Dynamic Power Management Using Adaptive Learning Tree *

Eui-Young Chung† Luca Benini‡ Giovanni De Micheli†

† {eychung,nanni}@stanford.edu
Stanford University
Computer Systems Laboratory
Stanford, CA 94305-4070, USA

‡ lbenini@deis.unibo.it
Università di Bologna
Dip. Informatica, Elettronica, Sistemistica
40136, Bologna, ITALY

Abstract

Dynamic Power Management (DPM) is a technique to reduce power consumption of electronic systems by selectively shutting down idle components. The quality of the shutdown control algorithm (power management policy) mostly depends on the knowledge of user behavior, which in many cases is initially unknown or non-stationary. For this reason, DPM policies should be capable of adapting to changes in user behavior. In this paper, we present a novel DPM scheme based on idle period clustering and adaptive learning trees. We also provide a design guide for applying our technique to components with multiple sleep states. Experimental results show that our technique outperforms other advanced DPM schemes as well as simple time-out policies. The proposed approach shows little deviation of efficiency for various workloads having different characteristics, while other policies show that their efficiency changes drastically depending on the trace data characteristics. Furthermore, experimental evidence indicates that our workload learning algorithm is stable and has fast convergence.

1 Introduction

The importance of system-level low-power design techniques has been increased by the widespread use of portable devices, which have limited battery life time [2, 4, 7]. *Dynamic power management* (*DPM*) [1] is a system-level low power design technique aiming at controlling performance and power levels of digital circuits and systems, by exploiting the idleness of their components. The heart of *DPM* is a *Power Manager* (*PM*) which monitors the overall system state and issues commands to control the power state of the system when it detects idleness. The control algorithm implemented by the *PM* is called a *power management policy*. *Adaptivity* is one of the most important issues in *DPM* because most external environments (user requests) are non-stationary.

Three classes of power management policies have been proposed in the past: time-out, predictive, and stochastic policies. The fixed time-out policy shuts down the system after a fixed amount of idle time [15]. Adaptive time-out policies are more efficient because they change the time-out according to the previous history. In contrast with time-out policies, predictive techniques do not wait for a time-out to expire, but shut down the system as soon as it becomes idle if they predict that the idle time will be long enough to amortize the cost of shutting down. Some predictive techniques are based on extensive off-line analysis of usage traces [8]. Adaptive prediction policies [3] overcome this limitation by adopting an exponential average prediction scheme. Both time-out and predictive policies have been applied only to systems with a single sleep state. A stochastic approach was proposed in [5] where general systems and user requests were modeled as Markov chains. This approach provides a polynomial-time exact solution for the search of optimal power management policies under performance constraints. The main drawback of this approach is the assumption that the Markov model of the workload is stationary and known. This limitation is addressed in [6], where adaptive Markov policies are investigated. In [6], a look-up table is constructed which contains pre-optimized stationary policies. Decisions are obtained by interpolation on the look-up table. The table is indexed by workload parameters which are estimated online with a sliding window algorithm. The main limitation of this adaptive technique is that it is based on a fixed-time update rule that can be power-inefficient. Furthermore, in some cases, neither the workload nor the system can be modeled accurately by the Markov chains.

In this paper, we present a novel adaptive predictive method applicable to systems (or components) with an arbitrary number of sleep states. Our policy is based on a new dynamic data structure called an *adaptive learning tree*. Using the tree, we can accurately predict the most appropriate low-power sleep state at the start of an idle period. Also, we propose an enhanced scheme which adopts a time-out filter for the purpose of eliminating very short idle periods from being candidates for prediction with minor power penalty. We tested out the technique on the well-known power management problem of hard disk spindown. Our experiments, with different hard-disk drives and measured workloads, show that our adaptive technique is robust and it consistently outperforms time-outs and predictive policies, in terms of both power savings and performance penalty.

2 Idle Period Grouping

In this section, we introduce the idle period clustering scheme which is the base of our proposed *DPM* approach for a multiple sleep-state system. A system can be abstracted as a two-state finite-state machine which is in busy state when a service is performed and it is in idle state otherwise. An idle period is defined as the period from the time when the system enters the idle state to the time when the system exits the idle state. Similarly, busy periods are the time intervals spent in busy state. Thus, the overall system behavior can be modeled as a time series of busy

* This work was supported in part by MARCO.

