Energy-Efficient Policies for Request-Driven Soft Real-Time Systems *

Cosmin Rusu, Ruibin Xu, Rami Melhem, Daniel Mossé
Computer Science Department, University of Pittsburgh
Pittsburgh, PA 15260
{*rusu,xruibin,melhem,mosse*}@*cs.pitt.edu*

Abstract

Computing systems, ranging from small battery-operated embedded systems to more complex general purpose systems, are designed to satisfy various computation demands in some acceptable time. In doing so, the system is responsible for scheduling jobs/requests in a dynamic fashion. In addition, with power consumption recently becoming a critical issue, most systems are also responsible for their own power management. In some rare cases, the exact arrival time and execution time of jobs/requests is known, leading to precise scheduling algorithms and power management schemes. However, more often than not, there is no a-priori knowledge of the workload. This work evaluates dynamic voltage scaling (DVS) policies for power management in systems with unpredictable workloads. A clear winner is identified, a policy that reduces the energy consumption one order of magnitude compared to no power management and up to 40% (in real-life traces) and 50% (in synthetic workloads) compared to the second-best evaluated scheme.

1 Introduction

From portable battery-operated computing systems to server farms in data centers, power consumption is rapidly becoming the key design issue. Based on the observation that the system, while designed for a peak load, is rarely fully utilized, power management schemes can successfully trade performance for power without causing the software to miss its deadlines.

Current power-efficient systems have management functions that can be invoked to turn off system components, or to choose among different power states for each component. Dynamic voltage scaling (DVS) is a technique that allows performance-setting algorithms to dynamically adjust the performance level of the processor. An increasing number of processors implement DVS, which can yield quadratic energy savings for systems in which the dominant power consumer is the processing unit [14, 24].

In the presence of real-time constraints, DVS performance-setting algorithms attempt to lower the operating frequency while still meeting request deadlines. For real-time systems with predictable (or known) workloads, DVS algorithms can fully explore power-performance tradeoffs. Unfortunately, in most real-life situations, there is no such a-priori knowledge. The workload is rather unpredictable in a variety of systems, from simple cell phones and PDA devices to more complex personal computers, servers and systems-on-a-chip [16, 21].

The example that prompted our investigation comes from our industrial research partners, dealing with satellite-based signal processing. Signal data collected through an external sensor is disseminated to several processing units for further analysis. A signal processing application is responsible for timely analysis of the signal data (also referred to as requests, or events). Currently we are investigating two such applications, known as *Event Extraction* and *CAF*, each provided with several realistic traces. For example, the Complex Ambiguity Function (or *CAF* for short) is an application that collects data in low orbiting satellites, correlates it with data collected from geo-stationary satellites, for object recognition. This object may be on Earth's surface or may be flying. The CAF application can determine an object's location with an accuracy from 4 to 7 significant digits (corresponding to 1K to 16K data point correlation, respectively).

In general, the average load is far less than the peak load that the system can handle, which brings about the opportunity for DVS algorithms to conserve energy. Thus, the major goal of the power-management policy is to keep up with the rate of request arrivals, while minimizing the energy consumption. In addition, requests are expected not to exceed a maximum response time (soft deadline).

This work evaluates DVS algorithms for systems with such unpredictable workloads. We first present related work

*Supported in part by DARPA PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736 and through NSF-ITR medium grant, award number 0121658.

Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems (ECRTS'04)
1068-3070/04 $20.00 © 2004 IEEE
IEEE COMPUTER SOCIETY

in Section 2. The system under consideration is described in Section 3. Prediction-based and stochastic DVS schemes are presented in Section 4. The schemes are compared in Section 5, using both synthetic and real-life traces. We conclude the paper in Section 6.

2 Related Work

Dynamic voltage-scaling (DVS), which involves dynamically adjusting the voltage and frequency of the CPU, has become a major research area. Quadratic energy savings [14, 24] can be achieved at the expense of just linear performance loss. For real-time systems, DVS schemes focus on minimizing energy consumption in the system while still meeting the deadlines. The seminal work by Yao et al. [24] provided a static off-line scheduling algorithm and a number of on-line algorithms with good competitive performance, for aperiodic tasks and continuous power functions. Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of off-line periodic requests are proposed in [13]. Non-preemptive power aware scheduling is investigated in [12]. All the above works assume and schedule for worst case task execution times (WCET). Automatic DVS for Linux with distinction between background and interactive jobs was presented in [9]. The goal in [9] is to minimize the energy consumption while maintaining peak system performance (thus, task deadlines are not considered).

