Online Strategies for Dynamic Power Management in Systems with Multiple Power Saving States

Sandy Irani, Sandeep Shukla, Rajesh Gupta

S. Irani, S. Shukla and R. Gupta are affiliated with the Information and Computer Science department at the University of California at Irvine E-mail: {irani, skshukla, rgupta}@ics.uci.edu

CONTENTS

Abstract

Online dynamic power management (DPM) strategies refer to strategies that attempt to make power mode related decisions based on information available at runtime. In making such decisions, these strategies do not depend upon information of future behavior of the system, or any a priori knowledge of the input characteristics. In this paper, we present online strategies, and evaluate them based on a measure called the competitive ratio that enables a quantitative analysis of the performance of online strategies. All earlier approaches (online or predictive) have been limited to systems with two power saving states (e.g., idle and shutdown). The only earlier approaches that handled multiple power saving states were based on stochastic optimization. This paper provides a theoretical basis for the analysis of DPM strategies for systems with multiple power down states, without resorting to such complex approaches. We show how a relatively simple "online learning" scheme can be used to improve the competitive ratio over deterministic strategies using the notion of "probability-based" online DPM strategies. Experimental results show that the algorithm presented here attains the best competitive ratio in comparison with other known predictive DPM algorithms. The other algorithms which come close to matching its performance in power suffer at least an additional 40% wakeup latency on average. Meanwhile, the algorithms which have comparable latency to our methods use at least 25% more power on average.

I. INTRODUCTION

Power management in embedded computing systems is achieved by actively changing the power consumption profile of the system by putting its components into power/energy states sufficient to meeting functionality requirements. For example, an idling component (such as a disk drive) can be put into a slow-down or shutdown state. Of course, bringing such a component back to the active state may require additional energy and/or latency to service an incoming tasks. The input to the problem we consider here is the length of an upcoming idle period and the decision to be made is whether to transition to a lower power dissipation state while the system is idle. There are several issues in coming to this decision intelligently. For instance, immediate shutdown – that is, shutdown as soon as an idle period is detected – may not save overall energy if the idle period is so short that the powering up costs are greater than the energy saved in the sleep state. On the other hand, waiting too long to power down may not achieve the best energy reductions possible. Thus, there exists a need for effective (and efficient) decision procedures to manage power consumption. Dynamic Power Management (DPM) attempts to make such decisions (usually under the control of the operating system) at runtime based on the dynamically changing system state, functionality and timing requirements [1], [2], [3], [4], [5], [6], [7], [8].

In a survey of DPM techniques in [9], the authors classify DPM strategies into two main groups: (a) *predictive schemes*, and (b) *stochastic optimum control* schemes. Predictive schemes attempt to predict the timing of future input to the system and schedule shutdown (usually to a single lower power state) based on these predictions. Stochastic optimum control is a well-researched area [9], [10], [11], [7], [12], [13]. The chief characteristic of these approaches is the construction (and validation) of a mathematical model of the system that lends itself to a formulation of a stochastic optimization problem. Then strategies to guide the system power profile are devised that achieve the most power savings in presence of the uncertainty related to system inputs. While several useful and practical techniques have been developed using predictive and stochastic optimum control schemes, it is difficult to develop bounds on the quality of the results without extensive simulations and/or model justification.

II. Competitive Analysis and Its Limitations

We approach DPM as an inherently "online" problem in that an algorithm governing power management must make decisions about the expenditure of resources before all the input to the system is available [14]. Specifically, the algorithm does not learn the length of an idle period until the moment that it ends. Analytical solutions to such online problems are often best characterized in terms of a *competitive ratio* [15] that compares the cost of an online algorithm to the optimal offline solution which knows the input in advance. Note that the offline algorithm is not a realistic algorithm since it knows the length of the idle period in advance. It is just used as a point of comparison to quantify the performance loss due to the fact that a DPM strategy must make decisions with only partial information. To be specific, we say that an algorithm is c-competitive if for any input, the cost of the online algorithm is bounded by c times the cost of the optimal offline algorithm for that input. The offline algorithm has access to the entire input before committing to any decisions. For DPM, the cost of an algorithm is the total amount of energy consumed. The *Competitive Ratio* (CR) of an algorithm is the infimum over all c such that the algorithm is c-competitive.

Competitive analysis [14] – that is, analysis to determine bounds on the competitive ratio – can be done either as a case analysis of the various adversarial scenarios [16], [17] or through theorem proving or automatic modeling checking means [5]. Competitive analysis has proven to be a powerful tool in providing a guarantee on the performance of an algorithm for any input. Competitive analysis is valuable in situations where it is impractical to obtain and process information for predicting future inputs. It also provides an assurance for the designers of the algorithm about the worst possible behavior of an online algorithm. For example, if an online strategy has a competitive ratio of 2, that assures the designer that no matter what input sequence is provided to the strategy, it will never cost the strategy more than twice the cost incurred by the best possible strategy.

There are two chief limitations of the competitive analysis approach to DPM that we seek to address in this paper. The first is that the results tend to be overly pessimistic due to the fact that they examine worst case behavior. Thus, there is a need to refine the CR bounds that take into account typical input behavior. In many applications there is structure in the input sequence that can be utilized to fine tune online strategies and improve their performance.