0-7803-5832-X /99/$10.00 ©1999 IEEE.

and idle periods. When an idle period is long enough to amortize the shutdown cost, the system can be shut down for power saving. For a system with a single sleep state, such as the one analyzed in [3] we can define a *threshold*, which is the minimum idle time required to reach the break-even point between shutdown cost and power savings. For a multiple sleep state system, we need as many thresholds as sleep states because each sleep state has a different shutdown cost. Figure 1 illustrates the need for multiple thresholds and the efficiency of multiple sleep states compared to the single sleep state when the system workload is known. Usually, shutting down to a deeper sleep state requires

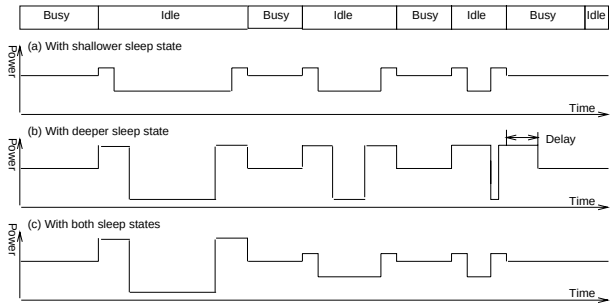


Figure 1. An example of system shutdown with multiple sleep states

more transition time and power consumption (i.e., higher cost). Energy consumption during the idle period can be calculated by estimating the area under the line corresponding to the selected sleep state. In Figure 1, the deeper sleep state is more efficient during the first idle period, but the shallower sleep state is more efficient during the second idle period. During the third idle period, by selecting the deeper sleep state, severe delay overhead and less power saving are observed. Finally, during the last idle period, no sleep state is helpful because the idle period is too short. From this example, it is obvious that multiple sleep states and multiple thresholds are required for more efficient *DPM*.

Let n be the number of sleep states; then the total number of power states in the system while it is idle is $n+1$ (*i.e.*, all sleep states plus the *fully on* state). Let $P = \{p_0, p_1, \cdots, p_n\}$ be the set of power states. The n threshold values (one for each sleep state) are determined based on the assumption that a deeper sleep state offers lower power consumption at the price of higher transition cost. For a given idle period, t_{idle}, energy consumption, E_i by selecting a power state p_i, $i = 0, 1, \cdots, n$, can be computed as follows.

$$E_i = td_i * pd_i + tu_i * pu_i + (t_{idle} - td_i - tu_i) * p_i \qquad (1)$$

Where, td_i is the transition time from idle state to power state p_i and tu_i is the transition time from power state p_i to idle state. Also, pd_i and pu_i are the power consumption levels corresponding to each transition and p_i is the power consumption while the system is in power state i. In our notation, power state p_i is a shallower sleep state than power state p_{i+1} ($p_i > p_{i+1}$) and power state 0 (p_0) is the idle state in which the system is not shut down. Hence, E_i should be greater than or equal to E_{i+1} and the equality holds when t_{idle} is the threshold between power state i and power state $i+1$. We can compute threshold values for every $i, i = 0, 1, \cdots, n$ by equating E_i with E_{i-1} and solving for t_{idle}. Let I_i be the threshold value between power state p_i and p_{i+1}. Then

$$I_i = \frac{(pd_{i+1} - p_{i+1}) * td_{i+1} + (pu_{i+1} - p_{i+1}) * tu_{i+1}}{p_i - p_{i+1}} - \frac{(pd_i - p_i) * td_i + (pu_i - p_i) * tu_i}{p_i - p_{i+1}} \qquad (2)$$

The time axis can be partitioned in $n+1$ disjoint intervals, bounded by the thresholds. We can then associate with a given idle period t_{idle} the index of the power state $IG(t_{idle})$ giving the best savings for that idle period:

$$IG(t_{idle}) = \begin{cases} 0 & if\ t_{idle} < I_0 \\ i+1 & if\ I_i < t_{idle} < I_{i+1} \quad for\ \ 0 \le i < n \\ n & if\ I_n < t_{idle} \end{cases} \qquad (3)$$

Thus, a sequence of idle periods can be transformed into a sequence of integers $0 \le IG(t_{idle}) \le n$, which represent the best power state that could be chosen for each idle period. Let s denote the sequence. If s has finite length l, it is denoted as s^l. Also, s_i denotes the ith value of the sequence and s_0 is the most recent event among all s_i's. The optimal power state for an idle period represented by s_i is p_{s_i}.

