Reinforcement Learning Algorithms for Dynamic Power Management

Maryam Triki[1], Ahmed C. Ammari[1,2]
[1]MMA Laboratory, INSAT
Carthage University, Tunis, Tunisia
[2]Department of Electrical & Computer Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah, SA
maryam.triki@gmail.com

Yanzhi Wang and Massoud Pedram
Department of Electrical Engineering
University of Southern California
Los Angeles, CA, USA

***Abstract*—In this paper we present a dynamic power management (DPM) framework based on model-free reinforcement learning (RL) techniques. For the RL algorithms, we employ both temporal difference learning and Q-learning for semi-Markov decision process in a continuous-time manner. The proposed DPM is model-free and do not require any prior information of the workload characteristics. The power manager learns the optimal power management policy that significantly reduces energy consumption while maintaining an acceptable performance level. Moreover, power-latency tradeoffs can be precisely controlled based on a user-defined parameter. In addition, the temporal difference (TD) learning is compared with the Q-learning approach in terms of both performance and convergence speed. Experiments on network cards show that TD achieves better power saving without sacrificing any latency and has faster convergence speed compared to Q-learning.**

I. INTRODUCTION

A Dynamic power management refers to the selective shut-off or slow-down of system components that are idle or underutilized. Such technique has proven to be a particularly effective way of reducing power dissipation at the system level [1]. An effective DPM policy should minimize power consumption while maintaining performance degradation to an acceptable level. There are several DPM methods in the literature classified into three categories: heuristic, stochastic, and learning based methods. The heuristic methods attempt to predict the length of the next idle time and shut-off the device if the predicted length of idle time justifies the cost [9]. They perform well only when the requests are highly correlated and do not take performance constraints into account. The stochastic approaches can take into account both power and performance. They model the request arrival times and device service times as stationary stochastic processes such as Markov Decision Processes (MDP) [2], [3], [4]. However, one of the main criticisms of these methods is the missing of exact knowledge about the workload details- which are usually varying with time- that result in a loss of accuracy.

A few research works have profited from the use of intelligence learning methods. In contrast to the aforesaid techniques, the machine learning-based methods can simultaneously consider power and performance, and perform well under various workload conditions. Dhiman et al. in [5] define an online policy algorithm which generates offline a set of experts (DPM policies) to select from. Tan et al. in [6] enhance the Q-learning algorithm and propose a DPM framework that learns the best-suited action for each system state based on the reward or penalty received. However, this work is based on a discrete-time model of the stochastic process, and thus has large decision making overhead. To overcome this problem, we extend this work to allow the power manager (PM) working in a continuous-time and event-driven manner with faster convergence rate, by exploiting reinforcement learning frameworks for semi-MDP (SMDP) [7]. In particular, we shall concentrate on TD(λ) and Q-learning approaches implemented using a reduced number of state action pairs in order to accelerate convergence. The tradeoff between energy consumption and performance can be controlled by a user-defined parameter. This paper will show that the proposed PM is more powerful in terms of power saving and has a faster convergence rate.

The rest of the paper is organized as follows. Section II explains basic background of reinforcement learning for SMDP. Section III explains our system model. Details of the proposed DPM framework are given in Section IV. The experimental results are presented in Section V, and we conclude our findings in Section VI.

II. REINFORCEMENT LEARNING BACKGROUND

A. Semi Markov Decision Processes

A Semi Markov Decision Process (SMDP) is a continuous time dynamic system composed of a countable state set X, and a finite action set A. Assume that the system is originally in state $x \in X$, and action $a \in A$ is applied. An SMDP [8] then evolves as follows:

- The system finds itself in the next state $x' \in X$ according to the transition probability function $P_{xx'}(a)$.

978-1-4799-2806-4/14/$31.00 ©2014 IEEE

- A reward rate function $r(x,a)$ is defined to specify the reward rate a system receives until the next transition occurs.
- When the system transit from x to x', the time until transition occurs has probability density function $f_{xx'}(.|a)$.

B. Reinforcement Learning Model

Reinforcement Learning is learning through experience accumulation [9]. It is based on the interaction between the agent and its environment. The general RL model, as illustrated in Fig.1, consists of an agent, a finite state space S, a set of available actions A, and a reward function $R: S \times A \rightarrow R$. The agent is the learner and decision maker. The environment is defined as anything the agent receives sensory information about. Actions refer to the decision that the agent will be called to make [10]. State represents the situations the agent can find itself in. More precisely, State is the available information about the agent's environment that helps in decision making.

