Dynamic Power Management Using Machine Learning

Gaurav Dhiman
Department of Computer Science and Engineering
University of California, San Diego
gdhiman@cs.ucsd.edu

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego
tajana@ucsd.edu

ABSTRACT
Dynamic power management (DPM) work proposed to date places inactive components into low power states using a single DPM policy. In contrast, we instead dynamically select among a set of DPM policies with a machine learning algorithm. We leverage the fact that different policies outperform each other under different workloads and devices. Our algorithm adapts to changes in workloads and guarantees quick convergence to the best performing policy for each workload. We performed experiments with a policy set representing state of the art DPM policies on a hard disk drive and a WLAN card. Our results show that our algorithm adapts really well with changing device and workload characteristics and achieves an overall performance comparable to the best performing policy at any point of time.

Keywords
Dynamic Power Management, Machine Learning

1. INTRODUCTION
Power consumption is a key issue in the design of computing systems today. While battery driven systems need to meet an ever increasing demand for performance with a longer battery life, high performance embedded systems contend with the issues of heating. Dynamic power management (DPM), defined as the selective shutdown of system components that are idle, has proven to be an effective technique for reducing system power dissipation. An effective DPM policy must maximize power savings while keeping performance degradation within acceptable limits. Design of such policies has been an active research topic. A number of heuristic and stochastic policies have been proposed in the past. All these policies tackle the DPM problem by selecting an appropriate timeout value after which the device can be put to sleep. This timeout can be fixed, adaptive or randomized. While simpler DPM policies like timeout and predictive policies do it heuristically with no performance guarantees, more sophisticated stochastic policies guarantee optimality for stationary workloads. Policies can outperform each other under different workloads and devices.

In this paper we propose a novel DPM technique that optimally exploits the existing DPM policies to achieve adaptability with varying workloads. The premise is to take a set of DPM policies and design a control algorithm that selects the best suited one for a given idle period. The control algorithm design is critical here since it bears the responsibility of effective evaluation and selection of policies, which directly determines the overall performance of the entire scheme. We employ a machine learning algorithm [2] to perform this control activity. The machine learning algorithm (referred to as "*controller*") has a set of DPM policies (referred to as "*experts*") to choose from and selects an expert which has the best chance to perform well for the current idle period. The controller evaluates the performance of the experts at the end of each idle period and based on that decides which expert should be activated next. It takes into account both power savings and performance penalty for this calculation. The controller can be employed separately on multiple devices in the system with each device having its own set of experts. Our machine learning algorithm will guarantee performance that is close to that of the best available expert for each device.

We implemented the controller with a set of experts representing state of the art DPM policies, for controlling power consumption of a hard disk drive and WLAN card under different workloads. We observed that under conditions where experts give mixed performance, the machine learning algorithm adapts to select the best performing expert at any point of time and delivers an overall performance (in terms of power savings and performance penalty) better than that of any single expert. Under conditions where a single expert consistently performs better than the other, the algorithm continuously selects it to achieve comparable results. The scheme, however, involves the overhead of performance evaluation of experts at the beginning of the active period. According to our experimental results the overhead is negligible (as low as 0.0001% of the total timeframe) when compared to the flexibility and the benefits offered. Thus, the use of machine learning algorithm with a carefully selected set of experts presents a novel, adaptive and robust DPM mechanism that can achieve good performance for a wide range of applications.

2. PREVIOUS WORK
A lot of research effort has been devoted to explore different DPM policies in the past. The existing DPM policies can be broadly classified into heuristic and stochastic policies. While the heuristic policies are simple to implement they provide no guarantees on the power/performance trade-off they offer. Stochastic policies are more complex to implement but they do guarantee optimality and performance bounds. However, the guarantee is only for stationary workloads and thus stochastic policies cannot adapt optimally.

A simple heuristic policy is a timeout policy, wherein a device is put to sleep if it is idle for more than a specified timeout period. The timeout period might be fixed [14] or adaptive [3] [4]. For instance, in [14], the device is put to sleep if it is idle for more

ICCAD'06, November 5-9, 2006, San Jose, CA

than T_{be} (break-even time). T_{be} is the minimum period for which the device should be turned off so that it compensates for the overhead associated with shutting down and turning on the device and saves energy compared to the case if it remains on. A drawback of such policies is that they waste energy while waiting for the timeout to expire. Predictive policies predict the duration of upcoming idle period as soon as the component goes idle. A shutdown decision can be made if the prediction indicates a long idle period. Srivastava et al. [5] proposed a policy, which uses a regression equation based on the component's previous active and idle periods to predict the current idle period length. In [6], Hwang and Wu used exponential average of predicted and actual lengths of the previous idle period to predict the current idle period length. In [7], Chung et al propose a scheme that uses an Adaptive Learning Tree to store a sequence of idle periods and is capable of managing multiple low power states. The algorithm predicts idle periods using finite state machines and selects the best suited low power state. Predictive policies, however, perform well only when the requests have a high degree of correlation. In summary, heuristic policies tend to be easy to implement and in some cases are adaptive. However, they do not offer any guarantee on optimality and by in large do not take performance overhead into account.

