Adaptive Probability Based Power Management Strategies

Sandra Irani
Sandeep K. Shukla
Rajesh K. Gupta

ICS

TECHNICAL REPORT

Technical Report # 01-58
(December 18, 2001)

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425

Information and Computer Science

University of California, Irvine

Adaptive Probability-Based Power Management Strategies (Tech Report# UCI-ICS-01-58)

Sandra Irani Sandeep K. Shukla Rajesh K. Gupta

Department of Information and Computer Science,
University of California at Irvine,
Irvine, CA 92697
E-mail: {irani,skshukla,rgupta}@ics.uci.edu

Contents

List of Figures

List of Tables

Abstract

Dynamic Power Management (DPM) is an important technique to reduce power consumption in embedded and portable systems. Power hungry devices such as disk drives, network interfaces, and other peripheral devices are often designed with multiple power saving states, and control knobs are provided for changing their power states under the operating system control. DPM strategies are "online" strategies since they must make decisions about the timing of transitioning to lower power consumption states during idle periods without knowing when the next request for service will arrive. In this paper, we present a novel approach to designing adaptive online DPM strategies. This approach dynamically learns the probability distribution of idle period lengths from recent request patterns. A probability-based scheme then uses this information to optimize power saving actions. We present experimental results comparing our strategy with various strategies in the literature, including our previous work. Our study includes measuring power usage as well as the additional latency introduced from the delay in powering back up when a new request for service arrives. Our strategy exhibits the lowest power consumption among all the online algorithms. The other algorithms which come close to matching its performance in power all suffer at least an additional 40% latency on average. Meanwhile, the algorithms which have comparable latency to our method all use at least 25% more power on average. Thus, our probability-based DPM strategy is the most successful algorithm in balancing power usage as well as latency incurred.

1 Introduction

The technical focus of the work presented here is on strategies that can be employed by the operating system, networking software and to some extent application software in effectively managing power usage. Due to constraints on power budgets of electronic systems, Dynamic Power Management (DPM) is increasingly gaining importance, as evidenced in the research literature [7, 23, 4, 3, 21, 19, 5, 20], as well as concerted industry efforts such as Microsoft's OnNow [13] and ACPI [8]. Significant work has been done in the area of DPM in search of strategies that yield the most reduction in power/energy with the least amount of runtime computational effort. These include heuristic shutdown policies [18], prediction based shutdown policies [7, 23], multiple voltage scaling [4], and stochastic modeling based policy optimization [16, 17]. However, there is a lack of formal methodology to evaluate the performance of such strategies and to guide the design of new strategies ones. Empirical analysis has been done extensively in [11], to compare various DPM strategies in an experimental framework. Although, this work gave an experimental foundation for quantitative comparison of various DPM strategies, there still is no theoretical foundation for designing DPM strategies. Most strategies are designed using experimental intuition and are then experimentally evaluated to be effective. In our work, we are trying to create a foundation on which to build strategies in a more systematic manner. As discussed in [3], the design and analysis of power management policy optimization techniques are still wide open fields of research. Our framework for analysis is that of Competitive On-line Computation [1] and most of our designs are based on examining the problem in the light of competitive analysis.

1.1 Dynamic Power Management(DPM) and Competitive Analysis

System Level Dynamic Power management is an "*online*" problem in the sense that critical decisions must be made before the entire input sequence is available. For the DPM problems we consider, the input to the strategy is the length of an upcoming idle period for a device and the decision to be made is whether to transition to a lower power dissipation state while the system is idle. For instance, the intervals between data transfer requests to a radio subsystem in a packet radio network interface are not known in advance. It may not be an effective power reduction strategy to shutdown power to such a device as soon as it is detected to be idle. If the idle period is too small, power up cost could surpass the energy saved by shutting down the subsystem. On the other hand, it is important to power down for long

idle periods when the device is not in use.