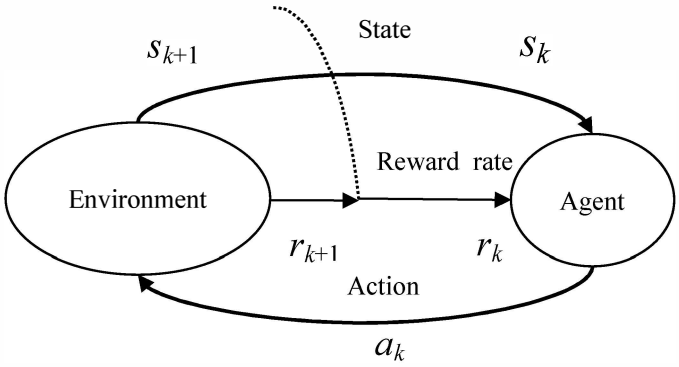


Fig.1. Agent-environment interaction model.

At each step of interaction with the environment, the agent receives a representation of the environment's state $s_t \in S$. It selects an action $a_t \in A(s_t)$ where $A(s_t)$ denotes the set of possible actions available at state s_t. As a consequence of the action taken, the agent moves to a new state s_{k+1} and receives from the environment a reward r_{t+1}which indicates the value of the state transition. The cumulative rewards affect the agent behavior and guide the action policy. The agent's goal is to optimize its behavior based on the received rewards. The RL agent continuously adjusts its policy so as to maximize the total amount of reward received over the long run.

A policy $\pi = \{(s,a)|s \in S, a \in A\}$ is a set of state-action pairs for all states in the RL framework. A deterministic policy specifies for each state the action that should be taken. In a stochastic policy the agent, given probabilities of choosing several actions; has to specify the optimal action. Thus, reinforcement learning algorithms are mainly based on estimating value functions.

We define the *value* of a state by the expected return when starting in that state and following π thereafter. It indicates how good it is to be in a given state. The value of a state s under a policy π, denoted $V^{(\pi)}(s)$ can be defined formally as :

$V^{(\pi)}(s) = E_\pi\{R_t|s_t = s\} = E_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\,|s_t = s\}$ (1)

where r_t is the reward given at time t and the discount factor γ is a value between 0 and 1. Consider an agent starting in state s and following π thereafter. The state value function $V^{(\pi)}(s)$ can be estimated from the agent's experience. The average of actual returns will converge to the state's value. Similarly, we define the action-value function for policy π as the value of taking action a in state s under policy π, denoted by $Q^{(\pi)}(s,a)$, such that

$$Q^{(\pi)}(s,a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\,\middle|\,s_t = s, a_t = a\right\} \quad (2)$$

If separate averages of reward are kept for each action taken in a state, then these averages will similarly converge to the state-action values.

Bellman Equation for V^π

For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} V^{(\pi)}(s) = E_\pi\{R_t|s_t = s\} &= E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\,\middle|\,s_t = s\right\} \quad (3)\\ &= E_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\,\middle|\,s_t = s\right\}\\ &= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left[R^a_{ss'}\right.\\ &\quad \left. + \gamma E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}\,\middle|\,s_{t+1} = s'\right\}\right]\\ &= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^{(\pi)}(s')\right] \end{aligned}$$

The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way. Notice that the value function $V^{(\pi)}(s)$ is the unique solution to its Bellman equation.

Bellman optimality equation

Let π^* denote the optimal policy. Then, it has a state-value function verifying:

$V^*(s) = \max_\pi V^\pi(s) \;\; \forall\, s \in S$ (4)

And has an optimal action-value function defined as:

$Q^*(s,a) = \max_\pi Q^\pi(s,a) \;\; \forall\, s \in S, a \in A$ (5)

The Bellman optimality equation for V^* is:

$$\begin{aligned} V^*(s) &= \max_{a\in A(s)} Q^{\pi^*}(s,a)\\ &= \max_{a\in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')] \quad (6) \end{aligned}$$

The Bellman optimality equation for Q^* is:

$Q^*(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')]$ (7)

The Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state. Computing the optimal policy by solving the Bellman Optimality equation leads to solve the RL problem. Unfortunately, we must have enough computational resources to solve such equation. In addition, we have to assume that the RL environment (i) is Markovian and (ii) we have the exact knowledge of its dynamics which is not realistic.