The second problem with our earlier work on CR-based DPM strategies [6], [18], [16] is that the system model consists of devices with only two power saving states, namely an idle state and a shutdown state. This is a limitation of most predictive strategies as well [9]. In contrast, most real systems consist of components with multiple power states. There are two kinds of systems here: systems with multiple shutdown states as well as systems with multiple operating states. For instance, Table I shows the power states of a portable hard drive [19] that consists of three low power states in which a device can be when it is not processing any requests. Radio modems represent devices that can often be in multiple active states (for instance, transmitting data at various power levels to the radio transmitter).

We do not address the problem of choosing among different operating states in this paper.

Microsoft's OnNow initiative gives standards for power saving states for a broad class of devices [20]. The Advanced Configuration and Power Interface (ACPI) Specification gives four distinct states which may be optional or required for different types of devices. These states are: an active state, two levels of sleep states and an off state. For example, for modems and similar communications devices, the standard specifies that the active and the off state are required and the lower powered sleep state is optional. For network devices, the active and off states are required and both intermediate sleep states are optional [21]. Thus, there is a clear industry standard that encourages device manufacturers to support multiple sleep states.

For systems with more than one device, our protocols can be applied to each device separately. Each device experiences periods of idle time which may or may not coincide with the idle periods experienced by other devices in the system. A power management strategy can then be applied to a particular device during the idle periods experienced by that device. There will be some dependence between performance for different devices in cases where a request requires more than one device to be satisfied. In these situations, a device which is in an active state may experience some latency because it has to wait for another device to transition from a low-power state to the active state before either can begin work. In this paper, we focus on performance for a single device although this phenomenon would be an interesting direction for future work.

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	1.9	0	0

TABLE I

VALUES FOR THE POWER DISSIPATION AND START-UP ENERGY FOR THE IBM MOBILE HARD-DRIVE.

A. Related Work

Many strategies for dynamic management of power and energy have been proposed and studied. These include predictive strategies [1], [2], stochastic modeling based strategies [11], [12], session clustering and prediction strategies [22], online strategies [6], and adaptive learning based strategies [23]. A survey of DPM strategies that covers the classes of algorithms in predictive and stochastic optimum control categories can be found in [9] . The authors in [24] present a quantitative comparison of various existing management strategies.

Many predictive dynamic power management strategies [1], [2], [6], [16], [11], [7], [22] 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 the predicted idle period length. In case the 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* (SVP) schemes. Of particular interest is the work of Chung, Benini and De Micheli in [23] that addresses multiple idle state systems using a prediction scheme based on adaptive learning trees. Their method has shown an impressively high hit ratio in its prediction.

In the past, stochastic control based methods as in [9], [10], [11], [7], [12], [13] have been limited because they make assumptions about the characteristic probability distribution of the input/job arrivals, the service time distribution of the device etc. Although the policies obtained are optimal given these assumptions, there are no guarantees that the assumptions will always hold. These problems have been addressed to some extent in [22], [23]. Recently the stochastic modeling approach has been extended so that the assumption that inter-arrival times are exponentially distributed can be removed [25]. Stochastic modeling has also been extended to systems which can be modeled by petri nets [26]. One drawback with these approaches is that they require solving computationally expensive optimization problems to derive the optimal solutions. If the usage patterns for the device change, the algorithm has to be re-optimized to adjust to the new behavior. In [7], an adaptive technique has been proposed which tries to overcome this limitation. The authors use sliding windows to keep a long enough history of the input arrivals, and estimate the parameters of the arrival Markov processes. However, this method is complex because they have to solve the optimization problem to find the optimal strategy in a clock-driven or event-driven manner.

There are also a few existing results on competitive analysis of DPM strategies in the online algorithms literature. Some of this work does not address DPM explicitly, but the general framework applies to DPM as well. An example of this is the now classical result in [15] which shows that 2 is the best competitive ratio achievable by any deterministic online algorithm. Probabilistic analysis for online DPM algorithms in two-state systems has been given in [16], [27]. They assume that the distribution over the upcoming idle period is known and optimize the algorithm based on that distribution. They give a method to determine the best online algorithm given a distribution and show that for any distribution, the expected cost of this online algorithm is within a factor of $e/(e-1) \approx 1.58$ of the expected cost of the optimal offline algorithm. This result is tight in that there is a distribution for which the ratio is exactly $e/(e-1)$, although for some distributions the ratio may be less.

B. Our Contributions and Paper Organization

In Section IV we present a deterministic algorithm for DPM for multi-state devices and show that it is 2-competitive. This result, which extends the earlier analysis from [15], is tight in the sense that there is no constant $c < 2$ such that there is a deterministic c-competitive algorithm that works for all multiple power down state devices. However, it may be possible to have a competitive ratio less than 2 for a specific device, depending on the parameters of the device (e.g. number of states, power dissipation rates, start-up costs, etc.). Note that this deterministic algorithm focuses on a single idle period, and does not depend on any history of previous idle periods. Thus, it is non-adaptive in that it does not use any information about previous idle periods to tune its behavior.

As we have argued above, competitive analysis often yields unduly pessimistic results. Indeed, our goal is to devise a DPM strategy whose total energy expenditure is much lower than twice the optimal offline algorithm. We show that by keeping track of a sequence of past idle periods one can improve the performance of the 2-competitive deterministic strategy in practice by adapting the behavior of the strategy to patterns observed in the input sequence. We do this in Section V by modeling the future input by a probability distribution that is learned from the recent history of idle periods lengths. In so doing, we avoid some of the problems with many of the previous strategies mentioned above.