Typically on-line algorithms are evaluated by their competitive ratios [1] which indicate the worst case performance of the algorithm over all possible sequences of inputs. The competitive ratio is the ratio of the cost of an online algorithm to the cost of the optimal offline algorithm that has a complete knowledge of the input in advance. Competitive analysis has been a powerful tool for proving the efficiency of algorithms without having to make assumptions about the input sequence. Competitive ratios for various DPM strategies can be found in [9, 18, 19]. However, this analysis is usually very pessimistic, similar to lower bound analysis in complexity theory, due to the fact that it is based on the worst case input sequence instead of focusing on typical input sequences. For example in the implementations described in [11], the simple competitive strategy of [9] seems to out perform many other strategies in practice.

One way to address this issue is to characterize typical input sequences by a probability distribution and optimize the online algorithm for that distribution. This has the advantage that algorithms are tuned for the input sequences that are thought to be more typical of those arising in the given application. Of course, the success of this method depends on the ability to accurately describe the input sequence by a probability distribution.

There is an extensive set of research work [15, 16, 17, 5, 22] on probabilistic formulation of the DPM problem, where the power state changes are modeled as Markov/Semi-Markov processes, and the input request arrivals are modeled as well known distributions, Binomial or Poisson, and inter-arrival times are then distributed as Geometric, Exponential etc. The formulation is then turned into an optimization problem, which is then solved to obtain system parameters to tune the DPM strategy. However, it is a strong assumption that inter-arrival times are exponentially or geometrically distributed, especially when the arrivals are very much application dependent. With this concern, we use recent trace history to learn a probability distribution describing idle period lengths and optimize power management decisions based on this distribution. A very appealing feature of this approach is that it makes no assumptions as to the form of the probability distribution which generates the idle period lengths.

1.2 Single Value Predictive Strategies

Previous work on prediction based dynamic power management can be categorized into two groups: adaptive and non-adaptive. Non-adaptive strategies set a threshold on the idle time interval for transi-

tioning from the active to the sleep state. For multiple state systems, there is a sequence of thresholds each of which indicates when to transition to the next lower power consumption state. In either case, non-adaptive strategies set these thresholds once and for all and do not alter them based on observed input patterns. Adaptive strategies, on the other hand, use the history of idle periods to guide the decisions of the algorithm for future idle periods. There have been a number of adaptive strategies proposed in the literature [7, 9, 2, 5].

Many adaptive dynamic power management strategies [7, 23, 19, 9, 5, 12] use a sequence of past idle period lengths to predict the length of the next idle period. These strategies typically describe their prediction for the next idle period with a single value. Given this prediction, they transition to the power state that is optimal for this specific idle period length. In the case that their prediction is wrong, they transition to the lowest power state if the idle period extends beyond a fixed threshold value. For the sake of comparison with other approaches, we shall call these predictive DPM schemes "**single-value prediction schemes (SVP)**". In particular, Chung, Benini and De Micheli [6] address multiple idle state systems using a prediction scheme based on adaptive learning trees. Their method has an impressively high hit ratio in its prediction.

One of the chief limitations of the single-valued prediction approach is that it fails to capture uncertainty in the prediction for upcoming idle period length. For example, if a very short idle period and a very long idle period are equally likely, these methods are forced to pick a single prediction and pay a penalty in the likely event that the prediction is wrong. We address this limitation by using a probability distribution to describe the upcoming idle period. The distribution allows for a much richer prediction so that the algorithm can optimize in a way that takes the nature of this additional information into account.

The idea of modeling idle period lengths by a probability distribution leads to two distinct questions:

1. If the upcoming idle period length will be generated according to a probability distribution known to the algorithm, how can this information be used to optimize power consumption?

2. Given historical data for previous idle periods, how can we use this information to reliably construct a probability distribution describing future idles periods? Note that, this probability distribution changes dynamically depending on applications running on the system. Hence, this probability distribution needs to be learned dynamically as well.

Our previous work addresses the first of these questions [20]. An algorithm is given which determines

thresholds to transition from one state to the next given that the next idle period will be generated by a fixed, known distribution. Analytical as well as empirical results are used to demonstrate that this algorithm does very well when the idle periods follow an a priori known probability distribution. The problem of obtaining obtain this probability distribution, however, was left open.