Real-time applications exhibit a large variation in actual execution times [8] and WCET is too pessimistic. Thus, a lot of research was directed at dynamic slack-management techniques [1, 11, 18, 22]. If a task's computational requirement is known, it was shown that one can safely commit to a constant optimal CPU speed during the execution of a task without increasing the energy consumption [1] assuming continuous speeds. If a CPU only offers a fixed, limited set of valid speeds, using the two speeds which are immediate neighbors to the optimal speed will minimize the energy consumption [15]. When the task's computational requirement is known probabilistically, there is no constant optimal speed and the expected energy consumption is minimized by gradually increasing the speed as the task progresses [17]. However, the optimal speed schedule computed in [17] assumes continuous speeds. Three solutions have been proposed for the case where the CPU only offers a limited set of valid speeds, namely, rounding to the closest discrete speed [17], rounding to the next higher available speed [25] and converting to the combinations of two immediate available speeds [11]. All the above solutions are intended for deterministic arrival times (such as periodic). In this work we are investigating non-deterministic workloads.

The area of online prediction and distribution estimation is closely related to our work. For rate-based systems, estimating arrival rates and request processing requirements are two important issues. Chandra et al. [4] applied a first-order autoregressive process to predict arrival rates and a histogram method to estimate the service demand. They observed that using service demand values from the recent past does not seem to be indicative of the demands of future requests. Generally there are two ways of estimating the distribution from a sample: parametric and nonparametric [6]. Nonparametric methods are more suitable for unpredictable workloads than parametric methods. Govil et al. [10] did a comparative study of several predictive algorithms for dynamic speed-setting of low-power CPUs and concluded that simple algorithms based on rational smoothing rather than "smart" predicting may be most effective.

Power management has traditionally focused on portable and handheld devices. IBM Research broke with tradition and presented a case for managing power consumption in web servers [3]. Elnozahy et al. evaluated five policies which employ various combinations of DVS and node vary-on/vary-off for cluster-wide power management in server farms [7]. Sharma et al. [20] investigates adaptive algorithms for dynamic voltage scaling in QoS-enabled web servers to minimize energy consumption subject to service delay constraints. Aydin et al. [2] attempted to incorporate variable voltage scheduling of periodic task sets to partitioned multiprocessor real-time systems.

3 System Model

Requests are processed on a DVS processor, with M discrete operating frequencies, $f_1, f_2, \ldots, f_M$. We assume the average power consumption of the system is known at each operating frequency. We denote the average power consumption at frequency f_i by P_i. Using IBM's Mambo cycle-accurate and power-accurate simulator [19], we observed very little variation in power when running applications at a constant frequency. Thus, describing energy consumption using average power values results in a precise estimation. The system power when idle is denoted by P_{idle}.

Requests are generated externally and buffered in the system memory for further processing. Once arrived, a request must finish processing by its deadline D (different requests may have different deadlines). Deadlines are soft, meaning that occasional deadline misses, while undesired, do not result in system failure. To reduce the number of deadlines missed, a good DVS policy typically chooses the maximum speed whenever it is possible that a request may miss its deadline (Section 4.2).

Arrival times and computation times for requests are not known beforehand. However, in the applications we use, in addition to the deadline, the system can determine, immediately upon request arrival, a request *type*. For example, the

Event Extraction application can determine solely from processing the header of a request if its computation is expected to be relatively short, or if there is a possibility that it may take much longer (thus, two types). For the *CAF* application there are three types: the first time an event occurs, the second time and all times after that (typically resulting in long, short to medium, and relatively short events, respectively). Request types are the sort of semantic information that helps improving the predictions about the workload. In general, types can be associated with the application processing the request. Moreover, requests of the same application may have several types, as in the case of *Event Extraction* and *CAF*.

Requests are scheduled on the processor in a first-come first-served (FCFS) fashion, without preemption, as subsequent requests may depend on results from previous requests. The DVS algorithm is invoked in one of two situations: when a timeout expires, or when an event specifically requests a speed change. The timeout mechanism can specify exactly at which moment in time to recompute the DVS schedule. Since we consider a non-preemptive system, only two events may trigger a speed change: the arrival of a new request, or the completion of the current request. The goal of the DVS algorithm is to select the minimum speed (i.e., minimum energy) that will not cause requests to miss their deadlines.

4 Performance-setting DVS Policies

This section describes the DVS algorithms under consideration in this work, with a brief discussion on their main advantages and drawbacks. The algorithms are divided in two categories: prediction-based and stochastic. Prediction-based algorithms adjust the performance based on workload estimations, which, in turn, are based on the execution history. Stochastic algorithms rely on statistical information about the workload (collected a-priori or on-line) in an attempt to minimize the expected energy consumption. The schemes are evaluated in Section 5.

4.1 Prediction-based Schemes

The schemes described in this section adjust the CPU speed based on predicted resource requirements. Thus, the success of such schemes depends on how accurately the future workload can be predicted. We classify the schemes in three broad categories: application-oblivious (AO), application-aware (AA) and reinforcement learning (RL) schemes.

4.1.1 Application-oblivious prediction (AO)

The simplest form of prediction we are considering is one that only monitors system utilization, unaware of the application running on the system. Typically, the monitored resource is the CPU itself. CPU utilization (monitored periodically and defined as busy time over a certain interval) is a great indicator of past resource usage. The CPU speed for the immediate future is then increased or reduced based on utilization trends. Without a complex prediction scheme, the underlying assumption is that the immediate future resembles the immediate past.

