Deriving a Near-optimal Power Management Policy Using Model-Free Reinforcement Learning and Bayesian Classification

Yanzhi Wang[1], Qing Xie[1], Ahmed Ammari[2], and Massoud Pedram[1]
[1]Department of Electrical Engineering, University of Southern California, Los Angeles, CA
[2]National Institute of the Applied Sciences and of the Technology (INSAT), 1080 Tunis cedex, Tunisia
{*yanzhiwa, xqing, pedram*}@*usc.edu*, *chiheb.ammari@insat.rnu.tn*

ABSTRACT

To cope with the variations and uncertainties that emanate from hardware and application characteristics, dynamic power management (DPM) frameworks must be able to learn about the system inputs and environment and adjust the power management policy on the fly. In this paper we present an online adaptive DPM technique based on model-free reinforcement learning (RL), which is commonly used to control stochastic dynamical systems. In particular, we employ temporal difference learning for semi-Markov decision process (SMDP) for the model-free RL. In addition a novel workload predictor based on an online Bayes classifier is presented to provide effective estimates of the workload states for the RL algorithm. In this DPM framework, power and latency tradeoffs can be precisely controlled based on a user-defined parameter. Experiments show that amount of average power saving (without any increase in the latency) is up to 16.7% compared to a reference expert-based approach. Alternatively, the per-request latency reduction without any power consumption increase is up to 28.6% compared to the expert-based approach.

Categories and Subject Descriptors: B.8.2 [Performance and Reliability]: Performance Analysis and Design Aides.

General Terms: Algorithms, Management, Performance, Design.

Keywords: Dynamic Power Management, Bayes Classification, Reinforcement Learning.

1. INTRODUCTION

Power consumption has become one of the critical concerns in design of electronic computing systems. High power consumption degrades system reliability, increases the cooling cost for high performance systems, and reduces the service time of batteries in portable devices. Dynamic power management (DPM), defined as the selective shut-off or slow-down of system components that are idle or underutilized, has proven to be an effective technique for reducing power dissipation at system level [1]. An effective DPM policy should minimize power consumption while maintaining performance degradation to an acceptable level. Design of such DPM policies has been an active research area.

Bona fide DPM frameworks should account for variations that originate from process, voltage, and temperature (PVT) variations as well as current stress, device aging, and interconnect wear-out phenomena in the underlying hardware. They must also consider workload type and intensity variations due to change in application behavior. In addition, DPM frameworks must cope with sources of uncertainty in the system under their control e.g., inaccuracies in monitoring data about the current (power-performance) state of the system. These sources of variability and uncertainty tend to cause two effects: (i) difficulty of determining the current global state of the system and predicting the next state given a DPM agent's action, and (ii) difficulty in determining the reward (credit assignment) rate of a chosen or contemplated action. Thus DPM policies that are statically optimized (and are considered to be globally optimal for the modeled system) may in reality not achieve optimal performance in the presence of such uncertainties and variations. Therefore, adaptive DPM methods which are able to learn about the input and environmental variations/uncertainties and change the policy accordingly are critically important for modern DPM systems.

Many DPM methods have been proposed in the literature. They can be broadly classified into three categories: ad hoc, stochastic, and learning based methods. Ad hoc policies are based on the idea of predicting whether or not the next idle period length is greater than a specific value (the break-even time T_{be}). A decision to sleep will be made if the prediction indicates an idle period longer than T_{be}. Among these methods Srivastava et al. [2] use a regression function to predict the idle period length while Hwang et al. [3] propose an exponential-weighing average function for predicting the idle period length. Ad hoc methods are easy to implement, but perform well only when the requests are highly correlated; they typically do not take performance constraints into account.

By modeling the request arrival times (rates) and device service times (rates) as stationary stochastic processes, stochastic policies can take into account both power consumption and performance. Stochastic DPM techniques have a number of key advantages over ad hoc techniques. First, they capture a global view of the system, thus allowing the designer to search for a global optimum which can exploit multiple inactive states of multiple interacting resources. Second, they compute the exact solution (in polynomial time) for the performance-constrained power optimization problem. Third, they exploit the vigor and robustness of randomized policies. On the flip side, the performance and power obtained by a stochastic policy are expected values, and there is no guarantee that the results will be optimum for a specific instance of the corresponding stochastic process. Second, policy optimization requires a priori Markov models of the service provider and service requester. Third, policy implementation tends to be more involved.

