Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control

Tibor Horvath, Tarek Abdelzaher, *Member*, *IEEE*,
Kevin Skadron, *Senior Member*, *IEEE*, and Xue Liu, *Member*, *IEEE*

Abstract—The energy and cooling costs of Web server farms are among their main financial expenditures. This paper explores the benefits of dynamic voltage scaling (DVS) for power management in server farms. Unlike previous work, which addressed DVS on individual servers and on load-balanced server replicas, this paper addresses DVS in multistage service pipelines. Contemporary Web server installations typically adopt a three-tier architecture in which the first tier presents a Web interface, the second executes scripts that implement business logic, and the third serves database accesses. From a user's perspective, only the end-to-end response across the entire pipeline is relevant. This paper presents a rigorous optimization methodology and an algorithm for minimizing the total energy expenditure of the multistage pipeline subject to soft end-to-end response-time constraints. A distributed power management service is designed and evaluated on a real three-tier server prototype for coordinating DVS settings in a way that minimizes global energy consumption while meeting end-to-end delay constraints. The service is shown to consume as much as 30 percent less energy compared to the default (Linux) energy saving policy.

Index Terms—Power management, voltage control, soft real-time systems, network servers, pipeline processing, distributed algorithms, optimization methods.

1 Introduction

Complex Web services are commonly realized by multitier Web server systems in order to functionally distribute computation across several computers. The different tiers perform different parts of request processing. For example, an e-business service usually consists of an HTTP server tier, an application server tier, and a database server tier. Client requests to these systems generally have highly varying and unpredictable resource requirements at each tier. Requests for static content, such as images or binaries, are often served by the first tier alone, with no resource usage in the others. On the other hand, an online purchase transaction would likely have a large processing demand on the application server and the database server, with the HTTP server only transferring a trivial amount of data.

In this paper, we consider the energy efficiency of multitier Web servers hosting soft real-time services with guaranteed end-to-end response times. These Web servers are often significantly overprovisioned in order to meet target response delay constraints even under peak loads. This practice, however, leads to poor overall energy efficiency since such systems are typically underutilized. The energy (and cooling) costs of large server farms are reported to be a significant part of their total upkeep and maintenance expenses [1], [2]. Excess power consumption not only hurts the operator economically, but it also limits the number of servers per unit volume (in the machine room) due to heat dissipation considerations [3]. Hence, there is an increasing need for solutions that reduce the system's energy consumption with as little effect on performance guarantees as possible.

Dynamic voltage scaling (DVS) is a powerful technique that allows significant energy savings by sacrificing some system performance. Reducing voltage requires a roughly proportional decrease in frequency, but power decreases quadratically with voltage. One of the key advantages of DVS (compared to other schemes, such as turning machines off) is that the overhead of performance adjustments is very low and, thus, it allows for an aggressive power saving policy.

Previous research has studied DVS in a single Web server or a single-tier Web server cluster with performance guarantees [1], [2], [3], [4], [5]. However, straightforward extensions of these are not sufficient to reasonably optimize power in server pipelines. In a pipeline, the end-to-end delay is composed of highly variable stage delays; therefore, independent stage delay control achieved by single-server algorithms cannot be effective in controlling the end-to-end delay. Further, since such independent DVS algorithms have no concept of end-to-end delay, their power optimization cannot be optimal because they lack the proper solution constraint. To our knowledge, no work has been done to address DVS in multitier Web servers with end-to-end delay constraints.

In this paper, we design, implement, and evaluate a coordinated distributed DVS policy for a traditional three-tier Web server system, based on distributed feedback control driven by a simple stage delay model. The policy is designed for realistic CPUs with discrete DVS frequency

- *T. Horvath and K. Skadron are with the Department of Computer Science, University of Virginia, 151 Engineer's Way, PO Box 400740, Charlottesville, VA 22904-4740. E-mail: tibor@cs.virginia.edu.*
- *T. Abdelzaher and X. Liu are with the Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801-2302.*

Manuscript received 14 Apr. 2005; revised 27 Feb. 2006; accepted 26 July 2006; published online 1 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0112-0405.
Digital Object Identifier no. 10.1109/tc.2007.1003.

settings. Decisions on frequency adjustments are made on each stage locally, governed by a decentralized self-coordination scheme. The self-coordination ensures that each stage can individually compute the globally optimal solution and apply it to itself, without the need for a central entity. We also present the formulation of the problem of determining the globally optimal DVS policy for such systems. We show experimental results from our prototype implementation confirming that our solution is efficient and stable. We experimentally verify that the proposed coordinated scheme outperforms uncoordinated single-machine power management. In particular, we compare it to the default Linux power management as a baseline. Additional energy savings in excess of 30 percent are observed.

It should be observed that the DVS-based approach explored in this paper does not exclude the usage of other power saving schemes. Typically, different schemes would be employed at different time-scales. For example, relatively long-term load fluctuation patterns (such as day/night fluctuations) can be accommodated by turning machines on or off to match the anticipated load, as proposed in earlier literature [4]. In such an on/off scheme, extra capacity would typically be left on each stage to accommodate shorter-term bursts. Hence, given a particular configuration of machines that are on, the protocol described in this paper can be used to determine their power-optimal DVS settings. Consequently, energy savings are increased by taking advantage of load fluctuations on shorter time-scales. Moreover, if machines in each tier are roughly load-balanced, their actions would typically be symmetric within the tier. Hence, in order to investigate coordinated DVS schemes across the pipeline, it is enough to consider a pipeline of one machine per stage. In thicker pipelines, assuming homogeneous servers with appropriate load-balancing in each stage, all machines within a stage will likely behave identically. With that in mind, we focus in this paper on deriving and implementing coordinated power-optimal DVS schemes for thin pipelines (i.e., those with one machine per stage) that respect end-to-end latency constraints.

The main contributions of this paper are the theoretical optimization of the energy efficiency of soft real-time multitier Web servers and the detailed case study and evaluation of our prototype testbed implementation.

The rest of the paper is organized as follows: Related work is presented in Section 2. Section 3 presents the general system architecture and DVS solution. Section 4 details the implementation. Performance evaluation is presented in Section 5. The paper concludes with Section 6.

2 Related Work

The importance of reducing both energy and power consumption in server systems is now well-known and has become a major research topic. Several papers [1], [2], [3] have made the case by pointing out the negative environmental effects, high operating costs, power density problems, and expensive infrastructure requirements of large server sites.

Earlier DVS research primarily addressed standalone, battery-operated, embedded mobile devices, which still remains an active research area, as demonstrated by the literature overview in [6]. Families of DVS algorithms integrated with an RTOS scheduler are proposed for periodic hard real-time task sets in [7], [8], [9]. DVS algorithms assuming similar task sets and a continuous frequency setting model are presented in [10] for multiprocessors. Recently, [11] presented the first feedback control-based DVS framework with EDF scheduling in hard real-time systems. A soft real-time energy-efficient scheduler for periodic tasks in embedded systems is presented in [12]. It employs a DVS algorithm similar to the most aggressive one[1] in [7], but it is based on CPU cycle demand distribution histograms built online. It can save more energy while providing statistical performance guarantees.

Much of the previous literature is focused on multimedia task sets. The authors of [13] devise a DVS algorithm for portable systems which relies on offline workload characterization and probabilistic online detection of arrival or service rate changes. Other DVS algorithms targeted at soft real-time systems predict near-future processing requirements (load) based on past history. PAST [14], one of the first such algorithms proposed by Weiser et al., simply assumes that the predicted (next) time window will have the same amount of idle time as the previous window had. Govil et al. presented and evaluated other prediction schemes [15], including AGED_AVERAGES, which uses a moving average of past samples with geometric decay, and PEAK, which expects short peaks in load and was shown to outperform PAST. Recently, the authors of [16] applied control theory to predict the future workload. They designed an algorithm, nqPID, that outperforms the aforementioned ad hoc algorithms, while its performance is also less dependent on parameter tuning. However, their results were validated only by simulation against a periodic task model. Feedback control techniques are used with DVS in [17], [18] to save energy while guaranteeing frame rate in multimedia workloads. The prediction is calculated based on a queuing model. In [17], similar energy savings are reported with reduced computation and improved quality of service over [13]. The authors of [18] use a dead-zone control method to provide strong real-time guarantees without requiring prior workload knowledge like many multimedia DVS schemes. However, since it controls buffer levels, it is not applicable to systems that do not tolerate buffering latency.

Several papers address DVS in standalone servers and server clusters. The authors of [1] present a soft real-time feedback control-based DVS policy combined with request batching. Simulation results show up to 42 percent savings of CPU energy in a standalone Web server when 90 percent of the response times are within the target deadline. They do not, however, validate their results by implementation in a real system nor do they measure total *system* energy savings. A real DVS policy is implemented in [5] for standalone Web servers with multiple QoS service classes which have soft real-time deadlines. The system builds on a proven schedulability bound for aperiodic tasks, due to

[1] The "Look-Ahead RT-DVS" algorithm, which is the most "aggressive" as it spends the greatest effort to exploit opportunities to reduce power consumption.

which it can sustain less than 2 percent deadline miss ratio. However, the work is restricted to a single tier server. Elnozahy et al. present and evaluate by simulation five different power management schemes for single-tier server clusters [4]. The schemes employ VOVO (*vary-on/vary-off*, i.e., turning nodes on and off depending on cluster load) and/or independent or coordinated (across the cluster) DVS. VOVO attempts to consolidate all workloads to just as many nodes as necessary, leaving enough slack for load spikes. An independent DVS policy (IVS) is completely node-local, while a coordinated one (CVS) is constrained to a small frequency range around the cluster average. VOVO combined with CVS is shown to be superior. However, they do not address service pipelines.

