Reinforcement Learning-Based Dynamic Power Management of a Battery-Powered System Supplying Multiple Active Modes

Maryam Triki[1],Ahmed C. Ammari[1,2]
[1]MMA Laboratory, INSAT
Carthage University, Tunis, Tunisia
[2]Department of Elect.& Comp. Engineering,
King Abdulaziz University, Jeddah, Saudi Arabia

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

***Abstract*—This paper addresses the problem of extending battery service lifetime in a portable electronic system while maintaining an acceptable performance degradation level. The proposed dynamic power management (DPM) framework is based on model-free reinforcement learning (RL) technique. In this DPM framework, the Power Manager (PM) adapts the system operating mode to the actual battery state of charge. It uses RL technique to accurately define the optimal battery voltage threshold value and use it to specify the system active mode. In addition, the PM automatically adjusts the power management policy by learning the optimal timeout value. Moreover, the SoC and latency tradeoffs can be precisely controlled based on a user-defined parameter. Experiments show that the proposed method outperforms existing methods by 35% in terms of saving battery service lifetime.**

Keywords-Dynamic power management; reinforcement learning, extending battery lifetime; battery-powered system design.

I. INTRODUCTION

Batteries are widely used as the only source of power in several applications. High power consumption reduces the battery service lifetime[1].Thus, reducing the power consumption in battery-operated portable devices has become a major concern. The goal of low-power design for battery-powered devices is to extend the battery service life while maintaining performance degradation within an acceptable level. Dynamic power management– which refers to a selective, shut-off or slow-down of system components that are idle or underutilized– has proven to be a particularly effective technique for reducing power dissipation in such systems [1].

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 the device down if the predicted length of idle time justifies the cost. 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]. The essential shortcoming of these methods consists in the need of an exact knowledge of the state transition probability function of the MDP. However, the workload of a complex system is usually varying with time and hard for accurate prediction. The machine learning-based DPM learns a policy online by trying to learn the best-suited action for each system state, based on the reward or penalty received. Such DPM methods can simultaneously consider power and performance, and perform well under various workload conditions [5], [6], [7], [8].

Although DPM techniques effectively reduce the system power consumption, they are not able to obtain the optimal policy for extending the service lifetime of a battery operated devices. This is because the characteristics of battery power source are not properly modeled and exploited in these techniques [1]. Due to the battery characteristics, a minimum power consumption policy does not necessarily result in the longest battery service life. To extend the service lifetime of battery-powered devices, authors in [9] proposed three policies: an open-loop policy, a closed-loop[2] policy and a combination of the two aforesaid policies. The open-loop policies attempt to reduce the average power consumption but they do not take into consideration the battery's state of charge (SoC) while managing the system power. Compared to the open-loop policies, the closed-loop policies are based on the observation of both battery's output voltage and system workload. The battery's output voltage is related nonlinearly with the charge state. As a consequence, the closed-loop policies help in maximizing

[1] Battery lifetime is the time one can use the battery before it is empty.[1]

[2] Closed-loop policy is the switching from a high quality factor system state to a low quality factor system state when the output voltage of the battery drops below some threshold.[10]

978-1-4799-2578-0/13 $31.00 © 2013 IEEE
DOI 10.1109/EMS.2013.74

the time of battery operation more effectively by adapting a component's shutdown scheme to the actual battery charge state. However, the presented work in [9] used all these policies in a heuristic manner. Rong et al. in[10] attempt to maximize the battery service lifetime while meeting a given service timing constraints. Unfortunately, this work is based on the assumption of a continuous-time MDP model given in advance for the power management system which may not be realistic.

This paper addresses the problem of extending the battery service lifetime in a portable electronic system while maintaining an acceptable performance degradation level. The proposed dynamic power management (DPM) framework is based on model-free reinforcement learning (RL) technique that performs learning and power management in a continuous-time and event-driven manner. It has fast convergence rate and less reliance on the Markovian property. The presented DPM framework can dynamically perform power management according to both system's workload and battery state of charge. It uses the reinforcement learning (RL) technique to learn online the optimal battery output voltage threshold value for a closed-loop policy. Experiments on measured data traces demonstrate the superior performance of the proposed dynamic power management method in comparison with prior works [10].