In [4], Benini et al. model a power-managed system as a controllable *discrete-time* Markov decision process (MDP) by assuming the non-deterministic service time of a request follows a geometric distribution. Qiu et al. in [5] model a similar system by using a controllable *continuous-time* MDP with Poisson distribution for the request arrival times and exponentially distributed request service times. This in turn enables the power manager (PM) to work in an event-driven manner, and thus reduce the decision making overhead. Other enhancements include time-indexed semi-MDP of Simunic et al. [6]. To cope with uncertainties in the underlying hardware state, DPM policies based on partially observable Markov decision process (POMDP) have been proposed in [7] and [8]. Note that in the aforesaid stochastic DPM approaches, request inter-arrival times and system service times are modeled as stationary processes that satisfy certain probability distributions. In addition, an optimal policy for the given controllable MDP can be found only if we have knowledge (and model) of the state transition probability function and the reward function for the MDP. Reinforcement learning is primarily concerned with how to obtain the optimal policy for a given MDP when such a model is not known in advance. The DPM agent must interact with its environment to obtain information which, by means of an appropriate algorithm, can be processed to produce an optimal policy.

Several recent works use machine learning techniques for adaptive policy optimization. Compared to simple ad hoc policies, machine learning-based approaches can simultaneously consider power and performance penalty, and perform well under various workload conditions. In [9], an online policy selection algorithm is proposed, which generates offline and stores a set of DPM policies (referred to as "experts") to choose from. The controller evaluates the performances of the experts at the end of each idle period and based on that decides which expert should be activated next. The performance of the expert-based approach is close to the best performing expert for any given workload. However, the effectiveness of such learning algorithm depends heavily on the expert selection. Besides, such an algorithm has a limited ability to achieve a good power-performance tradeoff.

Tan et al. in [10] propose to use an enhanced Q-learning algorithm for system-level DPM. This is a model-free RL approach since the PM does not require prior knowledge of the state transition probability function. However the knowledge of the state and action spaces and also the reward function is required. The Q-learning based DPM learns a policy online by trying to learn which action is best for a certain system state, based on the reward or penalty received. In this way the PM does not depend on any pre-designed experts; and can achieve a much wider range of power-latency tradeoffs. However, this work is based on discrete-time model of the stochastic process, and thus has large overhead in real implementations. Moreover, the number of state-action pairs in this system is large, which may result in large computational overhead and slow convergence speed.

In this paper, we present a novel approach for RL-based DPM in a partially observable environment. While possessing the merits of [10] (model free, and independent of any pre-designed experts), the proposed approach can perform learning and power management in a continuous-time and event-driven manner. Other novel characteristics of the proposed work are:

- The proposed method uses enhanced TD(λ) learning algorithm for semi-MDP [11] to accelerate convergence and alleviate the reliance on Markovian property.
- Workload prediction is incorporated in this work to provide partial information about service request (SR) state for the RL algorithm. Specifically, an online naïve Bayes classifier [14] is selected as the workload predictor because of its relatively high prediction rate, as well as the fact that the partial information it provides contains certain degree of certainty due to the use of *posterior probability* in such algorithm.
- State and action spaces of the RL algorithm are optimized i.e., the number of state-action pairs are greatly reduced.

It is interesting that our approach allows us to learn the optimal timeout policy, which is often the optimal DPM policy when the request inter-arrival time is non-exponentially distributed [6].

In the proposed method, the tradeoff between power consumption and latency can be controlled by a user-defined parameter. Experiments on both synthesized and real traces show that the proposed PM finds a much "wider" average power and latency tradeoff curve compared with prior work references.

The rest of the paper is organized as follows. Section 2 explains basic background of reinforcement learning for SMDP. Section 3 explains our system model, as well as the workload prediction method using an online Bayes classifier. The experimental results are presented in Section 4, and we conclude in Section 5.

2. THEORETICAL BACKGROUD

2.1 Semi-Markov Decision Processes

A stationary semi-Markov decision process (SMDP) is a continuous-time dynamical system comprised of a countable state set, *S*, and a finite action set, *A*. The decision maker (DM) can choose actions only when system changes state. Suppose that the system changes to state $s \in S$ at the current (*transition*) *epoch*, and action $a \in A$ is applied. An SMDP then evolves as follows.

- At the next epoch, the system transitions to *s'* with probability $p(s' \mid s,a)$ given that *a* is chosen in *s*. Furthermore, the next epoch occurs within *t* time units with probability $p(t \mid s,a,s')$ given *s*, *a,* and *s'*. Thus, the next epoch occurs at or before time *t* and the state equals *s'* with probability $f_{ss'}(t \mid a) \equiv p(s' \mid s,a)p(t \mid s,a,s')$. Let T(*s,a*) denote expected value of the current epoch duration. Then, $T(s,a) = \int_0^\infty (1 - \sum_{s' \in S} f_{ss'}(t \mid a))dt$. If this duration is distributed exponentially, then SMDP reduces to continuous-time MDP.
- When DM selects action *a* in state *s*, she accrues a reward at the rate of $r(s,a)$ as long as the system occupies *s* (before it transitions to *s'*.)

