Adaptive Power Management Using Reinforcement Learning

Ying Tan Wei Liu Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
Binghamton, New York 13902, USA
{ytan3, wliu4, qqiu}@binghamton.edu

ABSTRACT
System level power management must consider the uncertainty and variability that comes from the environment, the application and the hardware. A robust power management technique must be able to learn the optimal decision from past history and improve itself as the environment changes. This paper presents a novel on-line power management technique based on model-free constrained reinforcement learning (RL). It learns the best power management policy that gives the minimum power consumption for a given performance constraint without any prior information of workload. Compared with existing machine learning based power management techniques, the RL based learning is capable of exploring the trade-off in the power-performance design space and converging to a better power management policy. Experimental results show that the proposed RL based power management achieves 24% and 3% reduction in power and latency respectively comparing to the existing expert based power management.

Keywords
Power management, reinforcement learning, Q-learning, model-free

1. INTRODUCTION
Power consumption has become a major issue in the design of computing systems today. High power consumption increases cooling cost, degrades the system reliability and also reduces the battery life in portable devices. Almost all power reduction techniques associate with an adverse effect on system performance at certain degree. Low power design is a constrained optimization that minimizes power consumption while maintaining an acceptable performance.

Dynamic power management (DPM) has proven to be an effective technique for power reduction at system level [1]. It selectively shuts-off or slows-down system components that are idle or underutilized. The power manager needs to make wise decisions on when to put the devices into which power mode. The effective use of power management techniques at run time usually requires application and architecture specific information.

Robust power management must consider the uncertainty and variability that comes from the environment, the application and the hardware. For example, the workload of a complex system is usually unpredictable as it strongly depends on the nature of the application, the input data and the user context. The workload variation changes the device usage pattern and has the most significant impact on the system speed and power consumption. The contention of shared resources such as buses or I/Os in an MPSoC also increases the variability of hardware response time for communication and computation. Furthermore, the process, voltage, and temperature (PVT) variation results in a large fluctuation in hardware performance and power consumption. Therefore, statically optimized resource and power management are not likely to achieve the best performance when the input characteristics change. The ability to observe, learn and adapt to different hardware systems and different working environments is essential for a good power management controller.

Most of the previous power management work separates system modeling and policy optimization. Based on this approach, a model that characterizes both system workload and hardware responses is first constructed and the optimal power management policy is then derived. For example, in [2], the idle intervals of the incoming tasks are modeled and predicated using the exponential-average approach. The authors of [3] use a regression function to predict the next task incoming time.

For more complex systems, stochastic models are applied for system modeling, since the workload in such systems is more random and uncertain. Markov decision process ([4]) and Semi-Markov decision process ([5]) are adopted in workload modeling. In a more realistic situation, the workload or the incoming tasks are not fully observable to the power manager. In such cases, partially observable Markov decision process (POMDP) is investigated to model the partially observable systems ([6], [7]). The effectiveness of the above mentioned power management approaches relies heavily on an accurate workload model, which is usually not known in advance.

Some recent works adopt machine learning techniques to detect the unknown workload. The authors of [8] propose to use Baum-Welch algorithm to learn the hidden Markov model of the observed workload and find the optimal policy using quadratic constrained linear programming. Reference [9] proposes a user-based adaptive power management technique that considers user annoyance as a performance constraint. However, both approaches require offline training. Therefore, they are not suitable to be applied on-line for adaptive power management in a changing environment.

Online learning is a natural selection for adaptive policy optimization. In reference [10], an on-line learning algorithm dynamically selects the best DPM policies from a set of candidate policies (referred as experts), which are stored locally in the

ICCAD'09, November 2-5, 2009, San Jose, CA, United States

memory. Each expert has a weight factor, the value of which indicates the benefit gained if the correspondent expert was chosen during the last idle period. The one with the highest value has the biggest opportunity to control the device for the next idle period. Reference [11] proposes similar approach using a different learning algorithm. The expert based machine learning algorithm is able to find an appropriate DPM policy in short time without any workload information. However, our experiment shows that the effectiveness of such learning algorithms relies heavily on the pre-selected experts. Not being able to find good performance-power tradeoff is another concern.

In this paper, we present a novel approach for system level power management in a partially observable environment. In contrast to the previous approaches, the proposed power manager learns a new power control policy instead of learning how to choose from a set of existing policies. This is achieved by trying an action in a certain system state, and adjusting the action when this state is re-visited next time, based on the reward/penalty received. This is a model-free approach as the power manager learns the policy directly. The characteristics of the proposed work are described as follows.

