Power-aware QoS Management in Web Servers [*]

Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
e-mail:{*viveks, at4a, zaher, skadron*}@*cs.virginia.edu*

Zhijian Lu
Department of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22904
e-mail: *zl4j@virginia.edu*

Abstract

Power management in data centers has become an increasingly important concern. Large server installations are designed to handle peak load, which may be significantly larger than in off-peak conditions. The increasing cost of energy consumption and cooling incurred in farms of high-performance web servers make low-power operation during off-peak hours desirable. This paper investigates adaptive algorithms for dynamic voltage scaling in QoS-enabled web servers to minimize energy consumption subject to service delay constraints. We implement these algorithms inside the Linux kernel. The instrumented kernel supports multiple client classes with per-class deadlines. Energy consumption is minimized by using a feedback loop that regulates frequency and voltage levels to keep the synthetic utilization[1] *around the aperiodic schedulability bound derived in an earlier publication. Enforcing the bound ensures that deadlines are met. Our evaluation of an Apache server running on the modified Linux kernel shows that non-trivial off-peak energy savings are possible without sacrificing timeliness.*

Keywords: Dynamic Voltage Scaling, Web Servers, Aperiodic Task Scheduling, Utilization Bounds, Linux

1 Introduction

As chip-manufacturing firms continue to increase processing power in accordance with Moore's law, high energy consumption of chips has become an increasingly important concern. IBM first made the case for power management in web servers indicating dynamic voltage scaling as a means to save energy not only in embedded devices but in high performance web servers as well [5]. Further work [8, 9] has argued that power management in web servers is highly desirable due to technical, financial and environmental reasons.

In large data centers, node failures have been increasingly reported due to the high concentration of nodes and high power consumption per unit space. Energy costs alone could account for 23-50% of the revenue [9]. Hence, even moderate energy savings of can make a tangible difference. While the motivation for power management is clearly understood by the research community, there are still some areas where more work is required. One such area is reconciling energy minimization with meeting QoS guarantees in the case where input workloads do not exhibit periodic behavior. In this paper, we specifically target the research issues in incorporating power management in QoS-aware web servers and the systems issues in designing and implementing such web servers with energy-saving schemes. Our main contribution is the design, implementation, and experimental evaluation of an architecture for minimizing energy consumption subject to meeting per-class service delay constraints. The novelty of this architecture lies in its integration of timing requirements and power management in the context of *aperiodic* tasks. This integration is achieved by a feedback mechanism that relies on a recent utilization bound for schedulability of aperiodic tasks derived by the authors [2]. Prior work on power management in real-time computing has typically addressed periodic task sets. Our mechanisms are evaluated experimentally on a real prototype and not by simulation common to many prior publica-

[*]The work reported in this paper was supported in part by the National Science Foundation under grants CCR-0093144, CCR-0098269, CCR-0133634, and CCR-0306404.

[1]Also known as instantaneous utilization

tions.

The need to maintain timeliness guarantees is motivated by the requirements of contemporary computing services. Today, many high-performance computing services operate under revenue-critical time constraints. These constraints arise from the fact that customer behavior is affected by the quality of their sessions with the server. A very slow or overloaded server will drive away customers due to unsatisfactory performance. Hence, a trade or e-commerce server, for example, should produce a response to the clients within a sufficiently small amount of time, or else loss of customers (and revenue) may ensue. Servers which support multiple classes of clients may want to export multiple levels of delay guarantees such that higher-paying client classes are guaranteed a shorter delay. Acceptable delay bounds may be obtained from user studies and represent soft QoS requirements.

Synthetic utilization bounds [2, 1] have been shown to be an efficient mechanism to provide absolute delay guarantees. Synthetic utilization bounds can serve as control set points in dynamic voltage (and frequency) scaling loops that allow a web server to save power while maintaining the delay constraints on the different classes of clients. Such loops select a frequency/voltage combination among those supported by the hardware in a way sensitive to load and deadlines. This paper presents the first use of synthetic utilization bounds for power management, and the first in-kernel implementation and evaluation of a deadline-aware power management scheme in Web servers.

The remainder of this paper is organized as follows. Section 2 introduces the problem and presents the general design of the solution approach. Section 3 presents the system implementation. Performance results are presented in Section 4. Section 5 briefly relates this work to current ongoing research efforts on power management and scheduling. The paper concludes with Section 6, which summarizes the results and presents avenues for future work.

2 Problem and Solution Overview

The ultimate goal of this work is to develop mechanisms for energy saving that reduce energy costs in QoS-enabled web server farms. A typical web server farm is shown in Figure 1. It consists of a front-end node which forwards client HTTP requests to back-end nodes where these requests are served. When the load is less than system capacity, energy can be saved in two different manners; (i) some back-end machines can be turned off, and (ii) frequency and voltage can be scaled down on the remaining machines. The former approach has a high overhead and it introduces a high latency when turned-off machines are needed again. It should therefore be applied on a large timescale and should allow for a sufficient capacity margin to accommodate transient load disturbances and bursts. Such coarse-grained policies would typically use some simple hysteresis-like heuristics such as "turn off a machine if average load (averaged over some appropriately long time interval) is below a threshold, and turn on a machine when average load rises above another threshold". Such policies are investigated in [8]. Tuning these policies (e.g., for particular workload conditions) may be left to system administrators and is not addressed in this paper. Instead, we concern ourselves with finer-grained scheduling of machines that are turned on at a particular time.