A *policy* $\pi = \{(s,a) \mid a \in A, s \in S\}$ is a set of state-action pairs for all states of an SMDP. We use notation: $\pi(s) = a$ to specify the action that is chosen in state *s* according to policy π. We consider the class of *stationary* and *deterministic* policies. An optimal policy is the one maximizing the total expected reward.

2.2 Temporal Difference Learning for SMDP's

As illustrated in Figure 1, the general reinforcement learning model consists of an agent, a finite state space *S,* a set of available actions *A*, and a reward function $R : S \times A \to R$.

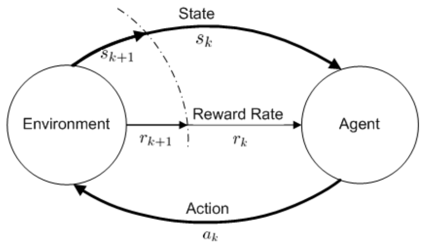

Figure 1: Agent-environment interaction Model.

Assume that the agent-environment interaction system evolves as a stationary SMDP, which is continuous in time but has a countable number of events. Then there exists a countable set of times $\{t_0, t_1, t_2, \ldots, t_k, \ldots\}$, known as *epochs*. At epoch t_k, system has just transitioned to state $s_k \in S$. The agent selects an action $a_k \in A$ according to some policy π. At time t_{k+1}, the agent finds itself in a new state s_{k+1}, and, in the time period $[t_k, t_{k+1})$, it receives a scalar reward with rate r_k.

Suppose system starts at time t_0. The *return* R is defined as the discounted integral of reward rate. Furthermore, the *value* of a state s under a policy π, denoted $V^\pi(s)$, is the expected return when starting from s and following π thereafter:

$$V^\pi(s) = E_\pi\{R \mid s_0 = s\} = \sum_{s' \in S} \int_{t_0}^{\infty} e^{-\beta(t-t_0)} V^\pi(s') df_{ss'}(t - t_0 \mid \pi(s))$$
$$+ \sum_{s' \in S} \int_{t_0}^{\infty} \int_{t_0}^{t} e^{-\beta(\tau - t_0)} r(s, \pi(s)) d\tau \, df_{ss'}(t - t_0 \mid \pi(s))$$

where $\beta > 0$ is a *discount factor*.

Similarly, we can define *value functions* for state-action pairs:

$$Q^\pi(s,a) = E_\pi\{R \mid s_0 = s, a_0 = a\}$$
$$= \sum_{s' \in S} \int_{t_0}^{\infty} \int_{t_0}^{t} e^{-\beta(\tau - t_0)} r(s,a) d\tau \, df_{ss'}(t - t_0 \mid a)$$
$$+ \sum_{s' \in S} \int_{t_0}^{\infty} e^{-\beta(t-t_0)} Q^\pi(s', \pi(s')) df_{ss'}(t - t_0 \mid a)$$

Now suppose that we want to estimate the value function $V^\pi(s)$ for some state, s. However, the agent has no prior knowledge about state transition probabilities, which are essential for characterizing an SMDP. Therefore, traditional value iteration or policy iteration methods cannot be used here. Instead a simple 1-step temporal difference learning method [11] (also known as the TD(0) rule) for SMDP may be used. Such a method generates an estimate $V^{(k)}(s)$ for each state s at epoch t_k, which is the estimate of the actual value $V^\pi(s)$ following policy π. Suppose state s_k is visited at epoch t_k, then the TD(0) rule updates the estimate $V^{(k)}(s_k)$ at the next epoch t_{k+1} based on the chosen action a_k, and the next state s_{k+1} as follows:

$$V^{(k+1)}(s_k) = V^{(k)}(s_k) + \alpha\left(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} V^{(k)}(s_{k+1}) - V^{(k)}(s_k)\right)$$

In the above expression, $\tau_k = t_{k+1} - t_k$ is the time that system remains in state s_k; $\alpha \in (0,1)$ denotes the *learning rate*; $\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k)$ is the sample discounted reward recieved in τ_k time units; and $V^{(k)}(s_{k+1})$ is the estimated value of the actually occurring next state. Notice that whenever state s_k is visited, its estimated value is updated to be closer to $\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k)$ $+e^{-\beta\tau_k} V^{(k)}(s_{k+1})$. The key idea is that the aforesaid expression is a sample of the value of $V^{(k)}(s_k)$, and it is more likely to be correct because it incorporates the real return. If the learning rate α is adjusted properly (slowly decreased) and the policy is kept unchanged, TD(0) converges to the optimal value function [12].