Stochastic policies take into account both power consumption as well as performance penalty. They model the request arrival and device power state changes as stochastic processes. Minimizing power consumption and delays then become stochastic optimization problems. In [8], Paleologo et al assume the arrival of requests as a stationary geometric distribution and model power management as a discrete-time Markov decision process. In [9], the work is extended to handle non-stationary request arrival. Continuous Time Markov Decision Process models arrival time as exponential distribution and runs the policy on event occurrences rather than at discrete time intervals [10]. Other proposed models include semi markov decision process model (SMDP) [11] and Time Indexed Markov Chain SMDP model (TISMDP) [12]. SMDP models transitions to and from low power state using uniform distributions, while the request arrivals are exponential. TISMDP models request arrival as a pareto distribution while the transition times of the service provider between the power states is modeled as a uniform distribution. The algorithm has low computation overhead and performs well when implemented on real devices [13]. Stochastic policies offer optimality for the power/performance tradeoff they have been derived for, but they do not keep their optimality properties as workloads become non-stationary, thus they have limited adaptability.

The main contribution of our work is that instead of designing a new power management policy, we take a set of well-known policies, each of which performs well for a given set of conditions and design a *policy selection mechanism*. The primary motivation for this work comes from the observation that no single policy fits perfectly all operating conditions. Our proposed scheme uses machine learning algorithm that guarantees to do nearly as well as the best performing policy on a given workload. This approach gives us the power to adapt with changing workloads and give an overall performance that is better than any single policy can offer. Ren et al. propose a similar setup in [16]. They design a hierarchical architecture, where the bottom layer is a set of stationary optimal DPM policies, precalculated offline from policy optimization in Markov decision processes, while the top layer adaptively switches among these stationary policies. In comparison, our work is more generic in nature since the policy set can comprise of any DPM policy rather than just a stationary stochastic policy. Moreover, the machine learning algorithm provides theoretical guarantee on overall performance converging to that of the best performing policy in the policy set. The basic idea of policy selection resembles that of hybrid branch predictors employed in microprocessors [17] [18].

The rest of the paper is organized as follows. Section 3 explains our system model. We then explain our experimental setup and results in Section 4. Finally we summarize our findings in Section 5.

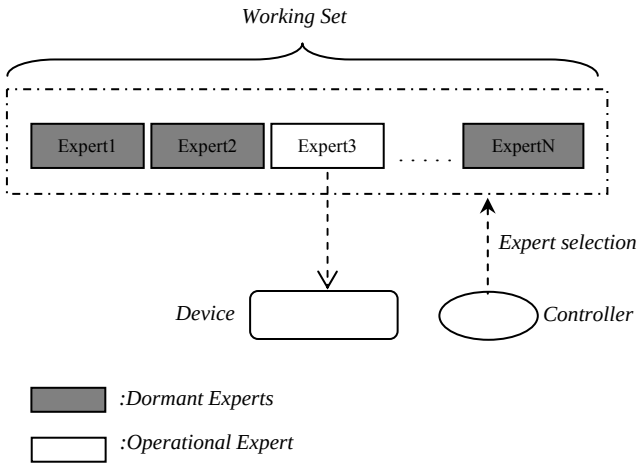

Figure 1. System Model

3. SYSTEM MODEL

The system we are modeling, shown in Figure 1, consists of three entities: controller (the machine learning algorithm), experts (the DPM policies) and the device whose power is being managed. The set of experts is collectively referred to as the "*working set"*. An expert can be a fixed timeout policy, an exponential predictive policy, a stochastic policy etc. When the device is busy, all the experts are inactive and thus are referred to as "*dormant experts"*. When an idle period occurs, the controller activates the expert that has the highest probability to perform well. This selected expert is referred to as the "*operational expert"*. The operational expert takes control over the device and makes the power management decisions for that idle period. For instance, in the scenario depicted in Figure 1, Expert3 is the operational expert and is managing power for the current idle period. After the idle period finishes, the operational expert returns to its default dormant state. This process is repeated for all idle periods. Note that for any idle period only one expert can be operational. This model can be independently applied to multiple devices in the system. Each device will have its own controller and working set, and the controller will select the most suited expert as the operational expert for the device it is controlling.

3.1 Machine Learning Algorithm for DPM

The most critical task in our methodology is the evaluation and selection of experts. The controller employs a machine learning

algorithm for this purpose. The algorithm is an adaptation of Freund and Schapire's on-line allocation algorithm [2]. Figure 2 contains the pseudo-code for the algorithm we use for the controller. The controller has *N* experts to choose from; we number these using the integers *i* = 1,2…..,N. The experts can be any DPM policy. The algorithm associates and maintains a weight vector $\mathbf{w}^t$ with the experts, where $\mathbf{w}^t = <w^t_1, w^t_2, \ldots w^t_N>$ consists of weight factors corresponding to each expert. The value of weight factor reflects the performance of an expert, with a higher value indicating a better performance. All of the weights of the initial vector $\mathbf{w}^1$ sum to one, as seen in Figure 2. Note that the weights need not sum to one after a few idle periods. In our implementation, we assign equal weights to all the experts at initialization.