It is easy to see that energy savings are maximized when load is exactly balanced among the back-end machines. This follows directly from the nonlinear power voltage relation and the fact that the sum of squares (or higher order functions) of numbers that add up to the same total is minimized when these numbers are equal. Efficient load balancing algorithms have been proposed for web server farms in past literature [6]. Experience with current farms shows that these algorithms are very successful in practice. Thus, we can assume that a good load balancing algorithm is used at the front-end, which makes the power management problem symmetric across the back-ends. In other words, it needs to be solved for only one machine. The solution is then symmetrically carried out by all.

We assume that incoming requests are assigned to back-end nodes depending on their connections. All requests on the same connection are served by the same back-end node. This approach alleviates problems with migrating connection state across different machines. Responses could be sent back to the client directly by using techniques like Network Address Translation (NAT). NAT is a widely used capability provided in Linux kernels as part of the Netfilter framework. It allows changing of the IP header fields of a packet (incoming or outgoing). The translation rules can be specified using the common iptables interface. NAT has been particularly useful in Linux clusters to re-direct requests to back-ends based on appropriate policies by modifying the destination address on the IP packets. Similarly outgoing packets at the backend can have their source address field modified to be the IP address of the front-end node.

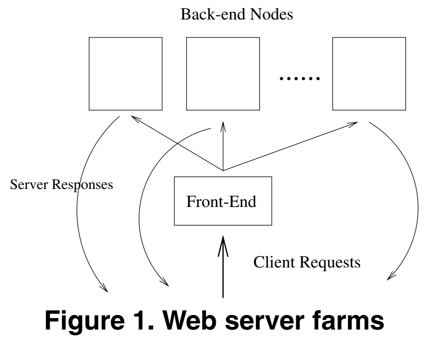

Figure 1. Web server farms

Since back-ends in a load-balanced farm have roughly equal load characteristics, the fundamental question answered in this paper is the following. Given aperiodically arriving clients at a particular back-end, partitioned into a number of classes with different maximum delay requirements, how to minimize the energy used by that back-end while maintaining service delay guarantees? Simple reasoning is then used to scale the results to a larger farm.

Our experimental results confirm that the incremental energy savings from turning a machine off are greater than those from slowing down all CPUs (and scaling down their voltage) to reduce farm capacity by the same amount. Since we do not turn machines off in our experimental evaluation, the savings we present are a pessimistic lower bound on savings achievable in a scheme that combines powering down some machines with load balancing across the remaining ones in which dynamic voltage scaling, described in this paper, is used to reconcile power management with meeting response-time requirements.

We implement our power management solution inside the Linux kernel and test the architecture with an actual Apache web server. The architecture would be symmetrically installed on all back-ends of the farm in a real application. To the authors' knowledge, this is the first work that designs and experimentally evaluates power management techniques in web servers that provide absolute delay guarantees.

2.1 In-kernel Architecture

In our architecture, the user-level server code remains unmodified, while the kernel combines a scheduling policy, an admission control scheme, and a dynamic voltage scaling scheme to minimize power while meeting deadlines. The decision to use a kernel-level solution has two main advantages. First, it is more efficient than application-level solutions. Second, it is more general. Utilizing a few loadable kernel modules, we can apply deadline-aware power minimization to different application servers, as opposed to having to instrument every Web server product. Our solution requires per-class prioritization while servers such as Apache schedule requests in FIFO order. To overcome this problem, we run multiple instances of the server at different priority levels such that each is responsible for serving a particular request class. This solution has been previously proposed in [23] and found to be successful in circumventing the lack of priority-based scheduling in several best-effort server systems. Figure 2 illustrates the resulting architecture applied to a single back-end server. The main components of this architecture are discussed in the subsections below.

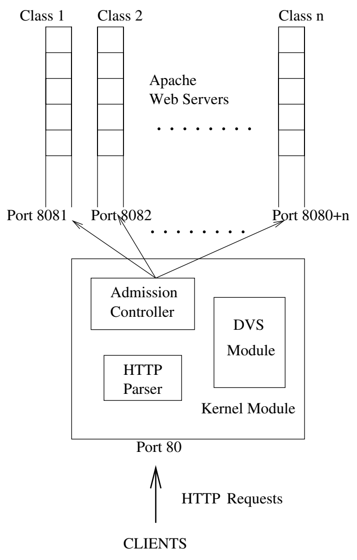

Figure 2. Architecture for single node web server

2.2 Extending kHTTPd

Linux (version 2.4) provides an in-kernel web server, called kHTTPd, as a dynamically loadable kernel module. The in-kernel web server was designed to serve static requests while dynamic requests were forwarded onto a user-space web server (for example, Apache). The motivation was to make the simple case of serving static files faster by serving them in the kernel itself.