A power-aware scheduler for distributed systems with hard real-time end-to-end delay constraints is proposed in [19]. It is capable of determining an optimal voltage schedule in a single task chain (such as a multitier Web server), but it assumes periodic task chains and requires worst-case execution times. Our work is different from the above literature in that we present and evaluate the first system implementation of a feedback control-based DVS policy in a multitier (i.e., pipeline) service that is not restricted to periodic task instances.

3 Architecture

Our multitier Web service architecture consists of a pipeline of several processing stages. The processing at each stage invokes services of the next stage in a request-response fashion. Requests from a client are addressed to the first stage. Depending on content, they may be processed by subsequent stages sequentially. Such processing is typically in response to calls to business logic scripts and database queries. Eventually, calls and queries return to their originating stage with a response to be sent back to the client.

The nontraditional element in our energy-efficient architecture is that the server machines in the aforementioned pipeline have DVS-capable processors. By employing our novel coordinated DVS policy, the servers minimize the overall power consumption of the Web service while satisfying the (soft) real-time end-to-end delay constraints on request processing. The controlled variable is the end-to-end response delay, with the set-point (i.e., the target value of the controlled variable) being a preconfigured end-to-end delay value. To prevent frequent DVS changes in response to delay fluctuations, a dead-zone is imposed. In other words, no corrective action is taken as long as the measured end-to-end delay lies within an acceptable range between a low and a high threshold. If either threshold is violated, the feedback loop changes DVS settings in the pipeline to recover from the violation.

3.1 Delay Characteristics

End-to-end delays are continuously measured at the first stage, where client requests enter and responses leave. The average CPU utilization, U_i, is measured at each stage i with sampling period T. The measured end-to-end delay, D, can be broken into a delay component, D_i, for each stage i. Hence, for an N-stage system, $D = \sum_i^N D_i$. In turn, the delay D_i, on stage i, can be broken into a CPU processing delay, denoted D_i^{CPU}, and a blocking delay, such as I/O blocking, denoted D_i^{block}. This delay is incurred by a request when waiting on or using a resource other than the CPU.

The DVS mechanism manipulates CPU speed and voltage only. Thus, it can only control the CPU delay components, D_i^{CPU}. In contemporary multitier servers, significant non-CPU delay components, D_i^{block}, are typically present due to network latency and database I/O. This happens to be a fortunate circumstance from the perspective of DVS schemes, as opposed to a disadvantage. The reason is that DVS schemes opportunistically *increase* CPU delay, D_i^{CPU}, whenever possible (by slowing processors down) in order to save energy. If the end-to-end delay is primarily a function of D_i^{block} and not D_i^{CPU}, more aggressive energy savings can be accomplished without adverse effects on overall delay performance. Observe that it could be argued that the reverse is also true. Namely, if the I/O blocking delay, D_i^{block}, is very large and if the disk is the bottleneck, the system will needlessly try to increase CPU speed when it is overloaded. This will decrease power savings without affecting the actual bottleneck delay. Fortunately, this situation is easy to prevent at runtime by disallowing machines with a low CPU utilization from speeding up their CPU. Hence, unless the CPU is the bottleneck at some machine, power savings will not be needlessly impaired. The algorithm described below adopts this restriction.

3.2 Simple Dynamic Voltage Scaling

In a first attempt to design an optimal feedback-based DVS scheme in terms of energy savings, we only assume that the CPU delay, D_i^{CPU}, at stage i is a convex function $g(U_i)$ of the CPU utilization, U_i, at that stage. In other words, stage delay increases progressively more steeply as CPU utilization increases. Formally, the second derivative, $d^2 g(U_i)/d^2 U_i$, is positive. For example, given a Poisson arrival process and exponentially distributed execution times, we know from queuing theory that $D_i^{CPU} = T_i/(1 - U_i)$, where T_i is a constant. Hence, $d^2 g(U_i)/d^2 U_i = 2T_i/(1 - U_i)^3$, which is positive for $U_i < 1$.

This assumption is generally true of busy servers and it is intuitively supported by the observation that real-life Web servers tend to increasingly saturate when operating at higher CPU utilizations, leading to steeply increasing latencies. When the algorithm described below is applied to workloads for which this assumption does not hold, the resulting system will still maintain the end-to-end delay within constraints, albeit with poorer performance and energy efficiency. For instance, the assumption may be false in completely sequential workloads (i.e., in workloads with no parallel tasks), where CPU utilization has no effect on delay. However, typical Web server workloads are highly concurrent in nature because of the large number of independent clients.

The convexity assumption leads to a simple set of rules for adjusting CPU speed to globally maximize energy savings subject to delay constraints. Namely, if the measured end-to-end delay, D, exceeds an upper threshold, step up the frequency of the most loaded machine. Similarly, if the delay drops below a lower threshold, step down the frequency of the least loaded machine.

Intuitively, when the end-to-end delay exceeds the desired value, some processor's frequency must be stepped up to decrease that processor's utilization and, consequently, decrease delay. The convexity of the utilization-delay function implies that stepping up the frequency of the most utilized processor is a good rule-of-thumb because it results in the maximum reduction in delay for the same reduction in utilization. Hence, hopefully, delay can be brought down to the set point with the least additional energy expenditure.

By the same token, when the end-to-end delay is below the threshold, stepping down the frequency of the least utilized processor is a good choice because it results in the least impact on delay for the same increase in utilization. Hence, this processor can presumably be slowed down the most, resulting in the most energy savings.

The main advantage of the above algorithm is simplicity. It uses two simple rules that require only per-machine total utilization measurements and a measurement of end-to-end delay. In particular, it does not need to know individual stage delays, task execution times, or processor power characteristics.

The algorithm does not actually lead to an optimal solution to the energy minimization problem because it implicitly assumes that energy savings are proportional to utilization changes. In general, this is not true. Fortunately, if the processors's power-frequency curve and the workload's utilization-delay function are known, the above optimization algorithm can be easily adapted to produce the optimum energy consumption, as shown below.

3.3 Optimality Conditions

Let us assume that the power consumption P_i of stage i is a general function of CPU utilization:

$$P_i = p_i(U_i). \tag{1}$$

Second, assume that the delay D_i^{CPU} of a stage i is approximately related to its utilization U_i by the queuing-theoretic equation:

$$D_i^{CPU} = \frac{T_i}{1 - U_i}, \tag{2}$$

where T_i is the mean service time of each stage. In reality, this equation is not exact since studies suggest that Web workloads in general follow a heavy-tailed distribution [20]. Unfortunately, queuing models with heavy-tailed interarrival and service times are very difficult to analyze [21]. However, our model provides a reasonable approximation for deriving a practical optimality condition for typical multitier Web workloads. The intuition is that, with similar distributions in each tier, the relative estimation error diminishes and the resulting deviation from the optimal system state is insignificant compared to the deviation arising from the inherent discreteness of the system (i.e., small number of available frequencies). Our experimental results also support this intuition by showing improved performance with the proposed model. For other workloads where the model is inappropriate, the same analysis can be carried out with a different delay approximation equation.

Summing over the entire pipeline, the total power consumption P of the N-stage system can be expressed by:

$$P = \sum_{i=1}^{N} p_i(U_i). \tag{3}$$

Our objective is to minimize that power consumption subject to the constraint $\sum_{i=1}^{N} D_i^{CPU} + D_i^{block} \leq L$, where L is the maximum desired latency. Taking the equality condition as the limiting case and substituting from (2), this constraint can be rewritten as:

$$\sum_{i=1}^{N} \frac{T_i}{1 - U_i} = K, \tag{4}$$

where $K = L - \sum_{i=1}^{N} D_i^{block}$, which we assume is a constant independent of frequency settings since blocking delays are not affected by CPU speed.

To solve the aforementioned constrained optimization problem, we first add the Lagrange multiplier, λ, which yields:

$$L(U_i, \lambda) = \sum_{i=1}^{N} p_i(U_i) + \lambda \left(\sum_{i=1}^{N} \frac{T_i}{1 - U_i} - K \right). \tag{5}$$

Using the Kuhn-Tucker Theorem, we can get

$$\frac{\partial L}{\partial \lambda} = 0 \tag{6}$$

and, for each i,

$$\frac{\partial L}{\partial U_i} = 0. \tag{7}$$

The solution of (6) is exactly the constraint given by (4), thereby ensuring that it is always satisfied. From (7), we get

$$p_i'(U_i) + \lambda \frac{T_i}{(1 - U_i)^2} = 0, \tag{8}$$

which implies

$$\frac{p_i'(U_i)(1 - U_i)^2}{T_i} = -\lambda. \tag{9}$$

Hence, the optimal solution to the general power minimization problem is the following equalizing optimality condition:

$$\frac{p_1'(U_1)(1 - U_1)^2}{T_1} = \ldots = \frac{p_N'(U_N)(1 - U_N)^2}{T_N}. \tag{10}$$

Next, to arrive at a specific solution, let us consider the following equation between system power consumption and CPU frequency:

$$P_i = A_i f_i^n + B_i, \tag{11}$$

where A_i and B_i are constants. The general rule of thumb with CMOS technology is that $P \propto V^2 f \propto f^3$, that is, power is proportional to the cube of clock frequency. The rationale is that raising the clock frequency also necessitates increasing the voltage. In reality, however, $f \propto V$ is a simplification; hence, our more general expression. This assumption is accurately satisfied in realistic systems, with n ranging between 2.5 and 3. The same power model with $n = 3$ is assumed for analysis in [4]. In general, it is possible to obtain the exponent n and constants A_i and B_i by curve fitting against empirical measurements obtained from profiling the system.