3 Adaptive Learning Tree

Predicting the values of a discrete event sequence is a fundamental problem in learning theory [11]. The idle period clustering technique mentioned in Section 2 transforms the sequence of idle periods into the sequence of discrete events. In other words, the problem to be solved is "which value will $IG(t_{idle})$ have in the next idle period for the current sequence s^l?" By predicting the next $IG(t_{idle})$, the system can choose the most appropriate sleep state. In previous studies, learning tree algorithms have been reported to find rules from experience [12, 13, 16, 17]. These algorithms are static in nature, and can be seen as techniques to organize knowledge and drive inference. To be effective, our algorithm must be highly dynamic, and be able to adapt rapidly to changes in the workload.

The learning tree that we propose can be applied to binary as well as multi-valued sequences. Idle periods are observed by the *PM* and they are transformed into integers $IG(t_{idle})$. This information can be seen as a sequence s^l. The *PM* predicts the next IG(t_{idle}) for the given s^l based on the current status of the learning tree. The learning tree is updated as soon as the prediction result is available. s^l is updated by shift operation whenever a new idle period is observed by *PM* such that $s_i \rightarrow s_{i+1}$ and the new value is stored as s_0. The basic assumption behind our algorithm is that we can predict the future idle periods with high accuracy by observing idle periods in the recent past. Our approach has some analogy with advanced branch prediction schemes widely used in computer architectures to reduce the penalty of mispredicted branches [14].

3.1 Basic Structure

An example of an adaptive learning tree is shown in Figure 2. The proposed adaptive learning tree consists of decision nodes (circles), history branches (solid lines), prediction branches (dashed lines), and leaf nodes (rectangles). The tree is levelized: the top decision node corresponds to s_0, nodes in the second level correspond to s_1 and so on. All leaf nodes are predictions for the next idle period regardless of their ancestor levels. Each leaf stores the Prediction Confidence Level (*PCL*). The higher the *PCL* is, the higher the confidence is for a prediction.

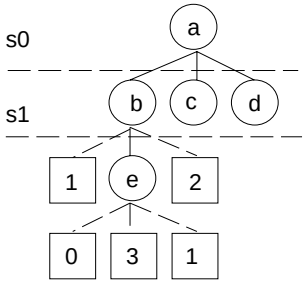


Figure 2. An adaptive learning tree (with two sleep states)

Each decision node can have both history branches and prediction branches, but the total number of branches is always n, and a prediction branch can only be used when the ancestor is a decision node and the descendant is a leaf node. Each branch of a decision node is associated with the index of a power state $IG(t_{idle}) = \{0, 1, \cdots, n\}$. From left to right, they are denoted as $b_i, i = 0, 1, \cdots, n$ regardless of their types.

3.2 Decision

A decision for a given sequence, s^l is taken based on a path matching procedure. A *path* for a given sequence, s^l is defined as a series of decision nodes such that from the top node, we recursively select a history branch b_{s_i} and move to the lower level decision node connected to b_{s_i}. The recursion is terminated when the b_{s_i} is a prediction branch or the level of the decision node corresponds to s_{l-1}. *Path length (pl)* is defined as the number of decision nodes included in the path. While matching the path, the leaf nodes connected to the decision node included in the path are checked and the leaf node which has the highest *PCL* is selected. When there are multiple leaf nodes which have the same highest *PCL*, the leftmost leaf node is selected. After path matching, the index of the selected leaf node becomes the prediction for the next event. For example, in Figure 2, the path, "$a \rightarrow b \rightarrow e$" is matched when the s^2 = "01" and its path length is 2. After path matching, the center leaf node of node e is selected, thus the tree predicts $IG(t_{idle}) = 1$ for the next idle period and issues a command to shut down the system to power state 1. Also, when the s^2 = "00" or s^2 = "02", the path, "$a \rightarrow b$" is matched. Note that node e is not included in the path for these sequences any more. Thus, the rightmost leaf node of node b is selected in this case. As shown in this example, pl, the number of old events used in decision is varied according to the given sequence. Also, two different sequences ("00" and "02") are classified in the same category and can share the resources of the tree to reduce memory usage.

