequently (if at all), and DVS provides the most savings. The combined policy provides savings of between 30.7% and 39% as the workload intensity varies from 1× to 6×, illustrating that the combination of request batching and voltage scaling captures the best features of both policies.

Figure 8(a) shows the energy savings yielded by the combined policy on all three workloads. The combined policy outperforms the individual request batching and voltage scaling policies across the range of workloads and intensities. However, while the energy savings for both Finance and Olympics98 are above 40% at low workload intensities, the savings drop to about 26% and 17% respectively as the workload intensifies to a scale of 6. The savings decrease because higher intensities require more processor involvement in order to service incoming requests. In particular, the baseline 90th–percentile response time (with no energy saving policy in effect) for the Olympics98-6× workload is 42ms, implying that the system has little room to save energy while maintaining the 90th–percentile response time at 50ms.

Figure 8(b) shows the variation in savings as the percentile goal is varied from 80% to 95% for the Olympics98 and Disk-Intense workloads. The Finance workload exhibits similar behavior. As the percent of response times that must fall at or below 50ms is tightened from 80% to 95%, the energy savings decrease. In the case of Disk-Intense at 6× intensity, the savings fall from 33.5% to 9.5%. This experiment illustrates that energy savings can be obtained not only by relaxing the response time goal, but also by relaxing the percent of requests that must be satisfied within that goal.

5.4 Projections for Faster Processors

The results we report in this paper are for a simulated server with a 600MHz processor. Current processors use clock rates as high as 3.0 GHz, and even faster processors will be available in the near future. In this section we project how our three energy management policies will perform in systems with faster processors. We use a hypothetical 3.0 GHz processor model that supports a low–power Deep Sleep mode and dynamic voltage scaling, and perform simulations using our three web server workloads. The key attributes of our 3.0 GHz processor

Processor State	Power Consumption
Busy (at 3GHz)	60W
Idle (Halted)	10W
Deep Sleep	5W
DVS Frequency Range	
1.5 GHz - 3.0 GHz, in 10 steps of 150MHz	

Table 4: Attributes of a hypothetical 3.0 GHz DVS processor

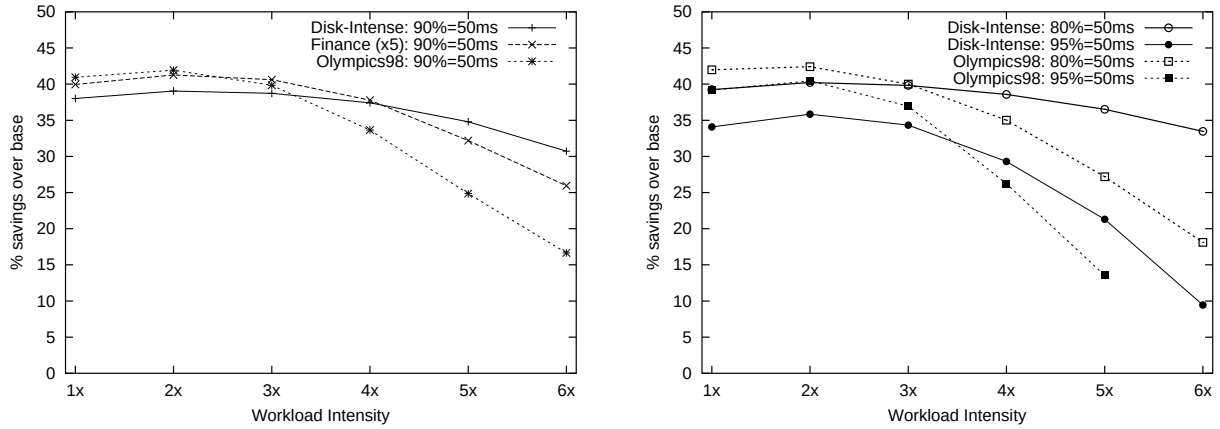


Figure 8: Energy savings from the combined policy. (a) Savings for all three workloads for the 90th–percentile response time goal of 50ms. (b) Savings range as percentile goal is varied for Disk-Intense and Olympics98.