For realistic RL algorithms, we need not only evaluate the performance of a predefined policy, but simultaneously learn the optimal policy and use that policy to control (make decisions.) To achieve this goal, the RL algorithm should learn the value of each *state-action pair*. Meanwhile system should choose an action at each state, either by choosing the one with maximum estimated value, or by using other semi-greedy policies [12].

2.3 TD(λ) for SMDP's

Because a real DPM problem is non-Markovian and non-stationary, we turn to the more powerful TD(λ) algorithm [12]. TD(λ) algorithm behaves more robustly in non-Markov cases. The learning rate of TD(λ) is also faster.

Suppose that we are in state s_k at epoch t_k, and we make decision a_k. In 1-step TD learning, we wait until the next epoch t_{k+1} and then perform a "1-step backup" to update the estimate $V^{(k)}(s_k)$. In 1-step backup the *target* is the immediate reward plus the discounted estimated value of the next state, i.e.:

$$R_k^{(1)} = \frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} V^{(k)}(s_{k+1})$$

Similarly, we could perform 2-step backup, in which we wait until epoch t_{k+2} and then perform a "backup" to update the value estimate $V^{(k)}(s_k)$. The *target* of 2-step backup is given by:

$$R_k^{(2)} = \frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} \frac{1-e^{-\beta\tau_{k+1}}}{\beta} r(s_{k+1}, a_{k+1}) + e^{-\beta(\tau_k + \tau_{k+1})} V^{(k)}(s_{k+2})$$

where the system transitions from state s_k under action a_k to state s_{k+1} and then under action a_{k+1} ends up in state s_{k+2}. This result can be easily generalized to n-step backup for arbitrary n. When $n \to \infty$, the n-step backup algorithm becomes Monte Carlo method (which relies on repeated random sampling to compute the optimal policy.) However, the n-step backup is rarely used directly because it is difficult to implement. Rather, people seek to find effective ways of averaging backups of different steps.

The TD(λ) algorithm may be understood as one particular way of averaging n-step backups. It contains all the n-step backups, each weighted proportional to λ^{n-1} ($0 < \lambda < 1$). The resulting target is:

$$R_k^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_k^{(n)}$$

TD(λ) learning algorithm can be implemented conveniently with the help of *eligibility traces*, as discussed in [12]. Among variant specific implementations, the one implemented in our system is Watkin's Q(λ) algorithm [13] modified for SMDP problems. This algorithm can perform simultaneous learning and control. In particular, the value update rule for an state-action pair in Watkin's Q(λ) algorithm is computed as follows:

$$\forall (s,a) \in S \times A: \; Q^{(k+1)}(s,a) = Q^{(k)}(s,a)$$
$$+ \alpha\left(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + \max_{a'} e^{-\beta\tau_k} Q^{(k)}(s_{k+1}, a') - Q^{(k)}(s,a)\right) e^{(k)}(s,a)$$

where $Q^{(k)}(s,a)$ is the value of state-action pair at epoch t_k, and $e^{(k)}(s,a)$ denotes the eligibility of that pair. Such eligibility reflects the degree to which state-action pair (s,a) has been chosen in the recent past. It can be updated online as follows:

$$e^{(k)}(s,a) = \lambda e^{-\beta\tau_{k-1}} e^{(k-1)}(s,a) + \delta((s,a),(s_k,a_k))$$

where $\delta(x,y)$ denotes the delta kronecker function.

3. SYSTEM MODEL

In this section, we explain how to extend RL techniques to solve the system-level DPM problem. Similar to many previous works, the system whose power is being managed consists of a service requester (SR), a service provider (SP), and a service queue (SQ). The SR generates different types of requests to be processed by the SP, and these requests are buffered in the SQ before processing. A power manager using RL algorithm, as well as a workload predictor is added to the system, as shown in Figure 2.

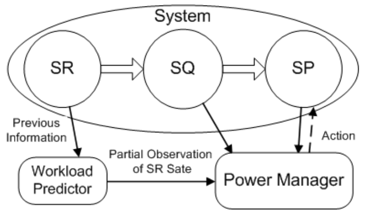

Figure 2. Abstract model of a power-managed system.