3.3 Learning

In conjunction with prediction, a learning process is needed to maintain the accuracy of the prediction. Whenever an event s_i occurs, the tree is updated to reflect the quality of prediction made when the previous event s_{i+1} occurred. When the prediction is correct, the learning tree should be updated to increase the possibility to choose the same leaf node for the given sequence. In the reverse case, the reverse action should be performed. This task is achieved by updating the *PCL* of the leaf nodes. *PCL* update is controlled by a finite-state machine as shown in Figure 3 (the update rule is analogous to that employed in branch prediction buffers for conditional branch prediction). When the prediction is correct, the *PCL* state is changed to the higher state, in the reverse situation, the *PCL* state is changed to the lower state. And when it reaches either end state, it keeps the current state. Thus, the *PCL* is an adaptive feature of the learning tree for non-stationary event sequences. Learning process is more compli-

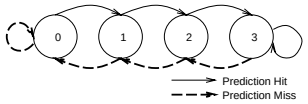


Figure 3. PCL operation

cated when misprediction occurs, because decreasing *PCL* of the selected node is insufficient. In other words, the adaptive learning tree has insufficient information to distinguish the given sequence from other sequences and the *PCL* of the leaf node which should have been selected (desired leaf node) is too low. Thus, two additional procedures are performed. Let us denote the desired value as $dv = IG(t_{idle})$. First, to increase distinguishability, increase the path length of the current path by replacing the leaf node on the prediction branch $b_{s_{pl+1}}$ connected to the last decision node in the path with a new decision node. Second, to increase the *PCL* of the desired leaf node, find all leaf nodes which are connected through the prediction branch, b_{dv} on the path and increase their *PCL*. An example of the updating procedure is shown in Figure 4. Suppose

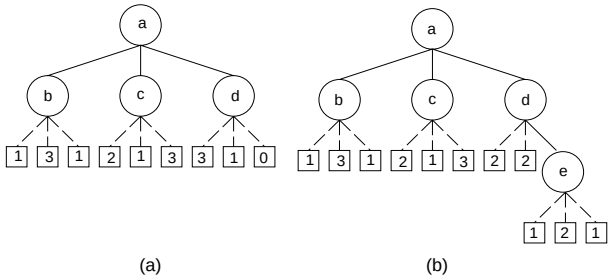


Figure 4. An example of learning for a prediction miss

there are three different sequences such that A = "20", B = "21" and C = "22" and the next event after sequence A and B is 0, but the next event after sequence C is 1. The path "$a \rightarrow d$" will be matched for all those sequences in Figure 4 (a) and the learning tree will predict 0 for every sequence. This prediction is correct when the given sequence is A or B; it is wrong when the given sequence is C. Thus, it is necessary to distinguish sequence C from A and B. When this prediction miss occurs, first, the *PCL* of the leftmost leaf node of node d is decreased. Then, the rightmost leaf node of node d is replaced with a new decision node because $s_1 = 2$. The leaf nodes of the new decision node have the initial *PCL* value (in this case, it is 1). Then, the second additional procedure is applied and the final *PCL* of leaf nodes are as shown in Figure 4. Due to these additional procedures, the adaptive learning tree grows in an unbalanced manner and this characteristic is efficient for keeping it small and naturally determines the correlation depth between the future event and old history depending on the sequence characteristics.

4 Power Manager

As mentioned in Section 1, the Power Manager (*PM*) is the heart of *DPM*. Thus, the adaptive learning tree is implemented

within the *PM* as shown in Figure 5. In Figure 5, service requester

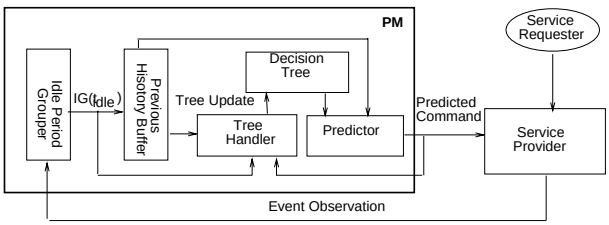


Figure 5. Power Manager Configuration

(*SR*) is the external environment which triggers the system and the service provider (*SP*) is the system itself which serves the requests from service requester. The idle period grouper (*IPG*) observes *SP* and extract idle periods. Then the idle interval for the observed idle period is calculated and it is passed to the previous history buffer (*PHB*) and the tree handler. *PHB* stores the observed idle sequence s^l. Whenever a new event arrives, it performs shift operation such that $s_i \rightarrow s_{i+1}$ and the new event is stored as s_0. Finally, the tree handler performs learning and the predictor performs the decision process as mentioned in Section 3.

Wakeup and miss correction

SP can be waken up in two different ways. One is when *PM* detects a new service request. The other is when *SP* stays in power state p_i longer than $I_i - tu_i$. The first case occurs when the predicted idle interval is greater or equal to the actual idle interval. And the second case occurs when the predicted idle interval is less than the actual idle interval. The second case is a prediction miss due to a conservative prediction. After *SP* is waken up, *PM* monitors the system until I_n (maximum threshold). During this period, if a new service request comes, the *SP* can serve this request without wakeup penalty, thus the inefficiency in power saving is compensated by eliminating wake-up performance penalty. Otherwise, the *PM* shuts down the *SP* to the deepest power state to save more power. This feature enables the exploitation of very long mispredicted idle periods.