C. Reinforcement Learning Techniques

Many research works have proposed different techniques and algorithms for solving the RL problem by approximating the chosen value function. Among these, we mention the Watkin's Q-learning algorithm [13] and the Temporal Difference learning. In this section, we will give a brief overview of these two techniques.

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

1) Q- Learning for SMDP's

We define the *return* R as the discounted integral of reward rate, whenever a selection of action is made by the agent. Furthermore, we define the *value* of a state-action pair (s, a) under a policy π, denoted by $Q^{\pi}(s, a)$, as the *expected return* when starting from state s, choosing action a (according to the policy π), and following π thereafter. The reinforcement learning problem is to determine the optimal policy that maximizes the value functions for all state-action pairs. To be realistic, the power manager has no predefined policy or knowledge about state transition characteristics unlike the stochastic DPM approaches. Therefore the agent has to simultaneously learn the optimal policy and use that policy to control.

Suppose that state sk is visited at epoch tk, then at that epoch the agent chooses an action either with the maximum estimated value $Qk(sk, a)$ for various actions $a \in A$, or by using other semi-greedy policies [11] . The Q-learning is based on an iteratively stochastic approximation to define an estimate for the optimal policy. The Q-learning rule for SMDP's:

$$\forall (s,a) \in S \times A:\ Q^{(k+1)}(s,a) = Q^{(k)}(s,a) + \alpha \left(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k,a_k) + \text{MAX}_{a'}\, e^{-\beta\tau_k} Q^{(k)}(s_{k+1},a') - Q^{(k)}(s_k,a_k) \right) \quad (8)$$

where α_k is the learning rate.

The reinforcement learning problem is to determine the optimal policy that maximizes the value functions for all state-action pairs. To be realistic, the power manager has no predefined policy or knowledge about state transition characteristics unlike the stochastic DPM approaches. Therefore the agent has to simultaneously learn the optimal policy and use that policy to control.

2) Temporal Difference Learning for SMDP's

The temporal difference learning TD(0), first described by Sutton [15], attempts to find solutions to (3). The TD learning rule updates the estimate $V_k(s_k)$ at the next epoch t_{k+1}, based on a sequence of sampled state transitions in the direction of the sample value $r(s_k, a_k) + \gamma V_{(k)}(s_{k+1})$.

The TD(0) rule for SMDP's:

$$V^{(k+1)}(s) = V^{(k)}(s) + \alpha \left(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k,a_k) + \text{MAX}_{a'}\, e^{-\beta\tau_k} V^{(k)}(s_{k+1}) - V^{(k)}(s_k) \right) \quad (9)$$

Various TD learning algorithm implementations are mainly different from one another by their updating methods. We choose to use the TD(λ) algorithm for SMDP due to a joint consideration of effectiveness, robustness and convergence rate. More specifically, the value update rule for a state-action pair at epoch t_{k+1} in the TD(λ) algorithm for SMDP is given as follows:

$$\forall (s,a) \in S \times A:\ Q^{(k+1)}(s,a) = Q^{(k)}(s,a) + \alpha \left(\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k,a_k) + \text{MAX}_{a'}\, e^{-\beta\tau_k} Q^{(k)}(s_{k+1},a') - Q^{(k)}(s_k,a_k) \right) e^{(k)}(s,a) \quad (10)$$

In the above expression, $\tau_k = t_{k+1} - t_k$ is the time that the system remains in state s_k ; $\alpha \in (0,1)$ denotes the *learning rate*; β is the *discount factor*; $\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k)$ is the sample discounted reward received in τ_k time units; $Q^{(k)}(s_{k+1}, a')$ is the estimated value of the state-action pair (s_{k+1}, a') in which s_{k+1} is the actually occurring next state. Moreover, in (1) $e^{(k)}(s, a)$ denotes the eligibility of each state-action pair (s, a) that reflects the degree to which this state-action pair has been chosen in the recent past. The eligibility is updated as follows:

$$e^{(k)}(s,a) = \lambda\, e^{-\beta\tau_{k-1}} e^{(k-1)}(s,a) + \delta\big((s,a),(s_k,a_k)\big) \quad (11)$$

Where $\boldsymbol{\delta}(\boldsymbol{x}, \boldsymbol{y})$ denotes the delta kronecker function.

