Model-Free Reinforcement Learning and Bayesian Classification in System-Level Power Management

Yanzhi Wang, *Member, IEEE* and Massoud Pedram, *Fellow, IEEE*

Abstract—To cope with uncertainties and variations that emanate from hardware and/or application characteristics, dynamic power management (DPM) frameworks must be able to learn about the system inputs and environmental variations, and adjust the power management policy on the fly. In this paper, an online adaptive DPM technique is presented based on the model-free reinforcement learning (RL) method, which requires no prior knowledge of the state transition probability function and the reward function. In particular, this paper employs the temporal difference (TD) learning method for semi-Markov decision process (SMDP) as the model-free RL technique since the TD method can accelerate convergence and alleviate the reliance on the Markovian property of the power-managed system. In addition, a novel workload predictor based on an online Bayesian classifier is presented to provide effective estimation of the workload characteristics for the RL algorithm. Several improvements are proposed to manage the size of the action space for the learning algorithm, enhance its convergence speed, and dynamically change the action set associated with each system state. In the proposed DPM framework, power-latency tradeoffs of the power-managed system can be precisely controlled based on a user-defined parameter. Extensive experiments on hard disk drives and wireless network cards show that the maximum power saving without sacrificing any latency is 18.6 percent compared to a reference expert-based approach. Alternatively, the maximum latency saving without any power dissipation increase is 73.0 percent compared to the existing best-of-breed DPM techniques.

Index Terms—Dynamic power management, reinforcement learning, supervised learning, Bayesian classification

1 Introduction

Power consumption has become one of the critical roadblocks in the design of electronic computing systems nowadays. High power consumption degrades system reliability, increases the cooling cost for high performance embedded systems, and also reduces the battery service life in portable devices. *Dynamic power management* (DPM), which refers to the selective shut-off or slow-down of system components that are idle or underutilized, has proven to be an particularly effective technique for reducing system-levelpower dissipations [1], [2]. An effective DPM policy should maximize power savings while maintaining the performance degradation within an acceptable level. Design of such effective DPM policies has been an active research area in the past decade.

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 take into consideration the workload type and intensity variations due to changes in application behavior. In addition, robust DPM frameworks must also 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 the DPM agent (controller)'s action, and (ii) difficulty in determining the reward (credit assignment) rate of a chosen or contemplated action. Therefore, DPM policies that are statically optimized (and are considered to be globally optimal for the modeled system) may in reality not achieve the optimal performance in the presence of such uncertainties and variations. Hence, adaptive DPM methods that are able to learn the input and environmental variations/uncertainties (in workload type/intensity and underlying hardware state) and change the policy accordingly have become critical for state-of-the-art system optimizations.

Many DPM methods have been proposed and investigated in the literature [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23]. 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 the next idle period length is greater than a specific value (the break-even time T_{be}) or not. A decision to the sleep state will be made if the prediction indicates an idle period longer than T_{be}. Among these methods, Srivastava et al. [3] used a regression function to predict the idle period length, whereas Hwang and Wu [4] proposed an exponential-weighted averaging function for predicting the idle period length. Ad hoc methods are easy to implement, but perform well only when

- *Y. Wang is with Syracuse University, Syracuse, NY 13210. E-mail: ywang393@syr.edu.*
- *M. Pedram is with the University of Southern California, Los Angeles, CA 90089. E-mail: pedram@usc.edu.*

Manuscript received 26 July 2014; revised 15 Jan. 2016; accepted 21 Jan. 2016. Date of publication 16 Mar. 2016; date of current version 14 Nov. 2016. Recommended for acceptance by T. Gonzalez.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2543219

the service requests are highly correlated. Moreover, they typically do not take system performance into account.

By modeling the service request arrival times (rates) and device service times (rates) as stochastic processes, stochastic DPM policies can take into account both power consumption and performance simultaneously. Stochastic DPM techniques have a number of key advantages over ad hoc techniques. First, they capture a global view of the system, thereby allowing the designer to search for a global optimum policy that 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 consumption obtained by the 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. Third, policy implementation tends to be more involved.

In [5], Benini et al. modeled a power-managed system as a controllable *discrete-time* Markov decision process (MDP) by assuming that the non-deterministic request inter-arrival times and request service times follow stationary geometric distributions. Qiu and Pedram in [6] modeled 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), i.e., the DPM agent, to work in an asynchronous and event-driven manner, and thereby, reduce the decision making overhead and improve dynamic response of the controller. Other enhancements include the time-indexed semi-MDP (SMDP) model of Simunic et al. [7]. To cope with uncertainties in the underlying hardware state, DPM policies based on the partially observable Markov decision processes (POMDP) have been proposed in [8], [9], [10]. Please note that in the above-mentioned stochastic DPM approaches, service request inter-arrival times and system service times are modeled as stationary processes that satisfy certain probability distributions. In addition, an optimal policy for given controllable MDP can be found only if we have knowledge (and model) of state transition probability function and reward function of MDP.

Several recent work use machine learning techniques for adaptive policy optimization. Compared with simple ad hoc policies, machine learning-based approaches can account for power and performance penalty simultaneously, and perform well under various workload conditions. In [11], [12], an online policy selection algorithm is proposed that 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. Reference [13] employs a similar idea of policy selection on hard disk power management for mobile devices. However, the effectiveness of such learning algorithm depends heavily on the offline selected experts. Besides, such an algorithm has a limited ability to achieve a good power-performance tradeoff.

Tan et al. in [14], [15] proposed to use an enhanced Q-learning algorithm for DPM. This is a model-free RL approach since the PM does not require prior knowledge of the state transition probability functions. However, the knowledge of the state and action spaces and also the reward function is required. The enhanced Q-learning based DPM learns a policy online by trying to learn which action is the best for a certain system state, based on the reward or penalty (cost) 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 a discrete-time model of the stochastic process, and therefore has the following limitations: (i) the discrete-time controller has relatively high overhead to make frequent and regular decisions, and (ii) discrete-time controller may not make timely decisions for fast state changes.

In this paper, we present a novel approach for reinforcement learning-based, system-level DPM in a partially observable environment. Similar to the previous DPM work, we consider the power management of a specific I/O device (component), e.g., hard disk, WLAN card, or USB devices. The proposed DPM framework possesses the merits of the reference work [14], [15], i.e., being model free, and independent of any pre-designed experts. Moreover, the proposed approach can perform policy learning and power management in a continuous-time and event-driven manner, and therefore, it enables us to learn a desirable timeout policy.[1] Other original characteristics of the proposed DPM framework are the following

- The proposed method utilizes the enhanced TD(λ) learning algorithm for SMDP [25] in order to accelerate convergence and alleviate the reliance on the Markovian property.
- Workload prediction is incorporated in this work to provide partial information about service requester (SR) state for the RL algorithm. Specifically, an online Bayesian classifier [28] is chosen as the workload predictor because of its relatively high prediction accuracy, low implementation cost, and the fact that the information it provides comes with a certain degree of certainty due to the use of *posterior probability* [28].
- State and action spaces of the RL algorithm have been optimized, i.e., the total number of state-action pairs has been significantly reduced compared to the methods presented in [14], [15].
- To further increase the convergence speed and (or) enhance performance, several other improvements have been incorporated, including *multiple-update initialization*, *dynamic action sets*, and *locally randomized action selection* as detailed in Section 5.

In the proposed method, the tradeoff between system (component) power consumption and latency can be precisely controlled by a user-defined parameter. Experiments on both synthesized and real workload traces show that the proposed DPM framework achieves a much "deeper and

1. The timeout policy is the optimal DPM policy when the service request inter-arrival times are stationary but non-exponentially distributed [7], and [7] derives the optimal timeout value using Markov decision process methods.

wider" tradeoff curve between average power consumption and latency of the power-managed system (component), compared with prior work references. The maximum saving in power dissipation without sacrificing any latency is 18.6 percent compared to the reference expert-based approach proposed in [11]. Alternatively, the maximum latency saving without any power consumption increase is 73.0 percent compared to the existing best-of-bread DPM techniques. The proposed method works for single device (component) without dynamic voltage and frequency scaling (DVFS) capability [24]. Power management of multi-core systems is out of the scope.

2 Recent Work on Adaptive DPM Algorithms

There have been many recent advancements on adaptive DPM algorithms. For example, reference work [18] applies model predictive control approach for DPM that captures the temporal dynamics of system and user state, the cost of power consumption, as well as the effect of power-saving actions on the user and system. Later work [19], [20] apply the model predictive control approach for dynamic voltage and frequency scaling and multi-core systems. Such model predictive control approach is essentially learning the best-fit parameters of the Markov decision process modeling of the power-managed system, and then derive the optimal control policy by solving this Markov decision process. Compared with the model-free RL framework, such model-based approach (i) requires a detailed Markov decision process model of the power-managed system given in prior and (ii) it requires a significantly larger learning space than the RL algorithm, since the former requires to learn the state transition probabilities when taking a specific action in each state, whereas the latter only requires to learn the optimal action to choose in each state.