In this work, the state of the entire system is characterized by a 3-tuple ($\mathbb{SR}$, $\mathbb{SQ}$, $\mathbb{SP}$), where $\mathbb{SR}$ is the service request generating rate (high, low, etc.) or the next inter-arrival time (short, long, etc.), $\mathbb{SQ}$ is the number of requests in the service queue, and $\mathbb{SP}$ is the system power state (busy, idle, sleep.) Note that the SP transition from idle to busy state is an autonomous transition. To be more realistic, we consider in this work that the SR state cannot be directly obtained by the PM. In contrast to previous work on POMDP [7][8], the PM has no prior knowledge of the characteristics of the SR. Therefore, workload prediction has to be incorporated to provide partial information to the PM so that the PM can learn in the observation domain of the SR.

3.1 Workload Prediction

The proposed system relies on workload prediction method to provide partial observation of actual SR state for the PM. Previous work on workload prediction in [2][3] assumes that a linear combination of previous idle times (or request inter-arrival times) may be used to infer the future ones, which is not always true. For example, one very long inter-arrival time can ruin a set of subsequent predictions. Thus in our work a naïve Bayes classifier, which can overcome the above effect and result in much higher prediction accuracy, is adopted as the workload predictor.

Naïve Bayes classifier is a generative classifying technique using the idea of *maximum a posteriori* (MAP). Given input feature $\boldsymbol{x}=(x_1, x_2, \ldots, x_n)$, the classifier's goal is to assign class label l from a finite set L for the output y, by maximizing the *posterior probability* $Prob(y=l|x_1,x_2,\ldots,x_n)$:

$$y_{\mathrm{MAP}} = \arg\max_l Prob(y=l \mid x_1, x_2, \ldots x_n)$$
$$= \arg\max_l \frac{Prob(x_1, x_2, \ldots x_n \mid y=l) \cdot Prob(y=l)}{Prob(x_1, x_2, \ldots x_n)}$$

where the denominator $Prob(x_1,x_2,\ldots,x_n)$ is the same for every class assignment of y. $Prob(y=l)$, which is the *prior probability* that the class of y is l, can be calculated from training set. Hence, we only need $Prob(x_1,x_2,\ldots,x_n|y=l)$, the conditional probability of seeing the input feature vector $\boldsymbol{x}$ given that the class of y is l.

A fundamental assumption that Bayes classifier made is that all input features are *conditionally independent* given the class y, i.e., $Prob(x_1|x_2,\ldots,x_n, y=l) = Prob(x_1|y=l)$. Therefore, we get: $Prob(x_1, x_2, ..., x_n \mid y=l) = \prod_j Prob(x_j \mid y=l)$, and we compute the MAP class of y as follows:

$$y_{\mathrm{MAP}} = \arg\max_l Prob(y=l) \cdot \prod_{j=1}^{n} Prob(x_j \mid y=l)$$

In the original algorithm, the prior and conditional probabilities are obtained by performing *Maximum Likelihood* estimation on the whole data set. However, in this work we have to implement the predictor in an online fashion. So when we observe a sequence of features ($x_1=m_1$, $x_2=m_2$, …,$x_n=m_n$) and output $y=l$, we update the conditional and prior probabilities as follows:

$$\forall i \in \{1,2,...,n\} \quad Prob(x_i = m_i \mid y=l) \leftarrow \alpha + (1-\alpha) \cdot Prob(x_i = m_i \mid y=l)$$
$$Prob(y=l) \leftarrow \beta + (1-\beta) \cdot Prob(y=l)$$

where $\alpha, \beta \in (0,1)$ denote the updating rate parameters.

In this work, we use previous request inter-arrival times as input features $\boldsymbol{x}=(x_1, x_2, \ldots, x_n)$, in which $x_i = 1$ if the corresponding interval length is greater than *break-even time* T_{be}; otherwise, $x_i = 0$. The output is the prediction whether or not the next inter-arrival time is greater than T_{be}. In real implementations, we use three output states "long, short, and unknown". We predict the next-inter-arrival time to be "unknown" if the difference between posterior probabilities that the next inter-arrival time is long and that it is short, is less than a predefined parameter ε.

3.2 RL-based DPM

To apply RL techniques for DPM frameworks, first we define *decision epochs* i.e., when new decisions are made and updates for the RL algorithm are executed. Note that the decision epochs are a subset of all possible transition epochs. In our case, the decision epochs coincide with one of the following four:

1. The SP entered the idle state ($\mathbb{SP}$ =idle) and $\mathbb{SQ} = 0$.
2. The SP entered the idle state and $\mathbb{SQ} \geq 1$.
3. The SP has just entered the sleep state and finds that $\mathbb{SQ} > 0$. This means at least one request has arrived during the transition from idle to sleep state, and therefore, the PM can decide whether to turn on the SP or to keep it in sleep state.
4. The SP is in sleep state and $\mathbb{SQ}$ transitions from zero to one (the SQ is initially empty, and a new request comes.)