In this paper, we develop a method to use recent history to form an estimate for a distribution governing future usage patterns for the device. This method is used in conjunction with the algorithm that uses a probability distribution to determine a power management strategy. We test the overall strategy empirically using data on file system access patterns obtained from [24]. The results are compared to the performance of a variety of other strategies suggested in the recent literature. Our results show that our method outperforms other methods in balancing power usage with latency.

Our experimental framework has Java applet interface which is available on our website [14] for users to upload their data and evaluate our algorithms and other known algorithms that we have implemented in our simulation framework.

2 System Model

We focus on strategies for a single peripheral device whose power is managed by the operating system. The device can be in one of the n power states denoted by $\{s_1, ..., s_n\}$. The power consumption for state i is denoted by α_i. The states are ordered so that $\alpha_i > \alpha_j$ as long as $i < j$. Thus, state s_1 is the *ready* state which is the highest power consumption state.

We are also given from the manufacturer's specification, the transition power p_{ij}, and transition times t_{ij}, to move from state s_i to s_j. Usually the power needed and time spent to go from a higher state to a lower state is negligible, whereas the power and time needed to transition to a higher state to lower state is high. Thus, we simplify the model by considering only the time and power necessary to power up the system. Furthermore, all of the algorithms considered in this paper have the property that they only transition to the ready state when powering up and never transition to an intermediate higher powered state. As a result, we only need the time and total energy consumed in transitioning up from each state i to the ready state. The total energy used in transitioning from state i to the ready state is denoted by β_i.

We note that in cases where the time and energy used in transitioning to lower power consumption

states is non-negligible, they can be easily incorporated by folding them into the corresponding power-up parameters. This can be done as long as the time and energy used in transitioning down is additive. That is, we require that for $i < j < k$, the cost to go from i to j and then from j to k is the same as the cost of going from i directly down to k.

The input to the DPM is a sequence of requests for service that arrive through time. With each request, we are told the time of its arrival and the length of time it will take to satisfy the request. If the device is busy when a new request arrives, it enters a queue and is served on a first-come-first-serve basis. In this case, there is no idle period and the device remains active through the time that the request is finished. This means that the number of idle periods is generally less than the number of requests serviced. Whenever a request terminates and there are no outstanding requests waiting in the system, an idle period begins. In these situations, the DPM is invoked to determine to which power consumption states the device should transition and at what times.

If the device is not busy when a new request arrives, it will immediately transition to the ready state to serve the new request if it is not already there. In the case where the device is not already in the ready state, the request can not be serviced immediately, but will have to incur some latency in waiting for the transition to complete. This delay will cause future idle periods to be shorter. In fact, if a request is delayed, some idle periods may in fact disappear. Thus, we have an interesting situation where the behavior of the algorithm effects future inputs (idle period lengths) given to the algorithm.

Another interesting phenomenon is that delaying servicing a request will tend to result in lower power usage. Consider the extreme case where the power manager remains in the deepest sleep state while it waits for all the requests to arrive and then processes them all consecutively. This extreme case is not allowed to happen in our model since we require that the strategy transition to the ready state as soon as any request appears. However, it illustrates the natural tradeoff which occurs between power consumption and latency. See [19] for a more extensive discussion of this tradeoff. Our experimental results explore this tradeoff for the set of algorithms studied.

3 Online Probability-Based DPM

The Online Porbability-Based Algorithm (OPBA) works as follows: a window size w is chosen in advance and is used throughout the execution of the algorithm. The algorithm keeps track of the last w

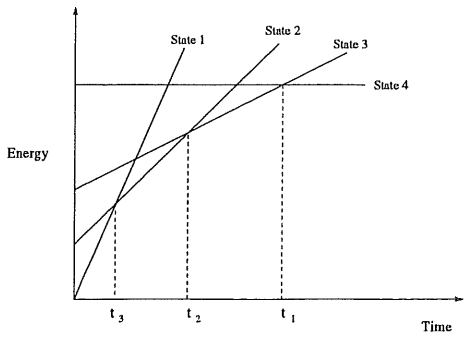

Figure 1. Energy consumption for each state for a four state system. Each state is represented by a line which indicates the energy used if an algorithm stays in that state as a function of the length of the idle period.

idle period lengths and summarizes this information in a histogram. The set of all possible idle period lengths $(0,\infty)$ is partitioned into n intervals, where n is the number of bins in the histogram. Let r_i be the left endpoint of the i^{th} interval. The i^{th} bin has a counter which indicates the number of idle periods among that last w idle periods whose length fell in the range $[r_i, r_{i+1})$. The bins are numbered from 0 through $n-1$. $r_0 = 0$ and $r_n = \infty$.