Our application-oblivious prediction scheme works as follows: the system utilization is monitored periodically, every p time units. If the system was fully utilized during the last monitored period, the speed is increased to the next higher discrete frequency. If the utilization u is less than 100%, the CPU speed s is updated as $s = \lceil s \cdot u \rceil_f$, where $\lceil x \rceil_f$ is the function that returns the smallest discrete frequency higher than x.

We experimented with many other variations on this scheme, such as using utilization thresholds to determine when to increase or reduce the CPU speed. An example of such a scheme for saving energy in web servers is mentioned in [3], where utilization was monitored every 20 milliseconds. Whenever the CPU utilization exceeds 95%, the CPU speed is increased to the next higher level. Similarly, when utilization falls below 80%, the speed is decreased one level. Another example of the utilization policy is Transmeta's firmware implementation (LongRun) [5]. CPU utilization is frequently monitored, resulting in performance speed-up/slow-down by one performance level.

From our experience, the main problem with the utilization-based policies is that they use a fixed monitoring period for the system lifetime. For workloads with large variations, this results in the system being either too aggressive or too slow to react. When the monitoring period is too small, the system may set the speed higher than necessary. If the utilization is monitored too infrequently, requests may have large response times. Whenever the monitoring period does not correspond to the workload, utilization-based schemes may not be efficient. Accordingly, a software DVS algorithm was shown to achieve 11%-35% more performance reduction over LongRun [9]. Another disadvantage is that request deadlines are not considered. On the other hand, the main advantage is simplicity. The policy does not require anything more than a timeout mechanism and a way of monitoring the system utilization. Even better, some CPUs already provide this functionality.

As a final remark, note that utilization need not refer only to the CPU. Other resources (hardware or conceptual) can be monitored. For example, we experimented with monitoring the number of requests buffered for processing in memory. An increasing/decreasing number of waiting requests indicates that there may be a need for higher/lower speeds. Monitoring these other resources allows optimization of the system based on those resources. Another example is when

Table 1: Request execution times (in millions of cycles)

	Event Extraction		CAF		
	Type 1	Type 2	Type 1	Type 2	Type 3
Min	2.9	2.0	8.2	4.1	1.3
Max	82.6	753.6	5045	210.2	32.9
Avg	9.7	123.2	820.2	45.0	5.8
Stdev	7.2	153.8	1251	78.5	6.2
%	79%	21%	5.4%	2.9%	91.7%

Table 2: Request inter-arrival times (in seconds)

	Event Extraction		CAF
	81 min	1030 sec	1800 sec
Min	0.13	0.1	0
Max	6.7	11	5
Avg	0.37	0.44	0.7
Stdev	0.62	0.77	1.74
events	13045	2307	2564

memory banks can be turned off: we would like to limit the number of requests that can fit (most of the time) in a single memory bank, and so monitoring memory usage may be useful.

4.1.2 Application-aware prediction (AA)

Rather than simply reacting to observed resource requirements, AA schemes attempt to predict future performance needs by monitoring request inter-arrivals and processing requirements. As with utilization-based schemes, the CPU speed is adjusted periodically, with a pre-specified period (or timeout) p. Every p time units, the number of requests (of each type) arriving in the next period is predicted based on recent such information. The average execution time of each request is similarly predicted. We also studied schemes that adjust the speed at irregular intervals corresponding to request completion times and to the arrival of a certain number of requests in the system. From our experience, a periodic adjustment scheme results in more energy savings.

Let N_i denote the predicted number of requests of type i arriving in the next period, each of which has average execution time (at maximum speed) a_i. Further, let L_i denote the number of requests of type i unfinished from the previous period. $\sum_i (L_i + N_i)a_i$ represents the predicted amount of work for the next period p. The speed s is recomputed as $s = \lceil \frac{\sum_i (L_i+N_i)a_i}{p} f_M \rceil_f$, where $\lceil x \rceil_f$ is as defined in Section 4.1.1.

L_i is known and N_i is predicted as the number of requests in the last period. a_i is predicted in one of three ways: the average for the whole trace (from offline profiling), the average of the last period, or a combination of the two (such as exponential decay). Tables 1 and 2 present statistical information about request execution and inter-arrival times for *Event Extraction* and *CAF* traces. As it turns out, the request execution times, as well as request inter-arrivals, are rather unpredictable, with large deviations from averages. Still, the above formula guarantees that the system will keep up with the rate, as it attempts to complete all waiting requests in the next period. However, since these schemes ignore the deadlines, the penalty incurred in missing deadlines may be very large. As expected, considering semantic information (i.e., request types) generally improves the quality of the prediction.