Similar to reference work [14], [15], other works such as [21], [22], [23] also apply discrete-time RL technique for DPM or DVFS for a digital multimedia system or a multi-core processor. Once again, the discrete-time RL technique results in a relatively large overhead in real implementations since the RL algorithm needs to make decision and evaluation in a periodic basis. This is especially true in the context of DPM for I/O devices, e.g., hard disk, WLAN card, or USB devices, in which the request processing time (e.g., reading/writing a page for the hard disk or sending/receiving a packet for the WLAN card) is order(s) of magnitude less than wakeup time or average request inter-arrival time. Moreover, the discrete-time controller may not make timely decisions for fast state changes (suppose a new request arrives but the controller can only make decision at the next discrete-time decision epoch, which results in additional delay.)

Finally, there is a set of reference work [16], [17] that address the adaptive control of power supply of embedded sensor nodes, in which the power supply can be made of battery, supercapacitor, or a hybrid system. This kind of adaptive control of power supply can be naturally combined with the RL-based DPM framework in this paper to derive a unified RL-based power management framework of the whole embedded system.

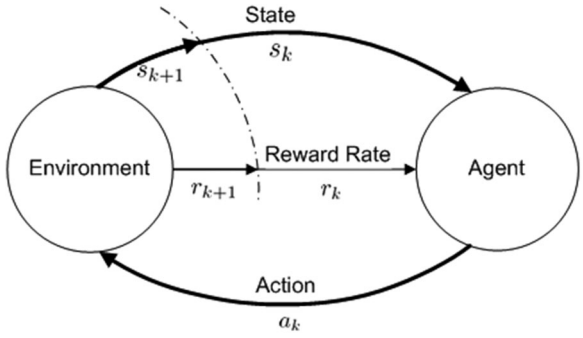


Fig. 1. Agent-environment interaction model.

3 Theoretical Background

3.1 Semi-Markov Decision Process

A stationary semi-Markov decision process is a continuous-time dynamic system comprised of a countable state set S and a finite action set A. The decision maker (DM) can choose action only when the system changes to a new 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 state s' with probability $p(s'|s,a)$ given that a is chosen in s. Furthermore, the next epoch occurs within t time units with probability $p(t|s,a,s')$ given s, a, and s'. Hence, the next epoch occurs at or before time t and the new state is s' with probability $f_{ss'}(t|a) \equiv p(s'|s,a) \cdot p(t|s,a,s')$. Let $T(s,a)$ denote the expected value of the duration that system occupies state s. We have $T(s,a) = \int_0^\infty \big(1 - \sum_{s' \in S} f_{ss'}(t|a)\big)dt$. If this duration is an exponentially distributed random variable, the SMDP reduces to a continuous-time MDP.[2]
- 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 state s'.)

A *policy* $\pi = \{(s,a)|s \in S, a \in A\}$ is a set of state-action pairs for all the states of an SMDP. We use notation $\pi(s) = a$ to specify the action that is chosen in state s according to policy π. An optimal policy is the one maximizing the total expected reward. We consider the class of *stationary* and *deterministic* policies unless otherwise specified. However, we shall see in Section 3.2 that we can learn *non-stationary* policy using RL techniques. Also, we shall see in Section 5.3 that the proposed improvement of locally randomized action selection allows us to learn randomized policy in the DPM framework.

3.2 Temporal Difference Learning for SMDP's

Fig. 1 illustrates the general model for reinforcement learning, which is comprised of an agent, a finite state space S, a set of all available actions A, and a reward function $R : S \times A \rightarrow R$.

Assume that the agent-environment interaction system evolves as a stationary SMDP, which is continuous in time

2. In the DPM framework, the distribution of T(s,a), e.g., the time to turn on, turn off the device or the process a service request, is generally far from exponential distribution [7]. In general, the time to turn on or turn off the device satisfies a uniform distribution or a Pareto distribution [7].

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, the system has just transitioned into state $s_k \in S$, and this information is captured by the agent. 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 that system operation 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 by $V^\pi(s)$, is the expected return when starting from state s and following policy π thereafter:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R|s_0 = s\} \\ &= \sum_{s' \in S} \int_{t_0}^{\infty} e^{-\beta(t-t_0)} \cdot V^\pi(s') df_{ss'}(t - t_0|\pi(s)) \\ &\quad + \sum_{s' \in S} \int_{t_0}^{\infty} \int_{t_0}^{t} e^{-\beta(\tau - t_0)} \cdot r(s, \pi(s)) d\tau \cdot df_{ss'}(t - t_0|\pi(s)), \end{aligned} \tag{1}$$

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

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

$$\begin{aligned} Q^\pi(s, a) &= E_\pi\{R|s_0 = s, a_0 = a\} \\ &= \sum_{s' \in S} \int_{t_0}^{\infty} \int_{t_0}^{t} e^{-\beta(\tau - t_0)} \cdot r(s, a) d\tau \cdot df_{ss'}(t - t_0|a) \\ &\quad + \sum_{s' \in S} \int_{t_0}^{\infty} e^{-\beta(t-t_0)} \cdot Q^\pi(s', \pi(s')) df_{ss'}(t - t_0|a). \end{aligned} \tag{2}$$

Now suppose that we want to estimate the value function $V^\pi(s)$ for some state s. However, the agent has no predefined policy or prior knowledge about state transition probabilities and transition times, which are essential for characterizing an SMDP. Therefore, traditional value iteration or policy iteration methods cannot be applied here. Instead, a simple one-step temporal difference learning method [26] (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 estimate of the actual value $V^\pi(s)$ following policy π. Suppose that 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} (which is observed by the agent) as follows [25]:

$$\begin{aligned} V^{(k+1)}(s_k) &= V^{(k)}(s_k) \\ &+ \alpha \left(\frac{1 - e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} V^{(k)}(s_{k+1}) - V^{(k)}(s_k) \right). \end{aligned} \tag{3}$$

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 received in τ_k time units; and $V^{(k)}(s_{k+1})$ is the estimated value of the actually occurring next state. Please note that whenever state s_k is visited, its estimated value is updated to be closer to $\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} V^{(k)}(s_{k+1})$, where $r(s_k, a_k)$ is the instantaneous reward received and $V^{(k)}(s_{k+1})$ is the estimated value of the actually occurring next state. 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 (i.e., it slowly decreases) and policy is kept unchanged, it is proved that TD(0) will converge to the optimal value function [26]

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 (i.e., make decisions.) To achieve this goal, the RL algorithm should learn the value of each *state-action pair* (s, a). Meanwhile, the agent should choose an action at each state in order to obtain high potential return. More specifically, such method generates an estimate $Q^{(k)}(s, a)$ for each state-action pair (s, a) at epoch t_k. Suppose that state s_k is visited at epoch t_k, then at that epoch the agent chooses an action either with the maximum estimated value $Q^{(k)}(s_k, a)$ for various actions $a \in A$, or by using other semi-greedy policies [26]. The ε- policy is a widely adopted semi-greedy policy, where the action with the maximum estimated value is chosen with probability $1 - \varepsilon$ and a random action is chosen with probability ε. This policy well balances between *exploitation versus exploration*, in order to overcome to risk of getting stuck in a sub-optimal solution.[3] Moreover, the TD learning rule updates the estimate $Q^{(k)}(s_k, a_k)$ at the next epoch t_{k+1}, based on the chosen action a_k, and the next state s_{k+1}.

3.3 TD(λ) Learning for SMDP's

Because a realistic DPM problem is non-Markovian and non-stationary, we turn to the more powerful TD(λ) algorithm [26]. TD(λ) behaves more robustly in non-Markovian cases because it seamlessly combines the simple one-step TD algorithm and the Monte Carlo method (which does not rely on the Markovian property assumption.) The learning rate of TD(λ) is also faster.

Suppose that we are in state s_k at epoch t_k, and the agent makes decision a_k. In one-step TD learning, we wait until the next epoch t_{k+1} and then perform a "one-step backup" to update the estimate $V^{(k)}(s_k)$. In one-step backup, the *target* is the immediate reward plus the discounted estimated value of the next state [25], i.e.,:

$$R_k^{(1)} = \frac{1 - e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} V^{(k)}(s_{k+1}). \tag{4}$$

Similarly, we could perform two-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 two-step backup is given by:

$$\begin{aligned} R_k^{(2)} &= \frac{1 - e^{-\beta\tau_k}}{\beta} r(s_k, a_k) + e^{-\beta\tau_k} \cdot \\ &\frac{1 - e^{-\beta\tau_{k+1}}}{\beta} r(s_{k+1}, a_{k+1}) + e^{-\beta(\tau_k + \tau_{k+1})} V^{(k)}(s_{k+2}), \end{aligned} \tag{5}$$

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}.

3. An RL agent must exploit the best action known so far to gain rewards while exploring all the possible actions to find a potentially better choice. If the action with temporarily highest Q value is always chosen, we take the risk of getting stuck in a sub-optimal solution.