1. The power manager does not require any prior knowledge of the workload or the system model. It learns the policy online with real-time incoming tasks and adjusts the policy accordingly. After a certain set-up time, the optimal policy can positively be found.
2. The power manager does not depend on any pre-designed experts, or any DPM policies. It learns the optimal policy by observing the workload and receiving the reward/penalty from the system.
3. We consider the system performance as a constraint while minimizing the overall power consumption. In this way, the total energy is saved without decreasing the quality of service of the system.
4. The trade-off between power consumption and system performance can be controlled by a user-defined parameter. The experimental results show that the proposed power manager finds near optimal power-performance tradeoff.
5. A power management system consists of the workload and the service provider. Although we do not have information of the workload *a priori*, we do know the state space and the transition speed of the SP. Based on this partial information, we improve the convergence speed of the Q-learning algorithm for the DPM problem. Comparing to the traditional Q-learning, the improved Q-learning provides 73% and 14% reduction in power and latency respectively.

The remainder of this paper is organized as follows. Section 2 gives some background of basic reinforcement learning and constrained Q-learning. Section 3 explains our problem formulation and policy optimization using constrained reinforcement learning. The experimental results and analysis are presented in Section 4. We conclude our work in Section 5.

2. BACKGROUND

In this section, we will give a brief introduction to the basic concepts of standard Q-learning, which is one of the most popularly used algorithms of reinforcement learning.

Reinforcement learning is a machine intelligence approach that has been applied in many different areas. It mimics one of the most common learning styles in natural life. The machine learns to achieve a goal by trial-and-error interaction with a dynamic environment.

The general learning model consists of

- An agent
- A finite state space S
- A set of available actions A for the agent
- A reward function $R: S \times A \rightarrow R$

The goal of the agent is to maximize its average long-term reward. It is achieved by learning a policy π, i.e. a mapping between the states and the actions. In our problem, the goal is to minimize the system energy dissipation.

Q-learning is one of the most popular algorithms that perform reinforcement learning. At each step of interaction with the environment, the agent observes the environment and issues an action based on the system state. By performing the action, the system moves from one state to another. The new state gives the agent a reward (a real or natural number) or punishment (a negative reward) which indicates the value of the state transition. The agent keeps a value function $Q^{\pi}(s,a)$ for each state-action pair, which represents the expected long-term reward if the system starts from state *s*, taking action *a*, and thereafter following policy π. Based on this value function, the agent decides which action should be taken in current state to achieve the maximum long-term rewards.

The core of the Q-learning algorithm is a value iteration update of the value function. The Q-value for each state-action pair is initially chosen by the designer and later, it is updated each time an action is issued and a reward is received, based on the following expression.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{old\ value} + \underbrace{\varepsilon_t(s_t,a_t)}_{learning\ rate} \times \left[\overbrace{\underbrace{r_{t+1}}_{Reward} + \underbrace{\gamma}_{Discount\ Factor} \underbrace{\max_a Q(s_{t+1},a)}_{Max\ Future\ Value}}^{Expected\ Discounted\ Reward} - \overbrace{Q(s_t,a_t)}^{Old\ Value} \right] \quad (1)$$

In the above expression, r_t is the reward given at time *t*, and $\alpha_t(s,a) \in (0,1)$ is the learning rates which may be the same value for all pairs. The discount factor γ is a value between 0 and 1. The next time when state *s* is visited again, the action with the maximum Q-value will be chosen, i.e. $\pi(s) = max_{a \in A}\, Q(s,a)$.

As a model-free learning algorithm, it is not necessary for the Q-learning agent to have any prior information about the system, such as the transition probability from one state to another. Thus, it is a highly adaptive and flexible algorithm.

Q-learning is originally designed to find the policy for a Markov Decision Process (MDP). It is proved that the Q-learning is able to find the optimal policy when the learning rate α is reduced to 0 at an appropriate rate, given the condition that the environment is MDP. However, it is important to point out that a computing system for power management is typically non-Markovian. Therefore it is not guaranteed that Q-learning will find the optimal DPM policy for our problem. We will justify why we choose Q-Learning to solve this problem in the next section.

3. PROBLEM FORMULATION AND POLICY LEARNING

In this section, we will discuss how to extend the traditional Q-learning algorithm to solve the dynamic power management problem.

3.1 Problem Formulation

Figure 1 shows the power management system we are interested in. Similar to many previous works in stochastic power management, our environment consists of a service requestor (SR), a service provider (SP), and a service queue (SQ). The service requestor generates different types of requests to be processed by the service provider. These requests are first buffered in a FIFO queue (SQ) before being processed. The state space of the environment is the composite space comprising of SR states, SQ states and SP states.