The efficiency of prediction-based DVS schemes greatly depends on the prediction accuracy. For unpredictable workloads, the schemes may be too aggressive or too slow to react, depending on the period p. As with utilization-based schemes, no request deadlines are considered. Moreover, implementation of the prediction policy is required.

4.1.3 Reinforcement learning schemes (RL)

RL schemes take a different approach to workload predictions. Rather than using specific formulas, they rely on more complex machine learning techniques to directly learn the DVS policy. The state of the system is first described by several parameters. The choice of such parameters is entirely up to the designer. However, note that the number of states exponentially increases with the number of parameters. For our system, we experimented with describing the system state by parameters such as: the current CPU speed, the number of requests waiting and average delays.

After dividing the system into states, an action is associated with each state. For our system, the action is the CPU speed adjustment. The goal of any machine learning scheme is to determine the policy (i.e., the mapping from states to actions) that results in minimizing or maximizing some metric that describes the system goal (in our case, minimize energy). Thus, a learning scheme has a third component, namely, an evaluation function that can be used to compare actions. The learning process works as follows: in a state never seen before a random action is taken. Later, the system can evaluate the consequences of the action. Next time the system finds itself in that state, it will hopefully avoid the incorrect actions.

Largely inspired from the work in [23] (QoS selection for each frame in a MPEG movie), we experimented with states described by the current CPU speed and the trace *progress*, defined in terms of average delays or number of requests queued. The QoS-setting in [23] is equivalent to our speed-setting goal, with the main difference that we can adjust the speed several times for a request, whereas the QoS level in [23] can be selected only once for each frame. However, while many MPEG traces have known arrival times and sim-

ilar execution times for frames of certain types, our traces are far more unpredictable, as shown in Tables 1 and 2. So far, our attempts to implement a RL scheme resulted in inconclusive results and thus we do not consider RL in the rest of this paper.

4.2 Stochastic DVS algorithms (S)

While still collecting statistical information about the workload (a-priori or online), these schemes differ from prediction-based schemes in the fact that they do not attempt to predict request processing requirements and inter-arrival times. The data collected is the probability distribution of request CPU cycles. Requests are classified based on their number of cycles, with any desired granularity. For example, with a granularity S (expressed in cycles), class C_0 would contain all requests whose cycles are up to S, class C_1 contains all requests with cycles in the range $(S, 2S]$ and class C_i represents requests with cycles in the range $(iS, (i+1)S]$. Counting the number of requests belonging to each class results in a histogram with $B = \lceil \frac{WC}{S} \rceil$ bins, where WC denotes the worst-case number of cycles of a request. We store the histogram as an array H of size B, where $H[i]$ denotes the number of request in class (bin, or position) i.

The probability distribution can be obtained from profiling or through on-line monitoring, as follows: every time a request finishes processing, its exact number of cycles e is known. To include the request in the histogram, its corresponding class count is updated as follows: $H[\lceil e/S \rceil] = H[\lceil e/S \rceil] + 1$. Of course, separate histograms can be maintained for each request type.

The cumulative density of probability function *CDF* associated with a histogram is defined as $CDF[k] = \frac{\sum_{i=1}^{k} H[i]}{\sum_{i=1}^{B} H[i]}$ (i.e., the probability that a request requires less than kS cycles, or the probability that a request belongs to one of the first k bins). When a new request enters the system, the function $1 - CDF[k]$ represents the probability that bin k will be executed (i.e., the request will probably execute for at least kS cycles).

We start by showing how to use DVS to minimize the expected energy consumption of a *single* request with a known histogram and deadline. It was previously shown that an optimal DVS schedule would gradually increase the speed [17, 11, 25]. While the work in [17] is only intended for continuous speeds, an exact solution for specific power functions is proposed in [11] and [25]. Since we are interested in systems with more general power functions (e.g., those that include other components beside the CPU), we are proposing a simple, novel DVS scheme that selects just two speeds among M possible speeds for each request: a primary and a secondary speed.

Using the notation introduced in Section 3, the scheme chooses a primary speed f_i and a secondary speed f_j so that to minimize the expected energy consumption. If WC is the worst-case execution time and D is the request deadline, and $WC/D \geq f_M$ (i.e., the worst-case number of cycles cannot be satisfied within the deadline even at maximum speed), the scheme selects $f_i = f_j = f_M$. If $WC/D \leq f_1$, then $f_i = f_j = f_1$. Otherwise, the expected energy for all combinations of primary and secondary speeds $f_i \leq WC/D$ and $f_j \geq WC/D$ is computed as follows.