We have used kHTTPd's connection handling and HTTP header parsing implementations for our purpose. We have modified kHTTPd to accept connections, parse the first HTTP request and make admission control decisions before forwarding the connection onto a user-space web server. The implementation is shown in Figure 3. The connection is handed off by inserting the socket data structure directly into the Accept Queue of the user-space web server.

This modified server runs in the kernel at port 80 where the first request on a connection is received. The server parses the first HTTP request and determines the priority of the request based on any value or combination of values in the HTTP header (for example - Cookie information) or network layer information such as the IP address of the client.

2.3 Dynamic Voltage Scaling

The heart of our algorithm is implemented in the DVS module which attempts to reduce voltage and frequency as much as possible but not enough to cause per-class response-time constraints. These constraints define a deadline associated with each request. Violation of a response-time constraint

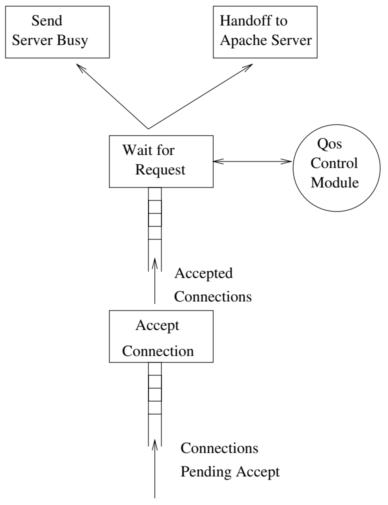

Figure 3. Session-based admission control

is a deadline miss. A trivial approach is to reduce speed until the first deadline misses are observed. This solution, by its very nature, however, requires that some misses be tolerated. Instead, we use a recently derived utilization bound for schedulability of aperiodic tasks. It is proved in [2] that maintaining a quantity called synthetic utilization below the bound guarantees that no deadline misses are observed. The bound presents a sufficient condition for schedulability. Exceeding it does not necessarily imply deadline misses. Hence, we take this bound as a control set point. When synthetic utilization is lower than the bound, processor speed is reduced until the utilization reaches the bound. Conversely, when synthetic utilization exceeds the bound, processor speed is increased. An appropriate sampling rate is chosen that is large enough to prevent excessive overhead due to voltage switching, but small enough (with respect to task deadlines) such that the system remains sufficiently responsive to load variations. We allow the synthetic utilization to exceed the bound within a sampling period instead of clipping the load at the bound by applying admission control. At the end of the sampling period the difference between the measured synthetic utilization and the utilization bound determines the extent of speed change required. An exception to that rule is when the processor operates at maximum speed. In this case we employ an admission controller to prevent synthetic utilization from exceeding the bound.

2.4 Session-based Admission Control

When admission control is called for, a choice we made was to do admission control at the granularity of sessions rather than individual HTTP requests. This is motivated by the fact that aborting already established sessions wastes all server resources expended on the aborted sessions. We assume HTTP/1.1 connections where requests could be pipelined and multiple requests could be sent over a single TCP connection to the server. Since HTTP/1.1 is becoming the norm in web servers, we focus on this version of the HTTP protocol and do not explicitly discuss HTTP/1.0 connections.[2]

In this paper, a session corresponds to a single HTTP/1.1 connection. Admission control for a session is done when the first HTTP request is received by the server and subsequent requests on the same connection are always admitted. While no admission control is done for subsequent requests on a persistent connection, these requests are accounted for by adding their synthetic utilization to the system utilization counter as they arrive in the system.

Thus, synthetic utilization $U(t)$ at time t is the sum of the ratio of computation times C_i over relative deadlines D_i for all requests, i, that have arrived but whose deadlines have not expired. (Let us call it the set of current tasks $V(t)$.) Hence:

$$U(t) = \sum_{T_i \in V(t)} \frac{C_i}{D_i} \tag{1}$$

If this summation is below the bound derived in [2], all requests meet their response-time constraints. The scheme mentioned above requires the knowledge of execution times of individual requests. This can be done by parsing the HTTP header and extracting the URL requested. The identity of the requested file or script gives some idea of the time it may take to serve it. However, for the case of high performance servers, individual requests have very low execution times on the CPU and during high load, the number of requests is large. Hence, one can make use of the law of large numbers and use the mean computation time instead of the actual value. We therefore formulate an approximate admission control policy as follows:

$$\sum_{T_i \in V(t)} \frac{C_{mean}}{D_i} \leq U_{bound} \tag{2}$$

or,

$$\sum_{T_i \in V(t)} \frac{1}{D_i} \leq \frac{U_{bound}}{C_{mean}} \tag{3}$$

In the rest of this paper, we use the mean computation time instead of individual computation times. The mean computation time is easy to calculate from statistical measurements of load and request rate on the web server.