To perform expert selection, the controller maintains a probability vector $\mathbf{r}^t$, that is obtained by normalizing the weight vector as shown below:

$$\mathbf{r}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$$

The probability vector, $\mathbf{r}^t = <r^t_1, r^t_2, \ldots r^t_N>$ where $0 \le r^t_i \le 1$, consists of probability factors associated with each expert for idle period '*t*'. At any point of time the best performing expert has the highest probability factor amongst all the experts. Thus the controller simply selects the expert with the highest probability factor as the operational expert for the next idle period. If the probability factor of multiple experts is equal, then it randomly selects one of them (step 1 in Figure 2). When the idle period occurs, the operational expert takes control of the device and takes the power management decision (step 2 in Figure 2).

Once the idle period finishes, the algorithm evaluates the performance of all the experts (step 3 in Figure 2). Dormant experts are evaluated on the basis of how they would have performed had they been selected. The evaluation takes into account both the energy savings and the performance delay. We evaluate loss with respect to an ideal offline policy that has zero delay and maximum possible energy savings. The loss incurred by each expert is collectively referred to as the loss vector $\mathbf{l}^t$. The value of loss factor (l^t_i) for each expert is influenced by the relative importance of energy savings and performance delay as expressed by factor α ($0 \le \alpha \le 1$). If l^t_{ie} and l^t_{ip} are the loss factors corresponding to energy savings and performance delay for an expert '*i*', then the joint loss factor is given by:

$$l^t_i = \alpha\, l^t_{ie} + (1 - \alpha)\, l^t_{ip}$$

Algorithm **Controller**

Parameters: $\beta \in [0,1]$

Initial weight vector $w^1 \in [0,1]^N$,

such that $\sum_{i=1}^{N} w_i^1 = 1$,

Do for *t* = 1,2,3…..

1. Choose expert with highest probability factor in $\mathbf{r}^t$.

$$\mathbf{r}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$$

2. Idle period starts -> operational expert performs DPM
3. Idle period ends -> evaluate performance of experts
4. Set the new weights vector to be

$$w_i^{t+1} = w_i^t \beta^{l_i^t}$$

Figure 2. Algorithm Controller

In our implementation we determine the energy loss l^t_{ie} by comparing the length of the idle period with the sleep time. If it is less than T_{be} (break-even time, defined in Section 2), then we do not save energy and thus $l^t_{ie} = 1$. For the values of sleep time T_{sleepi} of an expert *i* greater than T_{be} , and idle period, T_{idle} we use the following equation:

$$l^t_{ie} = 1 - T_{sleepi}/T_{idle}$$

Calculation of performance loss, l^t_{ip}, is based on whether the device sleeps or not. If the expert makes the device sleep, $l^t_{ip} = 1$ since we incur performance delay upon wakeup, otherwise it is set to 0. The loss calculation process is the step 3 in Figure 2. The final step in the algorithm involves updating the weight factors for each expert on the basis of the loss they have incurred:

$$w_i^{t+1} = w_i^t \beta^{l_i^t}$$

Thus, the weight factors corresponding to experts with higher loss factors get reduced while for the experts with lower loss factors get increased by this simple multiplicative rule. This gives higher probability of selecting the better performing experts in the next idle period. The value of β can be set between 0 and 1. The criterion for selecting the appropriate value is explained in [2]. For our experiments we used β = 0.75.

Once the weights are updated we are again ready to select the operational expert for next idle period by calculating the new probability vector $\mathbf{r}^t$ using step 1 in Figure 2. Note that all calculations related to selecting the operational expert are performed during the active periods and hence no overhead is incurred when the idle period actually occurs.

3.2 Performance Bound of Controller

From the previous sub-section we know that l^t_i is the loss incurred by each expert for the idle period *t*. Hence, the average loss incurred by our scheme for a given idle period '*t*' is:

$$\sum_{i=1}^{N} r_i^t l_i^t = \mathbf{r^t} . \mathbf{l^t}$$

The goal of this algorithm is to minimize its cumulative loss relative to the loss suffered by the best expert. That is, the controller attempts to minimize the *net loss*

$$L_G - \min_i L_i$$

where,

$$L_G = \sum_{t=1}^{T} \mathbf{r^t} . \mathbf{l^t}$$

is the total loss incurred by controller on T trials, and

$$L_i = \sum_{t=1}^{T} l_i^t$$

is individual expert *i*'s cumulative loss over T trials. It can be shown [2] that net loss of the algorithm is bounded by $O\left(\sqrt{T \ln N}\right)$ or that the average net loss per period decreases at the rate $O\left(\sqrt{(\ln N)/T}\right)$. Thus, as *T* increases, the difference decreases to zero. This guarantees that the performance of the machine learning algorithm is close to that of the best performing expert for any workload. This is in contrast to single policy based solutions which either adapt heuristically to changing workloads or guarantee optimality only for stationary workloads.