The last w idle periods are held in a queue. When a new idle period is completed, the idle period at the head of the queue is deleted from the queue. If this idle period falls in bin i, then the counter for bin i is decremented. The new idle period is added to the tail of the queue. If this idle period length falls into bin j, the counter for bin j is incremented. Thus, the histogram always includes data for the last w idle periods. Our experimental results include a study experimenting with values for w. Our results show that the algorithm is very robust under a variety of different ranges for the window size, w. (See Section 5).

Periodically, the histogram is used to generate a new power management strategy. We begin with a brief discussion about the offline power management strategy which can be found in greater detail in [20]. For each state, we can plot the total energy spent if the algorithm transitions immediately to state i, remains in state i for the duration of the interval and transitions to the ready state just in time to serve the next request. This energy expenditure is a linear function in t, the length of the idle period. See Figure 1 for a graph showing the energy expenditure functions for a system with four states.

The optimal offline algorithm which knows the length of the idle period in advance will select the state which corresponds to the line on the lower envelope of the curve. This naturally imposes an ordering on

the states: the order in which they appear on the lower envelope of the curve. Accordingly, we order the states $s_1, \ldots, s_m$, with s_1 being the highest power consumption state (the *ready* state) and the s_m being the deepest sleep state. We refer to the discontinuities of the lower envelope by $t_1, \ldots t_{m-1}$. These will also be referred to as the *offline thresholds*.

Our algorithm will transition from state to state in this order as long as the idle period continues. What remains to determine are the times (or thresholds) at which the online algorithm will transition to the next state. Since there are m states, $m-1$ thresholds must be defined. The i^{th} threshold indicates when the online algorithm will transition from state i to state $i+1$ if a new request has not yet arrived. If a new request arrives at any point in the idle period, the algorithm will immediately transition to the ready state and the idle period will end. Note that our algorithm only transitions to the ready state in response to an incoming request. Some of the other algorithms we consider in our study wake-up preemptively in anticipation of an incoming request. Although the preemptive wake-up is not helpful in power minimization, it does help reduce latency.

We now describe how the thresholds are chosen. Our algorithm uses a probability distribution (in our case, discrete histogram) to determine these thresholds. Given a two state system and a probability distribution generating the idle period length, it is known how to analytically determine the threshold which will minimize the expected cost [9]. We use this algorithm to find the threshold between each pair of consecutive states in a multi-state system. Let π be a probability distribution which generates the next idle period length. The value for the threshold τ_i is

$$\begin{aligned} & \arg\min_{\tau} \int_0^{\tau} \pi(t) t(\alpha_i - \alpha_{i-1}) dt \\ + & \int_{\tau}^{\infty} \pi(t)[\tau(\alpha_i - \alpha_{i-1}) + (\beta_{i-1} - \beta_i)] dt. \end{aligned}$$

Previous work ([20]) gives analytical and experimental results for this algorithm under the assumption that the idle period length is indeed generated by the distribution π.

This paper addresses the problem that π is unknown by using the histogram to estimate π. Recall that the histogram consists of a series of bins. Bin i covers the range from r_i to r_{i+1}. The counter for bin i is denoted by c_i. The threshold is selected among n possibilities: $r_0, \ldots, r_{n-1}$ (the lower end of each range). We estimate the distribution π by the distribution which generates an idle period of length r_i with probability c_i/w for each $i \in \{0, \ldots, i-1\}$. The sum of the counters is the window length w. Thus, the

threshold is taken to be

$$\arg\min_{r_t} \sum_{j=1}^{t-1} \left(\frac{c_j}{w}\right) r_j(\alpha_i - \alpha_{i-1})dt$$
$$+ \sum_{j=t}^{n} \left(\frac{c_j}{w}\right) [r_i(\alpha_i - \alpha_{i-1}) + (\beta_{i-1} - \beta_i)]dt.$$

A similar approach was taken in for a two state system in the context of determining virtual circuit holding time policies in IP-over-ATM Networks [10].