One of the chief limitations of a single-valued prediction (SVP) approach is that it fails to capture uncertainty in the prediction for the 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. Using a probability distribution to model the length of the upcoming idle period 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. Furthermore, we make no assumptions about the form of the distribution governing idle period length which means that our method can automatically adapt to a variety of applications, and applies to non-stationary input arrivals.

In Section V-A we derive analytical bounds for an online algorithm for DPM that knows the probability distribution governing the length of the upcoming idle period. The following Section V-B provides a systematic means of dynamically constructing an estimate for the probability distribution based on online observations and combining it with the algorithm from the previous subsection to build an online power management strategy. Experimental results in Section VII demonstrate the utility of our strategy in balancing power usage with latency.

III. System Model

Consider a device that can be in one of the $k+1$ power states denoted by $\{s_0,\ldots,s_k\}$. 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 *active* state which is the highest power consumption state. Any strategy for an individual idle period can be described by a sequence of thresholds each of which is associated with a power consumption state. As soon as the idle periods extends beyond a given threshold, the device transitions to the associated 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 energy needed and time spent to go from a higher (power) state to a lower (power) state is negligible, whereas the converse is not true. 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 active 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 active state (state 0). The total energy used in transitioning from state i to the active state is denoted by β_i.

We note that in cases where the time and energy cost incurred 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 algorithm is simply a sequence of idle periods. For each idle period, the DPM is notified when the idle period begins and then again when the idle period ends. This is the only information required by our DPM. In our experiments the idle periods are derived by our simulator which receives a time-stamped sequence of requests for service. With each request, the simulator is 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 active state to serve the new request if it is not already there. In the case where the device is not already in the active 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 disappear. Thus, we have an interesting situation where the behavior of the algorithm effects future inputs (idle period lengths) given to the algorithm.

Note also that delaying servicing a request will tend to result in lower power usage. For instance, 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 active state and begin to work on a request as soon as one appears. However, it illustrates the natural tradeoff which occurs between power consumption and latency. Our experimental results explore this tradeoff for the set of algorithms studied. A more extensive discussion of this tradeoff is presented in [6].

In the next two sections we present our analysis for the deterministic and probability-based algorithms. In this analysis, we do not take into account the delay incurred in returning to the active state. This is because the analysis focuses on an individual idle period. These sections address the problem of minimizing total energy expenditure given that the length of the upcoming period is arbitrary (deterministic case) or is governed by a fixed probability distribution (probability-based case). The empirical evaluations incorporate the effect of start-up latency as an entire sequence of requests for service arrive through time.

IV. The Deterministic Online Algorithm

We now present our deterministic algorithm for online DPM. The basic idea is that the online algorithm tries to mimic the behavior of the optimal offline algorithm. At each point in time t, the algorithm is in the state which the optimal algorithm would have chosen if the length of the idle period were exactly t.

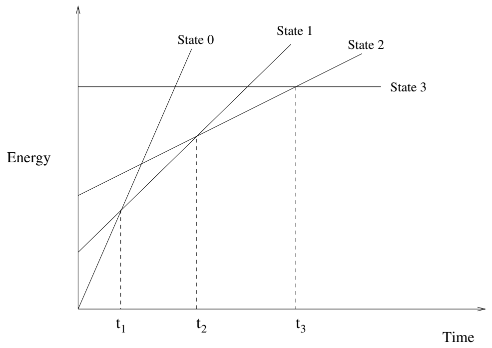

Fig. 1. Energy consumption for each state for a four-state system. Each state is represented by a line which indicates the energy used if an algorithm stays in that state as a function of the length of the idle period. For each state, the slope is the power dissipation rate and the y-intercept is the energy required to power up from that state.

To get the optimal cost, we plot each line $c = \alpha_i t + \beta_i$. This is the cost of spending the entire interval in state i as a function of t, the length of the interval. Take the lower envelope of all of these lines. Let us call this function, $LE(t)$. The optimal cost for an interval of length t is $LE(t) = \min_i\{\alpha_i t + \beta_i\}$. The online algorithm – called the Lower Envelope Algorithm (LEA) – will also follow the function LE. It will remain in the state which realizes the minimum in LE and will transition at the discontinuities of the curve. That is, LEA will remain in state j as long as $\alpha_j t + \beta_j = \min_i\{\alpha_i t + \beta_i\}$, for the current time t.

For $j = 1$ to k, let t_j be the solution to the equation $\alpha_j t + \beta_j = \alpha_{j-1} t + \beta_{j-1}$, where t_j is the time that LEA will transition from state $j-1$ to state j. We will assume here that we have thrown out all the lines which do not appear on the lower envelope at some point. This is equivalent to the assumption that $t_1 < t_2 < \cdots < t_{k-1} < t_k$. (See Figure 1).

Theorem 1: **The Lower Envelope algorithm is 2-competitive.**
The proof of Theorem 1 is shown in the Appendix.