3 Implementation Issues

In this section, we highlight some of the implementation challenges we faced when developing our architecture in-

[2]HTTP/1.0 connections can be thought of as a special case of HTTP/1.1 connections, in which a single request arrives per connection, and the keepalive timeout is zero.

side the Linux kernel. It must be mentioned that we denied ourselves the freedom of changing the core kernel and hence operated under the constraint of implementing everything as kernel modules. While being inflexible, this allows the solution to be generic to other applications and available as an option to the server administrator.

The main issues were - modifying kHTTPd and porting it to the 2.5 kernel versions of Linux, enabling voltage scaling, maintaining the synthetic utilization counter and the DVS module itself.

3.1 Enabling Deadline Monotonic Scheduling in Linux

We chose to run Apache servers on multiple ports at different static priorities. Standard Linux provides the capability to assign static priorities to processes and FIFO scheduling within each priority level. This is done by using the *sched_setscheduler* system call which takes in three arguments - PID of process, scheduling strategy to use to schedule the process, and priority. There are three scheduling strategies possible in Linux - SCHED_FIFO, SCHED_RR and SCHED_OTHER. SCHED_FIFO enables FIFO scheduling within each priority level with preemption by higher priority levels. SCHED_RR is similar to SCHED_FIFO except that when a request is preempted it is sent back to the end of the priority queue rather than at the beginning. SCHED_OTHER is the weighted round robin dynamic priority scheduling policy used by default in Linux.

Hence using SCHED_FIFO scheduling capability, multiple Apache servers can be run at different static priorities to essentially enable Deadline Monotonic Scheduling. While better scheduling policies like Earliest Deadline First could be used to schedule requests, we chose the mentioned alternative because high performance web servers today use standard operating systems that do not provide scheduling capabilities for policies like Earliest Deadline First. Real-time operating systems provide EDF capabilities but are not likely to be used on mainstream web servers. Since we wanted our implementation to be applicable to a wide set of scenarios and portable to current systems, we chose to use the standard Linux operating system and a fixed priority scheduling policy which is supported by it.

Moreover, while Earliest Deadline First is an optimal scheduling policy, implementing the policy could have some additional overhead as compared to a fixed priority scheduling policy as the priority depends on dynamic parameters like the arrival time of the task. While use of EDF is prevalent in hard real-time systems, we feel that deadline monotonic scheduling is sufficient for the case of high performance servers.

3.2 Porting kHTTPd

While kHTTPd was available with 2.4 kernels of Linux, the support for it has been discontinued from the 2.5 series of kernels due to issues of kernel bloat and arguments against having a web server in the kernel. Due to better support for voltage scaling drivers in Linux 2.5, we decided to port kHTTPd to the 2.5 version of Linux. Since there have been significant changes in 2.5 as compared to 2.4, this required adapting kHTTPd to the changes. Since kHTTPd needs socket-related data structures and functions, it is most affected by changes in these structures in Linux 2.5. Additionally, some of the code for kernel modules has changed needing modifications. In particular, the socket data structure in 2.4 was bloated contained information from other layers, such as the IP layer. This data structure has been pruned and additional functions have been added to access lower level information. The code in kHTTPd that needs this information had to be changed.

Further, signal handling for tasks in the task data structure has changed to include signaling for groups of tasks. The signal enabling/disabling code in kHTTPd had to be updated.

There have been many changes with respect to improving the performance of Linux for SMP systems as well as to make the kernel preemptible. We chose to disable these modules. We disabled SMP since we did not need it, and disabled preemptibility because we weren't adventurous enough!

3.3 Voltage Scaling Drivers

The AMD Athlon PowerNow drivers were used on 2.5 versions of Linux due to the good support for voltage scaling drivers. The drivers provide the capability of scaling the frequency and voltage. The range is shown in Table 1. We found the CPUFreq drivers to be the most useful and easy to use to modulate the frequency and hence the voltage of the processor. CPUFreq allows simple manipulation of the processor frequency via the /proc interface, via user-space applications. The driver exports two functions - cpufreq_get and cpufreq_set which can be used to get and set the frequency of the processor respectively. The voltage gets adjusted according to Table 1. We found these functions sufficient for our cause.

3.4 DVS Module

In the simplest design, the voltage scaling module could be implemented as a separate daemon with high priority. The daemon would wake up every specified interval and depending on the energy management policy (discussed in the next section) could modulate the frequency/voltage.

Since this design would require having a separate process and performing a context switch every interval, we opted instant that the DVS module be clubbed in together within kHTTPd.

3.5 Synthetic Utilization Update

Since we implement session-based admission control, once a connection is handed off to an Apache server, the admission controller does not have any control over subsequent requests. However, the synthetic utilization of subsequent requests has to be added onto the system utilization counter at the time each request arrives. This is done by extensions to the Netfilter module available in Linux as a dynamically loadable kernel module.