The time spent at the primary and secondary speeds, t_i and t_j, is first determined by solving the linear system described by the equations $t_i f_i + t_j f_j = WC$ and $t_i + t_j = D$, as in [15]. The first $t_i f_i$ cycles are executed at the primary speed, and the remaining cycles (up to WC) are executed at the secondary speed. The expected energy consumption of bin k is $E_k = P_i \frac{S}{f_i}(1 - CDF[k])$, if $(k+1)S \leq t_i f_i$ or $E_k = P_j \frac{S}{f_j}(1 - CDF[k])$, if $(k+1)S > t_i f_i$. The total expected energy consumption is $\sum_{k=1}^{B} E_k$. Depending on the probability distribution of request processing requirements, the primary and secondary speeds can differ by more than one discrete level. Because we consider probabilities, our solution is different from [15] and subsequent work, which picks adjacent speed levels for f_i and f_j. This is precisely the intuition behind the stochastic approach: if most requests are short enough to finish execution at the primary speed, a larger gap between the primary and secondary speeds will result in more savings compared to the adjacent speeds. The difference is larger for distributions where the worst-case is much higher than the average case, such as bimodal distributions. After computing all combinations of primary and secondary speeds (at most $M^2/4$, or four combinations for $M = 4$ discrete speeds), the one with the smallest expected energy consumption is selected as the final DVS schedule.

A straightforward implementation of the above DVS algorithm has complexity $O(BM^2)$. From our experience, $B = 100$ bins results in a good enough representation of the histogram. Combined with a small number of discrete speeds M, this leads to a very efficient algorithm. Note that the complexity can be improved to $O(B + M^2)$ if using $O(B)$ extra space. Furthermore, if the histogram is collected offline (i.e., no need to update the *CDF*), the complexity becomes just $O(M^2)$. This low complexity can be accomplished by storing a cumulative *CDF*, defined as $CCDF[k] = \sum_{i=1}^{k} CDF[i]$, which requires $O(B)$ extra space. Using the *CCDF*, the summation in $\sum_{k=1}^{B} E_k$ can be transformed into a $O(1)$ computation.

The description above considered only a *single* request. With *multiple* requests dynamically entering the system, we extend our algorithm as follows: aware of all waiting requests and their deadlines, the latest completion time of the

Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems (ECRTS'04)
1068-3070/04 $20.00 © 2004 **IEEE**
IEEE COMPUTER SOCIETY

Table 3: PPC405LP speed settings and voltages

Speed (MHz)	33	100	266	333
Voltage (V)	1.0	1.0	1.8	1.9
Power (mW)	19	72	600	750

first request is first computed, so that no request can miss its deadline, even in worst-case scenarios (i.e., all requests take their worst-case number of cycles). That is, with new requests arriving with their own deadlines, the DVS algorithm may have to consider an artificially-reduced deadline (to raise the speed) of the current request, to allow enough time at maximum speed for the waiting requests to meet their own deadlines. Note that this does not mean that all subsequent requests will run at maximum speed, as in practice average processing requirements will be less than their worst-case. Whenever a request finishes execution, the slack created can immediately be used by the next request. The DVS schedule is then computed as described in the single request system.

The complexity of computing the latest completion time of requests is $O(N)$, where N is the number of requests queued for execution. The artificially-reduced deadlines are computed assuming maximum speed and worst-case execution times, in reverse order of the queued requests, to ensure that all deadlines are met. Note that the computed latest completion time may be less than the request deadline, to accommodate for processing requirements of subsequent requests.

Speed change overheads are considered when recomputing the DVS schedule. Also, whenever a new request comes, a better estimation is obtained for the current executing request by considering only the remaining cycles when calculating the expected energy. For each request, the number of speed changes is at most M. As experimental results in the next section will show, the number of speed changes is much less in practice.

5 Experimental Results

Table 3 shows the power model used in the experiments, indicating the discrete speeds, corresponding voltages and average power consumption. The model is based on actual power measurements on IBM's PPC405LP embedded processor. The number of cycles for each request in the *Event Extraction* and *CAF* traces was obtained from the cycle accurate Mambo simulator [19].

To evaluate the schemes described in Section 4, we assume the period for monitoring the system utilization (Section 4.1.1) and request inter-arrival and processing times (Section 4.1.2) is $p = 1$ second. We choose this period because it results in good response times for our traces, without a negative impact on the energy consumption. Increasing the period generally results in higher response times, without a significant effect on the energy consumption. Reducing p slightly increases the energy consumption, without a significant effect on the already low response times.

The application-oblivious scheme (AO) updates the speed as described in Section 4.1.1. We also experimented with the threshold schemes, including the one used in [3]. Such schemes generally resulted in similar response times for requests, but significantly higher energy consumption. For the application-aware scheme (AA), the predicted number of requests (of each type) arriving in the next period is the number observed in the most recent period p. The predicted running times for each request type is the static average for the whole trace; very similar results were obtained for a dynamic average prediction (that is, the online average from the beginning of the trace until the current time). Recall that we do not evaluate RL schemes (Section 4.1.3) in this work.