If the workload arrival rate at stage i is λ_i cycles/sec, the utilization U_i of that processor is λ_i/f_i, where f_i is the service rate or frequency in cycles/sec. Equivalently, $f_i = \lambda_i/U_i$. Substituting in (11) yields the specific power-utilization function:

$$P_i = p_i(U_i) = A_i \frac{\lambda_i^n}{U_i^n} + B_i. \tag{12}$$

Note that the power consumption of a tier of machines has an indirect dependence on the other tiers: The behavior of the other tiers affects the utilization of the mentioned tier, which in turn affects its power consumption. Our model is simple in the sense that it does not contain a prediction component to capture how tiers affect each other's utilization. However, the model is fairly accurate in representing the true power consumption based on the observed utilization, which does reflect the intertier dependencies.

Substituting in the general solution given in (10), we get

$$\frac{p_i'(U_i)(1-U_i)^2}{T_i} = \frac{-nA_i\lambda_i^n U_i^{-(n+1)}(1-U_i)^2}{T_i}. \tag{13}$$

This finally leads us to the following equalizing optimality condition, which provides the optimal solution to our specific power minimization problem:

$$W_1H(U_1) = W_2H(U_2) = \ldots = W_NH(U_N), \tag{14}$$

where W_i is a weight which, after simplifying (14) by $(-n)$, is given by

$$W_i = \left(\frac{A_i\lambda_i^n}{T_i}\right)$$

and H is a transformation defined as

$$H(U_i) = \frac{(1-U_i)^2}{U_i^{n+1}}.$$

To minimize power consumption across the pipeline subject to the end-to-end delay constraint, a feedback loop is added to equalize the weighted transformed utilizations of all stages such that it satisfies (14). Utilization is manipulated by changing the CPU frequency settings.

3.4 Improved Algorithm with Miss Ratio Considerations

To converge on the condition expressed in (14), average local stage CPU utilization measurements, U_i, are broadcast by each machine at each sampling period. Average end-to-end delay D is computed by the first stage and also broadcast to all stages at each sampling period. Given this information, the distributed DVS algorithm on each machine computes the weighted transformed utilization, $W_iH(U_i)$, for each stage i. It is desirable to keep these values as equal as possible while observing that a given deadline miss ratio is not exceeded.

To ensure that a maximum tolerable miss ratio r is not exceeded, one can compute (from the expected workload distribution) the conditional probability that a deadline miss will occur in the next sampling interval given that the maximum delay observed in the current sampling interval is some fraction $\alpha_{hi} < 1$ of the actual deadline L. We denote this conditional probability by $P(D[k+1] > L | D[k] < \alpha_{hi}L)$, which is a function of α_{hi} (where $D[k]$ and $D[k+1]$ denote the delay measurements in the current and next samples, respectively). If the maximum acceptable deadline miss ratio is r, we would like to ensure that $P(D[k+1] > L | D[k] < \alpha_{hi}L) \leq r$. Given an analytically derived or empirically measured conditional probability function, the equality condition $P(D[k+1] > L | D[k] < \alpha_{hi}L) = r$ can be solved for α_{hi} simply by finding the point where the curve of this function reaches value r. The following two feedback rules are then applied:

- If $D > \alpha_{hi}L$ (overload), machine i with $\min_i\{W_iH(U_i)\}$ steps up its frequency to the next higher discrete setting. Note that, since H is monotonically decreasing in the range of U_i, this will increase the weighted transformed utilization, effectively balancing it as desired.
- If $D < \alpha_{lo}L$ (underutilization), machine i with $\max_i\{W_iH(U_i)\}$ steps down its frequency to the next lower discrete setting (where $\alpha_{lo} < \alpha_{hi}$). Note that, by symmetry, this will decrease the weighted transformed utilization, again meaning a balancing action.

The first rule guarantees that the conditions for a sustained miss ratio of r or more are always corrected to reduce the miss ratio. The second rule allows energy savings to be applied when the system is underutilized. By applying these rules simultaneously on each machine in the pipeline, we arrive at a distributed algorithm that converges on the globally optimal solution. Note that, if W_i are equal for all stages (such as in a homogeneous system with perfectly balanced load over the whole pipeline), the algorithm reduces to the one described in Section 3.2.

Finally, observe that, while we described the algorithm for a single class of clients with the same deadline, it is straightforward to generalize to multiple classes. The only change is that the first stage now measures the end-to-end delay for each class separately. This delay vector is broadcast to other stages. Let the deadline of class i be L_i and its measured end-to-end delay be D_i. Each stage executes the following two rules:

- If $\exists i : D_i > \alpha_{hi}L_i$ (overload), machine i with $\min_i\{W_iH(U_i)\}$ steps up its frequency.
- Else, if $\forall i : D_i < \alpha_{lo}L_i$ (underutilization), machine i with $\max_i\{W_iH(U_i)\}$ steps down its frequency (where $\alpha_{lo} < \alpha_{hi}$).

The first rule of the aforementioned algorithm can be further improved by excluding machines with a low CPU utilization (i.e., $U_i < U_{lo}$) from stepping up their speed. As mentioned earlier, this situation might arise if the disk was the true bottleneck, making the CPU speed irrelevant. With this improvement, CPU speed is adjusted only if the adjustment is likely to affect delay. The resulting algorithm has better energy savings in systems dominated by disk bottlenecks.

3.5 Feedback Control Model

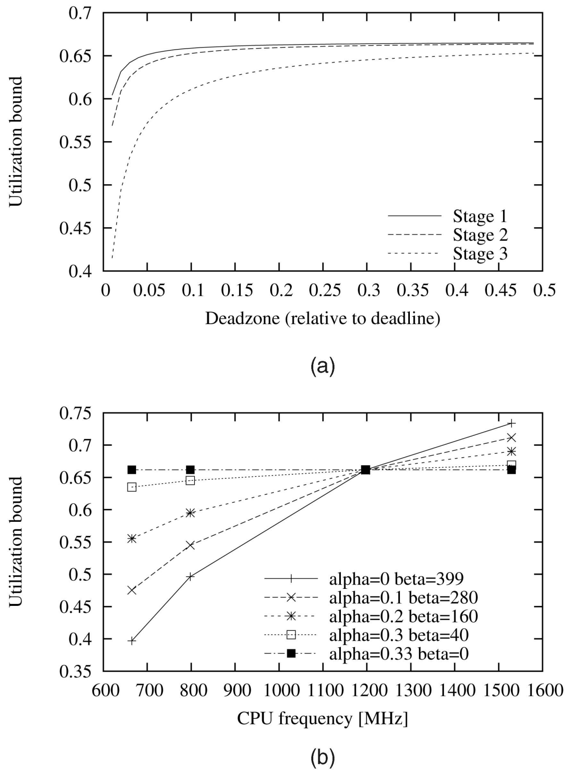
Having defined our DVS algorithm, we proceed to describe and analyze the design of the feedback control system we chose to implement. Note that our goal here is not to design an optimal controller, but rather to demonstrate the practical usefulness of our algorithm. Hence, we develop a simple yet effective controller, focusing on the design limitations of our target systems, such as the small number of available CPU frequencies. Other control-theoretic models of absolute delay control loops with different assumptions have been presented in [22].

The overall system is modeled by a discrete nonlinear feedback control loop with a deadzone. The input of the loop is the pair of threshold parameters α_{lo} and α_{hi}, which define the controller deadzone. The error signal received by the controller is then the difference between the measured end-to-end delay (feedback) and the center of this deadzone. Given the error signal, the controller determines the DVS adjustment as follows: If the error falls in the deadzone, no adjustment is made. If the error is greater, then the CPU frequency of one stage is adjusted by one stepping, as selected by the algorithm given in Section 3.4. Driven by the current CPU frequencies and the offered load, the multitier Web server (controlled system) processes requests with a certain end-to-end latency, which is sampled, averaged, and fed back to compose the aforementioned error signal.

Since the available CPU frequencies in the controlled system are limited, the presented saturated controller cannot become unstable. Limit cycles, where the system would only operate at the lowest or highest CPU frequency, are also impossible (assuming that other levels are available) because the frequencies are always adjusted by a single stepping only. However, analyzing stability is still worthwhile for identifying the presence of harmful oscillation in the system. To assure that our controller does not cause oscillatory behavior, it is sufficient to show that, under constant offered load, no frequency decrease can lead to a frequency increase. This means that, whenever the maximum end-to-end delay falls below the lower threshold, the average increase in delay caused by stepping down any CPU frequency should be smaller than the deadzone. Otherwise, the frequency decrease could drive the maximum end-to-end delay beyond the high threshold, which in turn would trigger a frequency increase, resulting in undesired oscillation in the average delay. The deadzone constraint can more formally be expressed as follows:

$$\frac{T_i}{1-\frac{\lambda_i}{f_i^{<j-1>}}} - \frac{T_i}{1-\frac{\lambda_i}{f_i^{<j>}}} \leq Z, \tag{15}$$

where $f_i^{<j>}$ is the jth CPU frequency setting in stage i and Z is the deadzone (relative to the deadline). In order to bound the average delay increase, we observe that, for any DVS architecture, we can find frequency bounding parameters α and β as follows:



(a)

(b)

Fig. 1. Effect of parameters on the utilization bound for stable CPU frequency reduction. (a) Varying the deadzone range. (b) Different frequency bounding parameters.

$$\exists \alpha, \beta : \forall j : f_i^{<\Delta j>} \leq \alpha f_i^{<j>} + \beta < f_i^{<j>}$$
$$\text{s.t.} \quad 0 \leq \alpha < 1 \quad \text{and} \quad 0 \leq \beta,$$

where $f_i^{<\Delta j>} = f_i^{<j>} - f_i^{<j-1>}$. Using these parameters, the following equation satisfies the deadzone constraint given by (15):

$$\frac{T_i}{1-\frac{\lambda_i}{(1-\alpha)f_i^{<j>}-\beta}} - \frac{T_i}{1-\frac{\lambda_i}{f_i^{<j>}}} = Z. \tag{16}$$

Solving (16) for λ_i, we get the arrival rate λ_i^B, for which the maximal delay increase arising from some frequency adjustment equals Z. Since the delay increase function on the left-hand side of (16) monotonically increases in λ_i, (15) is satisfied for all $\lambda_i \leq \lambda_i^B$. From this, $\lambda_i^B / f_i^{<j>}$ yields a utilization bound for each frequency setting j, below which the deadzone constraint is satisfied.