Netfilter provides hooks into the IP layer processing stages to implement functionality like packet filtering, connection tracking, and network address translation. Without going into too much detail on the capabilities of Netfilter, we mention how we use it to enable keeping track of synthetic utilization for HTTP requests. Netfilter allows incoming packets to be applied against a filter (usually based on IP/TCP header values) and corresponding actions to be taken if the match is successful. We add two modules to the Netfilter framework. The first module is the string matching module which provides the capability to search for any string in the IP or TCP payload of a packet. If a match is successful, a target module is invoked. We have implemented a new target module called - UTIL to do the actual synthetic utilization accounting. The accounting is simply to increment synthetic utilization upon the arrival of each request (by the ratio of average request processing time to the desired response time). The utilization is decremented by the contributions of requests whose deadlines have passed. The module also resets the synthetic utilization to zero if the processor has been idle.

Having loaded the string matching and the UTIL modules, the following command shall invoke the UTIL module if the string matching module finds the word "GET" in the TCP payload portion of the packet:

```
iptables -t filter -A INPUT -p tcp -m string –string
"GET" -j UTIL
```

Iptables is the tool to add filtering/forwarding rules for execution on packet arrival. INPUT refers to incoming packets and *filter* refers to the filter table.

3.6 Other Implementation Issues

- **Deadline Expirations:** A task contributes a synthetic utilization of $\frac{C_i}{D_i}$ from the instant of its arrival to when its deadline expires. Hence, when the deadline of a task expires, the synthetic utilization of the task has to be subtracted from the total synthetic utilization at the resource. In our implementation, a queue of timestamps corresponding to expiration of requests (arrival timestamp + deadline) is maintained for every priority level. Since, the deadline is the same for every priority level, the queue is already in sorted order. Whenever the Netfilter module sees a GET request, it increments the system synthetic utilization and adds a timestamp to the corresponding priority level timestamp queue. Every time kHTTPd starts the admission test stage (for all pending requests), a check is made to see if any timestamps have expired and if so those entries are purged and the synthetic utilization reclaimed. This enables quick and efficient implementation of deadline expirations.

- **CPU-Bound Requests:** It is also important to note that a web server typically has multiple bottlenecks. Depending on the type of requests, the server accept queue, the CPU, the Apache process pool or the network interface could be the bottleneck. Hence, CPU scheduling might not suffice to provide overall delay guarantees. However, for this work we shall focus only on the CPU as the bottleneck resource. Hence, in our admission control we assume large enough process pools and unsaturated network interfaces (this would be the case for small static requests and CPU-bound dynamic requests).

- **Efficient Rejection:** Admission control is done in the kernel so that rejection of requests takes up very few resources at the server. If we were to incorporate admission control using an Apache proxy, the implementation would be inefficient because each request would have to go through the involved request processing stage of Apache and the request would have to be copied into a user-space buffer. A simple HTTP header parser is much more efficient.

- **Kernel Module:** The admission controller and dynamic voltage scaling algorithm are implemented as kernel modules which can be dynamically loaded or un-loaded and does not required any changes in the kernel. Hence, we provide the flexibility of being able to turn off the feature easily if desired.

- **Header Parsing Overhead:** Each initial request on a persistent connection is parsed in the kernel to get the HTTP header values so as to determine the priority of the client. This is a price that has to be paid to support HTTP layer-based admission control. Classification based on the HTTP header values provides more flexibility and allows the service hosting entity to differentiate service across clients based on a richer set of policies.

Frequency (MHz)	Voltage (Volts)
1529	1.55
1197	1.35
798	1.2
665	1.2
532	1.2

Table 1. Frequency and voltage levels available with AMD Athlon/Powernow

- **Connection Handoff:** Once the socket is handed off to the respective Apache web server, the kernel module loses all control over subsequent requests. Hence admission control cannot be done at the granularity of each individual HTTP request using this infrastructure. However, as we have mentioned, we focus on session-level admission control and consider aborting of sessions undesirable. So, admission control is done only once for each session. The next module keeps track of subsequent HTTP requests so that their synthetic utilization is accounted for.

4 Performance Results

We have evaluated the implemented prototype experimentally on a testbed that consists of a Compaq laptop for the web server and some traffic generators connected by an Ethernet hub. The Compaq laptop has an AMD Athlon 1800+ processor and we use the PowerNow drivers to scale the voltage. We ran Linux 2.5.65 on the laptop (we chose the experimental version rather than the stable 2.4 version due to better support for voltage scaling available with 2.5). The frequencies and voltages that can be achieved using the PowerNow driver are shown in Table 1.

To measure the power being dissipated by the system, a sense resistor is used to determine the current level of the system. The voltage drop is measured across the sense resistor, amplified, and sent into a Data Acquisition Card. A National Instruments PCI-6034E DAQ card was used to sample data at a sampling rate of 1000/sec. The voltage is sent through a voltage divider circuit and then into the DAQ card. (The DAQ card accepts only +/-10V. The AC adapter has a nominal voltage of 18.5V.)