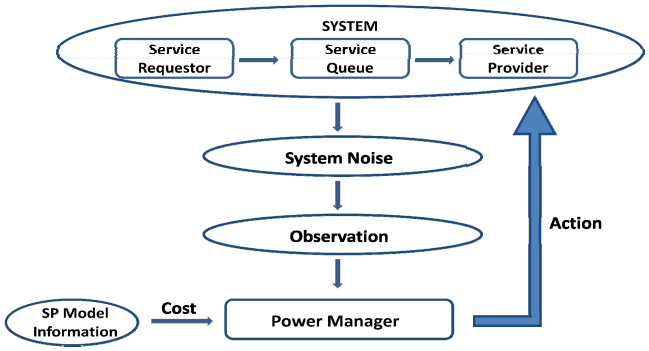


Figure 1 Illustration of system under power management.

The power manager observes the environment through a noisy channel therefore it only receives the partial observation. As reference [7] pointed out, such partial observation can be caused by delayed information or hidden states in the SP, SR or SQ. For example, the SR (i.e. the software application) has multiple request generation modes which are not distinguishable to the power manager (PM). Even if the PM is able to receive the perfect information of system states, it sometimes may choose to aggregate some system states in order to avoid the problem of state explosion. This, again, creates partial observation. We use a finite set $\mathcal{B} = \{o_1, o_2, o_3, \ldots, o_n\}$ to denote the set of observations. If the environment is Markovian, then the observation is a hidden Markov process.

The object of the power management problem is to minimize the average power consumption with respect to a given performance constraint. In this problem, we consider the average delay in the SQ as the performance constraint. Thus, the cost for an action taken in a state is measured by both the immediate power consumption and the caused delay. We assume that a power model of the SP has been characterized for each power mode. Based on the observed status of the SP, its power consumption can be estimated from the model. The delay cost is related to the number request in the service queue. More details about the cost function will be provided in section 3.2.

The action space can be denoted as a finite set $\mathcal{A} = \{a_1, a_2, a_3, \ldots, a_n\}$, with n representing the number of all power modes that the SP can be switched to. The power manager chooses an action for SP every time the system leaves a state and enters another.

The power manager learns the DPM policy based on its observation of the system, which is typically non-Markovian. First of all, the workload of most computing system exhibits long range similarity [14] and hence the service requestor in our power management system is most likely to be non-Markovian. Furthermore, even if the underlying system is Markovian, what the power manager observes may not be Markovian due to the noise and disturbance, such as state aggregation, during the observation. As we mentioned earlier, the Q-learning may not be able to find the optimal policy in a non-Markovian environment. Nevertheless we still choose Q-learning to solve this problem because of its simplicity and also because of its robustness to endure noise.

Reinforcement learning in a non-Markovian environment is an open problem. Many research works have investigated the feasibility of applying the traditional RL algorithms to solve the decision problem in a non-Markovian environment or a partially observable Markovian environment [15][16]. The author of [15] applies five RL algorithms in a noisy and non-Markovian environment and compares their performance and convergence speed. Their results show that the Q-learning exhibits the highest robustness at low noise level and medium robustness at high noise level. However, the convergence speed of Q-learning reduces the most drastically when the noise level increases. In [16] the similar results are reported. Q-learning is capable to achieve the same performance as the other two reference learning algorithms at the cost of slower convergence speed. Based on the study we conclude that the major limitation of Q-learning, when being applied in a non-Markovian environment, is its convergence speed.

Traditional Q-learning assumes no prior information of the environment. However, in a power management system, the model of SP can be pre-characterized. We know exactly how many power modes an SP has and how it switches its power mode given a power management command. In another word, we have partial information of the power management system. Based on this information, we are able to design an improved Q-learning algorithm with faster convergence speed. More details are provided in the next section.

3.2 Modified Q-Learning

In this section, we present our modification on the traditional Q-learning algorithm when applying it to solve the DPM problem.

Modify cost function to consider latency constraint

Our goal is to optimize the power while maintaining a certain level of overall performance. Thus, the original Q-learning is not suitable for our problem; since it only learns to minimize the power consumption without consideration of performance lost that may occur. To extend the Q-learning to address our problem, we define a Lagrangian cost

$$C(s, a; \lambda) = c(s, a) + \lambda d(s, a) \quad (2),$$

where $c(s, a)$ is the power consumption when action a is taken in state s, and $d(s, a)$ is the delay caused by the action. A straight forward way is to set the delay equal to the number of waiting requests in the queue because no matter which action is taken, the requests currently in queue will wait for at least one cycle. Under this definition, the actions *go_active* and *go_sleep* have the same performance penalty, which is not reasonable when the queue is

not empty. Therefore, we set different delay costs for actions *go_active* and *go_sleep* that are described as following:

$d(s, go\_active) = q$ and

$d(s, go\_sleep) = q * (1 + T_{tran})$,

Where q is the number of requests in queue and T_{tran} is the average power mode switching time.