In Fig. 1, we calculate (based on profiled parameter values) the aforementioned utilization bound for various parameters. Fig. 1a shows the results for different deadzone ranges. We can see from the graph that a deadzone range of 0.3, for instance, yields a utilization bound of 64 percent. Fig. 1b demonstrates how different feasible choices of the frequency bounding parameters affect the utilization bound. As the graph shows, some parameter values give tighter bounds than others. For example, for the CPUs used in our experiments, $\alpha = 0.33$ and $\beta = 0$ give the tightest overall bound, 66 percent, over the whole frequency range. Together, the graphs show that a deadzone range of 0.3 will prevent frequency oscillation as long as the utilization of the CPU being stepped down is lower than 64 percent. This condition is likely to be satisfied as, generally, the most underutilized stage's capacity is decreased to save power. The conclusion is that the parameter α_{lo} should be selected based on the appropriate deadzone range that prevents oscillatory behavior.

Note that we rely on single-step actuation as opposed to changing multiple CPU frequencies at the same time. This is especially significant since the number of available DVS frequency settings for a CPU is usually small and multiple-step actuation could easily overreact to load variations. Having said this, the controller gain can be altered by changing the sampling period. Smaller periods result in higher gain since actuation is more frequent (while the actuation step remains the same). It is important to choose a sampling period that does not cause stability problems. Specifically, to avoid unnecessary oscillation, it is sufficient to ensure that the sampling period is long enough that the effect of the last frequency adjustment appears in the newly measured end-to-end delay. Therefore, a suitable sampling period should be inferred from the expected workload arrival rate and target latencies.

4 Implementation

4.1 Infrastructure Overview

In designing the structure of our implementation, our primary goal was to make our DVS policy as independent of the actual server software as possible. This is preferable because it is unobtrusive to the server software that we want to leave intact and extensible because it does not need to be modified to accommodate a new server software. There is no need to modify any existing server software on the source code level as long as we can measure the end-to-end processing delay on the first stage without doing so. This may be done by taking advantage of certain hooks the server software provides for plugin modules. The Apache Web server [23], for example, does provide such hooks.

Our prototype three-tier platform is composed of three laptop computers with Mobile AMD Athlon XP DVS-capable processors. The processors have discrete frequency levels ranging from 532 MHz to 1,529 MHz, with settling time specified as 100 microseconds. Each computer runs Linux 2.6. We implemented two separate three-tier Web server systems on this platform: a Synthetic system and a TPC-W [24] system. In the Synthetic system, the first two computers run Apache 1.3 as an HTTP front-end and as an application server, respectively, while the third computer runs the MySQL 4.0 database server [25]. The TPC-W system consists of the first computer running Apache 1.3 as the HTTP front-end and image server, the second running JBoss 3.2 as an application server, and the third running MySQL 4.0. As for the actual TPC-W software, we adopted a J2EE-based implementation of the TPC-W 1.8 benchmark [26], which uses contemporary technologies such as entity EJBs with container managed persistence for the best performance. We have not been able to find a readily available client for this server. Thus, on the TPC-W client side, we used a compliant Remote Browser Emulator from a separate source [27]. Several modifications were necessary to both the server and the client to make them interoperable. Since the client was not capable of accepting browser cookies containing the session identifiers, we modified the server to support session tracking using URL encoding. Further, we modified the client's URI fragments and patterns, as well as resolving interface-level incompatibilities. Our DVS policy is implemented independently as a standalone daemon to be started on all servers. The daemons on each stage establish TCP connections with the previous and next stages. Once they form a pipeline, they start self-coordination and control of the local CPU frequency.

4.2 Measurements and Actuation

Measuring end-to-end delay in practice is a challenge. True end-to-end delay could only be measured with kernel support. Alternatively, measuring delay in user space is a flexible yet imprecise solution. Since Linux does not yet provide the necessary timestamping support for TCP packets, we chose the user-space solution, which gives a reasonable approximation if the network is not the bottleneck resource on the first stage.

To obtain end-to-end delay samples, processing delays of the first stage (and thus the whole pipeline) are measured by our Apache extension module attached to the "post read-request" and the "logger" hooks. The time elapsed between the invocation of these two hooks for a given request is its measured end-to-end processing delay. In many cases, the implementation must also support separate request classes with different deadlines. For instance, the TPC-W specification defines several "Web interactions" with different delay constraints. Hence, our extension module also provides a new command, which allows deadline specifications for separate request classes identified by regular expressions against the request URI. Every sample thus consists of a delay measurement and a corresponding deadline. The DVS daemon running on the first stage provides a local (System V) Message Queue IPC interface to gather these samples. The measured end-to-end delay statistics are then periodically sent to all subsequent remote stages via TCP/IP messages.

At the end of each sampling period, average stage CPU utilization is measured by the DVS daemon on all stages. The utilization values are obtained from the Linux kernel by reading its clocktick accounting statistics from the virtual file "/proc/stat." Averages for each period are computed by subtracting the values collected at the end of the previous period from those at the end of the current period. The average stage CPU utilizations, along with the stage's current CPU frequency setting, are periodically sent from all stages to each other also via TCP/IP messages.

The DVS algorithm is invoked at the end of each sampling period. Through the coordination mechanisms described above, all stages ideally have a consistent view of current CPU frequencies, average CPU utilizations, and the end-to-end delay statistics; hence, they can solve the current global DVS problem instance independently. When a stage's solution indicates that one of the rules needs to be activated on itself, then that stage adjusts its CPU frequency (i.e., steps it up or down to the next discrete setting). The actual CPU speed setting is implemented by invoking the standard "userspace" frequency scaling governor of the Linux CPUFreq device driver.

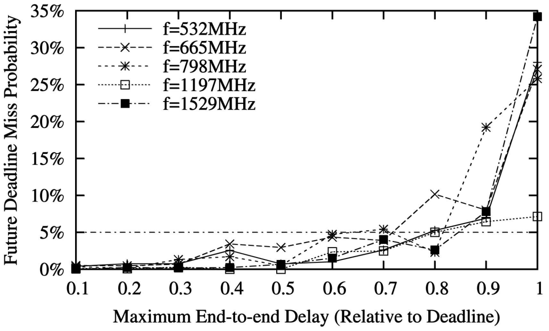


Fig. 2. Choosing the upper delay threshold.

4.3 Parameter Selection

When implementing our DVS algorithm, we must make an appropriate choice of the upper and lower delay thresholds, α_{hi} and α_{lo}, described in Section 3.4. Violations of these thresholds trigger reactions to overload and underutilization, respectively. As mentioned in Section 3.4, the upper threshold is chosen such that $P(D[k+1] > L|D[k] < \alpha_{hi}L) = r$, where L is the end-to-end deadline, $D[k+1]$ is the end-to-end delay in the next sampling period, $D[k]$ is the maximum end-to-end delay measured in the current sampling period, and r is the maximum tolerable deadline miss ratio. In other words, we would like the DVS algorithm to increase CPU speed when the conditional probability of a future deadline miss reaches the maximum tolerable miss ratio. Fig. 2 plots the aforementioned conditional probability for our workload as a function of the delay threshold. This curve was obtained empirically by observing the delays in every two successive sampling times. The conditional probability of a future deadline miss depends on CPU speed because, at lower speeds, individual requests contribute more to server delay, hence causing a larger delay variability. We imagine that, in high-performance servers where individual requests are very small compared to server capacity, the granularity of individual requests will play a smaller role. Let us take 5 percent to be the largest tolerable miss ratio. From Fig. 2, we see that a threshold of $\alpha_{hi} = 0.7$ guarantees that the maximum miss ratio will remain below 5 percent. The lower threshold is then selected by using the analysis in Section 3.5, where it was shown that 0.3 is an appropriate deadzone range for our workload. Hence, we use 0.4 as the lower relative delay threshold, which yields a deadzone range of $\alpha_{hi} - \alpha_{lo} = 0.3$.

In settings where the workload is not known at design time, the parameters must be determined online. The analysis presented above simply has to be automated by sampling the delays in the live system. The high threshold can then be computed from the estimated long-term conditional probability as shown above. Finally, an adaptive online algorithm can determine the minimal deadzone by measuring oscillations and adjusting the deadzone to avoid them, leading to an appropriate low threshold for the workload.