The proposed RL-based DPM framework operates as follows. At each decision epoch, the PM finds itself in one of the four aforesaid conditions; it will make a decision and issue commands to hardware to implement the decision. If it finds itself in case (1), then it will use the RL-based timeout policy described below. If it is in case (2), it will continue to keep the SP in the active state to continue processing requests in the SQ (we may not perform update for RL algorithm in this case because there is just one action to choose.) Otherwise (i.e., for cases 3 and 4), it will use the RL-based N-policy (again described below).

As pointed out in reference [6], the optimal policy when $\mathbb{SP}$ = idle for non-Markov environments is often a timeout policy, wherein the SP is put to sleep if it is idle for more than a specified timeout period. A list of timeout periods, as well as immediate shutdown (timeout value = 0), serve as the action set when $\mathbb{SP}$ = idle. The proposed PM learns to choose the optimal action among these by using a RL technique (see the pseudo-code.)

RL-based timeout policy

```
At decision epoch $t_k$ ($\mathbb{SP}$ = idle and $\mathbb{SQ}$ = 0),
    1. Choose an action from the action set (list of timeout values)
       based on the estimated state of the SR.
    2. Execute timeout policy based on the chosen timeout value.
At the next decision epoch $t_{k+1}$,
    3. The system finds itself either in the sleep state (no request
       came during the timeout period) or in the idle state (some
       request came in that period.)
    4. Regardless, it evaluates the chosen action using the "backup"
       method discussed in section 2 (note the dependency of the
       backup calculation on the estimates of the SR state.)
    5. Go back to step 1.
```

When the SP is in the sleep state, the action set for the PM is a set of integers. In the standard N-policy, the SP will wake up to process requests only if $\mathbb{SQ} \geq N$. We incorporate the N-policy in the RL-based framework as described by the following pseudo-code. Note that the partial knowledge of the SR state from the workload predictor can help in making decisions when $\mathbb{SP}$ = sleep, and therefore we incur less performance penalty under the same power consumption level. Also note that timeout values or estimated SR based re-decision may be incorporated in the RL-based *N*-policy for better performance.

RL-based *N*-policy

```
At decision epoch $t_k$ (here the SP is in sleep state),
    1. Choose an action from the action set (list of $N$ values) based
       on the estimated state of the SR.
    2. The SP turns on to process requests until SQ $\geq N$ (i.e., the SQ
       has accumulated at least $N$ requests.)
At the next decision epoch $t_{k+1}$,
    3. Evaluate the chosen action using the "backup" method
       discussed in section 2 (note the dependency of the backup
       calculation on the estimates of the SR state.)
    4. Go back to step 1.
```

In this work, we use "cost rate" instead of "reward rate" in the RL algorithms, which can be treated in the similar way. The cost rate is a linearly-weighted combination of power consumption and the number of requests buffered in the SQ. This is a reasonable cost rate because as reference [5] has pointed out the average number of requests in the SQ is proportional to the average latency for each request, which is defined as the average time for each request between the moment it is generated and the moment that the SP finishes processing it i.e., it includes the queuing time plus execution time. In this way, the value function $Q(s, a)$ for each state-action pair (s, a) is a combination of the expected total discounted energy consumption and total latency experienced by all requests. Since the number of requests and the total execution time are fixed, the value function is equivalent to a combination of the average power consumption and per-request latency. The relative weight between average power and per-request latency can be changed to obtain the Pareto-optimal tradeoff curve.

Note that although the DPM problem is not SMDP in essence, we can obtain much better results compared to the expert-based systems due to the robustness of TD(λ) learning technique used.

3.3 Multiple-Update Initialization

The use of various timeout values as actions in RL-based DPM algorithm enables us to do multiple updates. Suppose that the SP is in the idle state, and the PM takes an action corresponding to a specific timeout. If a request comes before the timeout expires, all actions corresponding to larger timeout values can evaluate (using the "backup" method described in section 2.) This is because all actions with larger timeout values, if taken, would result in the same immediate cost and the same discounted next state value compared to the selected action. The proposed multiple-update scheme can significantly accelerate convergence speed of RL algorithms. In this work such a scheme is for quick initialization.