As emphasized earlier, this algorithm does not take into account input patterns. The worst case scenario, obtained via Theorem 1, shows that the energy cost resulting from the online decisions can be no worse than 2 times the energy cost of the optimal offline strategy which knows the input sequence in advance. We show later that depending on request arrival patterns, this worst case bound may not really happen, and the empirical ratio of the online to offline costs may in fact be much lower. As shown in [12], [23], [11], input sequences are often interrelated, and hence modeling of the input pattern and exploiting that knowledge in the design of the algorithm can help bridge the gap between the performance of online strategy and that of the optimal offline strategy. In the next subsection, we discuss our probability-based algorithm and show that if the probability distribution governing the length of the idle period is known before hand, the worst case competitive ratio can be improved by 21%, with respect to the deterministic case. Moreover, we show through experimental evaluation that this worst case bound is pathological. In fact, we can bring the energy cost of the online algorithm within 27% of the optimal offline one.

V. The Probability-based Online Algorithm

A. Optimizing Power Based on a Probability Distribution

Let us assume that the length of the idle interval is generated by a fixed, known distribution whose density function is π and devise a method to optimize power management decisions. We first discuss systems with two states and then give our generalization to the multi-state case. Let β be the start-up energy of the sleep state and α the power dissipation of the active state. Suppose that the online algorithm uses τ as the threshold at which time it will transition from the active state to the sleep state if the system is still idle. In this case, the expected energy cost for the algorithm for a single idle period will be

$$\int_0^{\tau} \pi(t)(\alpha t)dt + \int_{\tau}^{\infty} \pi(t)[\alpha\tau + \beta]dt \tag{1}$$

The best online algorithm will select a value for τ which minimizes this expression. The offline algorithm which knows the actual length of an upcoming idle period will have an expected cost of

$$\int_0^{\beta/\alpha} \pi(t)(\alpha t)dt + \int_{\beta/\alpha}^{\infty} \pi(t)\beta dt \tag{2}$$

It is known for the two state case, that the online algorithm can pick its threshold τ so that the ratio of its expected cost to the expect cost of the optimal algorithm is at most $e/(e-1)$ [16]. That is, for any π, and any α and β,

$$\frac{\min_{\tau}\left\{\int_0^{\tau}\pi(t)(\alpha t)dt + \int_{\tau}^{\infty}\pi(t)[\alpha\tau+\beta]dt\right\}}{\int_0^{\beta/\alpha}\pi(t)(\alpha t)dt + \int_{\beta/\alpha}^{\infty}\pi(t)\beta dt} \leq \frac{e}{e-1} \tag{3}$$

This is optimal in that for any α and β, there is a distribution π such that this ratio is at least $e/(e-1)$.

Let us now consider the multi-state case. As in the previous section, let t_j be the solution to the equation $\alpha_j t + \beta_j = \alpha_{j-1}t + \beta_{j-1}$. t_j is the time that LEA will transition from state $j-1$ to state j. We will assume here that we have thrown out all the lines which do not appear on the lower envelope at some point. This is equivalent to the assumption that $t_1 < t_2 < \cdots < t_{k-1} < t_k$. For ease of notation, we will define t_0 to be 0 and t_{k+1} to be ∞. The cost (expected energy consumption) of the optimal offline algorithm is:

$$\sum_{i=0}^{k} \int_{t_i}^{t_{i+1}} \pi(t)[\alpha_i t + \beta_i]dt \tag{4}$$

Now to determine the online algorithm, we must determine k thresholds, where the threshold τ_i is the time at which the online algorithm will transition from state $i-1$ to state i. In the spirit of the deterministic online algorithm for the multi-state case, we will let τ_i be the same as the threshold which would be chosen if $i-1$ and i were the only two states. We call this algorithm the Probability-based Lower Envelope Algorithm (PLEA). The proof of the following theorem appears in the appendix.

Theorem 2: **For any distribution, the expected cost of the Probability-based Lower Envelope Algorithm is within a factor of $e/(e-1)$ of the expected cost for the optimal offline algorithm.**

B. Learning the Probability Distribution

While we have proven the competitive bounds regardless of the idle time probability distribution chosen, the practical problem of finding $\pi(t)$ to guide the PLEA algorithm remains. Our approach is to learn $\pi(t)$ online. Accordingly, the algorithm which uses PLEA in conjunction with our scheme to learn the probability distribution is called Online Probability-Based Algorithm (OPBA). It 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 idle period lengths and summarizes this information in a histogram. Periodically, the histogram is used to generate a new power management strategy.

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 in Section VII include a study experimenting with values for w.

The counter for bin i is denoted by c_i. The threshold for changing states 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 $i \in \{0,\ldots,n-1\}$. The sum of the counters is the window length w. Thus, the threshold is taken to be

$$\begin{aligned} &\arg\min_{r_t} \sum_{j=1}^{t-1} \left(\frac{c_j}{w}\right) r_j(\alpha_i - \alpha_{i-1}) \\ &+ \sum_{j=t}^{n} \left(\frac{c_j}{w}\right) [r_t(\alpha_i - \alpha_{i-1}) + (\beta_{i-1} - \beta_i)] \end{aligned} \tag{5}$$

In order to be useful as a decision procedure, this algorithm must be implemented efficiently. We have implemented the algorithm for finding the all k thresholds in time $O(kn)$, where $k+1$ is the number of states and n is the number of bins in the histogram. Two important factors determine the cost (in time expenditure) of implementing our method: (a) the frequency with which the thresholds are updated, and (b) the number of bins in the histogram. These need to be minimized for policy implementation efficiency. The algorithm runs the $O(kn)$ time algorithm every time the thresholds are updated. The frequency with which the thresholds are updated is the subject of one set of experiments discussed in Section VII. Note that the cost to implement our strategy is independent of w, the number of idle periods tracked in the histogram.