4.4 Control Performance

As we mentioned in Section 3.5, when choosing the sampling period, one major concern is to limit controller overshoot as much as possible. A conflicting concern is to get a short response time (rise time) when the system starts violating the performance requirements. We select a short sampling period during overload for the sake of high responsiveness to deadline misses. Our choice is $T = 200\,ms$ because, with the expected throughput of our workload, only a small number of requests exit during this time, which means the system quickly reacts after observing a few samples. It also results in a low controller overhead since coordination data will be measured and sent only five times per second. Since in modern systems the power consumption of the network interfaces is not significant compared to the main components (CPUs, memories, and disks), this communication overhead has very low effect on total system power. Also, since at most one frequency adjustment occurs in every sampling period, our 100 microsecond per period frequency transition overhead stays negligible even with this short period. However, such a small period is not suitable during underload because it leads to a small set of delay samples, which makes their average not sufficiently representative. Therefore, the sampling period during underload ranges from 4 to 10 seconds, depending on the deadlines of the workload. The reason is that this prevents the controller from decreasing system capacity before current request delays can be measured, as long as deadlines are met. Therefore, stability problems are avoided since the longer sampling period ensures that the effects of the previous frequency adjustment are seen before further adjustments. This much longer sampling period does not mean, however, that the system becomes unresponsive to deadline misses during underload because it is implemented in terms of the short periods by aggregating their samples. Hence, if deadline misses occur in any short period, the controller identifies an overload situation, which results in immediate corrective action at the end of that short period. Our results indicate that this yields a good compromise between soft real-time performance and energy savings.

Another design feature that impacts control performance is the issue of agreement in our distributed coordination scheme. Although synchronous coordination should be capable of guaranteeing coherence and consistency, it is expensive to enforce. Therefore, coordination (i.e., sharing of utilization values and end-to-end delay) is done asynchronously. Assuming that average utilization and average delay do not change abruptly from sample to sample (which can be ensured by an appropriate choice of the sampling period, discussed above), asynchrony has very little effect since state is not very time-sensitive. However, asynchrony does give rise to the possibility that, in an overload or underload situation, there might be no agreement on which stage should react (albeit there is likely to be an agreement on whether the system is underloaded or overutilized). As long as *any* stage decides to react, lack of agreement can only increase the extent of system reaction (as two or more machines decide to perform a corrective action). In other words, lack of agreement increases controller gain, which can be easily accounted for in stability analysis by substituting the expected value of system reaction for the actuation step size. The implication is that, if the impact is unacceptable, then a centralized design might be preferable.

Let us also remark that, since we do not assume that stage clocks are synchronized, the exact actuation times may vary throughout the pipeline. We note, however, that, in the worst case, any stage's reaction will be late by at most one sampling period since the last broadcast of end-to-end delay. Since we choose the sampling period to be small (compared to end-to-end deadlines) for fast system reaction, we argue that this delay is acceptable.

5 Evaluation

We evaluate two versions of our algorithm (the Feedback DVS version, implementing the naive policy, and the Weighted Feedback DVS version, implementing the optimal policy) by comparing them to a Baseline and an Independent DVS scheme. For the Baseline, we set the CPU frequency to the maximum on all stages. Let us point out that this does not necessarily mean that the CPUs will constantly run at that frequency. Linux (together with most modern operating systems) attempts to save power by default when the CPU is idle, even without a DVS policy. The exact way is platform and parameter-specific, but, usually, the CPU is turned off until a hardware interrupt occurs. Our platform uses the default method for x86 platforms: It executes the *hlt* instruction, which halts the CPU and puts it into a low-power state. Thus, our Baseline policy already performs such power management. For the Independent DVS scheme, we control the CPU frequencies independently, running an implementation of the PAST [14] DVS algorithm on each stage. All DVS algorithms are run on top of the Baseline policy. Thus, our reported power savings are those above the aforementioned policy.

The rationalization of our choice of comparison policies is that no other reasonable and applicable algorithm exists in previous literature to compare with. As we discussed in Section 2, reasonable previous solutions to multistage power optimization with real-time constraints are not applicable to aperiodic workloads with unknown worst-case execution times. On the other hand, algorithms devised for standalone servers or server clusters cannot reasonably satisfy end-to-end delay constraints in a multistage pipeline setting unless the end-to-end deadline is partitioned such that each stage works to satisfy a local deadline. Such partitioning must be done dynamically in a manner adaptive to current load, which makes it a nontrivial extension of the single-machine policy. The obvious extension of partitioning the end-to-end deadline a priori (e.g., by dividing by the number of stages) works very poorly because the stage load is not balanced, leading to poor performance and stability of such local schemes. Therefore, we deemed that comparisons with such algorithms would be unfair. Instead, we compare to two stable uncoordinated power management policies.

5.1 Workloads

To evaluate the expected real performance of our algorithm, we experiment with separate workloads for the two systems we implemented. The workload for the Synthetic system attempts to create a tunable server workload modeled after that of a typical three-tier Web server. While it is less representative of a specific application, it is very flexible. On the other hand, while the workload of the TPC-W system does not represent many different types of applications, it is a very realistic model of an online bookstore application. Our goal in implementing two different systems is to conduct a more comprehensive evaluation and to study the sensitivity of our algorithms with respect to the workload.

5.1.1 Synthetic Workload

As most serious services rely on large volumes of data, we create a reasonably sized database on the third stage. We have 500 tables, each table contains 1,000 records, and each record consists of 20 variable character fields. All records are initially filled with a key and 19 random fractional numbers. The physical size of the database (220 MB) prevents it from being entirely cached on our machine (the maximal observed cache memory size was 126 MB), making this stage I/O-intensive.

The second stage implements its application server functionality using CGI scripts, which perform data access and simulate data processing. The script first requests the database server to perform one of three different types of data manipulation actions: query record based on primary key, update record selected by primary key, and query records based on textual search pattern. The requested action is randomly chosen. In the first two cases, the key is randomly selected from the existing valid keys and, in the third case, the search pattern is a random 3-digit number as a substring. This randomization helps avoid invalid results due to disk caching by decreasing spatial locality of data accesses. These actions are, although minimal, representative of many real applications because they consist of both reads and writes, they involve both simple indexed lookups and complex nonindexed searches, and they can have highly varied execution times. Once the database access is finished, the script performs numeric calculations to simulate data processing. This processing, along with the processing done by the database client library (before sending a request to the database server), makes the second stage CPU-intensive, with the amount of CPU processing performed depending on the size of the data set received.

Finally, for the first stage, we create a small CGI script that sends an HTTP GET request to the second stage and copies the response to the client. It models the non-CPU intensive mediator and response-assembler role the HTTP server tier typically has.

Fig. 3 shows a histogram of the inherent end-to-end delay distribution of this workload in the Baseline case with no concurrent requests in the pipeline.

Test requests from the client are generated by the *httperf* [28] workload generator tool at various average rates. The request interarrival times are exponentially distributed. An individual TCP connection is created for each request.

5.1.2 TPC-W Workload

The database was populated as per the TPC-W requirements, with the scaling factors of 100 Emulated Browsers and 1,000 Items. The application server is logically further divided into two subtiers: It uses entity EJBs for database access (EJB tier) and servlets to provide access for clients to

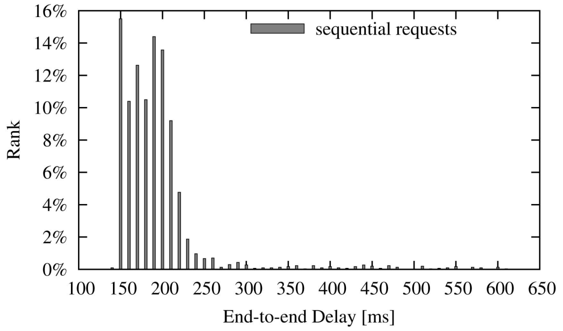


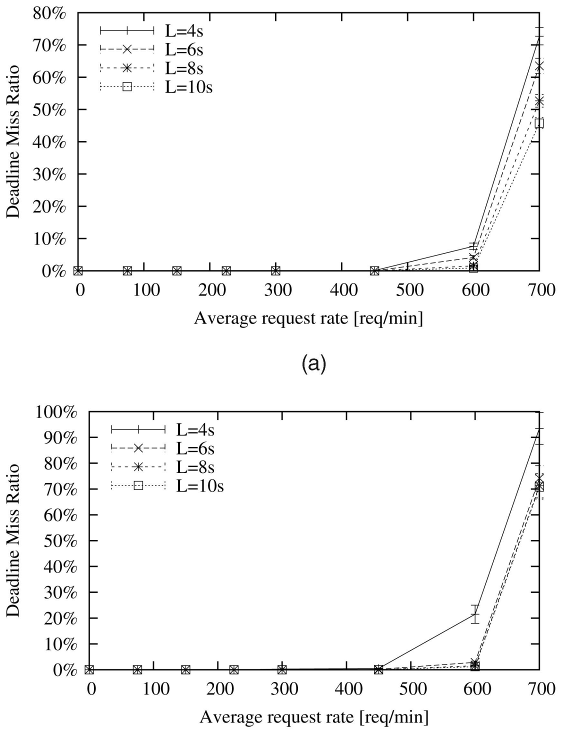
Fig. 3. Synthetic workload end-to-end delay distribution at sequential load.

the specified Web interactions (dynamic Web tier). In order to achieve better scalability, we increased the size of the database connection pool from the default 20 to 50. We also increased the connection timeout value from 5 to 20 seconds, the largest deadline in the TPC-W benchmark. The HTTP front-end server is loaded with all the static data: 1,000 item images, 1,000 thumbnails, and miscellaneous small images, such as buttons and icons. For dynamic requests (i.e., Web interactions), the HTTP server is set up as a proxy to the application server. To generate realistic client requests and to collect our statistics, we used the Remote Browser Emulator running in real time (i.e., no slow-down factor was used). The client workload profile used for the evaluation is the TPC-W Shopping Mix (the basis for the primary TPC-W metrics), which consists of an average mix of browsing and ordering activity. We varied the offered load by adjusting the number of Emulated Browsers, with other parameters (e.g., think time between user interactions) kept constant at their standard values.