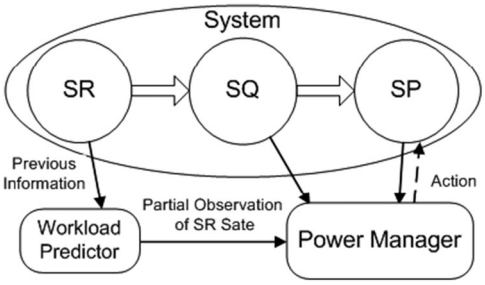


Fig. 2. Abstract model of a power-managed system.

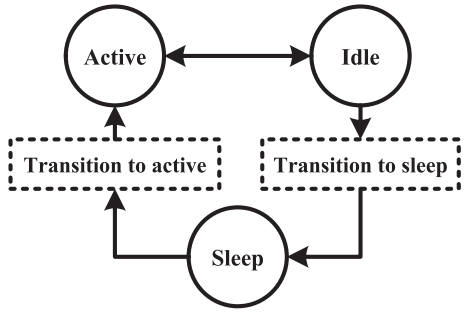


Fig. 3. State diagram of the SP.

This result can be easily generalized to n-step backup for arbitrary n. When $n \to \infty$, the n-step backup algorithm becomes the Monte Carlo method, which relies on repeated random sampling to compute optimal policy. However, n-step backup is rarely used directly because it is difficult to implement. Instead, people seek to find effective ways of averaging backups of different steps.

The TD(λ) learning 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} ($0 < \lambda < 1$). Notice that the one-step backup is given the largest weight $1 - \lambda$. The resulting target is:

$$R_k^{\lambda} = (1-\lambda) \cdot \sum_{n=1}^{\infty} \lambda^{n-1} \cdot R_k^{(n)}. \tag{6}$$

The TD(λ) learning algorithm can be conveniently implemented with the help of *eligibility traces*, as discussed in [26]. Among variant specific implementations, the one implemented in the proposed system is Watkin's Q(λ) algorithm [27] modified for SMDP problems due to a joint consideration of effectiveness, robustness and convergence rate. 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 given as follows:

$$\begin{aligned} &\forall (s,a) \in S \times A : \\ &Q^{(k+1)}(s,a) = Q^{(k)}(s,a) + \alpha \cdot e^{(k)}(s,a) \cdot \\ &\left(\frac{1-e^{-\beta \tau_k}}{\beta} r(s_k,a_k) + \max_{a'} e^{-\beta \tau_k} Q^{(k)}(s_{k+1},a') - Q^{(k)}(s_k,a_k) \right), \end{aligned} \tag{7}$$

where $Q^{(k)}(s,a)$ is the value of state-action pair (s,a) at decision epoch t_k, and $e^{(k)}(s,a)$ denotes the eligibility of that pair. Such eligibility reflects the degree of how " recently" and " frequently" state-action pair (s,a) has been chosen in the recent past. More specifically, if state-action pair (s,a) has been visited more recently before epoch t_k, $e^{(k)}(s,a)$ is higher. Or if (s,a) has been visited more frequently in the recent past, $e^{(k)}(s,a)$ is also higher. It can be updated online as follows:

$$e^{(k)}(s,a) = \lambda e^{-\beta \tau_{k-1}} e^{(k-1)}(s,a) + \delta((s,a),(s_k,a_k)), \tag{8}$$

where $\delta(x,y)$ denotes the delta kronecker function. Please note that parameters β and λ will affect the convergence speed. A larger β value will accelerate convergence but may degrade the solution quality (because it puts too much discounts for future), and parameter λ also has similar effect. In our work, the appropriate value of β is around 0.05 - 0.1 and value of λ is around 0.7. This is similar to general conclusion in reinforcement learning [28].

4 System Model

In this section, we explain how to extend the RL techniques to the system-level DPM framework. Similar to the previous work in DPM [11], [14], [15], the system whose power is being managed is comprised of a service requester (SR), a service provider (SP), and a service queue (SQ). Again similar to previous DPM work, we consider a specific I/O device (component), e.g., hard disk, WLAN card, or USB devices, as the SP.[4] In this paper, we focus on reducing the power consumption and finding a near-optimal power-latency tradeoff of the SP. In this framework, the SR generates different types of requests to be processed by the SP, and these requests are buffered in the FIFO queue SQ before processing. The exact generating time instances of service requests are not known *a priori*. A power manager (PM) using the RL algorithm, as well as a workload predictor is added to the system, as illustrated in Fig. 2.

The SP has three main states as shown in Fig. 3. It is in the active state while processing service requests, and after it has finished, it becomes idle. When the SP is in the idle state, it can autonomously and instantaneously transit to the active state as soon as any task arrives. Unfortunately, the SP has non-zero power consumption in idle state. It can, however, go to the sleep state from the idle state. A sleeping SP consumes little power compared to an idle one, but it suffers from large wakeup latencies along with high power consumption during the transition to active state. Our goal is to properly schedule the sleep time for the SP in order to reach the balance between latency and energy consumption.

As shown in Fig. 2, the SP (the I/O device) is controlled by the PM, which is implemented using RL-based DPM algorithm. The PM monitors the number of requests waiting in the SQ, the current SP state (active, idle, sleep), as well as the (estimated) current state of SR, and consequently makes decisions (adjusts the power state of the SP.) The SR state is the service request generating rate (high, medium, low) or the next inter-arrival time (short, long, etc.) There are two decision points for the PM: First, every time the SP

4. In this paper we focus on I/O components without dynamic voltage and frequency scaling (DVFS) capability [24]. Power management of DVFS-enabled modules, e.g., CPU, GPU, and multi-core systems is out of the scope of this work.

transits from active to idle state, the PM will make a decision on whether to let the SP go to sleep straightaway or set a timeout. If a timeout is set and no requests arrive during this period, the device will subsequently go to sleep. Second, while SP is in the sleep state, the PM decides whether or not to wake up the SP based on the number of waiting requests in the SQ. To be more realistic, we consider in this paper that the exact SR state cannot be directly obtained by the PM. In contrast to the previous work on POMDP [8], [9], the PM has no prior knowledge of the characteristics of the SR. Therefore, workload prediction has to be incorporated to provide partial information (estimation) of the SR state to the PM so that the PM can effectively learn in the observation domain of SR. We adopt the online Bayesian predictor for workload prediction, as shall be discussed in Section 4.1. After that we will discuss about the implementation details of the RL-based DPM method in Section 4.2.

4.1 The Bayesian Workload Predictor

The proposed DPM framework relies on workload prediction to provide partial observation of the actual SR state for the PM, and therefore, the SR state used by the PM to make decisions is essentially the observation state. The workload prediction method needs not be 100 percent accurate, because the RL technique is robust enough to handle not-so-accurate SR state estimations. Extensive experiments show that the partial observation (i.e., estimations) about the SR state can significantly enhance the performance of the PM.

Previous work on workload prediction in [3], [4] 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 significantly affect a set of subsequent predictions (because the linear combination will likely to be large.) Therefore a naive Bayesian classifier, which can overcome the above effect by encoding the inter-arrival times into "long, short, etc." values (therefore it will not treat a very long inter-arrival time differently than other "long" inter-arrival times) and result in a much higher prediction accuracy, is adopted in this work as workload predictor. Moreover, the proposed predictor is simple to implement, and works in an online manner.

Another benefit of the online Bayesian classifier is: The partial information it provides ensures certain degree of certainty, due to the use of *posterior probability* in such an algorithm. For example, suppose that the service request inter-arrival times are 80 percent short and 20 percent long. Then a trivial predictor, which predicts all the inter-arrival times to be short, will result in a high overall prediction accuracy of 80 percent. However, such a predictor cannot discriminate between the SR states, and therefore cannot provide useful information for the PM. For the Bayesian classifier, on the other hand, the prediction accuracy will always be higher than 50 percent when it makes a prediction of either short or long inter-arrival times. This is because of having only two possible outcomes "short" or "long" the simple case, and the prediction accuracy will be higher than 50 percent due to the using of maximum a posteriori (MAP) information. Hence, the Bayesian classifier can provide effective partial information about SR state discrimination that is required by the PM.[5]

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

$$\begin{aligned} y_{\text{MAP}} &= \underset{l}{\text{argmax}}\, Prob(y = l|x_1, x_2 \ldots, x_n) \\ &= \underset{l}{\text{argmax}} \frac{Prob(x_1, x_2, \ldots, x_n|y = l) \cdot Prob(y = l)}{Prob(x_1, x_2, \ldots, x_n)}. \end{aligned} \tag{9}$$

In Eqn. (9), the denominator $Prob(x_1, x_2, \ldots, x_n)$, which is the probability of witnessing the new input feature $\boldsymbol{x}$, is irrelevant for decision making since it is the same for every class assignment of y. $Prob(y = l)$, which is the *prior* (pre-evidence) *probability* of the hypothesis that the class of y is l, is easily calculated from the training set. Hence, we only need to calculate $Prob(x_1, x_2, \ldots, x_n|y = l)$, which is the conditional probability of seeing the input feature vector $\boldsymbol{x}$ given that the class of y is l.