Prediction filter

In many applications, the distribution of idle period intervals shows L-shaped curve as mentioned in [3, 8] which represents the ratio of very short idle periods is dominant in total idle periods distribution. Thus, the prediction quality for short idle periods can play an important role in deciding overall prediction accuracy. For this reason, fixed time-out policy is performed preceding the actual prediction. In other words, the command predicted by *PM* is not issued immediately, but the command issue is delayed for a small amount of time (threshold of the fixed time-out policy) to filter out very short idle periods. If a request is arrived during this waiting period, the predicted command is canceled, thus only the idle periods longer than the threshold can be used for shutdown. The threshold value used for the fixed time-out policy is the minimum threshold, I_0. Usually, I_0 is small, thus the sacrifice to filter out short idle periods is not a big penalty for power saving, but it prevents excessively aggressive shutdown.

5 Experimental Results

We applied the proposed scheme to two different Hard Disk Drives [5, 10] with the real trace data [9]. We chose two different types of disk traces from [9] - one is the trace for swap purpose

IBM in [5]			
State	ΔT	*Power*	*Threshold*
active	NA	2.5*W*	*NA*
idle (p_0)	*NA*	1*W*	*NA*
idleLP (p_1)	40*ms*	0.8*W*	680*ms*
standby (p_2)	2.2*sec*	0.3*W*	19088*ms*
sleep (p_3)	6*sec*	0.1*W*	95600*ms*
Toshiba in [10]			
active	NA	2.5*W*	*NA*
idle (p_0)	*NA*	0.9*W*	*NA*
standby (p_1)	1*sec*	0.3*W*	10667*ms*
sleep (p_2)	3*sec*	0.1*W*	70000*ms*

Table 1. HDD specifications

only disk and the other is the trace for swap and user data disk. Thus, the distributions of idle period length are different. Two different HDD specifications are shown in Table 1 with the threshold values computed by the equation 2. We implemented a simulator to estimate the performance of the proposed algorithm in terms of power consumption, delay overhead, and energy efficiency. Also, the simulator can simulate fixed time-out policies, the *best oracle* policy [6], and other predictive policies [3] for validation purpose. The best oracle policy is an ideal policy which cannot be implemented in practice because it assumes perfect knowledge of all idle periods, and it always takes the best decision. For the proposed approach, the size of *PHB* is determined 20 bits, thus the maximum path length of the adaptive learning tree was constrained to be less than or equal to 20. Since fixed time-out policy and the prediction policy in [3] does not support multiple sleep states, only the deepest sleep state is used for those policies.

The compared policies are: 1) best oracle ($O1$), 2) proposed approach without filter ($M1$), 3) proposed approach with filter ($M2$), 4) prediction policy in [3] with miss correction ($H1$), 5) $H1$ with pre-wakeup ($H2$), 6) time-out policy with time-out value = I_0 ($T1$), 7) time-out policy with time-out value = 1*sec* ($T2$), and 8) time-out policy with threshold value that is used in $H1$ ($T3$). $O1$ is the reference in comparison because any other shutdown technique cannot outperform $O1$ and $H2$ has the pre-wakeup feature in addition to the features of $H1$. Several quality measures as shown below were obtained from the simulation.

- **Hit ratio(*HR*):** is defined as the ratio between # of correct prediction to # of total prediction. Thus, it is not used for the fixed time-out policies. Also, the hit ratio of proposed approach can not be directly compared to that of [3] because they have different number of sleep states (unit: %).
- **Avg. power(*AP*):** is the average power consumption during *SP* is in idle state (unit: *W*).
- **Delay Overhead(*DO*):** is the ratio between the increased idle time after applying the policy and original idle time(unit: %).
- **Avg. delay / idle period(*AD*):** is the ratio between total increased idle time and total number of idle periods. It is a good quality measure for instant availability (unit: *sec*).
- **Energy(*EN*):** is the total energy consumed during idle periods normalized to the energy consumption by best oracle policy (unit: *J*).
- **Efficiency(*EF*):** is the ratio between the normalized energy in $O1$ and that of each policy. It represents well the efficiency of the policy compared to the ideal policy and good for considering the power saving and performance penalty together.