4. EXPERIMENTAL RESULTS

In this section we present the results with RL-based DPM with workload prediction and other improvements on two different devices: hard disk drive (HDD) and wireless adapter card (WLAN card.) Table 1 and 2 list the power and delay characteristics of both devices. In these tables T_{tr} is the time taken in transitioning to and from the sleep state while P_{tr} is the power consumption in waking up the device. T_{be} refers to the break even time.

Table 1. Power and delay characteristics of HDD.

P_{busy}	P_{idle}	P_{sleep}	E_{tr}	T_{tr}	T_{be}
2.15W	0.90W	0.13W	7.0J	1.6s	6.8s

Table 2. Power and delay characteristics of WLAN card.

P_{tran}	P_{rcv}	P_{idle}	P_{sleep}	E_{tr}	T_{tr}	T_{be}
1.6W	1.2W	0.90W	0W*	0.9J	0.3s	0.7s

*The WLAN card is turned off.

For the baseline system we use the expert-based DPM developed in [9]. Three policies are adopted as experts in the expert-based DPM: fixed timeout policy, adaptive timeout policy and exponential predictive policy [3], as shown in Table 3.

Table 3. Characteristics of the expert-based policy.

Expert	*Characteristics*
Fixed Timeout	Timeout = any value
Adaptive Timeout	Initial Timeout = T_{be}, adjustment = ±0.1 T_{be}
Exponential Predictive	$I_{k+1} = \alpha \cdot i_k + (1-\alpha) \cdot I_k$, $\alpha = 0.5$

In the following, we refer to the "simple RL-based power managed system" as the system which can only make decisions when SP = idle. This is for fair comparison with the baseline systems since they cannot make decisions when SP = sleep. The PM in the simple RL-based system makes decisions and performs updates for the RL algorithm according to the current state tuple (SR, SQ, SP), in which SP = idle, and SR is the estimated SR state provided by the Bayes classifier. Similarly we refer to the "entire power managed system" as the system which can make decisions when the SP are in both the idle or sleep states.

4.1 HDD

For the HDD, we implement our simulation based on the synthesized SR model, which is a continuous-time Markov model with three different states, each corresponding to a different request generating rate. The state transition matrix is given by:

$$\begin{bmatrix} -0.02 & 0.01 & 0.01 \\ 0.005 & -0.01 & 0.005 \\ 0.005 & 0.005 & -0.01 \end{bmatrix}$$

The request generation rates for the three states are 1.5, 0.1 and 0.025, respectively. The request inter-arrival times at each state satisfy arbitrary distribution (not necessarily be exponential.) The PM does not know what the exact state SR is in; rather it relies on the workload predictor for estimations of the SR state.

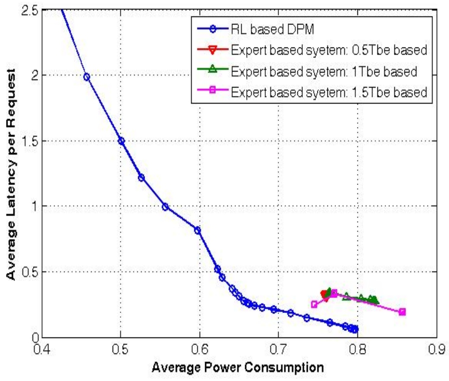

Figure 3. Tradeoff curves for HDD: entire system.

Figure 3 gives the power and latency tradeoff curves for the entire RL-based power management system with workload prediction, as well as three different expert-based DPM systems. The timeout values of the fixed timeout expert in those three systems are set to be $0.5T_{be}$, $1T_{be}$ and $1.5T_{be}$, respectively. We can see from the figure that the tradeoff curve of the RL-based system is more evenly distributed and has a much wider tradeoff range. Even with the same latency, the RL-based DPM system can achieve much lower power consumption than the references. The maximum power saving with the same average latency is 18.1%. The reason that the RL-based DPM system outperforms baseline systems is that performance of the expert-based baseline system depends heavily on the selection of experts (in fact the performance of the expert-based DPM will converge to, but not exceed, the expert with best performance), and each expert is not as robust as the RL based DPM. The learning rate of the RL algorithm is fast (in less than 100 SR requests), due to the carefully designed state space and other improvements been made.

4.2 WLAN Card

For the WLAN card, we have measured several real traces using the *tcpdump* utility in Linux. The measured traces include: 45-minute trace for online video watching, 2-hour trace for web surfing, and 6-hour trace for a combination of web surfing, online chatting and server accessing, 2 hours each. To reduce decision overhead, we incorporate a "minimal decision interval" of 0.1s for the RL algorithm and the workload predictor. The correct prediction rate of the online Bayes predictor can be 99.2% for the video trace, 79.8% for the web trace, 82.8% for the combined trace. In comparison, the correct prediction rate of an exponential predictor [3] for the combined trace is less than 65%.