A fundamental assumption of the naive Bayesian classifier is that all the input features are *conditionally independent* given the class y, e.g., $Prob(x_1|x_2, \ldots, x_n, y = l) = Prob(x_1|y = l)$. Then we have $Prob(x_1, x_2, \ldots, x_n|y = l) = \prod_j Prob(x_j|y = l)$, and we compute the MAP class of y as follows:

$$y_{\text{MAP}} = \underset{l}{\text{argmax}}\, Prob(y = l) \cdot \prod_{j=1}^{n} Prob\big(x_j|y = l\big). \tag{10}$$

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

$$\begin{aligned} \forall i \in \{1, 2, \ldots, n\} : Prob(x_i = m_i|y = l) \\ \leftarrow \alpha + (1 - \alpha) \cdot Prob(x_i = m_i|y = l) \end{aligned} \tag{11}$$

$$Prob(y = l) \leftarrow \beta + (1 - \beta) \cdot Prob(y = l), \tag{12}$$

where $\alpha, \beta \in (0, 1)$ denote the updating rate parameters. They are typically set to a value of 0.8 or higher. Of course, all the probability values should be normalized after updating. Please note that this online adaptive implementation will only converge to the same results as (10) when the

5. This advantage also holds when comparing Bayesian classifier with logistic regression [28] or linear regression that encodes inter-arrival times into "long, short, etc." values. The inherent assumption of logistic regression is still that a linear combination of past inter-arrival times can be used to infer future one, whereas Bayesian classifier is better in the case of patterned but not linear relationships. For example, if the inter-arrival time pattern is always "long, long, long, short" then the Bayesian classifier can predict the pattern while logistic regression cannot.

stable period is far longer than the convergence time, which could be satisfied in our experiments.

In this work, we use previous service request inter-arrival times as the input features $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, where $x_i = 1$ if the corresponding inter-arrival time is greater than the break-even time T_{be}; otherwise, we have $x_i = 0$. The predictor's 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 request inter-arrival time to be " unknown" if the difference between the posterior probabilities that the next inter-arrival time is long and that it is short is less than ε, i.e., $|Prob(y = 1|x_1, x_2 \ldots, x_n) - Prob(y = 0|x_1, x_2 \ldots, x_n)| < \varepsilon$, where ε is a small predefined parameter. In this way the prediction accuracy can be further enhanced. Experimental results about the proposed Bayesian workload predictor are reported in Section 6.

The Bayesian predictor requires negligible computation and storage overheads. Upon arriving of each sevice request, the computation complexity of the Bayesian predictor is $O(\#\text{offeatures} \times \#\text{ofpredictionoutcomes})$. In our implementation the number of features is 4 - 6 and the number of prediction outcomes is three, thereby resulting in negligible computation and storage overheads.

4.2 RL-Based DPM

In this section, we explain how to extend the RL techniques for the DPM framework.

We provide a number of key definitions and notation next. Let t_j^G $(j = 1, 2, 3, \ldots)$ denote the time when the jth request is generated by the SR. Suppose that we are currently at time t with $t_j^G \leq t < t_{j+1}^G$. Note that the PM has no exact information on t_{j+1}^G at the current time t. Instead, it only receives an estimation (long, short, unknown, etc.) on the service request inter-arrival time $t_{j+1}^G - t_j^G$ from the Bayesian classifier.

The PM monitors the following three *state parameters* of the power managed system:

- The SP state $SP(t)$, which is the component power state (active, idle, sleep, etc.)
- The SQ state $SQ(t)$, which is the number of requests buffered in SQ.
- The estimated SR state, which is represented in this paper by the estimation (long, short, unknown, etc.) on the request inter-arrival time $t_{j+1}^G - t_j^G$ from the online Bayesian predictor.

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

1. The SP entered idle state ($SP(t) = \text{idle}$) and $SQ(t) = 0$.
2. The SP has just entered the sleep state and finds that $SQ(t) > 0$.
3. The SP is in the sleep state and a new service request arrives.

We denote the kth $(k = 1, 2, 3, \ldots)$ decision epoch by t_k^D. There are two types of decision epochs. If at decision epoch t_k^D, we have $SP(t_k^D) = \text{idle}$ and $SQ(t_k^D) = 0$ (case 1), we call the decision epoch t_k^D an *idle-state decision epoch*. On the other hand, if at decision epoch t_k^D, we have $SP(t_k^D) = \text{sleep}$ (case 2 or case 3), we call the decision epoch t_k^D a *sleep-state decision epoch*.

In the TD(λ) learning algorithm adopted in the PM, at each decision epoch the power managed system is in some particular state, denoted by $s(t_k^D)$, which is used for the PM to make decisions and perform value updating. We call it the *RL state*. Obviously, the RL state can be characterized by the SP power state (idle, active, sleep), as well as other system state parameters. We provide a classification of RL states based on two definitions.

Definition 1. *The 1st class of RL state:*

We define that the RL state at decision epoch t_k^D, denoted by $s(t_k^D)$, belongs to the 1st class of RL states if t_k^D is an idle-state decision epoch. At this decision epoch t_k^D, we have $SP(t_k^D) = \text{idle}$ and $SQ(t_k^D) = 0$. Then the RL states belonging to the 1st class are further categorized by the estimated SR state from the online Bayesian predictor, which is, in fact, the estimation of the request inter-arrival time $t_{j+1}^G - t_j^G$ (if we suppose $t_j^G \leq t_k^D < t_{j+1}^G$.)

Definition 2. *The 2nd class of RL state:*

We define that the RL state at decision epoch t_k^D, denoted by $s(t_k^D)$, belongs to the 2nd class of RL states if t_k^D is a sleep-state decision epoch. We know that $SP(t_k^D) = \text{sleep}$ at this decision epoch t_k^D. Then the RL states belonging to the 2nd class are further categorized by (i) the estimated SR state from the online Bayesian predictor, and (ii) the $SQ(t_k^D)$ value, i.e., the number of requests waiting in the SQ at decision epoch t_k^D.

We only consider the $SQ(t_k^D)$ value in the range $0 < SQ(t_k^D) \leq Max\_SQ$, where Max_SQ is the predefined maximum $SQ(t_k^D)$ value. This is because if $SQ(t_k^D) > Max\_SQ$, the only possible action chosen by the PM at decision epoch t_k^D would be turning the SP on to the active state to process service requests. On the other hand, if $SQ(t_k^D) = 0$, the only possible action chosen by the PM would be keeping the SP in the sleep state until the next service request arrives.

The action space for the 1st class of RL states, denoted by A_1, is given by $A_1 = \{0 \cdot T_{be}, 0.2T_{be}, 0.5T_{be}, 0.8T_{be}, 1T_{be}, 2T_{be}, 3T_{be}, 5T_{be}, \ldots\}$ (as an example), where those actions correspond to different timeout values. Among those actions, $0 \cdot T_{be}$ corresponds to " immediate shut-down". Note that as pointed out in reference [7], the optimal policy when the SP is idle for non-Markovian environments is often a timeout policy, wherein the SP is put to sleep if it is idle for more than a specific timeout period. The proposed PM learns to choose the optimal action among action set A_1 by using a RL technique. Moreover, the use of timeout values as actions allows for " multiple updating" as well as the use of " dynamic action sets", as we shall discuss in the following section. Finally, the action space for the 2nd class of RL states, denoted by A_2, is given by $A_2 = \{\text{keepsleep}, \text{goactive}\}$. In other words, there are two possible actions in

TABLE 1
A Summary of States and Actions Used in the DPM Framework

	State differentiation (number)	Actions
Idle-state	Estimated SR state (3)	A set of timeout values (8)
Sleep-state	Estimated SR state (3) × No. of waiting requests (5)	" Keep sleep" or " Wake-up" (2)

these RL states: keeping the SP in the sleep state until the next request comes, or going to active to process the service requests buffered in the SQ.[6]

In the proposed DPM framework, we have significantly reduced the number of state-action pairs compared with the reference work [14], [15]. The total number of state-action pairs for the 1st class of RL states is $3 \times 8 = 24$ since (i) we use three estimated SR states long, short, and unknown, and (ii) there are eight possible actions in A_1, i.e., $|A_1| = 8$, as discussed before. On the other hand, if we set $Max\_SQ = 5$, the total number of state-action pairs for the 2nd class of RL states is $5 \times 3 \times 2 = 30$ since we have $|A_2| = 2$. Hence, the total number of state-action pairs in the proposed DPM framework is $24 + 30 = 54$, which is significantly less than that in [14], [15]. Table 1 provides a summary of the states and actions used in the proposed framework. In general, the number of decision epochs for an RL algorithm to converge is three to five times the number of state-action pairs (54 in our implementation.) In our RL algorithm, experimental results demonstrate that the RL algorithm will converge within 200 – 300 service request arrivals, demonstrating the fast convergence speed of the proposed method.