Minimization of the number of bins used in the histogram must be balanced with the fact that the finer grained the histogram is, the more accurate our choice of thresholds will be. Our experiments, as discussed in the following

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

TABLE II

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.

sections, show that a finer grained binning is more important in some ranges than it is in others. A way of addressing this problem (that was key to the success of the algorithm) is to use the thresholds of the Lower Envelope Algorithm to guide the selection of the bins. Recall that these thresholds are the $t_1, \ldots, t_k$ defined in Section IV. We then choose a constant c number of bins per state. The range from t_i to t_{i+1} is divided into c equal sized bins. Note that the running time of our algorithm depends linearly on the number of bins and hence depends linearly on c. For this reason, it is important to select a small value for c. We experimented with values for c ranging from 1 to 10 and the results only varied by 1%. We use a value of $c = 5$ in all our results. Table II shows a sample histogram from our experiments.

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

TABLE III

THIS FIGURE SHOWS, FOR EACH TRACE, THE PERCENTAGE OF IDLE PERIODS FOR WHICH THE OPTIMAL ALGORITHM CHOSE TO TRANSITION TO EACH STATE.

VI. EXPERIMENTAL DESIGN

A. Data Used in our Experiments

To demonstrate the utility of our probability-based algorithm, we use a mobile hard-drive from IBM [19]. This drive has four power down states, as shown in Table I. 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 [28]. From this data, we collected the 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.

Table III 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 active 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.

B. Algorithm Test Suite

We compare OPBA and LEA to several other predictive 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 LEA fall into this group. The second group are single-valued prediction algorithms. They use a single prediction for the upcoming idle period length and transition immediately to the optimal state for that length. 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 active state before the new request arrives in order to service the incoming request just as it arrives.

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

Exponential Decay (EXP): This algorithm, developed by Hwang, Allen and Wu [1] keeps a single prediction 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 (TREE): This method uses an adaptive learning tree to predict the value of the next period based on the sequence of recent idle period lengths just observed. Details of this method can be found in [23].

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, $t_1, \ldots, t_k$, described in Section V-B. 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. To see why the delay does not help significantly, we give some statistics for the Exponential Decay algorithm when run without the delay in Table IV. An entry in row i column j indicates the total number of times over all traces that the online algorithm predicted that the optimal state for an upcoming idle period would be j and the actual optimal state was i. Having a delay until time t_1 helps only when the algorithm predicts that the optimal state is Stand-by, Idle or Sleep and the actual optimal state is Active. The numer of times this happens is the sum of the values in Row 'Active', Columns 'Stand-by', 'Idle' and 'Sleep'. The degree of savings in latency is largest for idle periods counted in entry [Active,Sleep], less so for entry [Active,Stand-by] and even less so for

	Active	Stand-by	Idle	Sleep
Active	7631	3353	831	109
Stand-by	901	3144	4559	903
Idle	106	1414	1640	3416
Sleep	29	123	4897	7168

TABLE IV

PREDICTIONS VS. OUTCOMES FOR THE EXPONENTIAL DECAY ALGORITHM. THE TOTAL NUMBER OF IDLE PERIODS IS 54985 AND THE TOTAL NUMBER OF REQUESTS IS 373617.

[Active, Idle]. Given the values in the table, it is clear why having a delay does not offer a significant saving in the total latency.

Another feature which the authors of the Learning Tree Algorithm employ is to transition to the active 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 t_k, then the algorithm transitions to the lowest sleep state. To summarize, if there are $k+1$ states and the prediction is that state i will be the optimal state for the upcoming idle period, 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 active state (state 0). Finally, if a request has not arrived by time t_k, the algorithm will transition to the deepest sleep state, state k. We call this version of single-valued prediction algorithms to be the *Preemptive-Wake-Up* version.

The alternative Preemptive Wake-Up 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.

VII. EXPERIMENTAL RESULTS

A. Experimentation with Window Size

Figure 2 shows the average energy consumed per request as the window size is varied. Note that beyond a certain threshold window size (below which the predictions are not accurate anyway) the variation of energy consumption with respect to the increase in window size is not 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.

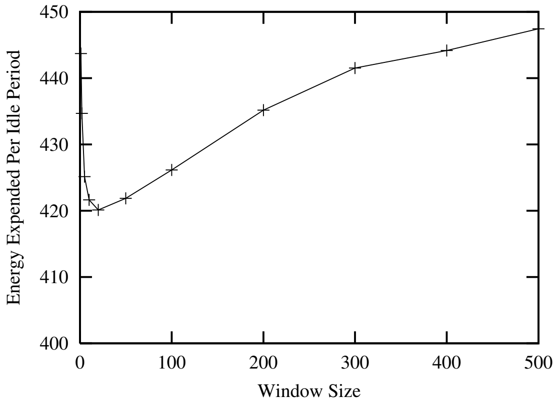

Fig. 2. Average energy consumed per request as a function of window size for the Online Probability-Based Algorithm. The thresholds are updated every 10 requests.