Naturally, we would also like to implement this algorithm as efficiently as possible. We have implemented the algorithm for finding the all $m-1$ thresholds in time $O(mn)$, where m is the number of states and n is the number of bins in the histogram. Two important factors which determine the cost (in time expenditure) of implementing our method is the frequency with which the thresholds are updated and the number of bins in the histogram. The frequency with which the thresholds are updated is the subject of one set of experiments which we perform. The results can be found in Section 5

Naturally, we would also like to minimize the number of bins used in the histogram. This must be balanced with the fact that the finer grained the histogram, the more accurate our choice of thresholds will be. One important fact that arose in implementing this strategy is that a finer grained binning is more important in some ranges than in others. A way of addressing this problem that was key to the success of the algorithm is to use the thresholds of the optimal strategy (the $t_1, \ldots, t_{m-1}$ from above) to guide the choice of bins and their ranges. We choose a constant c number of bins per state. In our case, we chose $c = 5$. The range from t_i to t_{i+1} is divided into c equal sized bins. Figure 3 shows a sample histogram from our experiments.

4 Experimental Design

4.1 Data Used in our Experiments

To demonstrate the utility of our probability-based algorithm, we use a mobile harddrive from IBM [25]. This drive has four power down states, as shown in Figure 3. Here, the start-up energy refers to the energy cost in transitioning from a state to the active state. For application disk access data, we used trace data from auspex file server archive which is available at [24]. From this data, we collected the

Bin	Range: Low End	Range: High End	Range Size	Count
1	0	11.2	11.2	35
2	11.2	22.4	11.2	2
3	22.4	33.6	11.2	4
4	33.6	44.8	11.2	7
5	44.8	**56**	11.2	4
6	56	478.8	422.8	5
7	478.8	901.6	422.8	3
8	901.6	1324.4	422.8	2
9	1324.4	1747.2	422.8	0
10	1747.2	**2170**	422.8	4
11	2170	4911	2741	7
12	4911	7652	2741	9
13	7652	10393	2741	2
14	10393	13134	2741	5
15	13134	**15875**	2741	4
16	15875	19050	3175	2
17	19050	22225	3175	3
18	22225	25400	3175	1
19	25400	28575	3175	0
20	28575	∞	∞	1

Figure 2. A snapshot of the histogram used in OPBA. Note that the offline thresholds are 56, 2179 and 15875 milliseconds. The number of bins per state is 5.

arrival times and lengths for requests for disk access for 0.4 million disk accesses divided into multiple trace files, corresponding to different hours of the day.

State	Power Consumption in Watts	Start-up Energy in Joules	Transition Time to Active
Sleep	0	4.75	5 S
Stand-by	.2	1.575	1.5 S
Idle	.9	.56	40 mS
Active	2.4	0	0

Figure 3. Values for the power dissipation and start-up energy for the IBM mobile harddrive at used in our experiments.

Figure 4 gives some data on these traces. The first column gives the number of requests. The subsequent columns give information about the behavior of the optimal algorithm when run on each trace. Specifically, they show for each state, the percentage of idle periods in which the optimal algorithm transitions to that state. In all the traces, there is a high percentage of short sequences for which the optimal strategy is to stay in the ready state (shown in column 2). The remaining percentages vary somewhat from trace to trace. All of the results reported in this paper are an average of the results on each individual trace, weighted by length.

4.2 Algorithm Test Suite

We compare OPBA to several other algorithms presented in the literature. The algorithms come in two groups. The algorithms in the first group use a series of thresholds which determine when the algorithm will transition from each state to the next lower power consumption state. OPBA and the Deterministic Competitive Algorithm, described below fall into this group. The second group are single-valued prediction algorithms. They use a single prediction for the upcoming idle period and transition immediately to the optimal state for that state. They differ only in how they select a prediction for the next idle period length.