The stochastic scheme (S) does not need a monitoring period, as it makes no prediction about the workload. Instead, a soft deadline is used as the maximum allowed response time for requests. Based on the embedded applications we are dealing with, we are considering the same deadline for all requests (note that the stochastic scheme can handle different deadlines for each request with no extra overhead). Since the worst-case execution time (at maximum speed) is 2.3 seconds for *Event Extraction* and 15 seconds for *CAF*, we show results for deadlines equal to approximately twice and four times the worst-case. The histogram for each request type was obtained statically. We expect very similar results if the histogram is collected dynamically (as described in Section 4.2). However, in the on-line version, the histogram will only be accurate after a sufficient number of requests (such as the first 100 seconds of the trace) finished execution. We used a fixed bin width of 10 million cycles, resulting in 76 bins for *Event Extraction* and 505 bins for *CAF* (worst case execution time is 753 million cycles for *Event Extraction* and 5045 million cycles for *CAF*, see Table 1). The corresponding space overhead for the histograms is 304 bytes for *Event Extraction* (2 types, 76 bins, 2 bytes for each bin) and 3030 bytes for *CAF* (3 types, 505 bins, 2 bytes per bin).

In all the schemes (except no-power-management), the system immediately switches to minimum speed when idle (i.e., $P_{idle} = P_1$). A time overhead of 1 millisecond is added for each speed change. In practice, CPUs that can adjust the voltage internally have a lower overhead (in the range of microseconds), but systems that require changing the voltage externally experience however overheads in the milliseconds range. We compute the energy overhead of a speed change as the system power at maximum speed times the time overhead. We note that both the time and energy

Table 4: DVS policies evaluation: (noPM) no-power-management (AO) application-oblivious prediction (AA) application-aware prediction and (S) stochastic with specified deadline, for the *Event Extraction* and *CAF* traces. SC means speed changes.

policy	Savings (per scale factors)				SC	Avg - Max
	1	2	4	0.8	/s	delay
81 min (EE)						
noPM	1x	1x	1x	1x	0	0.43 - 5.22
AA	3.8x	7.5x	13.2x	3x	1.27	0.72 - 5.43
AO	4x	8.4x	16x	3.1x	1.77	0.86 - 5.45
S 5s	4.4x	10.5x	21.2x	3.4x	1.05	1.42 - 5.88
S 10s	4.6x	11.7x	26.7x	3.5x	1.08	2.78 - 9.63
1030 sec (EE)						
noPM	1x	1x	1x	1x	0	0.36 - 2.51
AA	4.4x	8.5x	14.5	3.5x	1.1	0.67 - 3.00
AO	4.6x	9.7x	17.5x	3.7x	1.46	0.90 - 4.31
S 5s	5.4x	12.5x	23.1x	4.1x	0.95	1.4 - 4.5
S 10s	5.9x	14.8x	28.5x	4.3x	0.91	2.88 - 9.01
1800 sec (CAF)						
noPM	1x	1x	1x	1x	0	1.54 - 29.90
AA	4.3x	7.7x	12.9x	3.5x	0.6	1.58 - 29.89
AO	4.7x	8.5x	14.2x	3.8x	0.39	2.37 - 31.68
S 30s	4.9x	9.6x	18.2x	3.9x	0.11	3.06 - 34.41
S 60s	5.2x	11.6x	23.4x	4x	0.07	5.65 - 42.06

overheads do not have a significant effect for our traces, due to infrequent speed changes. The overhead of the policies themselves is in the microseconds range for each speed computation (at most two speed computations per request are necessary for the stochastic scheme).

Table 4 evaluates the schemes on the *Event Extraction* and *CAF* traces described in Tables 1 and 2. To vary the system load for the same trace, the original inter-arrival times between requests are multiplied with a *scale factor*, as in [7]. We considered the following values for the scale factor: 0.8, 1, 2, and 4, where 1 is the original trace. Reducing the inter-arrival below 0.8 results in a overloaded system that cannot keep up with the rate even if running at the maximum speed at all times.

The savings, shown in columns 2-5 of Table 4 for each scale factor, are normalized to the no-power-management scheme. The stochastic approach results in the most energy savings, up to 28.5x compared to no-power-management and up to 40% less energy compared to the second-best DVS scheme (the application-oblivious prediction). The stochastic scheme also results in the fewest speed changes per second (SC/s) among the DVS policies (see column 6, Table 4). This is because many requests do not reach the point where they switch to the secondary speed. Also, when the system is overloaded or under-utilized, the primary and secondary speeds are identical (i.e., f_1 for a under-utilized system and f_M for an overloaded system).

In most experiments there were no deadline misses (maximum and average delays are shown in column 7 of Table 4). We also experimented with tighter deadlines and noted that the maximum delay of the stochastic scheme closely matches the specified deadlines (unless the workloads is so high that the deadlines cannot be met). The stochastic scheme also resulted in considerably fewer deadline misses than the other policies, while still achieving energy savings over the other DVS policies (although less significant savings, depending on the tightness of the deadline).