In this paper, we use "*cost rate*" instead of "*reward rate*" in the RL algorithm. In fact, the cost rate is simply the negative value of the reward rate, and therefore is compatible with the RL framework. The cost rate $\cos t(t) = w \cdot P(t) + (1 - w) \cdot SQ(t)$ is a linearly-weighted combination of SP power consumption $P(t)$ and the number of requests $SQ(t)$ buffered in the SQ, where w is the weighting factor. This is a reasonable cost rate because as reference [6] has pointed out, the average number of service requests buffered 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 queueing time plus execution time. The above observation is based on the Little's Law [29]. 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 service requests. Since the total number of service requests and the total execution time are fixed, the value function is equivalent to a linear combination of the average power consumption and per-request latency. The relative weight w between the average power consumption and per-request latency can be adjusted to obtain the Pareto-optimal power-latency tradeoff curve.

6. We only use two actions "keep sleep" and "wake-up", instead of using a time-out policy, mainly due to convergence speed considerations. If we use a time-out policy similar to that used for idle-state decision epochs, the number of actions will be increased by several times, and then the convergence speed will be reduced even more significantly (this is because sleep state is less frequently visited than arrival rate of requests.)

Algorithm 1. The RL-Based DPM Algorithm.

At each decision epoch t_k^D (the power-managed system is in RL state $s(t_k^D)$:
Choose $a(t_k^D) = \text{argmin}_{a \in A_{class(s(t_k^D))}} Q^{(k)}(s(t_k^D), a)$ with probability $1 - \varepsilon$; choose a random action with probability ε.
If $class(s(t_k^D)) = 1$:
The chosen action corresponds to a specific timeout value, say, $x \cdot T_{be}$.
The SP waits in the idle state for a period of time with duration $x \cdot T_{be}$.
If some request comes during that period of time:
Keep SP in active state for processing requests until SQ is empty again. Then we arrive at decision epoch t_{k+1}^D.
The controller calculates power consumption including both idle and busy power states, and the request latency is the processing time in the busy state.
Else:
Go to the sleep state and wait until the SQstate > 0 (i.e., some request comes during the idle-to-sleep transition or after the SP is in the sleep state.) Then we arrive at decision epoch t_{k+1}^D.
End
Else:
If $a(t_k^D) =$ keep sleep:
Keep the SP in the sleep state until the next service request comes. Then we arrive at decision epoch t_{k+1}^D (another RL state with number of waiting requests incremented by 1).
Else:
Turn the SP active to process service requests until SQ is empty again. Then we arrive at decision epoch t_{k+1}^D.
End
End
$\tau_k = t_{k+1}^D - t_k^D$.
The cumulative cost $State\_Cost = \int_{t_k^D}^{t_{k+1}^D} e^{-\beta(t - t_k^D)} cost(t) \cdot dt$.
$\Delta = State\_Cost +$
$\min_{a\prime} e^{-\beta\tau_k} Q^{(k)}(s(t_{k+1}^D), a\prime) - Q^{(k)}(s(t_k^D), a(t_k^D))$.
For each RL state s, action $a \in A_{class(s)}$:
$e^{(k)}(s, a) \leftarrow$
$\lambda e^{-\beta\tau_{k-1}} \cdot e^{(k-1)}(s, a) + \delta((s, a), (s(t_k^D), a(t_k^D)))$.
$Q^{(k+1)}(s, a) \leftarrow Q^{(k)}(s, a) + \alpha \cdot \Delta \cdot e^{(k)}(s, a)$.

The details of the proposed RL-based DPM algorithm are provided in Algorithm 1, and an outline of the algorithm is given as follows. Suppose that we are now at decision epoch t_k^D, and the system under control is in RL state $s(t_k^D)$. At that decision epoch in the proposed algorithm, the PM maintains value estimates $Q^{(k)}(s, a)$ for each RL state s, and each action $a \in A_{class(s)}$, where $class(s)$ denotes which particular class of RL states (the 1st or 2nd) the RL state s is in. Then at decision epoch t_{k+1}^D, the value estimates $Q^{(k)}(s, a)$ of all state-action pairs (s, a)'s are updated according to the TD(λ) learning rule stated in Eqn. (7). The Q-value updating process of the RL-based DPM algorithm is illustrated in Fig. 4. Please 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 the utilized

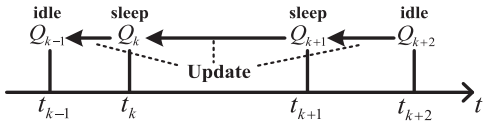


Fig. 4. *Q*-value updating diagram. An arrow from B to A means that A is updated based on the information provided by B.

TD(λ) learning technique, as well as other improvements been made (see Section 5).

The proposed RL-based DPM algorithm has low computation and storage complexity. As can be seen from Algorithm 1, the computation complexity at each decision epoch is $O(\#\text{ofstate-actionpairs})$, which is 54 in our implementation.

5 System Improvements

In this section, we discuss several improvements we made to the DPM framework including "multiple updating", "dynamic action set", and "locally randomized action selection".

5.1 Multiple-Update Initialization

The use of various timeout values as actions enables us to perform multiple updates. When the SP is in the idle state, the PM (i.e., the DPM agent) takes an action corresponding to a specific timeout value. If a request arrives before the timeout period expires, all the actions with a larger timeout value can evaluate (using the "backup" method as described in Section 3.) This is because all the actions with a larger timeout value, if taken, would result in the same immediate cost and the discounted next state value compared to the selected action (since the actually occurring next state would be the same.) On the other hand, for those actions with a smaller timeout value, the SP may enter the sleep state and the time-to-sleep may be a random variable. In this case, we cannot accurately estimate the cost and discounted next state value, and thus we do not update the corresponding Q values.

The proposed multiple-update scheme can accelerate the convergence speed of RL algorithm significantly, but some actions may be evaluated more often than the others. Therefore, in this work the multiple-update scheme is only utilized for quick Q value initialization.

5.2 Dynamic Action Set

In the continuous-time problem setting, the action space of timeout values should also be continuous. We use discrete action space because it is difficult for RL algorithms to learn from a continuous one. However, we may lose optimality through discretizing the continuous space because we may not be able to find the exact optimal action in a continuous space. Therefore, the " dynamic action set" method is proposed to mimic a continuous timeout action space, which may result in deeper power-latency tradeoff. Besides, it can alleviate the negative effect caused by action set with very few actions or actions that are not carefully selected.

There are mainly two operations updating the dynamic action set: *expansion* and *contraction*, as shown in Fig. 5. Consider the power managed system after multiple-updates initialization. At that time each state-action pair is associated

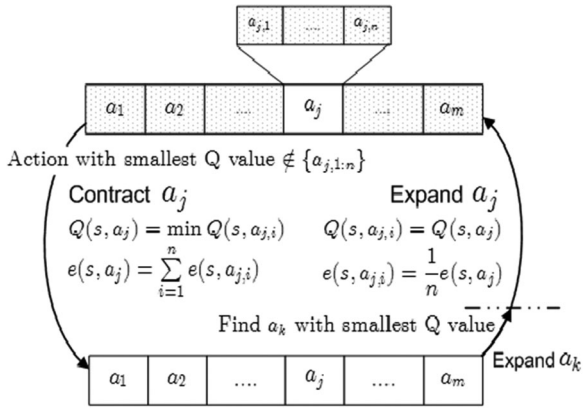


Fig. 5. Operations on the dynamic action set.

with a Q value $Q(s,a)$ and an eligibility trace value $e(s,a)$. Suppose that for state s, action a_j has the minimal Q value, which means when the power managed system is in state s, action a_j is the optimal action to choose among the action set $\{a_1, a_2, a_3, \ldots, a_m\}$. It is natural to think that the actual optimal timeout action in the continuous space should be in the neighborhood of a_j, and it is desirable to conduct a finer-grained search for the optimal timeout action in the neighborhood of a_j. Based on this observation, we expand a_j to a set of actions $\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$. For example, the action $0.5 \cdot T_{be}$ can be expanded to a set of actions $\{0.4 \cdot T_{be}, 0.45 \cdot T_{be}, 0.5 \cdot T_{be}, 0.55 \cdot T_{be}, 0.6 \cdot T_{be}\}$. After that, the PM begins to evaluate the new expanded action set $\{a_1, a_2, \ldots, a_{j-1}, a_{j+1}, \ldots, a_m, a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$ on that state. Please note that when expanding an action a_j, we have to generate the Q values and eligibility trace values for the new actions $\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$. We generate these values in the following way:

$$\text{for each } i : Q(s,a_{j,i}) = Q(s,a_j), e(s,a_{j,i}) = \frac{e(s,a_j)}{n}. \quad (13)$$

On the other hand, because of the non-stationary nature of real workload, it is likely that after some time, the optimal action at state s is not in the set $\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$ any more. In other words, the action with the smallest Q value does not belong to that set. In this case, still maintaining the expanded set $\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$ will result in additional computation and convergence overhead. Therefore, we need to contract the expanded set $\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$ to the single action a_j as shown in Fig. 5. Similarly, we need to generate the Q value and eligibility trace value for the action a_j :

$$Q(s,a_j) = \min_i Q(s,a_{j,i}), \quad e(s,a_j) = \sum_{i=1}^{n} e(s,a_{j,i}). \quad (14)$$

After performing contraction to the action set, we find a_k among the action set $\{a_1, a_2, a_3, \ldots, a_m\}$ with the smallest Q value and expand a_k to a set of actions $\{a_{k,1}, a_{k,2}, \ldots, a_{k,n}\}$ with the above-mentioned procedure. In this way, the PM maintains a dynamic action set in which actions are distributed densely near the optimal action in the continuous action space.