II. THEORETICAL BACKGROUND

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 $R: S \times A \to R$. A policy $\pi = \{(s,a)|a \in A, s \in S\}$ is a set of state-action pairs for all states in the RL framework. We use notation $\pi(s) = a$ to specify the action chosen in state s according to policy π.

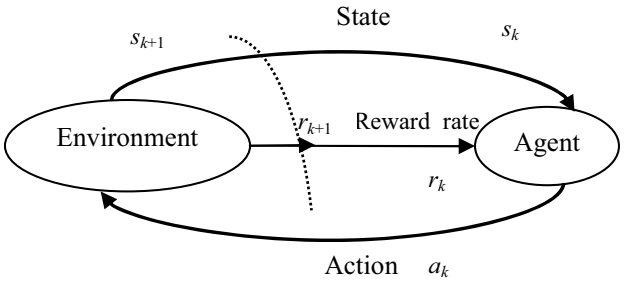

Fig.1.Agent-environment interaction model.

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.

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. An optimal policy is the one 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. The temporal difference (TD) learning method for SMDP generates an estimate $Q^k(s,a)$ for each state-action pair (s,a) at epoch t_k, which is the estimate of the actual value $Q^\pi(s,a)$ following policy π. Suppose that statess_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 [11] . 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}.

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:

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

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:

$$e^{(k)}(s,a) = \lambda e^{-\beta\tau_{k-1}} e^{(k-1)}(s,a) + \delta\big((s,a),(s_k,a_k)\big) \quad (2)$$

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

III. SYSTEM MODELING

We consider a portable, battery-powered electronic system and we focus on extending battery lifetime while maintaining performance degradation within an acceptable level. Basically, we propose an online adaptive approach that enhances the closed-loop policy. The PM uses reinforcement learning technique to accurately define the optimal battery voltage threshold value and accordingly controls the state transitions of the device.

A. The Global System Architecture

The basic system consists of a service requestor (SR) generating the requests, a service queue (SQ) to store the requests waiting for processing, and a service provider (SP) powered by a single battery to supply for the required services. For real systems, the SR most likely has a non-stationary behavior and the exact generating time instances of service requests are not known a priori.

B. Model of the Service Provider

The Service Provider (SP) has six main states as shown in Fig.2. The SP is active (busy) while processing services, and becomes idle after it finishes servicing a request. The active state is defined by two sub-states: *High Performance* (HP) state when the system is providing high quality services, and *Low Performance* (LP) state when the system provides low quality services. In this work, we use the SP response time as the service quality metric. Thus, in the HP state the system services the SR more rapidly than in the LP state.

In the *Idle* state, the system is still fully up and 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. The SP moves to the *Sleep* state-where it has reduced power consumption- only from the idle state. 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 *Off* state refers to a completely turned off system.

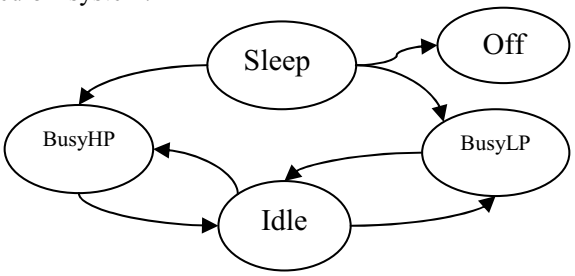

Fig.2. State diagram of SP.

C. Model of the Battery

1) Basic Model of the Battery

A battery state of charge (SoC) is usually changing with time depending both on user activities and on battery properties. Basically, the battery may be discharged when it is used. In addition, when the battery is given some rest it may recover an amount of its deliverable charge. These variations results into many states listed below:

- *FC state corresponds* to the fully charged state and means that the battery is fully charged.
- *EX* state denotes an exhausted battery in which the battery output voltage falls below a given voltage threshold (e.g., 80% of the nominal voltage).
- *FD state* represents the fully discharged state in which the battery has been completely discharged and thus, the system becomes no more operational and goes to its off state.
- *AC state: State* in which the battery output voltage exceeds the voltage threshold value.