The first part of our experimental study is to profile the execution times of different requests at different frequencies in order to determine the slow-down factor of computation time as processor frequency is reduced. Requests are generated using a standard workload generator, namely, httperf [17]. We use CGI scripts profiled at different frequencies and estimate changes in their computation times on the server. The profiling was done by requesting each CGI script at varying rates and noting the maximum rate the server can support before timeouts occur. The inverse of the maximum attainable rate gives us the processing time of each script at the frequency at which the experiment was conducted. This processing time is attributed to execution on the CPU and to various other overheads such as network and disk I/O operations. Figure 4 shows the maximum server throughput at different frequencies as a fraction of that at the highest frequency. Multiple curves are shown, each for a different frequency. The X-axis shows the computation time of the scripts (measured at the highest frequency), ordered from the less computationally intensive to the more computationally intensive. It can be seen that the curves tend to saturate in the region towards the higher execution times. This is because the decrease in server throughput becomes more a factor of processor speed as the scripts become more CPU intensive. With less computationally intensive scripts (the left of the graph), the difference between execution times at various voltage levels is gradually decreased. This is because the throughput in this region is also affected by I/O operations whose duration does not change with frequency scaling. Hence, the effect of frequency scaling on server throughput is reduced.

In subsequent experiments an average execution time was obtained for the current workload for the purposes of computing synthetic utilization. When the voltage was scaled from one level to another, the utilization was scaled in accordance with the ratios shown in Figure 4. The factors corresponding to compute-intensive scripts were used.

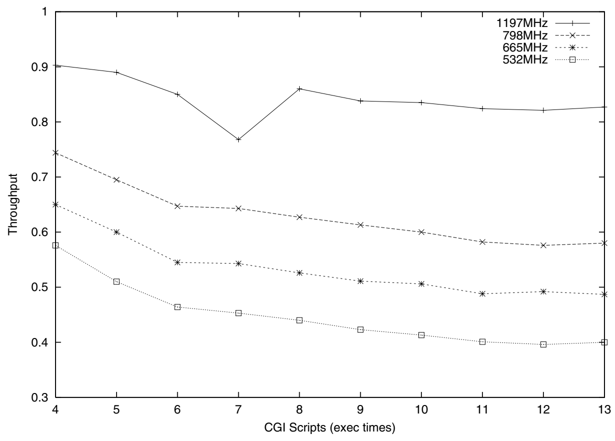

Figure 4. Ratio of throughput at different frequencies

Next, we demonstrate the performance of a DVS scheme that attempts to keep synthetic utilization around the utilization bound. In this experiment, we generate a session-oriented workload using httperf to read out logs of persistent connection-based requests. The sample workload consists of 1000 persistent connections. Each connection has

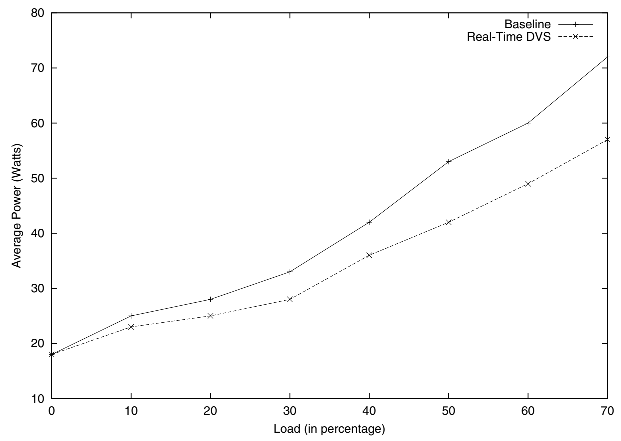

Figure 5. Average power consumption

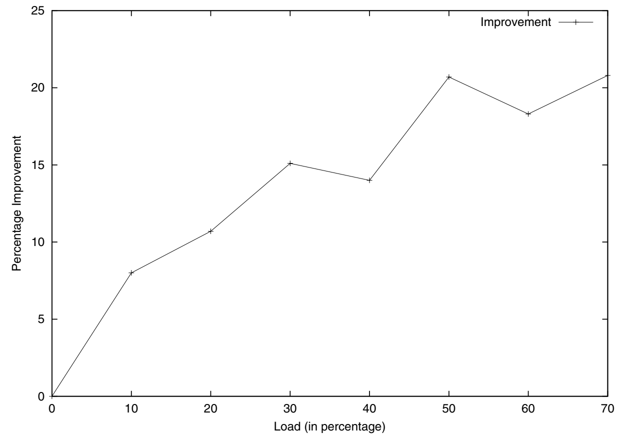

Figure 6. Energy savings compared to baseline

a length chosen randomly from the interval (1,10). Each of the files in the sessions could be one of the 10 scripts profiled above. The inter-arrival time of the sessions at the server can be specified as an exponential distribution with varying means. This provides the ability to generate different loads. Further, the same workload was generated from four client machines which were assigned different priorities. The deadlines for the four classes were 1, 2, 3 and 4 seconds respectively. The average power consumption and deadline miss ratio were measured in each experiment.