4. RESULTS

In this section we give the results we obtained with our controller algorithm on different devices commonly present in a wide range of embedded system with varying real workloads. We show how the controller performs with an "expert" set representing state of the art DPM policies. We use varying workloads and two devices to show that our methodology does well under different conditions. We also show that we can achieve reasonably good results with a working set of simple fixed timeout policies. The results indicate that our controller is capable of dynamically adapting while delivering sizeable energy savings over a range of power/performance tradeoff settings.

4.1 Experimental Setup

We performed our experiments using two devices: HP 2200A hard disk drive (HDD) and Cisco Aironet 350 series Wireless Adapter (WLAN) with workloads having different characteristics. For HDD we used traces originally collected in [15]. For WLAN we collected traces for different applications by running tcpdump on an XScale platform running Linux 2.4.21. The characteristics of the workloads selected for the experiments are described in Table 1. This is a broad range of workload characteristics. For example, HP-1 and HP-2 traces have very different interarrival times in terms of both average value and standard deviation ($\overline{t_{RI}}$ and $\sigma_{t_{RI}}$ respectively).

We run traces described in Table 1 for both the HDD and WLAN and then record the performance in terms of energy savings and performance delays for both the individual experts as well as the controller. Table 2 lists the power characteristics of both devices. P_{on} and P_{sleep} refer to the power consumed while the devices are on and in the sleep state respectively. P_{tr} is the power consumed in transitioning to and from the sleep state while T_{tr} is the time

Table 1 Workload characteristics

Device	Trace Name	Duration (in sec)	$\overline{t_{RI}}$	$\sigma_{t_{RI}}$
HDD	HP-1Trace	32311	20.5	29
	HP-2 Trace	35375	5.9	8.4
	HP-3 Trace	29994	17.2	2
WLAN	Web Surfing	4720	0.16	0.65
	Telnet	2767	0.16	0.49
$\overline{t_{RI}}$: Average Request Inter-arrival Time (in sec)				

taken for this transition. T_{be} refers to the break even time. The power/performance characteristics of the two devices are quite different from each other. For instance WLAN has a T_{be} that is less than half of that for HDD. For our experiments we assumed HDD to be idle after 1s of inactivity and WLAN after 100ms of inactivity. Once an idle period is detected, the controller is invoked to select the operational expert.

For our working set we selected fixed timeout, adaptive timeout [3], exponential predictive [6] and TISMDP [12] policies, representing different classes of state of the art DPM policies. While fixed and adaptive timeout policies represent the timeout class, exponential predictive policy represents the predictive class and TISMDP represents the stochastic class of policies. The fixed timeout policy simply waits for the specified timeout before switching off the device, while the adaptive timeout policy adjusts it by the given factor based on whether a correct shutdown decision was made in the previous idle period or not. The exponential predictive policy predicts the length of the upcoming idle period (I_{n+1}) using the actual (i_n) and predicted (I_n) lengths of the previous idle period. TISMDP is a stochastic policy which provides randomized timeouts optimal for given device and workload distributions. Further details are provided in the following sections.

Table 2 HDD and WLAN power characteristics

Device	P_{on}	P_{sleep}	P_{tr}	T_{tr}	T_{be}
HDD	1.6W	0.4W	2.4W	2.5s	1.6s
WLAN	0.9W	0 W*	3W	0.3s	0.7s
*WLAN card is turned off					

4.2 Selection with state of the art policies

4.2.1 HDD

We performed experiments on the HDD traces shown in Table 1 with the working set described in Table 3. The fixed timeout employs a timeout equal to seven times the break even time or T_{be} (see Section 2 for definition). The adaptive timeout policy uses the same timeout with an adjustment factor of $+0.1T_{be}$/$-0.1T_{be}$ depending on whether the previous idle period resulted in

Table 3 Working set characteristics

Expert	Characteristics
Fixed Timeout	Timeout = $7*T_{be}$
Adaptive Timeout [3]	Initial timeout = $7*T_{be}$; Adjustment = $+0.1T_{be}/-0.1T_{be}$
Exponential Predictive [6]	$I_{n+1} = a\ i_n + (1 - a)\boldsymbol{.I_n}$, With a = 0.5
TISMDP [12]	Optimized for delay constraint of 3.4% on HP-1

energy savings or not. Exponential predictive policy is implemented as described in [6] while TISMDP policy is optimized for 3.4% delay on the HP-1 trace. The main idea we are trying to show is that given a set of experts, the Controller always converges to select the best performing expert at that time.