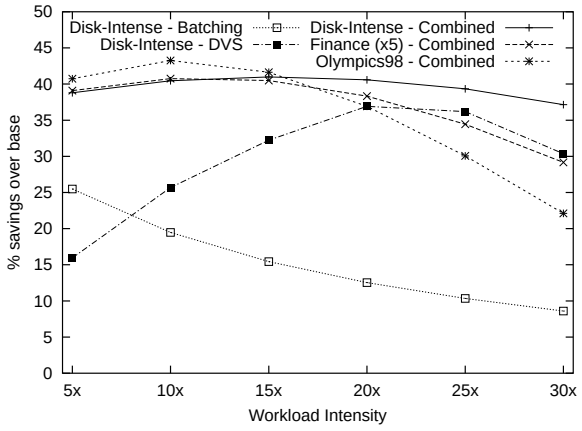


Figure 9: Energy savings from the combined policy for a 3.0 GHz processor for the 90th–percentile response time goal of 50ms.

model are presented in Table 4. These attributes are based loosely on reported power consumption for current high performance processors, with DVS support similar to that assumed for our 600MHz processor (Section 4.2). Note however that we have not validated our simulator against an actual 3.0 GHz processor, so the results in this section are intended only to establish the basic trend in effectiveness of our policies as CPU speed increases.

With faster processors, the savings from DVS (as a percentage of the total energy consumed) is likely to remain the same. The savings from request batching are likely to increase if the energy consumed during Deep Sleep is a smaller fraction of the idle power consumption. Indeed, this is true for most (if not all) Intel processors that are faster than the one we consider in this paper. Figure 9 shows the energy savings achieved by the combined policy on all three workloads and energy savings from DVS and request batching for the Disk-Intense workload using the 3.0 GHz processor model. These results confirm the expected trends described above. Calibrating and validating a new processor model in Salsa is a time intensive activity, and we have left a more complete evaluation of our policies on faster processors as future work.

5.5 Extensions to Other Environments

The results we report in this paper are for static workloads. Dynamic workloads introduce two new factors. First, requests take longer to service, causing significantly fewer periods where no requests are being processed. To handle dynamic requests, the batching policy we described in Section 3.1 will have to place the processor in Deep Sleep when all processes are blocked, as opposed to when no requests are being serviced. Second, dynamic requests can take significantly longer to process than static requests. Thus, the response time requirements must be relaxed beyond that we have used for static requests in order to permit the energy conservation policies to be applied. Furthermore, partitioning the QoS levels for static and dynamic requests will

prevent the service times of one class from overshadowing the responsiveness metric used by the policies.

Several vendors have introduced "blade" systems that incorporate low-power processors in a spatially dense form factor [21]. While the policies described in this paper can be employed with low-power processors, the resulting large CPU energy savings may not translate into significant *system* energy savings since the CPU energy consumption may be a small component of the system energy in these systems. However, when such systems are clustered, policies that target server clusters by switching off entire systems become applicable [4, 19], and can be used in conjunction with the single–node policies described in this paper.

In earlier work [5], we studied combinations of cluster–level and single–node power management policies and found that fine–grained single–node energy management techniques can be very complimentary to typical cluster–level energy management mechanisms, which tend to be more coarse–grained in their effect on energy consumption and system performance. In addition, frequent powering off of systems may adversely affect the reliability of certain components, particularly disk drives, making single–node power management techniques more desirable for achieving energy savings from small or short term fluctuations in workload. Finally, single–node energy management policies are necessary for web sites that do not receive sufficient traffic to require multiple servers.

6 Related Work

Policies to reduce CPU energy consumption using dynamic voltage scaling have been widely studied. Early work used the CPU utilization over a recent set of intervals as a predictor of future system load, and then set the processor voltage and frequency for the next time internal so as to handle this load [9, 23]. Flautner et. al. explored a DVS policy that sets CPU speed on a per–task basis rather than for time intervals [7]. Lorch et. al. describe PACE, an algorithm to optimize task-based DVS policies when task completion times cannot be accurately predicted [14]. While these techniques perform well for desktop workloads, they are unsuitable for server environments with many concurrent tasks. Our DVS policy keeps track of the response time of individual requests ("tasks"), but employs a feedback–driven control framework to meet responsiveness requirements in the aggregate, instead of for individual tasks.