A discrete-time slotted model is used throughout this work, which means all the decision making and system state updating occur on a cycle basis. A time slot n is defined as the time interval $[nT, (n+1)T]$, and the power manager makes decision for this time slot at the beginning of this interval at time nT.

Learning in the observation domain

As we mentioned previously, the power manager works in the observation domain. During each cycle, the agent first obtains an observation of the current system state, which may not be the real system state due to the noisy channel between the agent and the environment. Instead of maintaining a set of Q-values for each state-action pair, the learner keeps a set of Q-values for each observation-action pair, which is updated in a very similar way as the original Q-learning algorithm.

$$Q(o,a;\lambda) = \left(1-\varepsilon_{(o,a)}\right)Q(o,a;\lambda) + \varepsilon_{(o,a)}(c(o,a;\lambda) + \min_{a'} Q(o',a';\lambda)) \quad (3)$$

Next time when the same observation is re-observed, the action with the smallest Q-value is chosen and a new Q-value is again calculated. Note that we cannot update $Q(o,a;\lambda)$ until we obtained the observation *o'*. In another word, each observation cycle, we update the Q value for the observation-action pair of the previous cycle.

Speed up Q-learning for the DPM problem

The convergence of the Q-learning relies on the recurrent visits of all possible state-action pairs. Based on equation (1) we can see, each time a state *s* is visited and action *a* is taken, a corresponding Q value $Q(s, a)$ is updated. It is calculated as the weighted sum of itself and the best Q value of the next state *s'*, i.e. $Q(s,a) = (1-\varepsilon)Q(s,a) + \varepsilon(c(s,a) + \min_{a'} Q(s',a'))$. The frequency that state *s'* occurs after state-action pair (*s*, *a*) reveals the information of the system transition probability. In traditional Q-learning, only the Q value corresponding to the actual visited state-action pair will be updated. This is because the learner has no information of the system dynamics, and it totally relies on the actual execution trace to figure out the next state information for a given state-action pair.

The state of a power management system is a composition of the states of SP, SR and SQ. Among these three, only SR is unknown. The state space SP is the set of available power modes and its transition probability from state sp_1 to state sp_2 when action *a* is taken (i.e. prob(sp_1, sp_2, *a*)) can be characterized. We also know that SP and SR are independent to each other.

Based on the available information on SP and SQ, we proposed to update more than one Q values to speed up convergence. More specifically, for each visited state-action pair ((*sp*, *sr*, *sq*), *a*) we will update all Q values corresponding to the state-action pairs ((*sp'*, *sr*, *sq*), *a'*), $\forall sp' \in SP, \forall a' \in A$. This requires us to know the next state of the state-action pair ((*sp'*, *sr*, *sq*), *a'*) even if it is not visited.

Let (sp_{t+1}, sr_{t+1}, sq_{t+1}) represent the next state of the visited state-action pair ((*sp*, *sr*, *sq*), *a*) according to the system execution trace. Let (sp_{t+1}', sr_{t+1}', sq_{t+1}') represent the next state that needs to be projected for the virtually visited state-action pair ((*sp'*, *sr*, *sq*), *a'*). Given the current state *sp'* and action *a'*, it is not difficult for us to find the next state sp_{t+1}' as the SP model is pre-characterized. We also know that the service requestor works independently to the service provider. Regardless of the state of SP, the requests are generated in the same way. Therefore, sr_{t+1}' is equal to sr_{t+1}. To find out the value of sq_{t+1}' is more difficult. On one hand, it depends on the number of incoming requests, which is not affected by the SP power mode. On the other hand, it depends on the SP service rate, which is unknown and will be affected by the SP power mode. However, we know that the difference between sq_{t+1}' and sq_{t+1} is less than 1 because each cycle the SP completes at most one request. Therefore, here we simply set sq_{t+1}' to be equal to sq_{t+1}. With the next state information, the Q values of those virtually visited state-action pairs can easily be calculated.

Since the number of Q-values that would be updated in each cycle is $|SP| \times |A|$, with $|SP|$ the number of SP states and $|A|$ the cardinality of the action set, the complexity of the constrained Q-learning is O($|SP| \times |A|$). This is an affordable computing intension for an online power management algorithm.