Compared with a large action set with, say, $m \times n$ actions, using dynamic action set will result in much less convergence

TABLE 2
Power and Delay Characteristics of the 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

overhead, especially when we update the dynamic set every 50 or 100 service requests (not on a per request basis.) The overhead for performing ε-policy also decreases. Moreover, the usage of dynamic action set also makes the implementation of " locally randomized action selection" straightforward, as we shall discuss in the next section.

5.3 Locally Randomized Action Selection

As shown in the literature [6], the optimal policy minimizing power consumption with certain delay constraint is a randomized policy. However, it is difficult to learn a global randomized policy (selecting with certain probabilities among all actions) with RL techniques. Several RL techniques, such as the actor-critic method [26], provide the possibility of learning randomized policy through the use of *softmax selections*. However, the parameters are difficult to set in such algorithms. The system either has large overhead due to "over-randomness" or becomes nearly greedy (with too little randomness).

The utilization of dynamic action set allows for locally randomized action selection. Consider the expanded actions from an original action as described in Section 5.2. These expanded actions form a "local action set" while the others are in the "global action set". the global decision of whether selecting the local action set or the global action set is taken by the ε-greedy policy, i.e., select the local action set with probability 1-ε and the global set with probability ε. The actions in local action set are selected by the soft-max policy. If the global action set is selected, each of its actions are selected with equal probability.

The softmax selection method utilizes a Boltzmann distribution. When the system is in state s, the PM chooses action a from the local action set with probability $Prob(s,a)$ given by:

$$Prob(s,a) = \frac{e^{-Q(s,a)/\tau}}{\sum_{b \in \text{localactionset}} e^{-Q(s,b)/\tau}}, \tag{15}$$

which implies that more desirable action a with smaller $Q(s,a)$ value will be chosen with higher probability. With the proposed locally randomized action selection method, we have the potential to learn the optimal randomized policy, with much less computation overhead compared to the global version.

6 Experimental Results

In this section, we present experimental results of the RL-based DPM with workload prediction and other improvements on two different devices: a hard disk drive (HDD) and a wireless adapter card (WLAN card.) Tables 2 and 3 list the power and delay characteristics of both devices, which are the state-of-the-art values similar to [15], [24]. In these tables, T_{tr} is the time taken in transitioning to and from the sleep state, whereas E_{tr} is the energy consumption in waking up the device. T_{be} refers to the break-even time. The meanings of other characteristics can be easily inferred from their names.

TABLE 3
Power and Delay Characteristics of the 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 systems, we use the expert-based DPM developed in [11], [12]. Three policies are adopted as experts in the expert-based DPM: fixed timeout policy, adaptive timeout policy, and exponential predictive policy [4], as shown in Table 4. We also adopt the discrete-time reinforcement learning framework proposed in [14], [15] as baseline system. We use 0.2s as the decision period as an effective tradeoff between decision overhead and system performance, and use $Max\_SQ = 5$ (same as our work.)

In the rest of this paper, we refer to the "simple RL-based power-managed system" as the DPM system that can only make decisions when the SP is idle (and only the first class of RL states are utilized). This is for fair comparison with the baseline expert-based algorithms since the baseline expert-based algorithms cannot make decisions in the sleep state. The simple RL system has three different RL states on which we can make decisions: (idle, SR rate high), (idle, SR rate low), and (idle, SR rate hard to decide). Similarly, we refer to the "whole DPM system" as DPM system that can make decisions when SP is at both idle or sleep states as described in Section 4.2.

6.1 Hard Disk Drive

For the HDD, we simulate based on the synthesized SR model, which is a continuous-time Markov process model with three different states, each corresponding to a different service request generating rate. This kind of continuous-time Markov process model is a widely used modeling technique for workloads where request inter-arrival times are non-exponentially distributed [6]. 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 service request generating rates for the three states are given by 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 the SR is in; rather it relies on the workload predictor for estimations of the SR state.

TABLE 4
Characteristics of the Expert-Based Policy

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

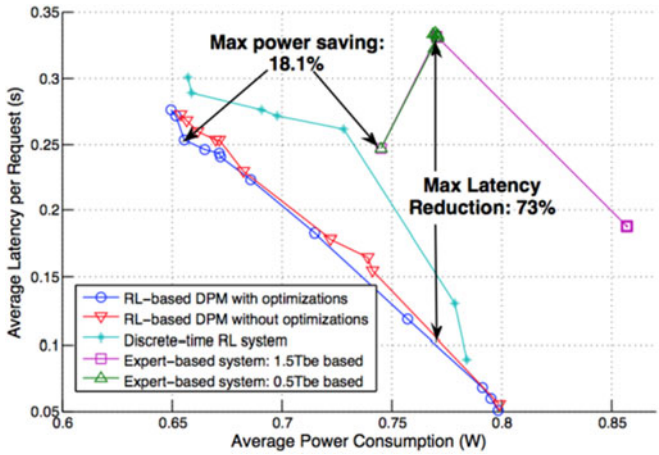


Fig. 6. Power-latency tradeoff curves for the HDD: The simple system.

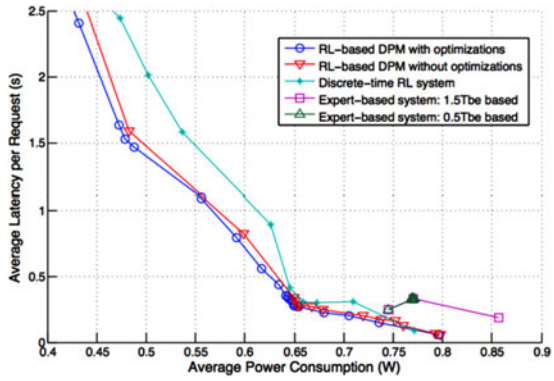


Fig. 7. Power-latency tradeoff curves for the HDD: The whole system.

Fig. 6 provides the power-latency tradeoff curves for the simple RL-based power-managed system with workload prediction, the two different expert-based DPM frameworks, as well as the discrete-time RL framework. The timeout values of the fixed timeout expert in those two systems are set to be $0.5 \cdot T_{be}$ and $1.5 \cdot T_{be}$, respectively, which are typical timeout values in the DPM framework [7], [11], [14]. We show experimental results on both DPM frameworks with and without the three further optimization methods (i.e., multiple-update initialization, dynamic action set, and locally randomized action selection). For the DPM framework without further optimization, we set the action (timeout) values to be $\{0 \cdot T_{be}, 0.2T_{be}, 1T_{be}, 2T_{be}\}$. We have the following observations from the figure: (i) the tradeoff curve of the RL-based power-managed system is more evenly distributed and has a much wider tradeoff range.[7] Even with the same average latency, the RL-based DPM framework can achieve much lower power consumption than the references. The maximum power saving with the same average latency is 18.1 percent. On the other hand, the maximum saving in average latency is more than 73.0 percent without any increase in the average power consumption. (ii) Comparing the performance of the DPM framework with further optimizations and that without further optimizations, we can see that the former consistently outperforms the latter although their performances converge at certain points. More specifically, the maximum power saving is 2.2 percent with the same average latency, whereas the maximum reduction in average latency is 12.7 percent when the average power consumption remains the same. The higher performance mainly results from the optimization of dynamic action set, which can alleviate the negative effect caused by action set with very few actions or actions that are not carefully selected.

When comparing with the discrete-time RL-based DPM framework, one can still conclude that our proposed RL-based DPM framework can achieve a "deeper and wider" tradeoff curve, and also achieve lower average power consumption when the average latency is the same (and vice versa). The main reasons are: (i) the discrete-time controller has relatively high overhead to make frequent and regular decisions, and (ii) discrete-time controller may not make timely decisions for fast state changes (suppose a new request arrives but the controller can only make decision at the next discrete-time decision epoch, which results in additional delay.)