Table 4 shows the results achieved in terms of energy savings and performance delay for the individual experts on the HDD traces. An oracle policy, which knows the trace in advance and is thus an ideal policy, has been added to provide a baseline comparison among all the policies. The oracle is *not* included into Controller's working set. The *%energy* indicates the amount of energy saved relative to the case where we do not have any DPM policy while the *%delay* shows the amount of performance delay caused relative to the total timeframe. The results highlighted in black show where we get the best energy savings while the results highlighted in grey address the case where we get the least performance delay. The results for the oracle policy indicate the maximum achievable energy savings for all traces. We can notice that the predictive policy does really well in terms of saving energy. For instance, on HP-1, it achieves around 66.6% energy savings, which is very close to what achieved by the oracle. It does equally well for the other traces as well.

Table 4 Energy Saving/Performance Delay for experts

(grey shade indicates least performance delay and black indicates maximum energy savings)

Policy	HP1 Trace		HP2 Trace		HP3 Trace	
	%delay	%energy	%delay	%energy	%delay	%energy
Oracle	0	68.17	0	65.9	0	71.2
Timeout	4.2	49.9	4.4	46.9	3.3	55
Ad Timeout	7.7	66.3	8.7	64.7	6	67.7
TISMDP	3.4	44.8	2.26	36.7	1.8	42.3

However, predictive policy is also the worst in terms of causing performance delay. This is because it is extremely aggressive in turning off the HDD and thus incurs a lot of overhead while waking up. In contrast TISMDP causes the least performance delay and consequently fetches the least energy savings. It can be observed in Table 3 that TISMDP was optimized for 3.4% delay on HP-1 trace and the results achieved confirm this. However, the figures are not the same for HP-2 and HP-3 traces which confirms that it is optimal for stationary workloads and does not

Table 5. Energy Savings/Performance Delay for Controller

	%delay	%energy	%delay	%energy	%delay	%energy
Low α	3.5	45	2.61	37.41	2.55	49.5
Medium α	6.13	60.64	5.86	54.2	4.36	61.02
High α	7.68	65.5	8.59	64.1	5.69	66.28

adapt with changing workloads. Fixed timeout performs reasonably well on both the accounts while adaptive timeout is quite close to predictive in terms of energy savings. So primarily what these results highlight is that different classes of policies, depending upon their characteristics, deliver different levels of performance. However, depending upon the application requirements or user preferences one might want the overall performance to be more delay sensitive or more energy sensitive. The problem with just having a single DPM policy is that it does not offer the flexibility to control this behavior. The Controller offers exactly this flexibility.

Table 5 shows results achieved on the same traces using the controller with different energy savings/performance delay (e/p) preference settings. As explained in Section 3.1, α value indicates the desirable e/p setting. A high value indicates a higher preference to energy savings, a low value indicates higher preference to performance while a medium value indicates a reasonable ratio of both. In our experiments we tested with values of α ranging from 0.3 (minimum) to 0.7 (maximum). We used values of α around 0.5 for the medium value. From the results we can observe that as we increase the value of α, we get higher energy savings and for lower values of α, we get low performance delays. For instance, on HP-2 trace we get 64.1% energy savings for high α, which is quite close to that achieved predictive policy. In contrast for low α, we get performance delay comparable to that of TISMDP. Remember that we limit the values of α between 0.3 and 0.7. For even higher values (close to 1) we will achieve energy savings even closer to that of predictive policy.

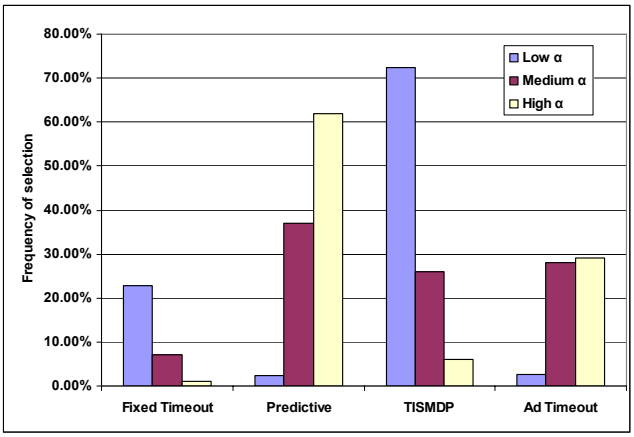

Figure 3. Selection Frequency of experts for HP-3 trace

Figure 3 shows how the Controller achieves these results taking example of HP-3 trace. It shows the frequency of selection of

Table 6 Working set characteristics

Expert	Characteristics
Fixed Timeout	Timeout = T_{be}

Adaptive Timeout [3]	Initial timeout = T_{be}; Adjustment = $+0.1T_{be}/-0.2T_{be}$
Exponential Predictive [6]	$I_{n+1} = a\ i_n + (1 - a)I_n$, with a = 0.5
TISMDP [12]	Optimized for delay constraint of 8.5% on www trace

experts according to the selected value of α. We can observe that for higher value of α, predictive expert is selected most often since it is aggressive in turning off the HDD and thus achieves better energy savings. Likewise, for lower values of α, TISMDP expert is selected with higher frequency since it is conservative in turning off the HDD and thus offers lower performance delays. For the medium value of α, we can see that it selects amongst all the policies to deliver a performance which offers a reasonable e/p tradeoff. Hence, we can see that α factor offers us a simple yet powerful control knob to obtain the desired e/p tradeoff.