Optimal Offline Algorithm (OPT): This algorithm is assumed to know the length of the idle period in advance. It selects the optimal power usage state for that idle period and then transitions to the

Trace Length	State 1	State 2	State 3	State 4
34575	0.804	0.158	0.011	0.024
68139	0.673	0.233	0.092	2.9E-05
36161	0.824	0.065	0.084	0.025
7250	0.727	0.045	0.057	0.169
10648	0.787	0.096	0.061	0.054
7154	0.571	0.043	0.179	0.205
72026	0.783	0.145	0.056	0.013
5130	0.643	0.155	0.063	0.137
46929	0.78	0.152	0.031	0.034
24821	0.59	0.208	0.151	0.048
9587	0.741	0.126	0.028	0.104
16110	0.608	0.107	0.199	0.084
18131	0.79	0.15	0.016	0.042
14774	0.858	0.056	0.019	0.066

Figure 4. This figure shows, for each trace, the percentage of idle periods for which the optimal algorithm chose to transition to each state.

ready state before the new request arrives in order to service the incoming request just as it arrives.

Deterministic Competitive Algorithm (DET): This algorithm, presented and analyzed in [20], simply uses the discontinuities of the lower envelope curve for the offline algorithm ($t_1, \ldots t_{m-1}$) as thresholds which determine when to transition from each state to the next.

Last Period (LAST): This is a single-valued prediction algorithm which simply uses the last period as a predictor for the next idle period.

Exponential Decay (EXP): This algorithm, developed by Hwang, Allen and Wu [7] keeps a single value for the upcoming idle period. After a new idle period ends, the prediction is updated by taking a weighted average of the old prediction and the new idle period length. Let p be the current prediction and l the length of the last idle period. p is updated as follows:

$$p \leftarrow \lambda p + (1 - \lambda) l,$$

where λ is a value in the range $(0, 1)$. We use a value of .5 for λ.

Adaptive Learning Tree: This method uses an adaptive learning tree to predict the value of the next period based on the recent sequence of idle period lengths just observed. Details of this method can be found in [6].

There are different possible versions of single-valued prediction algorithms which are worth mentioning here. In describing these variations, we refer to the offline thresholds, $t_1, \ldots, t_{m-1}$, described in Section 3. The authors of the Learning Tree Algorithm observe that there are many idle periods which are very short. In order to avoid transitioning to a lower powered state for such short idle periods, they keep their system in the active state until the first offline threshold t_1 has been passed. Only then do they transition to the predicted optimal state. We ran all single-valued prediction algorithms with and without this initial delay. We found that the results were not significantly different between the two versions. In this paper, we will only report results for the versions without this delay.

Another feature which the authors of the Learning Tree Algorithm employ is to transition to the ready state after the threshold for the predicted optimal state is reached. This is done in hopes that a job will arrive shortly thereafter and the system can avoid incurring any additional latency in powering up after

the new job has arrived. If no job arrives before the last threshold, then the algorithm transitions to the lowest sleep state. To summarize, if there are m states and the prediction is that state i will be the optimal state, the algorithm will transition immediately to state i. If a new request has not arrived by time t_i, the algorithm will transition back to the ready state (state 1). Finally, if a request has not arrived by time t_{m-1}, the algorithm will transition to the deepest sleep state, state m. We call this version of single-valued prediction algorithms to be the *Preemptive-Wake-Up* version.

The alternative to this is to transition immediately to state i if that is the predicted optimal state and then to transition directly to the deepest sleep state if a request has not arrived by time t_i. We call this version the *Non-Preemptive-Wake-Up* version. Naturally, the preemptive-wake-up version will use more power but will tend to incur less latency on average. We report results for both versions of all the single-valued predictive algorithms used in the study.

5 Experimental Results

5.1 Experimentation with Window Size

Figure 5 shows the average energy consumed per request as the window size is varied. The most notable feature is that the variation is not very large, indicating that our method is fairly robust to choices in window size. Our method performs best with a relatively small value for the window size, indicating that it is the most recent history which is the most relevant predicting upcoming idle period length. It also indicates that the distribution over idle period lengths is not necessarily stable over time. However, the results get worse if the window size gets below 10, showing that there need to be enough values to get a representative sample. We select a value of 50 for the window size which is used for the remainder of the results presented.