Fig. 7 provides the power-latency tradeoff curves for the whole DPM-based power-managed system with workload prediction, as well as two expert-based DPM frameworks which is the same as discussed before. Comparing these two figures, we can see that with the help of making decisions at sleep state, power consumption can be further minimized to less than 2/3 of the minimal power consumption achievable by the simple RL-based DPM framework by sacrificing performance.[8] In fact, around 1s average latency is acceptable for many delay-tolerant applications on the HDD or WLAN cards. This is especially true for portable devices in which power and energy consumption minimization is the primary goal, and therefore reducing power consumption by 1/3 to 1/2 is worthwhile. When comparing the performance of DPM frameworks with and without further optimizations, we can observe a higher reduction of power consumption up to 4.2 percent when the average latency remains the same.

6.2 WLAN Card

For the WLAN card, we have measured several real traces using the *tcpdump* utility in Linux. Although tcpdump fails to capture the MAC layer retransmissions, it affords a fairly good approximation of WLAN activities. The measured traces include a 45-minute trace for online video watching, a 2-hour trace for web surfing, and a 6-hour trace for a combination of web surfing, online chatting, and server accessing, 2 hours for each.

Due to the handshaking procedure of the TCP protocol, as well as the bursty nature of data networks, often tens of TCP packets can be transmitted or received by the WLAN card in one second. Obviously, making tens of decisions in

7. The key to compare two trade-off curves is as follows: Under the same average power consumption the proposed system can achieve significant reduction in average request latency. Under the same average latency, the proposed system can achieve significant reduction in average power consumption. In other words, given any actual system specification (with target at power minimization or latency minimization or a combination), the proposed system can achieve lower power consumption (lower average latency) while the average latency (power) is the same, compared with baseline systems.

8. In this case, when comparing with the N-policy, our proposed RL-based DPM framework can achieve more than 30 percent average latency reduction under the same average power consumption.

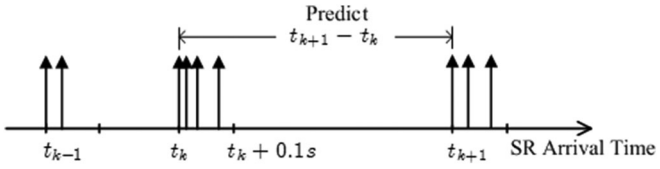


Fig. 8. Illustration of the minimum decision interval.

one second will result in huge overhead for the system. Since packets often "group together" in a very short period of time, we incorporate a "minimal decision interval" of 0.1 s for the RL algorithm and the online Bayesian workload predictor. As shown in Fig. 8, a new service request comes at time t_j, and the workload predictor predicts the time interval $t_{j+1} - t_j$ (t_{j+1} is the arrival time of the first service request generated after time $t_j + 0.1$ s) based on the knowledge of $t_j - t_{j-1}$, $t_{j-1} - t_{j-2}$, etc. Similarly, suppose that at time t_j, the SP is in the idle state, and a new service request arrives. Then the SP turns to the active state for processing that request. After the request processing finishes, the SP cannot turn to the sleep state immediately. Instead, it should wait in the idle state until time $t_j + 0.1$ s and finishes processing all the service requests generated in the time period $\left[t_j, t_j + 0.1\text{s}\right)$, before it could take any action, either going sleep right away or wait in idle state for a specific timeout.

For each specific trace (web trace, video trace, and combined trace), we initialize the conditional probabilities of the Bayes predictor to be 1/3 (assuming three possible outcomes), and test on the trace. The accuracy is defined as the ratio of correctly predicted inter-arrival times to the total inter-arrival times (i.e., number of requests - 1) over the whole trace. The correct prediction rates of the online Bayesian predictor are 99.2 percent for the video trace, 79.8 percent for the web trace, and 82.8 percent for the combined trace. In comparison, the correct prediction rate of an exponential predictor [4] for the combined trace is less than 65 percent.

For the WLAN case, we also compare with the RL-based DPM framework with oracle predictor with 100 percent prediction accuracy (the detailed RL implementation is the same as our proposed framework), in order to illustrate the effect of workload prediction. Fig. 9 provides the power-latency tradeoff curves for the simple RL-based power-managed system with Bayesian workload prediction, the oracle

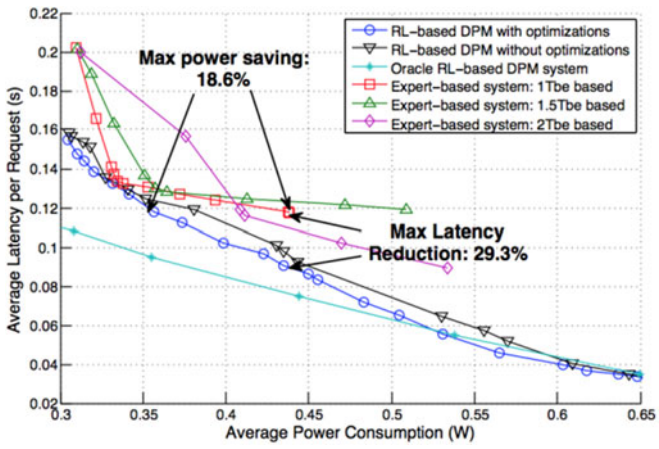


Fig. 9. Power-latency tradeoff curves for the WLAN card: The simple system.

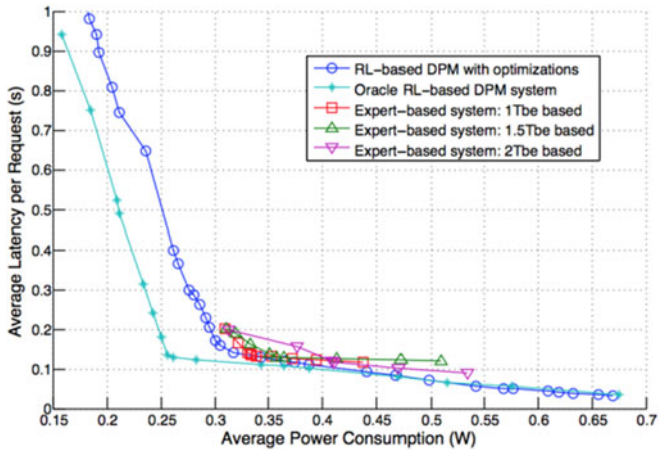


Fig. 10. Power-latency tradeoff curves for the WLAN card: The whole system.

framework, as well as three different expert-based DPM frameworks. We derive these experimental results using the combined trace. The timeout values of the fixed timeout expert in those three systems are set to be $0.5 \cdot T_{be}$, $1 \cdot T_{be}$ and $1.5 \cdot T_{be}$, respectively. We illustrate experimental results on both DPM frameworks with and without the three further optimization methods (multiple-update initialization, dynamic action set, and locally randomized action selection). For the DPM framework without further optimization, we set the action (timeout) values to be $\{0 \cdot T_{be}, 0.3T_{be}, 1T_{be}, 3T_{be}, 5T_{be}\}$. On the other hand, the tradeoff curves for the whole DPM system are given in Fig. 10. We only show experimental results on the DPM framework with the three further optimization methods because the performance difference is much smaller compared with the range of X-axis and Y-axis in this figure. Comparing these two figures, we can see that with the help of making decisions at the sleep state, the RL-based DPM framework can further minimize power to half of the minimum power consumption achievable by the simple RL-based DPM framework by sacrificing performance. When comparing RL-based DPM framework with baseline systems, we can see again that the RL-based DPM method can achieve a "wider and deeper" power-latency tradeoff curve than the latter. When comparing to the expert-based approach in which the fixed timeout expert has a timeout value of $1 \cdot T_{be}$, the maximum power saving with the same latency is 18.6 percent; while the maximum per-request latency saving with the same power consumption is 29.3 percent. When comparing the performance of the DPM frameworks with and without further optimizations, we conclude that the former consistently outperforms the latter framework. Moreover, the maximum power saving is 7.6 percent with the same average latency, whereas the maximum reduction in average latency is 14.7 percent when the average power consumption remains the same. These gaps are more significant than the HDD experimental results based on synthesized SR model. Finally, when comparing the performance of the proposed DPM frameworks with the oracle frameworks with 100 percent prediction accuracy, we conclude that the performance of our proposed framework is close to the oracle framework in the "low latency" tradeoff region, i.e., the average power consumption is relatively high while the

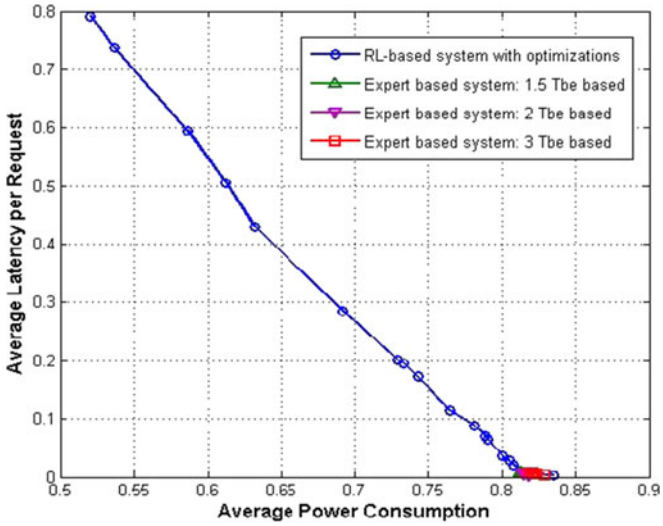


Fig. 11. Power-latency tradeoff curves for the WLAN card on the video trace.

average latency is low. On the other hand, the oracle framework can outperform our proposed framework in the "low power" tradeoff region, indicating that making decisions (mainly sleep decisions) judiciously in this region heavily rely on the performance of workload predictions.