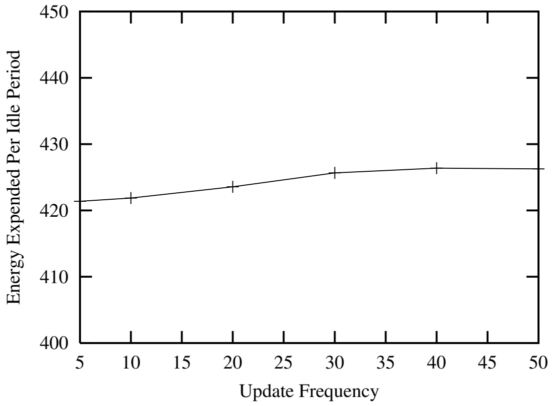

Fig. 3. Average energy consumed per request as a function of frequency of update for the Online Probability-Based Algorithm. The window size is 50.

B. *Experimentation with Threshold Update Frequency*

Figure 3 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 frequency of update of 50.

C. *Evaluation of Algorithm Performance*

Table V shows the comparison of energy consumption and wake-up latency effects across a number of predictive DPM algorithms. The first column of numbers in Table V 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

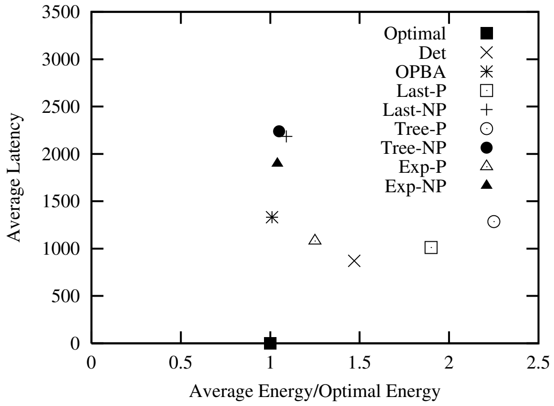

Fig. 4. Energy is measured in Joules and Latency is measured in milliseconds.

preemptively before a request arrives. This means that the optimal algorithm incurs no additional latency due to waking up the disk drive. 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 latency is averaged over the number of requests, not the number of idle periods. These two numbers are different since some requests may arrive while the device is busy working on other jobs or powering-up. In these cases, the incoming request would not correspond to an idle period. As an example, we refer back to the statistics for the Exponential Delay algorithm in Table IV. The total number of idle periods over all traces is $54,985$, whereas the total number of requests is $373,617$. Note that the algorithm only experiences the full 5 seconds of transition time when a request arrives and the algorithm is in the Sleep state. (Refer to Table I for the transition times for the various states). For single-value prediction algorithms without predictive wake-up, this only happens if the algorithm predicts that the Sleep state will be the optimal state for the upcoming idle period or when the idle period actually lasts so long that the algorithm discovers that the sleep state was in fact the optimal state. The number of times this happens is the sum of the numbers in entries which are either in Column 'Sleep' or Row 'Sleep' (or both). The algorithm only experiences the 1.5 sec delay in transitioning from the Stand-by state to the Active state for those idle periods counted in entries [Active, Stand-by], [Idle, Stand-by] and [Stand-by, Stand-by].

The final results in Table V are also shown graphically in Figure 4 which plots the power usage (middle column from Table V) against the average latency (last column from Table V). 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 (LEA and the preemptive versions of LAST, TREE and EXP) 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.

Finally, we study the the effect of variable wake-up time since many devices vary somewhat in the time it takes to

Algorithm	Average Energy	Competitive Ratio	Average Latency
Optimal	438.47	1	0
LEA	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

TABLE V

ENERGY IS MEASURED IN JOULES AND LATENCY IS MEASURED IN MILLISECONDS.

ε	LEA	OPBA	EXP-Preempt	EXP-Non-Preempt
0	879	1490	1092	1905
.25	887	1543	1089	1945
.5	894	1585	1108	2001
.75	931	1631	1151	2116

TABLE VI

LATENCY AS VARIATION IN TRANSITION TIME INCREASES. EACH TRANSITION TIME IS MULTIPLIED BY A NUMBER GENERATED UNFORMLY AT RANDOM FROM THE RANGE $[1-\varepsilon, 1+\varepsilon]$. LATENCY IS MEASURED IN MILLISECONDS.

transition from one state to another. We study the top performers for latency and power when random noise is added to the time to transition to the active state. Every time a transition is performed, we multiply the time to transition by a number generated unformly at random from the range $[1-\varepsilon, 1+\varepsilon]$ for different values of ε. The results are shown in Table VI. We did not include the power usage because it does not vary significantly for different values of ε. All of the methods suffer a slight degradation in performance as ε increases. The variation seems to have a similar effect on all algorithms studied.

VIII. CONCLUSION AND FUTURE DIRECTIONS

We have presented a deterministic online algorithm for dynamic power management on multi-state devices and proved that it is 2-competitive, and this bound is tight. We improve upon this bound considerably by devising a probability-based scheme. To support this strategy in an online DPM framework, we provide a method to efficiently construct a probabilistic model for the length of te upcoming idle period based on online observations. Our experiments show that the algorithm presented in this paper attains the best performance in practice, in comparison to other known predictive DPM algorithm. The other algorithms which come close to matching its performance in power all suffer at least an additional 40% wakeup latency on average. Meanwhile, the algorithms which have comparable latency to our methods all use at least 25% more power on average. Our future plans include extension of this work into systems with multiple active as well as power-down states.

Our simulation code can be downloaded from our website to test the effects of various power management strategies on different kinds of trace data [29].

IX. Acknowledgements

This work was supported by NSF grant CCR-0098335, SRC, and DARPA/ITO supported PADS project under the PAC/C program. In addition, the first author is partially supported by NSF grant CCR-0105498 and by ONR Award N00014-00-1-0617.