5.2 Experimentation with Threshold Update Frequency

Figure 6 shows the average energy consumed per idle period as the frequency of updating the thresholds is varied. As one would expect, as the interval between updates grows, so does the power usage. However, there do not to seem to be large differences in the cost, so we adopt a frequence of update of 50.

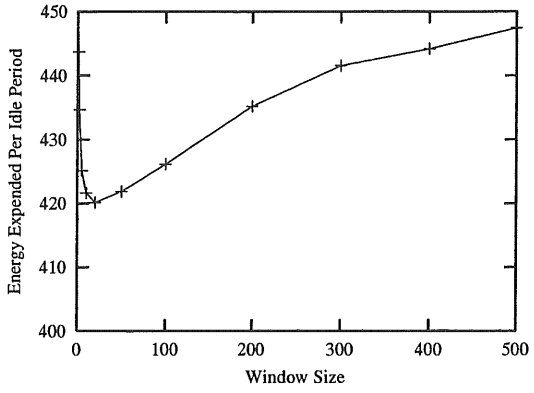

Figure 5. Average energy consumed per request as a function of window size for the Online Probability-Based Algorithm. The thresholds are updated every 10 requests.

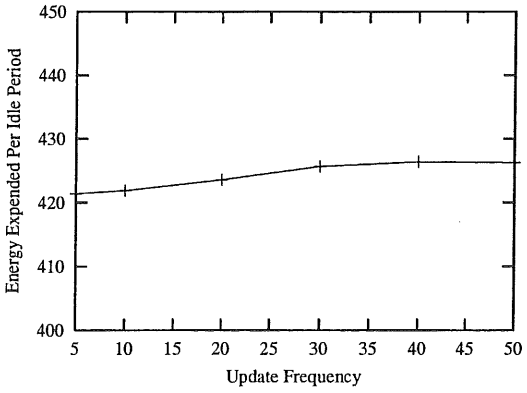

Figure 6. Average energy consumed per request as a function of frequency of update for the Online Probability-Based Algorithm. The window size is 50.

5.3 Evaluation of Algorithm Performance

Figures 7 and 8 give the main results from our study. The first column of numbers in Figure 7 is the average energy used per request for each of the algorithm. The second column is the ratio of this figure to the average energy consumed per request by the optimal offline algorithm. Interestingly, there were some traces where this ratio is less than one (although they always averaged out to be greater than 1 over all the traces). The reason it is possible for an algorithm to be better than the optimal offline algorithm on power consumption is because the optimal algorithm is always forced wake-up preemptively before a request arrives. This means that the optimal algorithm incurs no additional latency. Recall that since there is a power-latency tradeoff, this will tend to penalize the optimal algorithm with respect to power usage. The average latency per request is shown in the final column of the figure.

The results are also shown graphically in Figure 8 which plots the power usage (middle column from Figure 7) against the average latency (last column from Figure 7). The OPBA exhibits the lowest power consumption among all the online algorithms. The other algorithms which come close to matching its performance in power (the non-preemptive versions of LAST, TREE and EXP) all suffer at least an additional 40% latency on average. Meanwhile, the algorithms which have a lower average latency than OPBA (DET and the preemptive versions of LAST, TREE and EXP), they all use at least 25% more power on average. Thus, OPBA is the most successful algorithm in balancing power usage as well as latency incurred.

6. Acknowledgement

This work was partially supported by SRC, NSF and DARPA.

References

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[2] L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.

Algorithm	Average Energy	Ratio	Average Latency
Optimal	438.47	1	0
DET	647.41	1.47	870.91
OPBA	446.45	1.01	1332.41
LAST: Preempt	868.51	1.9	1010.87
LAST: Non-Preempt	477.96	1.09	2183.92
TREE: Preempt	990.53	2.25	1285.01
TREE: Non-Preempt	460.44	1.05	2239.57
EXP: Preempt	550.75	1.25	1080.16
EXP: Non-Preempt	458.98	1.04	1897.29

Figure 7. Energy is measured in Joules and Latency is measured in milliseconds.