III. System Model

A. Dynamic Power Management Problem

A power managed system is a collection of interacting components (some of which are power-manageable) to provide requested services and performance levels [12]. It can be modeled as a power state machine. States are corresponding to different levels of power consumption and operated performance. State transitions are controllable and are associated with overheads in terms of energy transition

and delay cost. The object of dynamic power management problem is to minimize the average energy consumption while maintaining an acceptable performance level. The decisions on when and how to perform state transitions are taken dynamically by the power manager policy. The main difficulty in low-power system design is to define the optimal policy that provides the best tradeoff between power and latency.

B. System Model

The power-managed system consists of a service requester (SR), a service queue (SQ) and a service provider (SP). The SR generates requests to be serviced by the SP. Inter arrival times between requests are stochastic process and characterize the system's environment. The SQ is a FIFO queue that stores requests waiting for processing. In this work we consider the SP as shown in Fig.2. It has three states:

- *Active* state: the system is processing the requests.
- *Idle* state: the system is still operational, but there are no service requests to deal with. The transition between the active and idle states is autonomous, i.e., as soon as the system completes servicing all of the waiting requests, it enters the idle state. Similarly, the system goes from *idle* to *active* as soon as a service request arrives.
- *Sleep* state: The SP moves to the *Sleep* state- where it has reduced power consumption- only from the idle state.

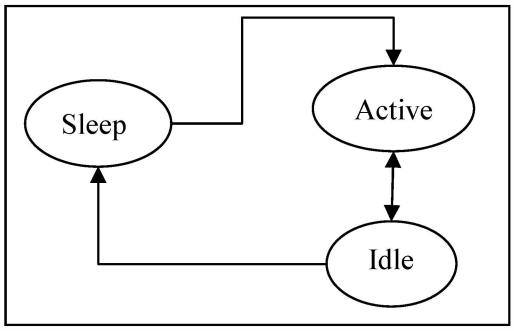


Fig.2. State diagram of SP.

The duration that the SP is kept in the idle state before it enters the sleep state determines the tradeoff between the service latency and power dissipation of the SP. The transition between these different power states are controlled by the power manager (PM).

IV. DPM FRAMEWORK USING REINFORCEMENT LEARNING

Our goal is to find an optimal policy for minimizing the energy consumption while maintaining an acceptable performance level. In this work, we consider the average number of waiting requests in the service queue as the performance measure. We define a cost function which is a linearly weighted combination of (i) the immediate power consumption and (ii) the caused delay.

Upon a transition from state s to state s' upon selection of the action a, the cost function $C(s,\ a;\ \alpha)$ is defined as follows:

$$C(s,a;\propto) = \propto * d(s,a) + (1-\propto) * p(s,a) \qquad (12)$$

Where $d(s,\ a)$ denotes the caused delay, $p(s,\ a)$ is the amount of power consumption when choosing the action a from the state s and α is the user defined parameter enabling the power-delay tradeoff.

The proposed RL framework operates as follows. At each decision epoch, the PM chooses an action for SP to implement the decision according to the following four cases:

1. The SP is in the idle state and the SQ does not contain any request.
2. The SP is in the idle state and the SQ contains at least one request.
3. The SP has just entered the sleep state and finds out some waiting requests in the SQ. This means, during the SP transition from idle to sleep state, at least one request has arrived.
4. The SP is in the sleep state and a new request arrives.

The state space of the environment is a composite space of SR states, SQ states and SP states. The power manager is the agent. It has to learn the DPM policy based on its observation of the system. Due to noise and disturbance, the PM observes a non-Markovian environment [6]. As reference it is pointed out in [7], the optimal policy in the idle state for non-Markovian environments is often a timeout policy. A timeout policy has a time-out value τ. In timeout policy, the device is put to sleep if its idle period is longer than the specified timeout value τ. The major difficulty is to accurately define the optimal timeout[1] period. An undesirable situation is where the SP is put to sleep after a short period of time, only to be awakened immediately, e.g. the next service request arrives early. Thus, the system has to pay for the extra energy and latency of waking up the SP to process the new coming requests. On the other hand, if the power manager sets the timeout to be long and yet no service requests arrives in that period, then the SP has unnecessarily wasted power by not going to sleep.