We further improve the convergence speed of the proposed Q-learning algorithm by adopting a variable learning rate. Compared to the traditional Q-learning, the learning rate $\varepsilon_{(o,a)}$ is not fixed in our algorithm. Instead, it is dependent on the observation-action pair (o, a) and is calculated as

$$\varepsilon_{(o,a)} = \frac{\mu}{Visit(o,a)} \quad (4),$$

where $Visit(o,a)$ is the number of times that the observation-action pair (o, a) has been visited, and μ is a given constant.

Our experimental results show that, the modified Q-learning algorithm (i.e., updating multiple Q-values in each cycle at a variable learning rate) converges faster than the traditional Q-learning (i.e., updating only one Q-value each time at a fixed learning rate). Moreover, the learner finds better policy because the learning process is based on more information.

4. EXPERIMENTAL RESULTS AND ANALYSIS

We evaluate the proposed Q-learning algorithm using both synthetic workloads and real workloads. As a baseline reference, we also implemented the learning algorithm that is proposed in [10] to the best of our understanding. In the rest of the paper, we will refer to our algorithm as modified Q-learning and refer to the algorithm in [10] as expert-based DPM. Three other algorithms are also implemented as comparisons: fixed timeout policy, adaptive timeout policy, and exponential predictive policy. The fixed timeout policy employs a timeout equal to three times the break even time T_{be}. The adaptive timeout policy uses $3T_{be}$ as its initial timeout and +/-1 cycle as adjustments, depending on whether the previous idle period achieved energy saving or not.

The exponential predictive policy was implemented as described in [2]. Table 1 gives the characteristics of these policies.

Table 1 Characteristics of different policies.

Policy	Characteristics
Fixed Timeout	Timeout = 3 * T_{be}
Adaptive Timeout	Initial timeout = 3 * T_{be} Adjustment = +/-1 cycle
Exponential Predictive	$I_{n+1} = \alpha i_n + (1-\alpha) I_n$ $\alpha = 0.5$
Expert-based Learning	Uses the above three policies as experts.

4.1 Experiment Using Synthetic Workload

The SP used in the simulation is a hard disk drive (HDD) It has two power modes, *active* and *sleep*. Its power consumption and switching time are reported in Table 2. The unit for T_{tran} and T_{be} is cycle. The service rate of SP is 0.7/cycle. The SQ can hold up to 4 waiting requests; therefore there are five different states of SQ including the state which stands for an empty queue.

Table 2 Characteristics of the Service Provider.

Device	P_{active}	P_{sleep}	P_{tran}	T_{tran}	T_{be}
HDD	1.6	0.4	2.4	2.5	4.2

Our first set of experiments is performed using synthesized workload. The requests are generated by an SR model that has three states, s_1, s_2, and s_3 whose incoming request rates are 0.01, 0.10, and 0.25 respectively. The following matrix gives the transition probability of the three states.

$$P = \begin{bmatrix} 0.98 & 0.01 & 0.01 \\ 0.01 & 0.98 & 0.01 \\ 0.01 & 0.01 & 0.98 \end{bmatrix}$$

Based on our model, the SR has three request generation modes which generate requests at three different speeds. One of the modes has relatively higher request generation speed than the other two. The SR will stay in a request generation mode for about 50 cycles on average and then switch to another mode. Note that the power manager does not know the SR model. Furthermore, since our system is partially observable, we assume that the power manager is not able to distinguish the two modes, s_1 and s_2, which generate requests at a relatively low speed. Hence, the power manager will receive two observations about the SR, one is high workload mode and the other is low workload mode.

We developed a cycle-based simulator that reports the average power consumption, the latency (i.e. average number of waiting requests in queue), and the request loss rate due to overflow in the service queue. We run the simulation for 50000 cycles. Because the stochastic nature of the simulated system, all the results reported in this paper are the averages over 20 different simulations with different random seeds.

The parameter λ in the Lagrangian cost function $C(c, a; \lambda)$ controls the power-latency tradeoff of the modified Q-learning. We vary the λ from 0.01 to 50 and generate a set of different power-performance tradeoff points in the design space. The learning rate $\varepsilon_{(o,a)}$ is calculated as equation (4), with $\mu = 0.25$.

For the expert-based DPM [10], three different experts are selected. They are: fixed timeout policy, adaptive timeout policy, and exponential predictive timeout policy. An agent learns how to choose the most proper expert based on the workload. We evaluate three different versions of expert-based DPM with the timeout of the fixed timeout expert set to be $1T_{be}$, $3T_{be}$, and $5T_{be}$, respectively. The characteristics of the other two experts are the same. The expert-based DPM also has a parameter $\alpha \in (0,1)$ that can be used to control the tradeoff between the power and performance. In the experiment, we vary α from 0.1 to 0.9.