The simulation results are shown in Table 2. First, the effect of

IBM in [5]: Trace data 0 (swap and user data purpose)								
	O1	M1	M2	H1	H2	T1	T2	T3
HR	100	85.6	96.2	97.5	97.5	-	-	-
AP	0.172	0.217	0.194	0.298	0.998	0.247	0.244	0.234
DO	0.0	1.0	1.0	1.8	1.3	6.1	5.9	1.8
AD	0.0	0.227	0.236	0.428	0.294	1.440	1.393	0.413
EN	1.000	1.273	1.136	1.561	5.202	1.527	1.504	1.384
EF	1.000	0.786	0.880	0.641	0.192	0.655	0.667	0.723
IBM in [5]: Trace data 1 (swap only purpose)								
	O1	M1	M2	H1	H2	T1	T2	T3
HR	100	84.5	94.8	60.7	60.7	-	-	-
AP	0.125	0.148	0.132	0.227	1.014	0.132	0.128	0.149
DO	0.0	0.6	0.5	2.2	1.7	1.3	1.1	1.0
AD	0.0	1.525	1.420	5.891	4.678	3.530	3.093	2.741
EN	1.000	1.190	1.067	1.855	8.242	1.069	1.038	1.203
EF	1.000	0.840	0.937	0.539	0.121	0.935	0.963	0.831
AE	1.000	0.813	0.909	0.590	0.157	0.800	0.815	0.777
Toshiba in [10]: Trace data 0 (swap and user data purpose)								
	O1	M1	M2	H1	H2	T1	T2	T3
HR	100	91.0	99.3	97.6	97.6	-	-	-
AP	0.158	0.200	0.186	0.249	0.898	0.205	0.206	0.234
DO	0.0	0.6	0.5	1.0	0.7	2.1	3.0	1.5
AD	0.0	0.135	0.109	0.234	0.159	1.486	1.705	0.360
EN	1.000	1.273	1.183	1.458	5.242	1.325	1.343	1.503
EF	1.000	0.786	0.845	0.686	0.191	0.755	0.744	0.686
Toshiba in [10]: Trace data 1 (swap only purpose)								
	O1	M1	M2	H1	H2	T1	T2	T3
HR	100	87.6	98.0	59.9	59.9	-	-	-
AP	0.118	0.133	0.126	0.192	0.915	0.125	0.121	0.135
DO	0.0	0.3	0.3	1.1	0.9	0.5	0.6	0.5
AD	0.0	0.803	0.685	3.011	2.387	1.403	1.553	1.381
EN	1.000	1.131	1.071	1.645	7.819	1.065	1.032	1.150
EF	1.000	0.884	0.934	0.607	0.128	0.939	0.970	0.870
AE	1.000	0.835	0.890	0.647	0.160	0.847	0.857	0.778

*AE: Average of EF for trace data 0 and EF for trace data 1

Table 2. Comparisons of the various policies

filter is shown from $M1$ and $M2$. In $M2$, by filtering out very short idle periods, the hit ratio is increased by about 10% and this is also reflected in efficiency. Second, $M2$ shows comparable or better hit ratio compared to $H1$, even though $M2$ is in harder situation to increase the hit ratio because it has more choices than $H1$. And, the correction after miss is considered as hit in $H1$. Nevertheless, the efficiency of $H1$ is much lower than that of $M2$. Also it is observed that the hit ratio of $H1$ is drastically decreased when trace data 1 is simulated. This is an indication that the non-stationary property of trace data 1 is much stronger than that of trace data 0. In contrast, the hit ratio of $M2$ is decreased by about 1%. Thus, the proposed approach adapts well according to the variation of SR. $H2$ has very poor efficiency even though the hit ratio is same to that of $H1$. This is because the pre-wakeup scheme wakes up SP even when it meets very long idle periods. Third, in average, $M2$ outperforms any other time-out policy by about $5-17\%$. When trace data 1 is applied, policy $T2$ is slightly better than $M2$(1%). This fact can be explained by the distribution of idle intervals as shown in Figure 6. For IBM HDD, the ratio of

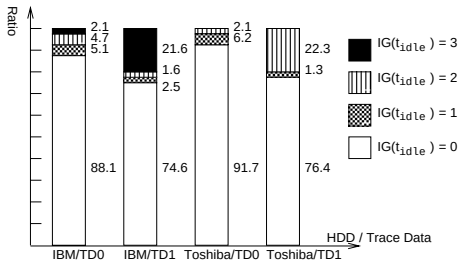
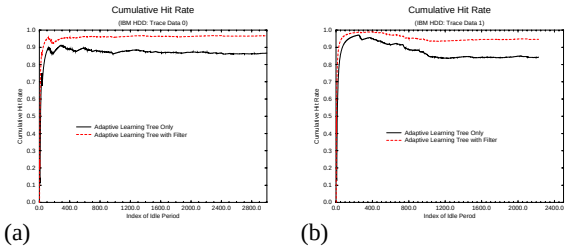