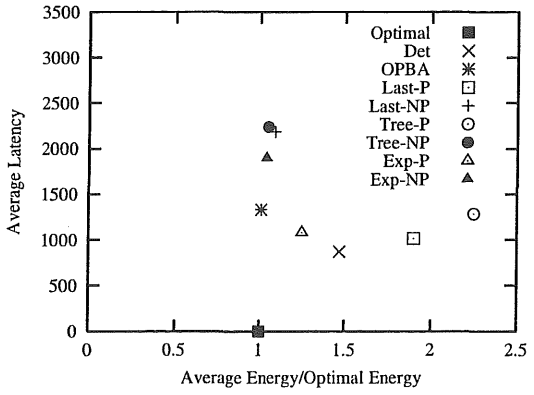

Figure 8. Energy is measured in Joules and Latency is measured in milliseconds.

[3] L. Benini, G. De Micheli, and E. Macii. Designing Low-power Circuits: Practical Recipes. *IEEE Circuits and Systems Magazine*, 1(1):6–25, Mar. 2001.

[4] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Publications, 1998.

[5] E. Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Proceedings of the Design Automation and Test Europe*, 1999.

[6] E.-Y. Chung, L. Benini, and G. D. Micheli. Dynamic Power Management Using Adaptive Learning Trees. In *Proceedings of ICCAD*, 1999.

[7] C.-H. Hwang, C. Allen, and H. Wu. A Predictive System Shutdown Method For Energy Saving of Event-Driven Computation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 28–32, 1996.

[8] Intel and Microsoft and Toshiba. Advanced Configuration and Power Interface Specification. Website, 1996.

[9] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized competitive algorithms for non-uniform problems. In *First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[10] S. Keshav, C. Lund, S. Phillips, N. Reaingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in ip-over-atm networks. *IEEE Journal on Selected Areas in Communications*, 13:1371–1382, 1995.

[11] Y. Lu, E. Chung, t. Simunic, L. Benini, and G. DeMicheli. Quantitative Comparison of Power Management Algorithms. In *DATE - Proceedings of the Design and Automation and Test in Europe Conference and Exhibition*, 2000.

[12] Y. Lu and G. DeMicheli. Adaptive Hard Disk Power Management on Personal Computers. In *Proceedings of the Great Lakes Symposium on VLSI*, 1999.

[13] Microsoft. OnNow Power Management Architecture for Applications. Website, 1997.

[14] Online Strategies for Dynamic Power Management Website (OSDPM). Java Applet based DPM Strategy Evaluation Website.

[15] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Proceedings of Design Automation Conference*, 1998.

[16] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of Design Automation Conference*, pages 555–561, June 1999.

[17] Q. Qiu, Q. Wu and M. Pedram. Stochastic Modeling of a Power-Managed System: Construction and Optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999.

[18] D. Ramanathan. High-level timing and power analysis for embedded systems. In *PhD thesis, University of California at Irvine*, 2000.

[19] D. Ramanathan, S. Irani, , and R. K. Gupta. Latency Effects of System Level Power Management Algorithms. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, 2000.

[20] S. Irani and S. Shukla and R. Gupta. Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Saving State. In *Proceedings of the Design Automation and Test Conference Europe (DATE02)*, 2002.

[21] S. Shukla and R. Gupta. A Model Checking Approach to Evaluating System Level Power Management for Embedded Systems. In *Proceedings of IEEE Workshop on High Level Design Validation and Test (HLDVT01)*. IEEE Press, 2001.

[22] T. Simunic, L. Benini, and G. D. Micheli. Event Driven Power Management of Portable Systems. In *In the Proceedings of International Symposium on System Synthesis*, pages 18–23, 1999.

[23] M. B. Srivastava, A. P. Chandrakasan, and R. W. Broderson. Predictive Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Trans. on VLSI Systems*, 4(1):42–54, 1996.

[24] *Auspex File Traces from the NOW project, available at.* http://now.cs.berkeley.edu/Xfs/AuspexTraces/auspex.html, 1993.

[25] *Technical specifications of hard drive IBM Travelstar VP 2.5inch, available at.* http://www.storage.ibm.com/storage/oem/data/travvp.htm, 1996.