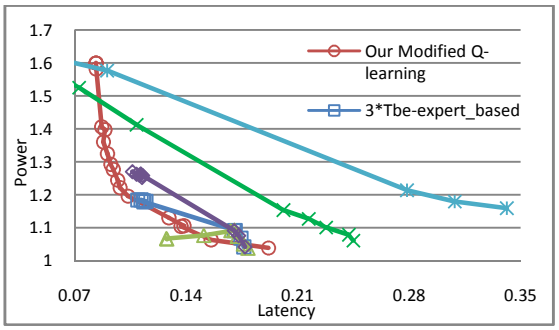


Figure 2 Power /Latency tradeoff curves for synthesized workload.

Figure 2 gives the power-latency tradeoff curves for the modified Q-learning, the traditional Q-learning, Q-learning with multiple Q-values updated in each step, and the 3 different expert-based DPM. It shows that our modified Q-learning algorithm finds better policy that achieves lower power consumption while maintaining the same average latency as the expert-based DPM most of time. Moreover, our modified Q-learning algorithm can achieve even smaller latency which the expert-based DPM is not capable of, at the cost of higher power consumption. This is because our modified Q-learning algorithm sometimes pre-wakes up the SP when there are requests coming in, whereas the expert-based DPM can only wake up when the learner actually observes the incoming requests. It is useful in the cases where the power consumption is not the major concern, but the prompt reaction is most important.

The power-latency tradeoff points found by the modified Q-learning are evenly distributed over the design space while those points found by the expert-based DPM are trapped into only two locations. Hence, our learning algorithm provides more flexibility in terms of power-delay tradeoff. The performance of the three expert-based DPM differs significantly in the high performance side. It indicates that the expert selection has heavy impact on the efficiency of the expert-based DPM.

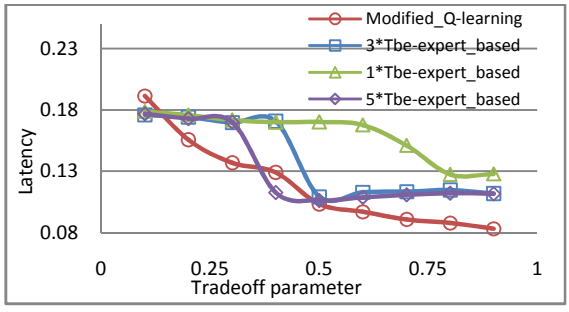


Figure 3 Latency vs. tradeoff parameter for synthesized workload.

Figure 3 shows the relation between the system latency and the control parameter α for the expert based DPM. It also gives the

relation between the system latency and the control parameter λ for the modified Q-learning. As we can see, the system latency is a monotonically decreasing function of λ. However, the similar trend cannot be found between the performance and the α value for the expert-based DPM.

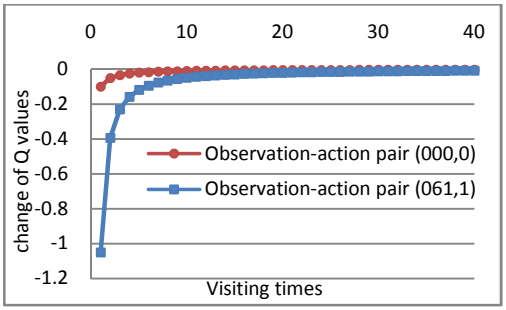


Figure 4 Convergence of Q-values for two observation-action pairs.

Every learning algorithm requires a period of time to learn before it converges. We monitored the changes of Q-values each time the Q-values are updated. Figure 4 shows the Q-values convergence speed of two most visited observation-action pairs (000, 0) and (061, 1). Observation (000) represents that there are no incoming requests, no waiting requests in queue, and HDD is in *sleep* mode. Observation (061) represents that there are no incoming requests, the queue is full, and HDD is in *active* mode. Action 0 and 1 represent *go_sleep* and *go_active*, respectively. From this figure, we can see that both Q-values converge within 30 updates.

Our modified Q-learning algorithm can quickly adapt to workload changes. We tested our modified Q-learning algorithm on concatenated workload, which consists of a high workload trace (average incoming rate is 0.3) and a low workload trace (average incoming rate is 0.02). With the high workload, the algorithm reaches its stable state with average latency equal to 0.5 within 3000 cycles. After changing to low workload, modified Q-learning again reaches the stable state within 1500 cycles.