A number of recent papers have considered techniques for reducing energy consumption of web servers in large data centers. Pinheiro et. al. [19] proposed a simple policy for managing energy use in server clusters by powering machines on and off. Chase et. al. have employed this mechanism in the context of an economic framework in which web sites "bid" on resources based on their current workload [4]. In prior work, we explored the combination of the power-on/power-off technique with voltage scaling [5] to reduce energy consumption for a cluster of servers. The energy management techniques proposed in this paper are applicable to individual web server systems, and therefore complement efforts to manage energy for web server clusters.

Several researchers have developed tools for simulating the power consumption of computer systems. Brooks et. al. have developed Wattch, a microprocessor power analysis tool based on a microarchitecture simulator [3]. Flinn and Satyanarayanan describe PowerScope, a tool for profiling the energy usage of applications [8]. Our Salsa simulator is substantially faster, because it is targeted specifically for web serving workloads.

7 Conclusions

This paper describes three policies for reducing the energy consumption of server processors during web serving. The first policy uses dynamic voltage scaling (DVS), an existing energy-saving mechanism, but employs it in a feedback–driven control framework. The second policy is based on request batching, a new mechanism we introduce in this paper, that groups requests under periods of low workload intensity and executes them in batches, otherwise placing the processor in a Deep Sleep mode to conserve energy. These two policies target complementary ranges of workload intensities. While DVS is most effective for moderately intense workloads, request batching saves energy for light workloads. The third policy leverages both request batching and DVS mechanisms in a policy that saves energy across a wide range of workload intensities. All three policies conserve energy while maintaining a given quality of service level, as defined by a percentile–level response time.

The policies are evaluated using Salsa, a web serving simulator that has been validated for both energy and response times against an actual web server. The effectiveness of the policies are measured using three day–long workloads derived from real web server systems. When 90% of the requests must be serviced within a response time of 50ms, the DVS policy saves from 8.7% to 38% of the CPU energy used by the base system. The request batching policy provides from 3.1% to 27% savings for the same response–time requirement. However, the two polices provide these savings for different regions of the workloads. The combined policy is effective for all three workloads across a broad range of intensities, saving from 17% to 42% of the CPU energy.

Acknowledgements

We wish to thank our shepherd, Amin Vahdat, and the anonymous reviewer for many valuable comments and insights which have improved the quality of this paper.

References

[1] Alteon WebSystems Inc. Gigabit ethernet/PCI network interface card: Host/NIC software interface definition, 1999.

[2] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers. In Robert Graybill and Rami Melhem, editors, *Power-Aware Computing*. Kluwer/Plenum Series in Computer Science, January 2002.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *27th International Symposium on Computer Architecture*, pages 83–94, 2000.

[4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, October 2001.

[5] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proceedings of the Workshop on Power-Aware Computing Systems*, February 2002.

[6] Puget Sound Energy. Schedule 449: Retail Wheeling Service, 2002. Also see 'Schedule 448: Power Supplier Choice'.

[7] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[8] J. Flinn and M. Satyanarayanan. PowerScope: A tool for profiling the energy usage of mobile applications. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 2–10, 1999.

[9] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the ACM International Conference on Mobile Computing and Networking*, pages 13–25, November 1995.

[10] Intel Corporation. Wired for management baseline version 2.0, 1999.

[11] Intel Corporation. Intel pentium 4 processor datasheet. Order Number: 249887-003, April 2002.

[12] The IRCache project. http://www.ircache.net/.

[13] A. Iyengar, M. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, June 1999.

[14] J. Lorch and A. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of the ACM SIGMETRICS 2001 Conference*, June 2001.

[15] Mesquite Software Inc. *CSIM18 Simulation Engine*, 1994.

[16] J. Mitchell-Jackson. Energy needs in an internet economy: A closer look at data centers. Master's thesis, University of California, Berkeley, July 2001.

[17] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *SIGMETRICS First Workshop on Internet Server Performance*, pages 59—67. ACM, June 1998.

[18] K. Nowka. Private communication. IBM Research, Austin, TX.

[19] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[20] R. Rajamony and M. Elnozahy. Measuring client perceived response times on the WWW. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS'01)*, March 2001.

[21] RLX Technologies Inc. Serverblade product description. http://www.rlx.com/products-/products_blades.php, 2002.

[22] BS Web Services. SLA. http://www.bsws.de/en-/products/sla-ahp.shtml.

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *First Symposium on Operating Systems Design and Implementation*, pages 13–23, Monterey, California, U.S., 1994.