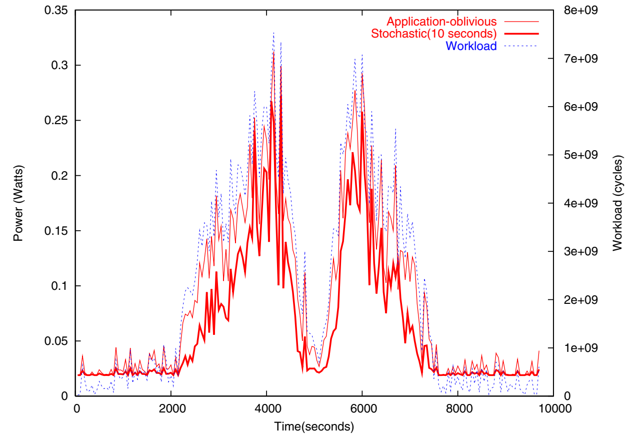

Figure 1: Power vs workload for the stochastic and application-oblivious DVS policies on the *Event Extraction* 81 minutes trace

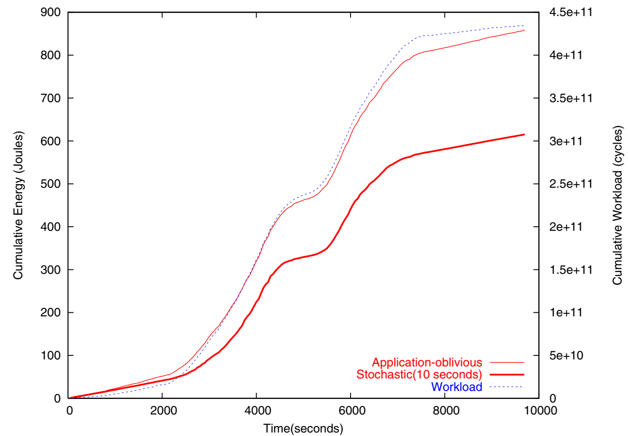

Figure 2: Energy consumed for the *Event Extraction* 81 minutes trace

As seen in Figure 1 for the 81 minutes *Event Extraction* trace, the system power closely matches the workload (processing cycles arriving every second), with lower power consumption for the stochastic policy. To see more clearly how stochastic schemes do better over time, in Figures 2

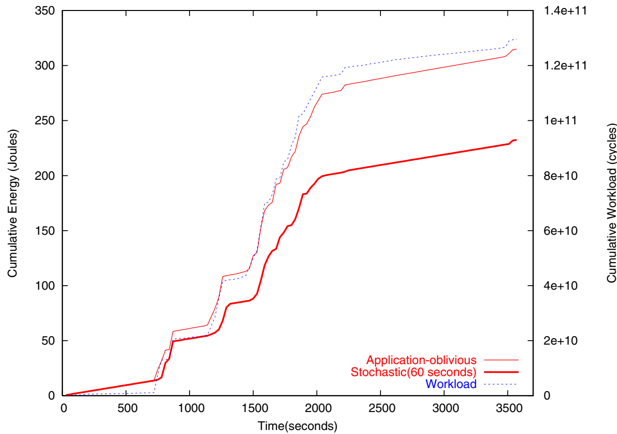

Figure 3: Energy consumed for the *CAF* 1800 seconds trace

and 3 we plot the cumulative energy consumption under the stochastic and application-oblivious policies for the 81 minutes *Event Extraction* and 1800 seconds *CAF* traces (for a scale factor of 2). The cumulative workload is also shown for comparison. We do not show the arrival rate of requests, since it has little correspondence with the workload.

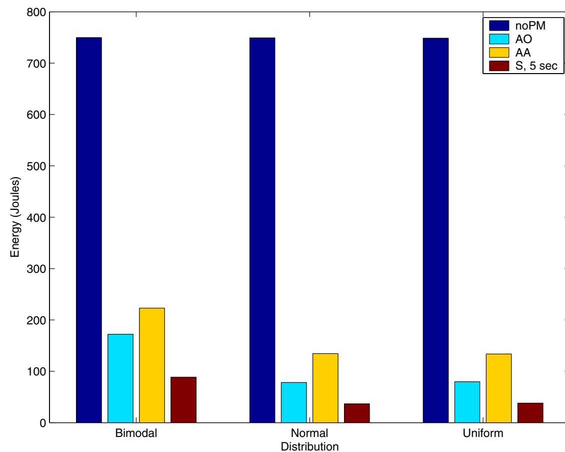

Figure 4: Comparison of the DVS policies using synthetic traces

Finally, we evaluate the DVS policies on a set of synthetic traces. For the request cycles we artificially generate three typical distributions, namely, bimodal, normal (Gaussian) and uniform distributions, each with a minimum of 5 million cycles and a worst-case of 200 million cycles (20 bins). We use a uniform random number generator to generate the arrival rate for each second, which results in a unpredictable workload. The deadline (maximum response time) for the stochastic scheme is 5 seconds. For each experiment, the average system load is around 30% of the maximum load that the system can handle. Figure 4 shows the average power consumption of the DVS policies (1000 experiments were averaged for each distribution). The stochastic scheme achieves up to 20x savings compared to no-power-management and, on average, 50% more savings compared to the second-best scheme (AO).