4.2 Experiment Using Real Workload

Our second set of experiments is performed using real workloads that are collected from different desktop workstations. Using Windows Performance Monitor, we collected hard disk read/write request sequences from two different desktop workstations whose hard disk usage level differs significantly. We stopped collection when the generated file size reaches 5MB, which is equivalent to 70,000 read/write requests in the sequence. Different workstations take different time to generate those 70,000 requests.

The first trace was collected in the afternoon when a set of applications were running simultaneously with high disk I/O activities, resulting in a short collection time (i.e., 18 minutes). We label this trace as "trace_high". The other trace was collected at night when only two applications were running workstation and it took more than 200 minutes to complete the collection. We label this trace as "trace_low". The two trace represent high workload and low workload respectively.

To reduce the complexity of learning, our power manager performs state aggregation to limit the state space of SR and SQ. It has only 7 observations of SR, that is, the total number of incoming requests (read/write together) is 0 , 1, 2, 3, 4, 5, and the incoming requests are more than 5. Similarly, there are only 8 observations of the SQ, the queue is empty, the queue has 1 (or 2, 3, 4, 5, 6) waiting requests waiting and the queue has more than 6 waiting requests. We assume that the service rate of a read request and a write request is 0.5 and 0.3 respectively.

Figure 5 gives the power-latency tradeoff curves of seven algorithms for two traces. The first four algorithms are the same as described for Figure 2. We also compare the performances of fixed timeout policy, adaptive timeout policy, and exponential predictive policy. As shown in the figures, the proposed modified Q-learning finds better policy that achieves lower power consumption as well as smaller average latency. The same as synthetic workload, the modified Q-learning is able to achieve even smaller average latency at the cost of power consumption.

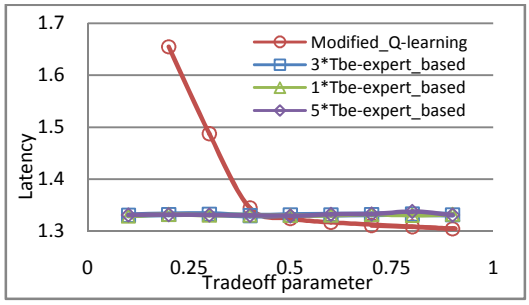
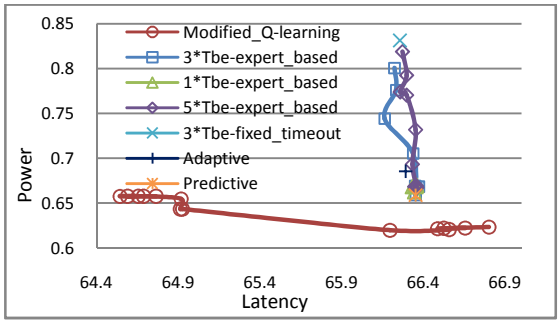
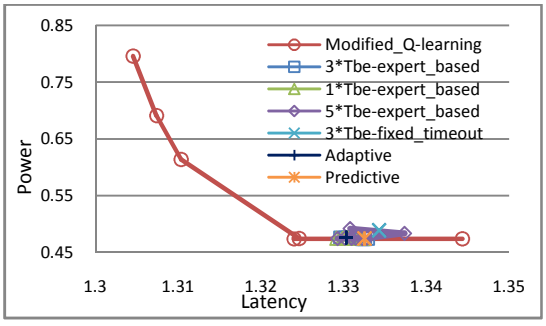


Figure 6 Latency vs. tradeoff parameter for trace_low.

The tradeoff parameter λ in the proposed modified Q-learning controls the power-latency tradeoff efficiently. Figure 6 shows the relation between the system latency and the control parameter α for the expert based DPM as well as the relation between the system latency and the control parameter λ for the modified Q-learning. As shown in the figure, the system latency is a monotonically decreasing function of λ, while the parameter α has almost no impact on the latency control.



(a) trace_high



(b) trace_low

Figure 5 Performance comparison of different algorithms.

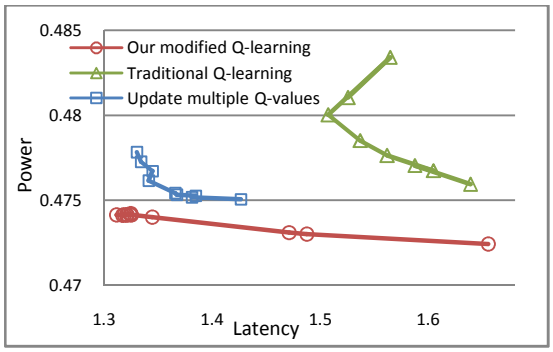


Figure 7 Power/Latency curves of three Q-learning algorithms.