When the SP is in the idle state, the action set can be denoted as a finite set $A_{idle} = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}$ of timeout values. The PM has to learn the optimal action (timeout value) by using the RL techniques. If the PM finds itself in case (1), it will use the RL-based timeout policy (see the pseudo code). If the SP is in the idle state and some request arrives (case 2), the PM turns the SP active to process the waiting requests.

[1] The timeout refers to the minimum duration of idle time before entering into the sleep mode.

RL based Timeout policy

Input: the action set A (a set of $timeout$ values),

At each decision epoch t_k (SP=idle and SQ=0),

1. Choose an action a, which corresponds to a specific timeout value, from the action set A.
2. Let the PM execute the timeout policy with timeout value a.

At the next decision epoch t_{k+1},

3. The system finds itself either in the sleep state (no request came during the timeout period) or in the idle state (some request came in that period.)
4. Regardless, it evaluates the chosen action using the RL update method discussed in section II.
5. Go back to step 1.

When the SP is in the sleep state, we use the N-policy, wherein the SP is turned into the active state if the number of waiting requests in the SQ is more than a specified value denoted N. Otherwise, it will stay in the sleep state until the coming of the new request. The action set in case (3) and (4) can be denoted as a finite set of integers $A_{sleep} = \{n_1, n_2, n_3, \dots, n_n\}$ corresponding to different specified N values. The proposed PM learns the optimal threshold N value of waiting request among these by using the RL based N-policy explained in the following pseudo-code.

RL based N- policy

Input: the action set A (List of N values),

At each decision epoch t_k (SP=sleep),

1. Choose an action n, which corresponds to a specific timeout value, from the action set A.
2. **If** SQ $\geq$ n:
   The PM turns the SP active for processing requests. Then we have reached decision epoch t_{k+1}.
   Else
   The PM keeps SP in the sleep state.
   End

At the next decision epoch t_{k+1},

3. Evaluates the chosen action using the RL update method discussed in section II.
4. Go back to step 1.

V. Simulation Results

In this work, we developed the proposed DPM framework using both Q-learning and TD(λ) techniques, and we evaluate their performance and convergence speed. Simulation is done using real workloads running on a wireless adapter card (WLAN card). Table1 lists the power and delay characteristics of the considered device. In this table, P_{sleep}, P_{busy} and P_{idle} denote the power consumed when the SP is respectively in its sleep state, busy state and idle state. Tt_r is the time taken in transitioning to and from the sleep state while E_{tr} is the power consumption in waking up the device. T_{be} refers to the break even time. We conduct simulation on real traces measured using the *tcpdump* utility in Linux. The measured traces include: 45-minute trace for online video watching, 2-hour trace for web surfing, and 6-hour trace for a combination of web surfing, online chatting and server accessing, 2 hours each.

TABLE I. Power and delay characteristic of the WLAN card.

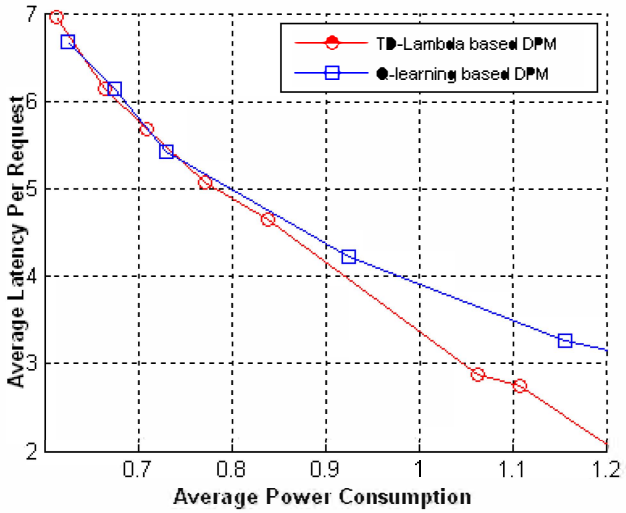
P_{sleep}	P_{busy}	P_{idle}	E_{tr}	T_{tr}	T_{be}
0 W*	1.6 W	0.9 W	0.9 J	0.3 s	0.7 s

*The WLAN card is turned off.