4.2.2 WLAN

We performed similar experiments on WLAN to show that our scheme performs well regardless of a device type or workload characteristics. Our working set had experts with characteristics listed in Table 6. We ran our algorithm on the web surfing, telnet and a combined workload, which had the other two workloads concatenated (see workload characteristics in Table 1). Combined workload allows us to analyze how the Controller adjusts with changing workload characteristics.

Table 7 Energy Saving/Performance Delay for experts

(grey shade indicates least performance delay and black indicates maximum energy savings)

Policy	www Trace		Telnet Trace		Combined	
	%delay	%energy	%delay	%energy	%delay	%energy
Oracle	0	41.64	0	20.44	0	29.82
Timeout	10.13	23.47	9.69	4.82	9.98	14.64
Ad Timeout	11.63	28.72	10.73	4.74	11.46	17.56
TISMDP	8.5	19.04	7.41	3.37	7.6	10.02

Table 7 shows the results achieved in terms of energy savings and performance delay for the individual experts on the WLAN traces. As in Table 4, the results in black highlight the case where we get the best energy savings while the results in grey highlight the case where we get the least performance delay. Again, the results for oracle policy indicate the maximum achievable energy savings for all the traces. The oracle policy is not a part of the working set; it just serves as a baseline. We can observe that for WLAN, predictive policy is no longer the best performing policy in terms of energy savings. In fact for the telnet trace it is so bad that we actually incur negative power savings (-9.24%). That means we would have been better off without using any DPM policy. TISMDP does really well in terms of keeping performance delay low across all the workloads.

Table 8. Energy Savings/Performance Delay for Controller

Preference	www Trace		Telnet Trace		Combined	
	%delay	%energy	%delay	%energy	%delay	%energy
Low α	6.48	15.77	4.77	1.57	6.17	9.23
Medium α	6.78	16.38	5.35	2.69	6.28	10
High α	10.38	26.58	5.67	3.33	8.9	16.1

Adaptive timeout expert does really well in terms of energy savings and so does the fixed timeout expert. If we compare the amount of energy savings for HDD (see Table 4) and WLAN (see Table 7), we can observe that in former case the best results are quite close to the energy savings of the oracle policy. But for WLAN the difference is bigger. In fact for the telnet trace it is significantly more. This is primarily due to difference in nature of activity on a HDD and WLAN for the traces used. For HDD we get long idle periods during which a lot of power can be saved by turning it off. In contrast on WLAN, for the traces we used, the level of activity is quite high. This effectively translates into fewer opportunities to save power and higher probability of potential mispredictions. Since predictive policy is extremely aggressive in its DPM decisions, it suffers the most.

Table 8 shows results achieved on the same traces using the controller with different values of α. From the results we can observe that as we increase the value of α, we get higher energy savings and for lower values of α, we get low performance delays. For instance, on www trace we get 26.58% energy savings for high α, which is quite close to that achieved by adaptive timeout. In contrast for low α, we get performance delay comparable to that of TISMDP. In fact in case of both the traces, it is even lower than that of TISMDP. This is primarily due to the fact that none of the experts are outstanding in terms of reducing the delay (like TISMDP was for HDD). So at various points of time, different experts perform well, and our controller adapts to select each expert, hence delivering an excellent overall performance.

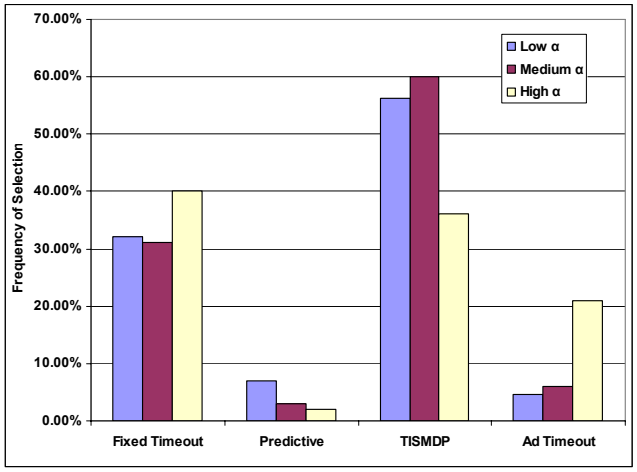

Figure 4. Selection Frequency of experts for combined trace

Figure 4 shows the frequency of selection of experts according to the selected value of α for the combined trace. It is interesting to observe the selection statistics for a low value of α and compare it with the results achieved for HDD. For HDD (see Figure 3), the results were dominated by TISMDP. For WLAN (see Figure 4) different experts perform well at different instances so the Controller continually selects the currently best performing expert.