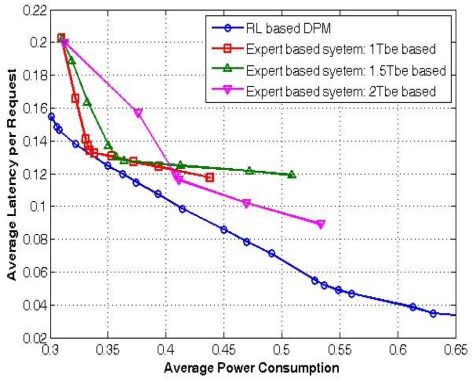

Figure 4. Tradeoff curves for WLAN card: simple system.

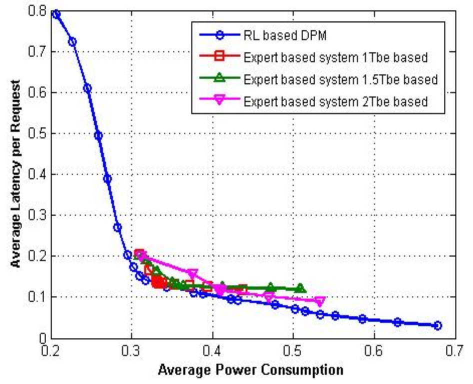

Figure 5. Tradeoff curves for WLAN card: entire system.

Figure 4 gives the power and latency tradeoff curves for the simple RL-based DPM system with workload prediction, as well as three different expert-based systems. The tradeoff curves for the entire system are given in Figure 5. Comparing these two figures, we can see that with the help of the RL-based *N*-policy, system can minimize average power to two thirds of the minimal power achievable by the simple RL-based system by sacrificing performance. When comparing the RL-based DPM with reference systems, we can see again that the former approach can achieve a "wider and deeper" power and latency tradeoff curve than the latter. When comparing to the expert-based approach in which the fixed timeout expert has timeout value of $1T_{be}$, the maximum power saving with the same latency is 16.7%; while the maximum latency saving with the same power consumption is 28.6%.

5. CONCLUSION

In this paper a novel adaptive DPM technique using reinforcement learning is proposed. The underlying system model is that of a semi-Markov Decision Process (which enables modeling the system evolution in continuous time and allows the time spent in a particular system state to follow an arbitrary probability distribution.) The TD(λ) learning for SMDP problems is selected as the basic RL algorithm in the proposed system. The proposed DPM is model-free and requires no prior information of the state transition probability function for the SMDP. A workload predictor based on an online Bayes classifier is presented to provide estimates of the request inter-arrival times to the DPM agent. Experiments show that the proposed PM finds a much "deeper and wider" power and latency tradeoff curve compared with reference expert-based DPM systems.

Acknowledgement – This work is sponsored in part by a grant from the National Science Foundation.

REFERENCES

[1] L. Benini, A. Bogliolo and G. De Micheli, "A survey of design techniques for system level dynamic power management," *IEEE Trans. on VLSI Systems*, Vol. 8, Issue 3, pp. 299-316, 2000.

[2] M. Srivastava, A. Chandrakasan and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. on VLSI*, 1996.

[3] C. H. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *ICCAD '97*.

[4] L. Benini, G. Paleologo, A. Bogliolo and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. on CAD*, Vol. 18, pp. 813-833, Jun. 1999.

[5] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," in *DAC '99*.

[6] T. Simunic, L. Benini, P. Glynn and G. De Micheli, "Event-driven power management," *IEEE Trans. on CAD*, 2001.

[7] H. Jung and M. Pedram, "Dynamic power management under uncertain information," in *DATE '07*, pp. 1060-1065, Apr. 2007.

[8] Q. Qiu, Y. Tan and Q. Wu, "Stochastic Modeling and Optimization for Robust Power Management in a Partially Observable System," in *DATE '07*, pp. 779-784, Apr. 2007.

[9] G. Dhiman and T. Simunic Rosing, "Dynamic power management using machine learning," in *ICCAD* '06, pp. 747-754, Nov. 2006.

[10] Y. Tan, W. Liu and Q. Qiu, "Adaptive power management using reinforcement learning," in *ICCAD '09*, pp. 461-467, Nov. 2009.

[11] S. Bradtke and M. Duff, "Reinforcement learning methods for continuous-time Markov decision problems," in *Advances in Neural Information Processing Systems 7*, pp. 393-400, MIT Press, 1995.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[13] C. Watkins, *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England, 1989.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, August 2006.