Figure 6. Distribution of Idle intervals

idle periods in *idle intervals 1 and 2* in trace data 0 is two times more than that in trace data 1. Also, for Toshiba HDD, the ratio of



(a) (b)

Figure 7. (a) Cumulative hit ratio(IBM HDD: trace data 0) (b) Cumulative hit ratio(IBM HDD: trace data 1)

Trace data 0	2nd case		3rd case	
	O1	M2	O1	M2
Hit ratio (%)	100.0	98.1	100.0	99.3
Avg. power (W)	0.183	0.194	0.174	0.213
Delay overhead (%)	0.0	0.9	0.0	0.9
Avg. delay / idle period(s)	0.0	0.206	0.0	0.211
Energy (J)	1.0	1.066	1.0	1.235
Efficiency	0.940	0.882	0.989	0.801
Trace data 1	2nd case		3rd case	
	O1	M2	O1	M2
Hit ratio (%)	100.0	96.5	100.0	97.3
Avg. power (W)	0.125	0.129	0.125	0.138
Delay overhead (%)	0.0	0.5	0.0	0.5
Avg. delay / idle period(s)	0.0	1.450	0.0	1.359
Energy (J)	1.0	1.034	1.0	1.110
Efficiency	1.0	0.967	1.0	0.901
Average Efficiency (IBM)	0.970	0.925	0.995	0.851

Table 3. Comparisons for design guide

idle periods in *idle interval 1* is about four times more than that in trace data 1. It means trace data 1 rarely has intermediate length of idle periods, In other words, the idle periods in trace data 1 are either very short or very long because it is the trace of swap operation only. For this reason, the fixed time-out policy shows good efficiency for trace data 1. But for trace data 0, the efficiency of fixed time-out policy degrades rapidly because the ratio of the intermediate length of idle periods can not be ignored, while the proposed approach shows almost same efficiency. From these results, we can conclude that the proposed approach has superior reliability.

Next, we tested the adaptive speed of our proposed approach and the stability over the time. We used the hit ratio variation over the time for this purpose. As shown in Figure 7, the proposed approach achieves high hit ratio after experiencing less than 1000 idle periods. Moreover, after it reaches high hit ratio, the variation of hit ratio is very small. The variation in trace data 1 is somewhat larger than trace data 0 because the non-stationary property of trace data 1 is stronger than trace data 0.

Next, we performed another experiment to provide a design guidance in deciding number of sleep states and sleep levels. For this purpose, we simulated three different cases for IBM HDD. First case is the same case as in the above experiment. In second case, the standby sleep mode (power state p_2) is eliminated and in third case, the idleLP sleep mode (power state p_1) is not used. The simulation results are shown in Table 3. To avoid duplication with Table 2, the results of the first case is omitted in Table 3. From the comparison of the best oracle policies, it is shown that increasing the number of sleep states is ideally more efficient. This is because increasing the number of sleep states enables to handle more various lengths of idle periods. Also, it

shows choosing deeper intermediate sleep state (standby instead of idleLP) makes it possible to save more power. This situation depends on the distribution of idle intervals. In this experiment, deeper sleep states are preferred, because idle periods in both *idle intervals 1 and 2* have similar ratios in the distribution. Nevertheless, the $M2$ of the second case shows better efficiency than the third case. The reason is that third case wastes more idle periods when filtering out impulse-like idle periods because its I_0 is much larger than the I_0 of the second case. Even though the hit ratio is increased by a large I_0 (because of perfectly filtering out short idle periods), the increased ratio is only a small amount because the proposed approach already preserves high hit ratio. Thus, to adopt the proposed approach, choosing shallower sleep states or choosing deeper sleep state with small time-out value for filtering is recommended. And it is also shown that increasing the number of sleep states is not always the best choice, because it increases the difficulty of the decision process. The results of $M2$ from the first case and second case supports this argument because their efficiency is almost same and the hit ratio of the first case is lower than second case.