6 Conclusions and Future Work

We evaluated several DVS policies for power management in systems with unpredictable workloads. A stochastic DVS scheme was proposed that uses a simple, yet effective algorithm to obtain the DVS schedule, based on collected statistical information about the workload.

The scheme has little overhead (in the microseconds range for each request) and was shown to outperform prediction-based algorithms, ranging from simple application-oblivious prediction to more complicated application-aware prediction. Our experimental results show that the stochastic scheme achieves one order of magnitude energy reduction over no-power-management and up to 50% more savings compared with the best prediction-based scheme. In general, our observation is that prediction-based policies are too aggressive, with lower delays and higher energy consumption. Aware of deadlines, the stochastic policy can better explore power-performance tradeoffs.

We are currently studying the interplay of DVS policies with on-off schemes for multiprocessor systems. As part of this work, we are also investigating request distribution policies and load balancing mechanisms.

References

[1] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of the 22nd Real-Time Systems Symposium (RTSS'01)*, 2001.

[2] Hakan Aydin and Qi Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *International Parallel and Distributed Processing Symposium (IPDPS'03)*, pages 113–121, March 2003.

[3] P. Bohrer, E. Elnozahy, M. Kistler, C. Lefurgy, C. McDowell, and R. R. Mony. The Case for Power Management in Web Servers. In *Power Aware Computing, Kluwer Academic Publications*, 2002.

[4] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. An Online Optimization-based Technique For

Dynamic Resource Allocation in GPS Servers. Technical Report TR02-30, Department of Computer Science, University of Massachusetts Amherst, 2002.

[5] Transmeta Corporation. LongRun Technology. http://www.transmeta.com/crusoe/longrun.html.

[6] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification. Second Edition*. John Wiley and Sons, New York, 2001.

[7] E.N. Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-Efficient Server Cluters. In *Workshop on Power-Aware Computer Systems (PACS'02)*, 2002.

[8] R. Ernst and W. Ye. Embedded Program Timing Analysis based on Path Clustering and Architecture Classification. In *Computer-Aided Design (ICCAD'97)*, pages 598–604, 1997.

[9] K. Flautner and T. Mudge. Vertigo: Automatic Performance-Setting for Linux. In *Proceeding of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[10] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[11] Flavius Gruian. Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors. In *International Symposium on Low Power Electronics and Design (ISLPED'01)*, Aug. 2001.

[12] I. Hong, D. Kirovski, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable Voltage Core-based Systems. In *Design Automation Conference (DAC'98)*, 1998.

[13] I. Hong, M. Potkonjak, and M. B. Srivastava. On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. In *Computer-Aided Design (ICCAD'98)*, pages 653–656, 1998.

[14] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of the 19th IEEE Real-Time systems Symposium (RTSS'98)*, Madrid, Spain, December 1998.

[15] Tohru Ishihara and Hiroto Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *International Symposium on Low Power Electronics and Design (ISLPED'98)*, pages 197–202, August 1998.

[16] Raimondas Lencevicius and Alexander Ran. Can Fixed Priority Scheduling Work in Practice? In *Proceedings of the 24th IEEE Real-Time systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[17] Jacob Lorch and Alan Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *ACM SIGMETRICS*, June 2001.

[18] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler Assisted Dynamic Power-Aware Scheduling for Real-Time Applications. In *Workshop on Compilers and Operating Systems for Low Power (COLP'00)*, Philadelphia, PA, October 2000.

[19] Hazim Shafi, Patrick Bohrer, James Phelan, Cosmin Rusu, and James L. Peterson. The Design and Validation of a System Performance and Power Simulator. *IBM Journal of Research and Development*, 47(5/6), 2003.

[20] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Liu. Power-aware QoS Management in Web Servers. In *Proceedings of the 24th IEEE Real-Time systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[21] David C. Sharp, Edward Pla, and Kenn R. Luecke. Evaluating Mission Critical Large-Scale Embedded System Performance In Real-Time Java. In *Proceedings of the 24th IEEE Real-Time systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[22] D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. In *IEEE Design and Test of Computers*, pages 18(23):20–30, March 2001.

[23] Clemens C. Wust, Liesbeth Steffens, Reinder J. Bril, and Wim F. J. Verhaegh. QoS Control Strategies for High-Quality Video Processing. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, Catania, Italy, July 2004.

[24] F. Yao, A. Demers, and S.Shankar. A Scheduling Model for Reduced CPU Energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.

[25] Wanghong Yuan and Klara Nahrstedt. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, Bolton Landing, NY, October 2003.

Proceedings of the 12th 16th Euromicro Conference on Real-Time Systems (ECRTS'04)
1068-3070/04 $20.00 © 2004 **IEEE**

IEEE COMPUTER SOCIETY