4.3 Selection with Fixed Timeout Policies

The goal of DPM policies is to determine if a device should go to sleep on a given idle period and how long it should wait before doing so. Thus, inherently all DPM policies are a variant of a timeout policy. The manner in which this timeout value is determined varies from policy to policy. While the adaptive timeout policy heuristically adjusts the value of timeout based on previous performance, TISMDP employs a randomized timeout. For predictive policy the timeout is zero or the entire idle period depending upon a positive or a negative prediction of the length of the idle period. Based on these observations we next test our controller with a working set comprising of multiple simple fixed timeout policies with different timeout values. Such policy set should perform reasonably well compared to the state of the art policies used in the previous sections. Keeping this in mind, we performed experiments on the HDD with a working set of experts that had fixed timeouts ranging from T_{be} to 180s. A timeout of T_{be} guarantees that the energy consumption is not be greater than a factor of 2 when compared to an ideal offline policy [14]. Timeout of 180s is the minimum we can set for a HDD on Windows XP.

Figure 5 illustrates the performance of these individual timeout policies against the performance of the controller corresponding to the HP-1 trace. The x-axis represents the performance delay as a percentage of the total time frame, while y-axis gives the percentage of energy savings compared to a case where no DPM policy is present (always on case). The grey line in Figure 5 shows (e/p) points for different fixed timeout experts. The lowest point towards the left in the graph (e/p = 20%/0.2%) corresponds to the default Windows XP policy with a timeout of 180s, while the highest point towards the right (e/p = 65%/7%) corresponds to timeout of T_{be}[14]. By varying the fixed timeouts we obtain significant changes in the e/p ratio. The black line in Figure 3 shows e/p points for the controller algorithm with three different values for α. For high values of α, we get energy savings as high as 65% (around 45% more than the default Windows XP policy) while for a low value of α it gives an e/p ratio comparable to the policies with large timeouts (around 0.5%). Cleary, α factor offers us a simple and powerful control knob to obtain a desired e/p tradeoff.

We now compare results for the set of fixed timeout policies shown in Figure 5 with the results obtained in section 4.2.1 employing a working set representing more sophisticated policies (see Tables 4 and 5) for HP-1 trace. We can observe that in Figure 5 we get a broader range of e/p points ranging from 20%/0.2% (timeout = 180s) to 65%/7% (timeout = T_{be}) compared to 44.8%/3.4% (TISMDP) to 66.6%/8% (predictive) in Table 4. The range is smaller in the latter case because of the smaller size of the working set and the characteristics of the experts chosen (see Table 3). Correspondingly we can notice that the e/p points for Controller also closely follow the range offered

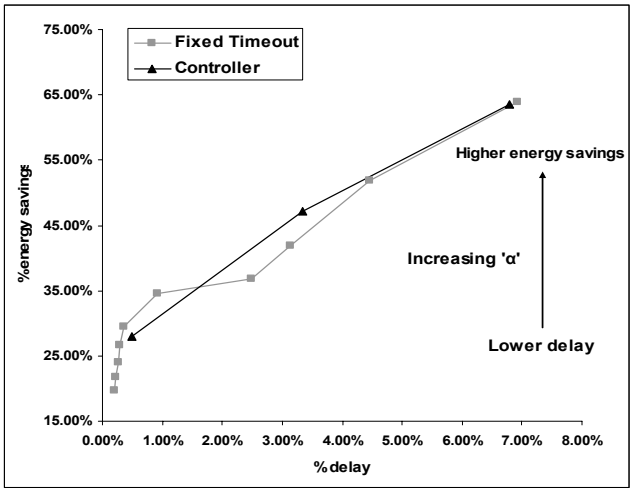

Figure 5. Energy/Performance delay curve for HP-1 trace

by their respective working sets (see Figure 5 and Table 5). This indicates that the performance of the Controller depends directly on the performance of the experts comprising the working set. For instance in Figure 5, the lowest performance delay offered by the working set is around 0.2% and our Controller closely achieves that for low α. For section 4.2.1, the lowest delay offered by the working set was 3.4% and again our Controller closely matched that for low α (see Table 5). This indicates that expert selection is extremely critical and we must include experts with characteristics matching our e/p requirements in the working set. For instance it is a good idea to have both conservative and aggressive experts in the working set, since that gives Controller a broader range of e/p points to select from according to the current user preference (α). We can further observe that e/p points achieved by the Controller for high α in both Figure 5 and Table 5 are quite similar. Hence the use of controller with a working set of fixed timeout policies (with a wide range of timeouts) offers a simple adaptive scheme that delivers good results with reasonable performance bounds.