As we mentioned in section 3.2, we modified the traditional Q-learning in two ways. First, the learning rate ε is modified as an adaptive factor associated with the observation-action pair. Second, we update multiple Q-values instead of only one Q-value in each learning step. Figure 7 shows the power-latency tradeoff curves of three Q-learning algorithms: the traditional Q-learning, Q-learning with multiple Q-values updated in each step, and our proposed modified Q-learning. All the results in this figure were obtained from the same trace (trace_low). As shown in the figure, the proposed modified Q-learning over-performs the other two Q-learning algorithms in both the energy saving and latency control. Also, the tradeoff parameter λ in modified Q-learning controls the power-latency tradeoff more effectively. Figure 8 shows the Q-value of observation-action pair (000, 0) in each cycle. As we can see, comparing to the other two learning algorithms, the changes of Q-value for the proposed modified Q-learning is smoother. Moreover, it converges much faster to the stable state.

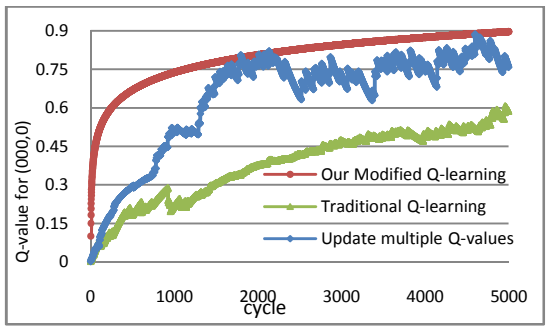


Figure 8 Q-values of observation-action pair (000, 0).

5. CONCLUSIONS

In this paper, we present a novel on-line power management technique using reinforcement learning. It learns the best power management policy that gives the minimum power consumption without knowing the workload information *a priori*. Our experimental results show that, compared to the existing expert based DPM, the proposed RL based power management adapts to changing workload and finds better power-performance tradeoff points.

6. 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*, Vol. 8, Issue 3, pp.299-316, 2000.

[2] C.H. Hwang, and A.C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proceedings of International Conference on Computer-Aided Design*, Nov. 1997.

[3] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 4, pp. 42-55, Mar. 1996.

[4] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE transactions on Computer-Aided Design*, Vol. 18, pp. 813-833, Jun. 2001.

[5] T. Simunic, L. Benini, and G. De Micheli, "Event-driven power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, pp. 840-857, Jul. 2001.

[6] H. Jung, and M. Pedram, "Dynamic power management under uncertain information," in *Proceedings of Design Automation & Test in Europe (DATE'07)*, Apr. 2007.

[7] Q. Qiu, Y. Tan and Q. Wu, "Stochastic Modeling and Optimization for Robust Power Management in a Partially Observable System," in *Proceedings of Design Automation & Test in Europe (DATE'07)*, Apr. 2007.

[8] Y. Tan, and Q. Qiu, "A framework of stochastic power management using hidden Markov model," in *Proceedings of Design Automation & Test in Europe (DATE'08)*, Apr. 2008.

[9] G. Theocharous, S. Mannor, N. Shah, P. Gandhi, B. Kveton, S. Siddiqi, and C-H. Yu, "Machine Learning for Adaptive Power Management," *Intel Technology Journal*, Vol. 10, pp. 299-312, November 2006.

[10] G. Dhiman, and T. Simunic Rosing, "Dynamic power management using machine learning," in *Proceedings of International Conference on Computer-Aided Design (ICCAD'06)*, Nov. 2006.

[11] V.L. Prabha and E. C. Monie, "Hardware Architecture of Reinforcement Learning Scheme for Dynamic Power Management in Embedded Systems," EURASIP Journal on Embedded Systems, Vol. 2007.

[12] Dynamic Power Management in Embedded Systems

[13] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design*, Vol. 18, pp. 813-833, Jun. 1999.

[14] G.V. Varatkar and R. Marculescu, "On-chip Traffic Modeling and Synthesis for MPEG-2 Video Applications," *IEEE Transactions on Very Large Scale Intergration Systems*, Vol. 12, No. 1, January 2004.

[15] M. Pendrith, "On reinforcement learning of control actions in noisy and nonMarkovian domains" *Technical Report NSW-CSE-TR-9410,* School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia, 1994.

[16] K. Sikorski and T. Balch, "Model-Based and Model-Free Learning in Markovian and Non-Markovian Environments," *in Proceedings of Agents-2001 Workshop on Learning Agents*, May 29, 2001.