5.2 Measurement Setup

We place our three server laptops on one network segment, making sure that unintended traffic does not flood it. The workload generator is run on a dedicated client computer located in a separate network segment. To filter out possible measurement errors due to lack of client resources, we verify that close to 100 percent of system time is available for request generation on the dedicated computer during each test.

To measure the power consumption of the laptops, we use three custom measurement circuits that sense the current flowing from each laptop power supply (AC adapter). The accuracy of the measurements was within 5 percent, similar to the solution described in [3]. Since the adapters provide constant voltage (18.5 V), we need not measure it. Observe that the adapter's voltage remains the same even when the CPU is performing DVS. Hence, our measurements reflect the true total power consumption of the laptop, including that of the CPU and other circuits. During power measurements, we remove the batteries from the laptops since we do not want to measure power consumed to charge them and we want the laptops to obtain power exclusively from the AC adapter. Also, since server systems usually do not include a display, we turn off the LCD backlighting, which drains a significant amount of power. We do not, however, turn off the display adapter, by which our power savings could be improved further without affecting performance.



(a)

(b)

Fig. 4. Performance of the comparison algorithms (synthetic). (a) Baseline. (b) Independent DVS.

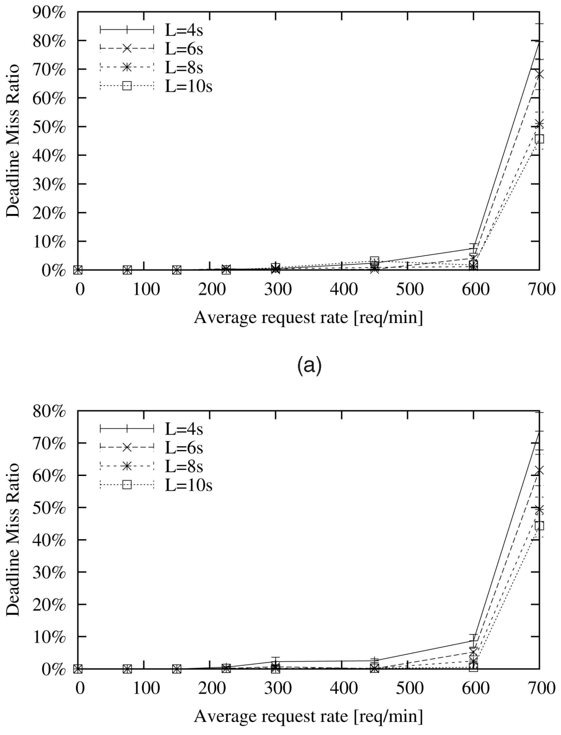
Current readings for all three laptops are performed simultaneously at a rate of 2,000 samples per second per channel, using three channels of a National Instruments PCI-6034E data acquisition card installed in a separate computer. The average stage power consumptions for the test duration are then calculated offline. Performance data, such as the deadline miss ratio, are collected from the output of the workload generator tool.

5.3 Performance Results

5.3.1 Synthetic Workload Results

Next, we evaluate the energy savings and deadline miss ratio of the synthetic 3-tier service that runs our DVS algorithm. Each data point in our results is obtained by running several experiments for 3-5 minutes and plotting the average values along with error bars. Since we want to show the stable behavior of the system, we eliminate transient cold-start effects by running a short (18-30 s) lead-in workload prior to starting each experiment.

Figs. 4a and 4b plot the deadline miss ratio of the two comparison policies as a function of the average request rate, which we vary from 0 (no load) to 700 requests/minute (severe overload). We perform several sets of experiments for different deadlines ranging from 4 to 10 seconds. These deadlines are natural for our setup for a number of reasons. First, delays in multitier Web servers are the sum of the delays of individual stages. As the database tier typically has much larger delay due to I/O than other stages, a 4-second deadline easily translates to a



(a)

(b)

Fig. 5. Performance of our novel algorithms (synthetic). (a) Feedback DVS. (b) Weighted feedback DVS.

subsecond delay bound to the first two stages in our three-tier prototype. Second, the typical e-business server workloads that we model usually include computationally complex operations that work on large data sets. These operations can cause delays to be on the order of seconds in these systems. Note that the TPC-W specified delay constraints are also in a similar range (3-20 seconds). Another factor is that our testbed computers are slow compared to real-life Web server hardware. Obviously, on faster machines, shorter deadlines are possible. Nevertheless, real studies with e-business Web site users in [29] show that these deadlines are in the tolerable range in most cases. The Baseline graph (Fig. 4a) shows that the system begins to saturate at 450 requests/minute in each case and that saturation is naturally slower with higher deadlines.

Fig. 5a presents the miss ratio of the (simple) Feedback DVS algorithm. This first version is one where all weights W_i are assumed to be equal (as an approximation). The advantage of this version is that it does not require knowledge of the power characteristics of the CPUs and characteristics of machine workload. If such information is available, however, it is possible to compute the coefficients W_i derived in Section 3.4. Fig. 5b shows the resulting improved (optimal) Weighted Feedback DVS algorithm using weights derived from empirical measurements. We can see that, when the system is underloaded, our algorithms have a slightly higher deadline miss ratio than the comparison algorithms, but still within our specified tolerable limit, 5 percent. This means that the end-to-end delays are successfully controlled so that deadlines are statistically (at least 95 percent of the time) still met. Therefore, the increased miss ratios are acceptable in soft real-time systems such as our multitier Web service. As we see next, the increased miss ratios are the results of improved power savings with our novel algorithms.

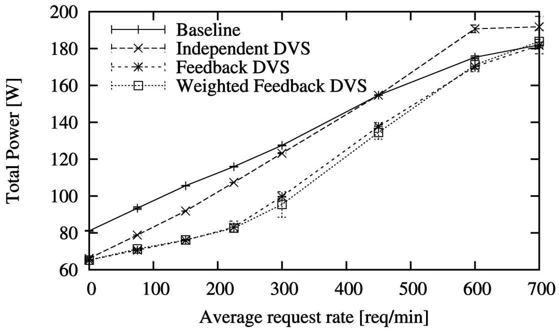


Fig. 6. Total system power consumption (synthetic).

To illustrate what energy savings are achieved, in Fig. 6, the total power consumption of the three servers using the feedback DVS policies is compared to the total power consumption using the two comparison policies. For each load level, the power samples are obtained by performing individual measurements for each deadline. The lines connect the averages of these samples, while the error bars show the minimum and maximum values. (We note that, at many data points, the power measurements were so consistent across our experiments that the corresponding error bars are not visible graphically.) We can verify that the Baseline power saving policy in fact saves a considerable amount of power in itself when the system is underutilized: It achieves an approximately 80 W base power consumption out of the highest observed power of over 180 W.

Finally, Fig. 7 displays the overall power savings attained by the two feedback DVS policies and the Independent DVS policy. We can see that both of our algorithms can achieve above 30 percent total power savings under medium load. The graph also demonstrates that the improved algorithm in fact slightly outperforms the original algorithm. As conjectured, both of our algorithms also have a great advantage over the Independent policy. Let us observe that approximately 20 percent power is saved even when the system is idle because background processes and periodic kernel operations such as the timer handlers all run at lower frequency. The highest relative power savings are realized at medium load (150-225 requests/min). The shape of the curves is explained by the fact

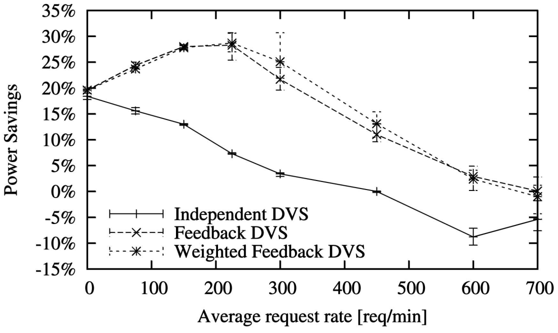


Fig. 7. Total DVS power savings (synthetic).

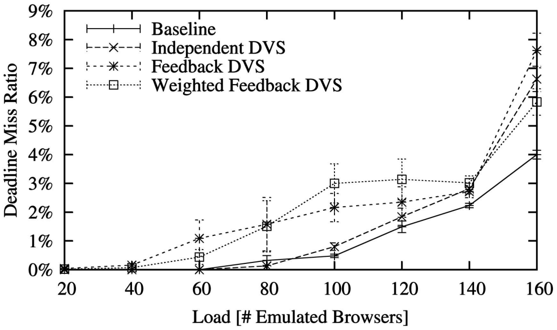


Fig. 8. Comparative performance (TPC-W).

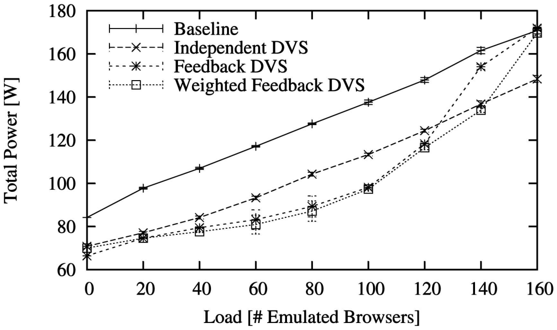


Fig. 9. Total system power consumption (TPC-W).

that, in light load (0-75 requests/min), the CPU is often idle; therefore, most of the power saving opportunities are exploited by the Baseline policy and little can be added by the DVS algorithms. As load increases, there is less chance for the *hlt* instruction to be executed. Our policy wins because it can run the processor at a lower frequency. Progressing toward heavier loads (above 300 requests/min), there is no longer much opportunity to lower processor frequencies. Therefore, the power savings diminish. Since most server farms are normally overprovisioned, a substantial power reduction is possible using our schemes.