References

[1] 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*, 1996, pp. 28–32.

[2] 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*, vol. 4, no. 1, pp. 42–54, march 1996.

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

[4] L. Benini, G. De Micheli, and E Macii, “Designing Low-power Circuits: Practical Recipes,” *IEEE Circuits and Systems Magazine*, vol. 1, no. 1, pp. 6–25, Mar. 2001.

[5] 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)*. November 2001, IEEE Press.

[6] 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.

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

[8] S. Irani and S. Shukla and R. Gupta, “Competitive analysis of dynamic power management strategies for systems with multiple power saving states,” in *Proceedings of the Design Automation and Test Europe Conference*, 2002.

[9] L. Benini, A Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, vol. 8, no. 3, pp. 299–316, 2000.

[10] T. Simunic, L. Benini, and G. De Micheli, “Event Driven Power Management of Portable Systems,” in *In the Proceedings of International Symposium on System Synthesis*, 1999, pp. 18–23.

[11] L. Benini, A Bogliolo, G. Paleologo, and G. De Micheli, “Policy optimization for dynamic power management,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.

[12] Q. Qiu and M. Pedram, “Dynamic Power Management Based on Continuous-Time Markov Decision Processes,” in *Proceedings of Design Automation Conference*, June 1999, pp. 555–561.

[13] 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.

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

[15] S.J. Phillips and J.R. Westbrook, *chapter 10 of Algorithms and Theory of Computation Handbook*, chapter On-line algorithms: Competitive analysis and beyond, CRC Press, Boca Raton, 1999.

[16] 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*, 1990, pp. 301–309.

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

[18] D. Ramanathan, S. Irani, and R. Gupta, "An Analysis of System Level Power Management Algorithms and their effects on Latency," *IEEE Trans. on Computer Aided Design*, vol. 21, no. 3, march 2002.

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

[20] *OnNow and Power Management, available at*, www.microsoft.com/hwdev/tech/onnow, 2002.

[21] *Device Class Power Management Reference Secification, available at*, www.microsoft.com/hwdev/resources/specs/pmref, 2002.

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

[23] E-Y Chung, Luca Benini, and Giovanni De Micheli, "Dynamic Power Management Using Adaptive Learning Trees," in *Proceedings of ICCAD*, 1999.

[24] Y. Lu, E.Y. 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.

[25] P. Glynn T. Simunic, L. Benini and G. De Micheli, "Dynamic power management of portable systems," in *In the Proceedings of MOBICOM*, 2000.

[26] Qinru Qiu, Qing Wu, and Massoud Pedram, "Dynamic power management of complex systems using generalized stochastic petri nets.," in *Proc. 37th Design Automation Conference (DA'2000), 5-9 June 2000, Los Angeles, CA*, 2000, pp. 352–356.

[27] S. Keshav, C. Lund, S. Philliips, 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*, vol. 13, pp. 1371–1382, 1995.

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

[29] OSDPM Website at UC Irvine, http://www.ics.uci.edu/ osdpm, "Java Applet based DPM Strategy Evaluation Website," .

APPENDIX

I. PROOFS

Proof of Theorem 1:

First we establish that the worst case for the algorithm will always be just after a transition time. Consider the time $t_j+\gamma$, for some $0 \leq j \leq k$ and $0 \leq \gamma < t_{j+1}-t_j$. For any value of γ in the given range, the optimal cost will be

$$\alpha_j(t_j+\gamma)+\beta_j.$$

For any value of γ in the given range, the online cost will be:

$$\sum_{l=0}^{j-1}\alpha_l(t_{l+1}-t_l)+\alpha_j\gamma+\beta_j.$$

The ratio of these two will be maximized for $\gamma = 0$.

Now suppose that the interval ends just after t_j for some $1 \leq j \leq k$. Using the cost for the online and offline determined above, the ratio of the online cost to the offline cost will be

$$\begin{aligned} & \frac{\sum_{l=0}^{j-1} \alpha_l (t_{l+1} - t_l) + \beta_j}{\alpha_j t_j + \beta_j} \\ = \quad & 1 + \frac{\sum_{l=0}^{j-1} \alpha_l (t_{l+1} - t_l) - \alpha_j t_j}{\alpha_j t_j + \beta_j} \end{aligned}$$

Thus, it is sufficient to prove that

$$\sum_{l=0}^{j-1} \alpha_l (t_{l+1} - t_l) - \alpha_j t_j \leq \alpha_j t_j + \beta_j. \tag{6}$$

Each t_l was chosen so that

$$(\alpha_{l-1} - \alpha_l) t_l = \beta_l - \beta_{l-1},$$

so we can substitute these values into Inequality 6 to get that

$$\begin{aligned} & (\beta_1 - \beta_0) + (\beta_2 - \beta_1) + \cdots + (\beta_j - \beta_{j-1}) - \alpha_0 t_0 \\ \leq \quad & \alpha_j t_j + \beta_j. \end{aligned}$$

Collapsing the telescoping sum, we get that

$$\beta_j - \beta_0 - \alpha_0 t_0 \leq \alpha_j t_j + \beta_j.$$

Using the fact that $\beta_0 = 0$, $t_0 = 0$ and that $\alpha_j \geq 0$ and $t_j \geq 0$, the inequality holds. ■

Proof of Theorem 2:

Consider a system in which there are only two states: $i-1$ and i. Both online and offline must pay at least $\alpha_i t$ for an interval of length t. In addition, each must pay at least β_{i-1} for the start-up cost. These costs which are incurred by both algorithms, regardless of their choices will only serve to decrease the competitive ratio. In determining, τ_i, we will disregard these additional costs. Consider the system where the power consumption rate in the ON state is $\alpha_{i-1} - \alpha_i$ and is 0 in the OFF state. The energy required to transition from the ON to the OFF state is $\beta_i - \beta_{i-1}$. We will choose τ_i to the be the transition time for the optimal online policy in this system. Thus, we will choose τ_i to be

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

The online cost for this new system is the above expression evaluated at $\tau = \tau_i$:

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

Let t_i be defined to be $(\beta_i - \beta_{i-1})/(\alpha_{i-1} - \alpha_i)$. Note that this is the same definition in the previous proof: the point where the lines $\alpha_i t + \beta_i$ and $\alpha_{i-1} t + \beta_{i-1}$ meet. The offline cost for the new system is

$$\begin{aligned} OFF_i \quad = \quad & \int_0^{t_i} \pi(t) t (\alpha_{i-1} - \alpha_i) dt \\ + \quad & \int_{t_i}^{\infty} \pi(t) (\beta_i - \beta_{i-1}) dt. \end{aligned}$$

We are guaranteed that the ratio of the expected online to offline costs is at most $e/(e-1)$ [16], [27].

Since the ratio of $\mathrm{O}N_i$ to $\mathrm{O}FF_i$ is at most $e/(e-1)$ for each i, we know that

$$\frac{\sum_{i=1}^{k}\mathrm{O}N_i}{\sum_{i=1}^{k}\mathrm{O}FF_i} \le \frac{e}{e-1}.$$

We will now prove that $\sum_{i=1}^{k}\mathrm{O}N_i$ is exactly the expected cost for PLEA on the multi-level system. We will also prove that $\sum_{i=1}^{k}\mathrm{O}FF_i$ is exactly the expected cost of the optimal algorithm for the multi-level system.

We will rephrase $\mathrm{O}N_i$ by separating the integral into the intervals from τ_{j-1} to τ_j. To simplify notation, τ_0 will denote 0 and τ_k will denote ∞.

$$\begin{aligned}\mathrm{O}N_i &= \sum_{j=1}^{i}\int_{\tau_{j-1}}^{\tau_j}\pi(t)[t(\alpha_{i-1}-\alpha_i)]dt\\ &+ \sum_{j=i+1}^{k+1}\int_{\tau_{j-1}}^{\tau_j}\pi(t)[\tau_i(\alpha_{i-1}-\alpha_i)+(\beta_i-\beta_{i-1})]dt.\end{aligned}$$

In the sum over all $\mathrm{O}N_i$, we will group together all the contributions from each $\mathrm{O}N_i$ over the interval $[\tau_j,\tau_{j+1}]$ for $1\le j\le k+1$. Note that this is the interval that the algorithm will spend in state j. This value will be:

$$\begin{aligned}&\sum_{i=1}^{j}\int_{\tau_j}^{\tau_{j+1}}\pi(t)[\tau_i(\alpha_{i-1}-\alpha_i)+(\beta_i-\beta_{i-1})]dt\\ +&\sum_{i=j+1}^{k}\int_{\tau_j}^{\tau_{j+1}}\pi(t)[t(\alpha_{i-1}-\alpha_i)]dt.\end{aligned}\tag{7}$$

Thus, we have that

$$\sum_{i=1}^{k}\mathrm{O}N_i = \sum_{j=0}^{k}f(j),$$

where

$$\begin{aligned}f(j) &= \sum_{i=1}^{j}\int_{\tau_j}^{\tau_{j+1}}\pi(t)[\tau_i(\alpha_{i-1}-\alpha_i)+(\beta_i-\beta_{i-1})]dt\\ &+ \sum_{i=j+1}^{k}\int_{\tau_j}^{\tau_{j+1}}\pi(t)[t(\alpha_{i-1}-\alpha_i)]dt.\end{aligned}$$

Putting the summations inside the integrals and collapsing the telescoping sums, the expression in (7) becomes:

$$\int_{\tau_j}^{\tau_{j+1}}\pi(t)cost(t)dt,$$

where

$$\begin{aligned}cost(t) &= (\beta_j-\beta_0)+\tau_1\alpha_0\\ &+ \sum_{l=1}^{j-1}(\tau_{l+1}-\tau_l)\alpha_l+(t-\tau_l)\alpha_l.\end{aligned}$$

Note that

$$(\beta_j-\beta_0)+\tau_1\alpha_0+\sum_{l=1}^{j-1}(\tau_{l+1}-\tau_l)\alpha_l+(t-\tau_l)\alpha_l$$

is exactly the energy expended by PLEA if the idle period t is in the range $[\tau_j,\tau_{j+1}]$. Thus, the expected cost for PLEA is:

$$\sum_{j=0}^{k}\int_{\tau_j}^{\tau_{j+1}}\pi(t)cost(t)dt \quad = \quad \sum_{i=1}^{k}\mathrm{O}N_i.$$

The proof that the expected offline cost is equal to $\sum_{i=1}^{k}\mathrm{O}FF_i$ is the same as the proof for the online cost, except that the integrals are separated into intervals according to the t_i's instead of the τ_i's.