As another illustrative example, we demonstrate in Fig. 11 the power-latency tradeoff curves of the whole RL-based DPM framework on the video trace. Because of the frequency request sending/receiving during online video watching, always keeping the SP in the idle state (without letting it go into the sleep state) will be the optimal policy in terms of latency minimization. Although the RL-based DPM framework cannot outperform this " always on" policy in terms of latency saving, it could reduce the average power consumption to 50 percent of the original power consumption by sacrificing performance. Again, this is achieved by the capability of making decisions at the sleep state.

For video watching, the video requests typically arrive earlier than the time when we watch the corresponding content. The few-second-gap is needed for video decoding and re-framing. In order to save power consumption (this is especially true for portable devices), it is possible for the WLAN card to turn to sleep and turn active again when it has accumulated a batch of video requests. In this content, an average latency of 0.8s may be acceptable with the benefit of reducing power consumption by nearly half, since it generally means that we watch all video contents 0.8s later, which is not so significant compared with the network and decoding delay. On the other hand, video presentation and conference is delay sensitive and PM should set the goal to only minimize delay. In this case the optimal policy, i.e., always ON, can still be derived.

7 Conclusion

In this paper, we propose a novel adaptive DPM technique using the model-free reinforcement learning technique. We adopt the TD(λ) learning technique for SMDP as the basic RL algorithm in the DPM framework. The proposed DPM method is model-free and requires no prior information of the state transition probability function or the reward function. Moreover, we propose a workload predictor based on an online Bayesian classifier to effectively provide estimations of the future service request inter-arrival times to the DPM agent (i.e., the PM.) Several improvements, including multiple-update initialization, dynamic action set, and locally randomized action selection, are also incorporated. The proposed DPM framework is capable of exploring a tradeoff in the power-latency design space of the power-managed system by adjusting a user-defined parameter. Experimental results on both synthesized and real workload traces show that the proposed DPM framework finds a much "deeper and wider" power and latency tradeoff curve compared with reference expert-based DPM methods.

Acknowledgments

This work is supported in part by the Software and Hardware Foundations program of the NSF's Directorate for Computer & Information Science & Engineering.

References

[1] Y. Wang, Q. Xie, A. C. Ammari, and M. Pedram, "Deriving a near-optimal power management policy using model-free reinforcement learning and Bayesian classification," in *Proc. 48th ACM/EDAC/IEEE Des. Autom. Conf.*, 2011, pp. 41–46.

[2] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system level dynamic power management," *IEEE Trans. VLSI Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[3] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI*, vol. 4, no. 1, pp. 42–55, Mar.1996.

[4] C. H. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 1997, pp. 28–32.

[5] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Comput.-Aided Des.*, vol. 18, pp. 813–833, Jun. 1999.

[6] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," in *Proc. 36th Des. Autom. Conf.*, 1999, pp. 555–561.

[7] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event-driven power management," *IEEE Trans. Comput.-Aided Des.*, vol. 20, no. 7, pp. 840–857, Jul. 2001.

[8] H. Jung and M. Pedram, "Dynamic power management under uncertain information," in *Proc. Des., Autom. Test Europe*, 2007, pp. 1–6.

[9] H. Jung and M. Pedram, "Resilient dynamic power management under uncertainty," in *Proc. Des., Autom. Test Europe*, 2008, pp. 224–229.

[10] Q. Qiu, Y. Tan, and Q. Wu, "Stochastic modeling and optimization for robust power management in a partially observable system," in *Proc. Des., Autom. Test Europe*, 2007, pp. 779–784.

[11] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 747–754.

[12] G. Dhiman and T. S. Rosing, "Dynamic voltage scaling using machine learning," in *Proc. Int. Symp. Low Power Electron. Des.*, 2007, pp. 207–212.

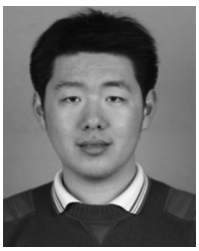
[13] A. Weissel and F. Bellosa, "Self-learning hard disk power management for mobile devices," in *Proc. 2nd Int. Workshop Softw. Support Portable Storage*, 2006, pp. 33–40.

[14] Y. Tan, W. Liu, and Q. Qiu, "Adaptive power management using reinforcement learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2009, pp. 461–467.

[15] W. Liu, Y. Tan, and Q. Qiu, "Enhanced Q-learning algorithm for dynamic power management with performance constraint," in *Proc. Des., Autom. Test Europe*, 2010, pp. 602–605.

[16] S. Yue, D. Zhu, Y. Wang, and M. Pedram, "Reinforcement learning-based dynamic power management in mobile computing systems equipped with hybrid power supply," in *Proc. Int. Conf. Comput. Des.*, 2012, pp. 81–86.

[17] C.-T. Liu and R. C. Hsu, "Adaptive power management based on reinforcement learning for embedded system" *Proc.* 21st Int. Conf. *Springer New Frontiers Appl. Artif. Intell.*, 2008, pp. 513–522.
[18] G. Theocharous, S. Mannor, N. Shah, P. Gandhi, B. Kveton, S. Siddiqi, and C.-H. Yu, "Machine learning for adaptive power management," *Intel Technol. J. Autonomic Comput.*, vol. 10, no. 4, pp. 299–311, 2006.
[19] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *Proc. Int. Symp. Low Power Electron. Des.*, 2012, pp. 97–102.
[20] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling," in *Proc. Int. Conf. Hardware/Software Codes. Syst. Synthesis*, 2013, pp. 1–10.
[21] U. A. Khan and B. Rinner, "A reinforcement learning framework for dynamic power management of a portable, multi-camera traffic monitoring system," in *Proc. Int. Conf. IEEE Green Commun. Conf.*, 2012, pp. 557–564.
[22] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, pp. 1–25, 2014.
[23] R. Ye and Q. Xu, "Learning-based power management for multi-core processors via idle period manipulation," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2012, pp. 115–120.
[24] C. Xu, X. Lin, and L. Zhong, "Device drivers should not do power management," in *Proc. ACM SIGOPS Asia-Pacific Workshop Syst.*, Jun. 2014, pp. 1–7.
[25] S. Bradtke and M. Duff, "Reinforcement learning methods for continuous-time Markov decision problems," in *Proc. Advances Neural Inf. Process. Syst.*, 1995, vol. 7, pp. 393–400.
[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[27] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge Univ., Cambridge, England, 1989.
[28] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, Aug. 2006.
[29] L. Kleinrock, *Queueing Systems, Volume I: Theory*. New York, NY, USA: Wiley, 1975.



Yanzhi Wang received the BS degree with distinction in electronic engineering from Tsinghua University, Beijing, China, in 2009, and the PhD degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2014, under the supervision of Prof. Massoud Pedram. He is currently an assistant professor at the Department of Electrical Engineering and Computer Science at Syracuse University, Syracuse, NY, USA. His current research interests include system-level power management, neuromorphic computing, near-threshold computing, digital circuits power minimization and timing analysis, etc. He received best paper awards at 2014 IEEE International Symposium on VLSI (ISVLSI) and 2014 IEEE/ACM International Symposium on Low Power Electronics Design (ISLPED), top paper award at 2015 IEEE Cloud Computing Conference (CLOUD), and multiple best paper nominations. He is a member of the IEEE.

Massoud Pedram received the PhD degree in electrical engineering and computer sciences from the University of California, Berkeley in 1991. He is the Stephen and Etta Varra professor in the Ming Hsieh Department of Electrical Engineering at the University of Southern California, He holds 10 U.S. patents and has published four books, 13 book chapters, and more than 140 archival and 380 conference papers. His research ranges from low power electronics, energy-efficient processing, and cloud computing to photovoltaic cell power generation, energy storage, and power conversion, and from RT-level optimization of VLSI circuits to synthesis and physical design of quantum circuits. For this research, he and his students have received seven conference and two IEEE Transactions Best Paper Awards. He is a recipient of the 1996 Presidential Early Career Award for Scientists and Engineers, a fellow of the IEEE, an ACM Distinguished Scientist, and currently serves as the Editor-in-Chiefs of the *ACM Transactions on Design Automation of Electronic Systems* and the *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. He has also served on the technical program committee of a number of premiere conferences in his field and was the founding Technical Program Co-chair of the 1996 International Symposium on Low Power Electronics and Design and the Technical Program Chair of the 2002 International Symposium on Physical Design.