5.3.2 TPC-W Workload Results

In this section, we describe the experimental results obtained from our 3-tier TPC-W service. Every data point reports the results of multiple repeated experiments, showing the average behaviors and the observed deviations. Each individual test run consists of a 10-minute ramp-up period, a 30-minute measurement interval, and, finally, a 5-minute ramp-down period. The load placed on the system is identical throughout each test run, but data collection takes place solely in the measurement interval. The ramp-up period is used to warm up the system to the desired operating point, while the purpose of the ramp-down period is to keep the system at that point even as the measurement finishes. Thus, test startup and shutdown effects are eliminated from the results. These test runs are much longer than the ones we performed with the synthetic workload in order to meet the TPC-W requirements. The long measurement interval is necessary, for instance, to collect sufficient samples from each individual Web interaction type.

Fig. 8 displays the performance (deadline miss ratio) of each algorithm we considered with the TPC-W workload. Since every TPC-W Web interaction type (class) has a specified end-to-end deadline, the plotted data points reflect the aggregate miss ratios, i.e., the number of interactions that missed their deadline per the total number of interactions. The graph shows that our performance goal (5 percent miss ratio) is met by each policy when the system is not overloaded.

Figs. 9 and 10 show the results of the power measurements. The first plots the absolute total system power consumption, while the second visualizes the gains of each considered algorithm relative to the Baseline scheme. We can make similar observations to the ones about the Synthetic system. Both of our new algorithms exhibit improved power savings over both comparison policies. Again, our Weighted algorithm slightly outperforms our simple one, as expected. Bigger differences can be expected between the two policies in heterogeneous systems, where the simple algorithm would become less useful. For this workload, the Independent DVS policy saves somewhat more power at the highest loads, but with a slightly higher deadline miss ratio than the Weighted DVS algorithm. In conclusion, overall we have obtained very similar results for the TPC-W workload as for the Synthetic one, which suggests that our schemes are not sensitive to a specific workload type.

5.3.3 Discussion

To understand why our algorithms perform better than the competition, the key is to realize that the inputs available to each algorithm are different. Since the local (independent) algorithms have no knowledge of the global performance of the system (i.e., whether end-to-end deadlines are met), they can only assess how much CPU capacity the machine should provide based on local metrics, such as the CPU utilization. As periods of high CPU utilization can occur in some machines without causing global performance problems (deadline misses), in such cases, the local algorithms waste power by unnecessarily increasing the overall capacity. By contrast, our global (coordinated) algorithms allow machines to run at lower capacity and, hence, lower power as long as the actual performance constraints are not violated. As we saw on the graphs, the difference, i.e., the wasted power to provide unnecessary capacity, varies with the offered load. Moving from low to medium load, it increases because the number of periods with high CPU

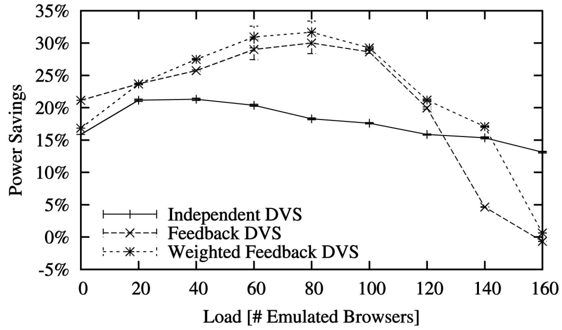


Fig. 10. Total DVS power savings (TPC-W).

TABLE 1
Number of CPU Speed Adjustments with the Weighted Algorithm

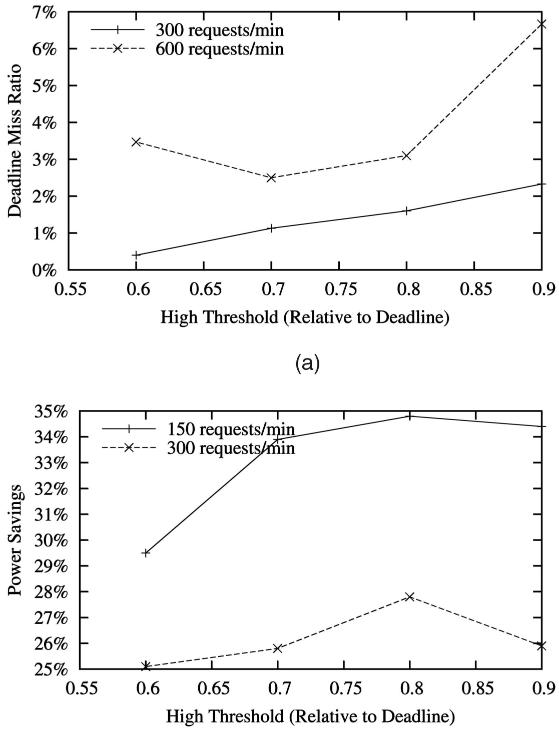
Load	*Adjustments/minute*
20	0.5
40	1.3
60	4.1
80	9.2
100	20.8
120	25.0
140	18.8
160	3.5

utilization but no performance constraint violations grows. At medium load, the growth stops as violations start occurring and, moving toward high load, the difference decreases because the increased capacity gradually becomes more necessary and thus less wasteful.

One might wonder if our algorithms have any advantage over a static scheme that configures the CPUs to run at some fixed clock speed. Since such a scheme is open-loop by nature, our algorithms possess all the advantages of closed-loop systems, most notably: decreased sensitivity to parameter variations in the controlled system and improved rejection of transient disturbances. In other words, since the static scheme must be calibrated for a specific system and workload, it could not achieve optimal performance because of persistent errors in calibration and transient noise in the system. Further insight can be gained by measuring the average number of CPU speed adjustments during experiments with constant average load. The results shown in Table 1 indicate that significant variability exists in the system even if the load is kept constant. (Note that we verified that the adjustments were not controller-induced oscillations.) Therefore, an open-loop algorithm such as the aforementioned static scheme is unlikely to achieve similar performance.

5.3.4 Parameter Sensitivity

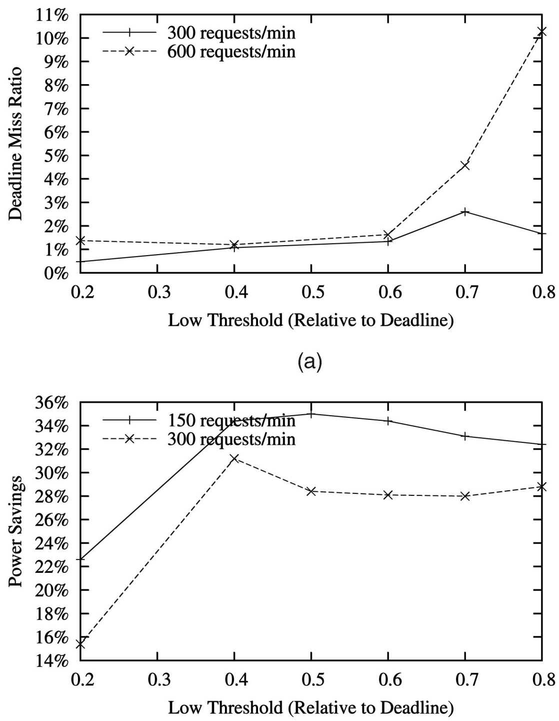
We performed additional experiments to evaluate the effects of varying the thresholds of our algorithm. As we shall see, decreasing these thresholds generally reduces both deadline misses and power savings. In Fig. 11, we verify that 0.7 is a good choice for the high threshold because it yields a lower miss ratio than the higher ones and it saves more power than the lower ones. Next, in Fig. 12a, we can see that a low threshold of at most 0.6, which corresponds to a deadzone of at least 0.3 (since the high threshold is fixed at 0.9), yields a low miss ratio. At the same time, Fig. 12b shows good power savings when the low threshold is at least 0.4. Reconciling with our selected high threshold of 0.7 and our minimum deadzone requirement of 0.3, we verify that 0.4 is indeed a good choice for the low threshold.



(a)

(b)

Fig. 11. Effects of different high thresholds (low threshold = 0.5). (a) Effects on deadline miss ratio. (b) Effects on power savings.



(a)

(b)

Fig. 12. Effects of different low thresholds (high threshold = 0.9). (a) Effects on deadline miss ratio. (b) Effects on power savings.

5.3.5 Observations

Let us note that, for both systems (Synthetic and TPC-W), data points that are compared to calculate the power savings for a specific load level are obtained from experiments of approximately the same duration and amount of work, with negligible differences. Thus, the measured average power consumption was proportional to the total energy spent during compared experiments. From

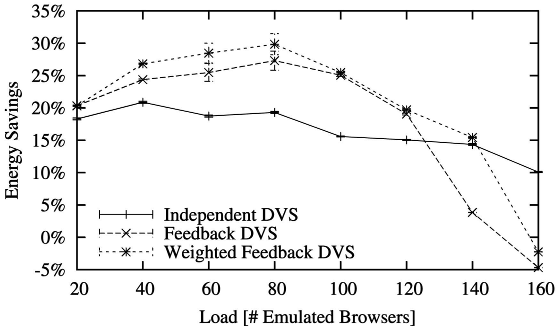


Fig. 13. Total DVS energy savings (TPC-W).

this, it can be seen that the total energy savings are approximately equal to the total power savings presented. Our intuition is also supported by Fig. 13, where we plot the energy savings in the TPC-W experiments. The basis of comparison is the metric of total system energy spent per Web interaction, which is obtained by dividing the average total system power by the average Web interactions per second (WIPS, a standard TPC-W metric).