4.4 Advantages and Disadvantages

Controller provides a flexible scheme where using a simple control knob (α) we indicate our desired e/p setting, while the Controller takes care of selecting the best performing expert at any point in time. The best performing experts vary with changing workload and device characteristics (see Table 4 and Table 7). The Controller is capable of recognizing these changes in workload and adapts accordingly to select the new best performing expert. We have shown using experiments on HDD and WLAN how Controller adapts to different devices and workload characteristics.

Controller, in the best case, performs as well as the best performing expert in the working set. Consequently if the expert selection is poor, the Controller cannot improve beyond the best available expert. For instance in Table 5, for a low α, we achieve low performance delay since TISMDP expert performs well in terms of delay and is thus selected the most often. However, if we remove TISMDP from the working set, then the results for performance delay would increase irrespective of the value of α. Thus expert selection is really critical to achieving good overall

performance. A big benefit of our work is that now designers can focus on optimizing policies for specific workloads. Our controller is then able to detect whenever that workload occurs and employ the appropriate expert to perform DPM, hence achieving the best possible performance.

Another concern is the controller overhead in terms of both energy and time to perform the evaluation of experts. No overhead is caused by our mechanism during the idle time, since the operational expert is selected in the active periods (see section 3.1). In our experiments we measured the average controller overhead at 0.0001% of the total timeframe for HDD and 0.0006% for WLAN. The overhead is higher for WLAN since the number of idle periods is greater and thus the controller is invoked more often. In either case the overhead is negligible relative to the overall timeframe. Some optimizations can be performed to further reduce this overhead. For instance, Figures 3 and 4 show that some experts get selected a lot more frequently than others. Hence, if an expert gets selected consecutively for more than some preset number of times, then we can assume it will be selected in the next idle periods as well, and the evaluation of experts can be thus suspended for some time.

5. CONCLUSION AND FUTURE WORK

DPM problem is one of selecting an appropriate idle period and the timeout value before the device can be shut down. DPM policies use different mechanisms to determine the value of timeout. However, with changing workloads, it is difficult to devise a single policy that can perform consistently well. In this paper we have proposed a novel DPM technique that takes a set of policies which perform well under different workloads and dynamically selects which policy should be active at run time using machine learning. Our algorithm guarantees us that its performance at any point of time is closest to that of the best performing policy.

We showed through our experiments using a set of various policies on HDD and WLAN that different policies perform better for different workloads. Our algorithm adapts to select the best performing policy each time. Furthermore, it allows us to optimally use experts that are best for specific workloads. In this way we can provide potentially large power/performance benefits in situations where state of the art policies do not consistently perform well.

As part of future work, we plan to further extend this scheme to incorporate multiple low power states and dynamic voltage/frequency scaling as well.

6. REFERENCES

[1] L. Benini and G. De Micheli, Dynamic Power Management:Design Techniques and CAD Tools, Kluwer Academic Publishers, 1997.

[2] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[3] F. Douglis, P. Krishnan, and B. Bershad. Adaptive Disk Spin-Down Policies for Mobile Computers. In *Computing Systems*, volume 8, pages 381–413, 1995.

[4] R. Golding, P. Bosch, and J. Wilkes. Idleness is not Sloth. In *USENIX Winter Conference*, pages 201–212, 1995.

[5] M. B. Srivastava, A. P. Chandrakasan, and R.W. Brodersen. Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.

[6] C.-H. Hwang and A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *International Conference on Computer-Aided Design*, pages 28–32, 1997.

[7] E.-Y. Chung, L. Benini, and G. D. Micheli. Dynamic power management using adaptive learning tree. In *International Conference on Computer-Aided Design*, pages 274–279, 1999.

[8] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Design Automation Conference*, pages 182–187, 1998.

[9] E.-Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Design Automation and Test in Europe*, pages 77–81, 1999.

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

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

[12] T. Simunic, L. Benini, and G. D. Micheli. Dynamic Power Management of Laptop Hard Disk. In *Design Automation and Test in Europe*, 2000.

[13] Yung-Hsiang Lu, Eui-Young Chung, Tajana Simunic, Luca Benini, and Giovanni De Micheli. Quantitative Comparison of Power Management Algorithms. In *Design Automation and Test in Europe*, pages 20–26, 2000.

[14] A. Karlin, M. Manesse, L. McGeoch and S. Owicki. Competitive Randomized Algorithms for Nonuniform Problems. In *Algorithmica,* pp. 542-571, 1994.

[15] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference*, 1993.

[16] Z.Ren, Krogh, B.H and Marculescu, R.. Hierarchical adaptive dynamic power management. In *Design Automation and Test in Europe*, pages 136-141, 2004.

[17] S. McFarling. Combining Branch Predictors. WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.

[18] P,-Y. Chang, E. Hao, and Y.N. Patt. Alternative Implementations of Hybrid Branch Predictors. 28th ACM/IEEE International Symposium on Microarchitecture, Nov. 1995.