The average power consumed (as measured by the DAQ setup mentioned previously) is shown in the Figure 5. The plot shows the power consumed when the workload is executed with our DVS scheme, versus when it is executed with the default setup on Compaq AMD laptops. Observe that in the default setup, power consumption is decreased at low loads due to the HLT instruction which shuts down the clock grid when the processor is idle. As mentioned earlier, a DVS scheme on back-ends in a web server farm would typically be used in conjunction with a heuristic that switches some machines off or on depending on load conditions. Since such switching is time consuming and expensive, the heuristic will attempt to leave enough machines on, such that an adequate capacity leeway is provided to accommodate bursts. Due to this leeway, we expect that the load on an individual server will rarely approach maximum capacity. The X-axis in Figure 5 therefore goes only up to loads that are 70% of maximum machine capacity. A comparison reveals that we outperform the default scheme by up to 20% in that load range (Figure 6). When the load is very low, the performance of both schemes is the same. The convergence between the compared schemes is because when processor idle time increases, both schemes become equally opportunistic in saving energy. When the load grows, the real-time DVS scheme takes advantage of its additional knowledge of the actual deadlines to reduce power consumption beyond what is attainable simply by monitoring processor idle time. Hence, additional savings of 15-20% are made possible beyond the default power saving scheme. If load is increased further (not shown in figure), both schemes eventually become identical again because the potential for energy saving is eliminated at very high loads.

In the DVS experiments conducted above, the deadline miss ratio was less than 2%, which confirms that our scheme is able to maintain the system in the schedulable region while reducing power consumption.

Finally, we compute the savings that might be expected in a server farm running our algorithm. While we surveyed many server load statistics available from different web sources, in this paper we show one example that presents trade volume statistics of stock-market servers. Figure 7 shows the trade volume of the Dow Jones Industrial Average over the last two years. The average volume is approximately 300 million trades/day, which amounts to an average of 625,000 trades/second assuming a uniform trading volume throughout an 8-hour day. Assuming that a server farm running at capacity can support around 620 million trades/day (the maximum volume that occurs in the trade graph), a volume of 300 million corresponds to 46% load. From Figure 6, a reduction of approximately 17% is observed at that load over the default power management scheme.

If machines could be turned off with zero overhead and if the remaining machines could meet all deadlines even when 100% loaded, an optimal algorithm would turn off all machines that are not needed (i.e., 100% - 46% = 54%) leading to 54% energy savings. This can be construed as an unattainable upper bound. Realistic algorithms that combine turning off a fraction of the machines with running the algorithm proposed in this paper on the remaining machines are therefore expected to save up to 54% of total energy according to measurements from our laptop.

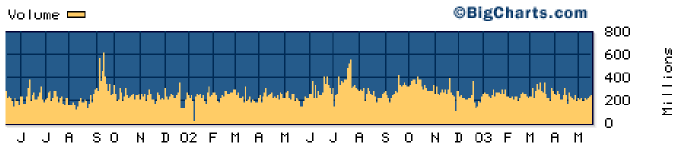

Figure 7. Server Load Statistics (Dow Jones Industrial Average)

5 Related Work

Initial work on power management has focussed on saving power in battery-constrained wireless embedded devices [10]. Dynamic voltage scaling was developed as an architectural tool for energy saving [7, 14]. This work was soon adopted for real-time computing giving rise to power management algorithms that are cognizant of time constraints. A common feature of most existing algorithms is that they assume a periodic task model.

Several variations of real-time DVS schemes have been investigated for periodic tasks. The basic problem of maximizing energy savings subject to meeting periodic task deadlines is presented in [15, 19, 16]. In [3], an optimal algorithm is described for assigning processor speeds to periodic tasks such that power is minimized. Extensions of power minimization to real-time tasks on multiprocessors are published in [24]. Dynamic adjustment for early task completions is investigated in [4]. Extensions to periodic tasks with blocking sections are presented in [22]. In [20] optimization under overload is addressed, which picks a subset of tasks, as well as a processor speed for each, such that energy constraints and deadlines are met while system utility is maximized. A similar performance optimization algorithm is presented in [12], except that some task execution parameters are assumed to be negotiable. The goal is to find the optimal parameter settings for each task to maximize system utility under energy and resource constraints. A performance comparison of several dynamic voltage scaling schemes for real-time systems is presented in [13]. Practical issues that arise from non-ideal characteristics of actual DVS implementations are addressed in [21].

As mentioned earlier, a common focus of the above algorithms is on periodic tasks. The effort in [11] is one of the few works which has focused on aperiodic tasks. However, they do so without considering QoS requirements. The authors of [18] focus on minimizing energy while meeting aperiodic task deadlines. However, their voltage scaling algorithm has high complexity and is unsuitable for large task sets.