Four important points are made from the experimental results. First, nontrivial power savings can be achieved using our DVS scheme while maintaining the miss ratio at a low rate. Second, the optimal savings occur when the *weighted transformed* utilizations of all machines are equal and *not* when utilizations are perfectly balanced. This interesting observation is confirmed both theoretically and experimentally. Third, balancing machine utilizations is an adequately good heuristic that is very easy to implement, largely independently of load and machine characteristics. Finally, the scheme does not require any modifications to server code. We therefore believe that our algorithms are both practical and efficient, which makes them a good candidate for implementation in real-life systems.

6 Conclusions

In this paper, we presented a distributed DVS control algorithm that minimizes overall power consumption in a server pipeline subject to end-to-end latency constraints. While the algorithm was described for a single class of clients, straightforward extensions to multiple classes are possible. A formal derivation of optimality conditions was given, together with a feedback control architecture that drives the system to satisfy these conditions. Interestingly, it was shown that the optimal power savings do not always coincide with the load balanced condition of equal utilization on all servers. However, in practice, such load balancing is a good approximation. A functional prototype of this system was implemented and experimentally evaluated. Empirical measurements confirm theoretical results and show that our system consumes up to 30 percent less energy than the default Linux power saving mode. These savings have a significant effect on the operation cost of large server farms.

This work will be extended to larger server clusters with multiple machines per stage and multiple classes of clients with different timing constraints. Our future work also includes the investigation of more advanced control strategies and the development of a comprehensive power management solution by integrating our DVS algorithms with the handling of other system power states.

Acknowledgments

This work was funded in part by US National Science Foundation grant no. CNS-0306404, US Army Research Office grant no. W911NF-04-1-0288, and a grant from Intel MRL. The authors would also like to thank the anonymous reviewers for their helpful comments.

References

[1] M. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," *Proc. Fourth USENIX Symp. Internet Technologies and Systems,* Mar. 2003.

[2] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems," *Computer,* special issue on Internet data centers, vol. 37, no. 11, Nov. 2004.

[3] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony, "The Case for Power Management in Web Servers," *Power-Aware Computing,* R. Graybill and R. Melhem, eds., Kluwer Academic, Jan. 2002.

[4] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. Workshop Power-Aware Computing Systems,* Feb. 2002.

[5] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-Aware QoS Management in Web Servers," *Proc. IEEE Int'l Real-Time Systems Symp.,* p. 63, 2003.

[6] O.S. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," *Proc. IEEE,* special issue on real-time systems, vol. 91, no. 7, pp. 1055-1069, July 2003.

[7] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," *Proc. ACM Symp. Operating Systems Principles,* pp. 89-102, 2001.

[8] H. Aydi, P. Mejia-Alvarez, D. Mosse, and R. Melhem, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '01),* p. 95, 2001.

[9] P. Mejia-Alvarez, E. Levner, and D. Mosse, "Power-Optimized Scheduling Server for Real-Time Tasks," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '02),* p. 239, 2002.

[10] D. Zhu, R. Melhem, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems," *Proc. IEEE Real-Time Systems Symp. (RTSS '01),* p. 84, 2001.

[11] Y. Zhu and F. Mueller, "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.,* pp. 84-93, May 2004.

[12] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," *Proc. ACM Symp. Operating Systems Principles,* pp. 149-163, 2003.

[13] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G.D. Micheli, "Dynamic Voltage Scaling and Power Management for Portable Systems," *Proc. 38th Conf. Design Automation,* pp. 524-529, 2001.

[14] M. Weiser, B. Welch, A.J. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *Proc. USENIX Symp. Operating Systems Design and Implementation (OSDI '94),* pp. 13-23, Nov. 1994.

[15] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," *Proc. First Ann. Int'l Conf. Mobile Computing and Networking,* pp. 13-25, 1995.

[16] A. Varma, B. Ganesh, M. Sen, S.R. Choudhury, L. Srinivasan, and J. Bruce, "A Control-Theoretic Approach to Dynamic Voltage Scheduling," *Proc. Int'l Conf. Compilers, Architectures, and Synthesis for Embedded Systems,* pp. 255-266, 2003.

[17] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron, "Control-Theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems,* pp. 156-163, 2002.

[18] Z. Lu, J. Lach, M.R. Stan, and K. Skadron, "Reducing Multimedia Decode Power Using Feedback Control," *Proc. 21st Int'l Conf. Computer Design*, pp. 489-496, Oct. 2003.

[19] D.-I. Kang, S. Crago, and J. Suh, "Power-Aware Design Synthesis Techniques for Distributed Real-Time Systems," *Proc. ACM Workshop Languages, Compilers, and Tools for Embedded Systems (LCTES '01)*, pp. 20-28, June 2001.

[20] A. Williams, M. Arlitt, C. Williamson, and K. Barker, *Web Workload Characterization: Ten Years Later.* Springer, Aug. 2005.

[21] M.J. Fischer, D.M. Masi, D. Gross, and J.F. Shortle, "One-Parameter Pareto, Two-Parameter Pareto, Three-Parameter Pareto: Is There a Modeling Difference?" *The Telecomm. Rev.*, pp. 79-92, Mitretek Systems, 2005.

[22] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queuing Model Based Network Server Performance Control," *Proc. 23rd IEEE Real-Time Systems Symp.*, Dec. 2002.

[23] The Apache Software Foundation, "The Apache HTTP Server," http://www.apache.org/, 2007.

[24] Transaction Processing Performance Council, "TPC Benchmark W (Web Commerce)," http://www.tpc.org/tpcw/, 2007.

[25] MySQL AB, "MySQL Database Server," http://www.mysql.com/, 2007.

[26] A.A. Totok, "J2EE-Based Implementation of the TPC-W Benchmark," New York Univ., Feb. 2004, http://www.cs.nyu.edu/~totok/professional/software/tpcw/tpcw.html.

[27] ObjectWeb Consortium, "TPC-W Benchmark," based on the implementation from Univ. of Wisconsin-Madison, http://jmob.objectweb.org/tpcw.html, 2007.

[28] D. Mosberger and T. Jin, "httperf: A Tool for Measuring Web Server Performance," *Proc. First Workshop Internet Server Performance*, pp. 59-67, June 1998.

[29] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design," *Proc. Ninth Int'l World Wide Web Conf. Computer Networks*, pp. 1-16, May 2000.

Tibor Horvath received the BEng degree in engineering informatics from the Kando Kalman College of Engineering, Budapest, Hungary, in 1999 and the MS degree in computer science from the University of Szeged, Hungary, in 2001. He is currently working toward the PhD degree at the University of Virginia. His research interests include power-aware systems, computer architecture, and wireless sensor networks.

Tarek Abdelzaher received the BSc and MSc degrees in electrical and computer engineering from Ain Shams University, Cairo, Egypt, in 1990 and 1994, respectively. He received the PhD degree from the University of Michigan in 1999. He has been an assistant professor at the University of Virginia. He is currently an associate professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He has authored/coauthored three book chapters and more than 60 refereed publications in leading conferences and journals in several fields including real-time computing, distributed systems, sensor networks, and control. He is editor-in-chief of the *Journal of Real-Time Systems*, served as program chair of RTAS 2004, finance chair of IPSN 2006, sensor networks vice chair of RTSS 2005 and ICDCS 2006, system vice-chair of DCoSS 2006, and general chair of RTAS 2005. He is currently finance chair of Sensys 2006, program chair of RTSS 2006, and general chair of IPSN 2007. His research interests lie broadly in understanding and controlling the temporal properties of software systems in the face of increasing complexity, distribution, and degree of embedding in an external physical environment. He is a member of the IEEE and the ACM.

Kevin Skadron received the BSEE and the BS degree in economics from Rice University, Houston, Texas, and the MA and PhD degrees from Princeton University, Princeton, New Jersey. He joined the Department of Computer Science, University of Virginia, Charlottesville, in 1999 and is now an associate professor. His research interests focus on the implications of technology trends and physical constraints (e.g., power, temperature, reliability) for future, highly multicore architectures. Dr. Skadron was the recipient of three Best Paper Awards, US National Science Foundation ITR, CRI, and CAREER Awards, cochaired MICRO 2004 and PACT 2002, was program cochair for PACT 2006, and is founding associate editor-in-chief of *IEEE Computer Architecture Letters*. He is a senior member of the IEEE and a member of the ACM.

Xue Liu received the BS degree in applied mathematics and the MEng degree in control theory and applications from Tsinghua University in 1996 and 1999, respectively. He is a PhD candidate at the University of Illinois at Urbana-Champaign. His research interests include real-time and embedded computing, performance and power management of server systems, sensor networks, fault tolerance, and control. He has authored/coauthored more than 20 refereed publications in leading conferences and journals in these fields. He received the Ray Ozzie Fellowship, the Saburo Muroga Fellowship, the C.W. Gear Outstanding Graduate Award, and the Mavis Memorial Fund Scholarship Award from the University of Illinois at Urbana-Champaign. He is a member of the IEEE.