Finally, the proposed adaptive learning tree algorithm was implemented on the ACPI-compliant [18] Pentium II laptop computer with a Fujitsu MHF 2043 hard disk. The operating system running on the computer was a beta version of Microsoft Windows V5. The proposed algorithm was written in C language and easily ported on the computer thanks to the software controlled power management architecture introduced in [19]. Also, fixed time-out policy was implemented in the same environment for the comparison purpose. the threshold value used for the fixed time-out policy was 30 seconds as suggested in [20]. The workload trace used in this comparison was collected for two different users and its length was about 11 hours. As expected, the hit rate of the proposed algorithm was 94.5% for the given trace and the average power consumption while using the proposed algorithm was 8% less than that using the fixed time-out policy. But the delay overhead of the proposed algorithm was 0.5% larger than that of the fixed time-out policy. Notice that the hard disk used in this experiment provides only a single sleep state. We believe that both average power consumption and delay overhead of the proposed algorithm will outperform the fixed time-out policy by a larger margin when both algorithms are applied to the multiple sleep state hard disk.

6 Conclusion

In this paper, we presented a novel power management policy which is useful for multiple sleep state components. The proposed approach is based on an adaptive learning tree and idle period clustering, and it has been validated through extensive experiments using two different HDD models and two kinds of real disk trace data. The experimental results show that the proposed approach outperforms fixed time-out policy and other prediction methods. Also, it is shown that the prediction accuracy is reliable in the sense that the proposed approach is much less affected by strongly non-stationary workloads. Moreover, the proposed approach reaches reasonable hit ratio before experiencing more than 1000 idle periods. Finally, we implemented the proposed algorithm on a laptop computer with a power-manageable hard disk and showed its feasibility in a real system environment.

References

[1] L. Benini and G. De Micheli, *Dynamic Power Management of Circuits and Systems: Design Techniques and CAD Tools*, Kluwer, 1997.

[2] A. Chandrakasan and R. Brodersen, *Low-Power Digital CMOS Design*. Kluwer, 1995.

[3] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", in *Proceedings of the Int.l Conference on Computer Aided Design*, pp. 28-32, 1997.

[4] W. Nebel and J. Mermet (Eds.), *Low-Power Design in Deep Submicron Electronics*. Kluwer, 1997.

[5] 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, June 1999.

[6] E.Y. Chung, L. Benini, A. Bogliolo and G. De Micheli, "Dynamic Power Management for non-stationary service requests" *DATE - Proceedings of the Design Automation and Test in Europe Conference and Exibition*, 1999, pp. 77-81.

[7] J. M. Rabaey and M. Pedram (editors), *Low-Power Design Methodologies*. Kluwer, 1996.

[8] M. Srivastava, A. Chandrakasan. R. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation", *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 42-55, March 1996.

[9] John Wilkes, "HP Laborataries Disk I/O traces", available at http://www.hpl.hp.com/personal/John_Wilkes/traces (1997)

[10] Toshiba America, "HARD DISK DRIVES", available at http://www.toshiba.com/taecdpd/hddover.htm#6.4 (1999)

[11] N. Cesa-Bianchi *et al*, "On Bayes Methods for On-Line Boolean Prediction", *Algorithmica*, Vol 22, pp.112-137, 1998

[12] N. H. Bshouty *et al*, "On Learning Decision Trees with Large Output Domains", *Algorithmica*, Vol 20, pp.77-100, 1998

[13] A. Ehrenfeucht and D. Haussler, "Learning Decision Trees from Random Examples", *Information and Computation*, Vol 82, pp.231-246, 1989

[14] D.A. Patterson and J. L. Hennessy, Computer Architecture A Quantative Approach, 2nd edition, Morgan Kaufmann Publishers, 1996

[15] K. Li, R. Kumpf, P. Horton, and T. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers", *USENIX Winter Conference*, 1994, pp. 279-292.

[16] J.R. Quinlan and R. Rivest, "Inferring Decision Trees Using the Minimum Description Length Principle", *Information and Computation*, Vol 80, pp.227-248, March, 1989

[17] J.R. Quinlan, "Induction of Decision Trees", *Machine Learning*, Vol 1, pp.81-106, March, 1986

[18] ACPI, "ACPI: Advanced Configuration & Power Interface", available at http://www.teleport.com/ãcpi (1999)

[19] Y.-H. Lu, T. Ŝimunic̀ and G. De Micheli, "Software Controlled Power Management", *International Workshop on Hardware/Software Codesign*, pp.157-161, 1999

[20] P. Greenawalt, "Modeling Power Management for Hard Disks", *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp.62-65, 1994