A list of timeout values is used defining the possible actions in the idle state as follows:{ 0, 0.3, 0.45, 0.6, 0.8, 1, 3 and 5}T_{be}. The *sleep* action set corresponds to the minimum number of waiting requests in the SQ to make the SP move from sleep to active state. For this experiment, the sate-action values considered are listed as follows :{ 1, 2, 3, 4 and 5}.

A. Performance Evaluation

The cost function is a linearly weighted combination of power consumption and caused delay according to (12). A user-defined parameter -corresponding to the chosen weight controls the power- latency tradeoff. When varying this parameter, we can get a set of different energy-performance tradeoff points in the design space as reported in Fig 3.



It is clear from Fig.3 that the tradeoff curves obtained using both Q-learning and TD(λ) are widely distributed over the design space providing multiple and widely distributed power-performance tradeoff points. However, under the same average latency, it is shown that the TD(λ) learning technique finds better policy t achieving lower energy consumption in comparison to what it is obtained using Q-learning. Moreover, at low delay constraints, the TD(λ) achieves much more better energy saving.

B. Convergence Speed Evaluation

In this set of experiment, we run both Q-learning and TD(λ) technique under similar workload and device characteristics and we reported their minimum required time for learning the optimal policy.

TABLE II. Experimental Results for convergence speed for Q-learning and TD(λ)

	Required Trial's Number	Required Time
Q-learning	5390	0.473s
TD(λ)	3575	0.406s

As shown from table II, the Q-learning based DPM framework needs 5390 trials to learn the best policy while the TD(λ) based DPM framework requires only 3575 trials to converge. This result confirm that TD(λ) requires smaller trial number to convergence. However, TD(λ) is more complex as it requires more time to compute similar number of trails than Q learning. But, even that TD(λ) is more complex, the optimal policy is learnt faster as a convergence time response of 0.406s is required in comparison with 0.473s for Q learning.Finally and in comparison with Q learning, the time response required for TD(λ) to learn the optimal policy is about 14% faster.

VI. CONCLUSION

In this paper, a continuous-time reinforcement learning (RL) dynamic power management (DPM) framework is proposed. The proposed DPM is model-free and does not require any prior information of the workload characteristics. The power manager can simultaneously consider power and performance, and perform well under various workload conditions. Moreover, power-latency tradeoffs can be precisely controlled based on a user-defined parameter.

Particularly, the TD(λ) and Q-learning based RL DPM frameworks are implemented and compared in terms of both performance and convergence speed rate. It is shown that TD(λ) and Q learning are both more powerful in terms of power saving and help find widely distributed power-latency tradeoff curves in comparison with reference studies in the field. Experimental results show also that the TD(λ) learning for SMDP has a faster convergence time response when compared to the Q-learning approach.

REFERENCES

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

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

[3] Q. Qiu, and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes", DAC, 1999, pp. 555-561.

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

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

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

[7] T. Simunic, L. Benini, P. Glynn and G. De Micheli, "Event-driven power management," *IEEE Trans. on CAD*, Vol. 20, pp. 840-857, Jul. 2001.

[8] S.M.Ross, "Applied Probability Models with optimization applications", Holden Day,San francisci,1970.

[9] L. P. Viswanathan and E. C. Monie,"Reinforcement temporal Difference Learning Scheme for Dynamic Energy Management in Embedded Systems," in *International conference on VLSI Design*, 2006.

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

[11] T. Simunic, and S. Boyd, "Managing power consumption in networks on chips", in DATE, 2002.

[12] Q. Qiu, Q. Wu, M. Pedram, "Stochastic modeling of a power-managed system—construction and optimization," IEEE Trans.. Computer- Aided Design Integr. Circuits Syst., vol. 20, no. 10, pp. 1200-1217, Oct. 2001.

[5] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," in *DAC '99,* pp. 555-561, 1999.

[6] T. Simunic, L. Benini, P. Glynn and G. De Micheli, "Event-driven power management," *IEEE Trans. on CAD*, Vol. 20, pp. 840-857, Jul. 2001.

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

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

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

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

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

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

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

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

[15] R. S. Sutton, "Learning to predict by the method of temporal differences", Machine Learning, pp. 3:9-44, 1988.