Bohrer et. al [5] have made the case for power management in web servers. Subsequent work out of IBM Research labs has proposed algorithms for power efficient load balancing and request batching techniques to save power while maintaining a fair level of performance [9, 8]. As part of this work, a web server simulator, called Salsa, was built, which has been extensively validated for both energy and response time against measurements from a commodity web server. While the simulator gives a good idea of the power savings for validated web server workloads, the lack of a real testbed makes it hard to identify systems issues in applying the power management policies to a real server.

This paper attempts to bridge the gap between the real-time community and the power saving efforts for high performance web servers by presenting a policy for power saving that maintains QoS guarantees on HTTP requests. Further, this paper presents a system implementation of the scheme and discusses the problems from the systems perspective.

6 Conclusions

This paper addressed the important problem of energy saving in web servers subject to meeting real-time constraints. The work is an extension of prior approaches that addressed a similar optimization in the case of periodic tasks. A recently derived utilization bound for schedulability of aperiodic tasks was used to determine the extent of processor slow-down that will not interfere with meeting individual deadlines. This bound provides a basis for a new variation of dynamic voltage scaling schemes that minimizes energy consumption subject to aperiodic deadline constraints. Evaluation of the approach on an actual server prototype reveals that significant energy savings are possible while meeting request deadlines. Server load traces show that large variations in server workload allow the server to operate at a reduced speed a significant fraction of the time, which makes our scheme particularly useful.

Future work of the authors will focus on identifying efficient algorithms that maximize energy savings by combining the mechanism described in this paper with switching machines off or on depending on load conditions. While in this paper we provide a proof of concept evaluation, in subsequent work we shall explore the sensitivity of various algorithm parameters such as sampling time and input workload distribution to the amount of savings attained.

References

[1] T. Abdelzaher and V. Sharma. A synthetic utilization bound for aperiodic tasks with resource requirements. In *Euromicro Conference on Real Time Systems*, Porto, Portugal, July 2003.

[2] T. F. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *IEEE Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.

[3] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[4] H. Aydin, R. Melhem, D. Mosse, and P.-M. Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Real-Time Systems Symposium*, London, UK, December 2002.

[5] P. Bohrer, E. Elnozahy, M. Kistler, C. Lefurgy, C. McDowell, and R. R. mony. The case for power management in web servers. *Power Aware Computing, Kluwer Academic Publications*, 2002.

[6] H. Bryhni, E. Klovning, and O. Kure. A comparison of load balancing techniques for scalable web servers. *IEEE Networks*, pages 58–64, July 2000.

[7] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *International Symposium on Low Power Electronics and Design*, July 2000.

[8] M. Elnozahy, M. Kistler, and R. Rajamony. Energy efficient server clusters. In *Second Workshop on Power Aware Computing Systems*, February 2002.

[9] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *Fourth USENIX Symposium on Internet Technologies and Systems*, Seattle, March 2003.

[10] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *17th ACM Symposium on Operating Systems Principles (SOSP)*, 1999.

[11] D. Grunwald, P. Levis, and K. I. Farkas. Policies for dynamic clock scheduling. In *4th Symposium on Operating Systems Design and Implementation*, October 2000.

[12] D.-I. Kang, S. P. Crago, and J. Suh. A fast resource synthesis technique for energy-efficient real-time systems. In *Real-Time Systems Symposium*, Austin, Texas, December 2002.

[13] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Real-Time Technology and Applications Symposium*, San Jose, CA, September 2002.

[14] J. R. Lorch and A. J. Smith. Improving dynamic voltage scheduling algorithms with pace. In *ACM SIGMETRICS*, June 2001.

[15] T. C.-L. Ma and K. G. Shin. A user-customizable energy-adaptive combined static/dynamic scheduler for mobile applications. In *Real-Time Systems Symposium*, Orlando, FL, November 2000.

[16] P. Mejia-Alvarez, E. Levner, and D. Moss. Power-optimized scheduling server for real-time tasks. In *Real-Time Technology and Applications Symposium*, San Jose, CA, September 2002.

[17] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *WISP*, pages 59–67, Madison, WI, June 1998. ACM.

[18] T. Pering and R. Brodersen. Energy efficient voltage scheduling for real-time operating systems. In *4th IEEE Real-Time Technology and Applications Symposium*, June 1998.

[19] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *4th IEEE Real-Time Technology and Applications Symposium*, June 1998.

[20] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *Real-Time Systems Symposium*, Austin, Texas, December 2003.

[21] S. Saewong and R. Rajkumar. Practical voltage scaling for power-aware real-time systems. In *Technical Report, Department of Electrical and Computer Engineering, Carnegie Mellon University*, 2002.

[22] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptable sections. In *Real-Time Systems Symposium*, Austin, Texas, December 2003.

[23] R. Zhang, T. F. Abdelzaher, and J. A. Stankovic. Kernel support for open qos-aware computing. In *Real-Time Technology and Applications Symposium*, Washington, DC, May 2003.

[24] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *Real-Time Systems Symposium*, London, UK, December 2002.