2) Circuit Model of the Battery

We model the battery by connection of circuit elements as shown in Fig.3. The equivalent circuit model of the Li-ion battery is given in [12].

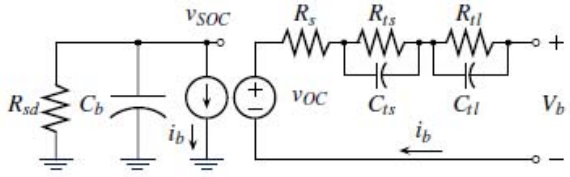

Fig.3. Equivalent circuit model of the Li-ion battery.

We use the following non-linear equations to define all the circuit model component values [13]. In these equations, b_{ij} are empirically-extracted regression coefficients and C_{init} represents the nominal energy capacity of the battery.

$$v_{oc} = b_{11}\, e^{b_{12}\, v_{soc}} + b_{13}\, {v_{soc}}^3 + b_{14}\, {v_{soc}}^2 + b_{15}\, v_{soc} + b_{16} \quad (3)$$

$$R_s = b_{21}\, e^{b_{22}\, v_{soc}} + b_{23} \quad (4)$$

$$R_{ts} = b_{31}\, e^{b_{32}\, v_{soc}} + b_{33} \quad (5)$$

$$\mathrm{C_{ts}} = \mathrm{b_{41}\, e^{b_{42}\, v_{soc}} + b_{43}} \quad (6)$$

$$R_{tl} = b_{51}\, e^{b_{52}\, v_{soc}} + b_{53} \quad (7)$$

$$C_{tl} = b_{61}\, e^{b_{62}\, v_{soc}} + b_{63} \quad (8)$$

$$C_b = 3600.\, C_{init} \quad (9)$$

In these equations, v_{oc} denotes the voltage open circuit and v_{soc} denotes the voltage at Soc.

IV. DPM Framework Using Reinforcement Learning

This paper focuses on extending service lifetime of a single battery-powered electronic system. The goal is to find an optimal policy for minimizing the energy delivered from the battery while maintaining an acceptable average number of waiting requests in the service queue. Basically, the problem we are addressing in this paper is to identify the battery output voltage threshold -for a closed-loop policy- that should be set in order to make the battery service lifetime optimal. To do so, we are going to implement a reinforcement learning technique that shall learn the optimal threshold voltage that guarantees the best tradeoffs between the battery lifetime and the system-performance. First, a system with a given pre-specified timeout policy is used. The RL technique is thus implemented to define the optimal voltage threshold value for that fixed timeout value. Then, the framework is enhanced with an optimal Timeout-learned policy that integrates a learned voltage threshold policy to get the best system performance.

A. System with a pre-specified given timeout policy

Let *V* denote the threshold value of the battery output voltage. A list of *V* values will serve as the action set *A* for the PM in controlling the SP for switching from the high performance state (BusyHP) to a low performance state (busyLP). The PM learns to choose the optimal action $a \in A$, which corresponds to the optimal *V* value, by using an RL technique. Notice that, to accurately learn the best supply voltage, we must run multiple battery charging/discharging cycles. The proposed RL framework operates as follows. At each decision epoch, the PM will issue commands to the SP to implement the decision according to the following four cases:

1. The SP is in the idle state and a request comes before the timeout period expires. Thus, the PM decides to turn the SP into the active state.
2. The SP is transitioning to the active state. According to the battery output voltage, the PM decides either or not to turn the SP into the low performance state. More precisely, if the battery output voltage falls below a given threshold value, the SP operates in the LP state.
3. The SP is in the idle state and the timeout has expired. Thus, the SP is put to sleep.
4. The SP is in the sleep state and a request comes. In this case, the PM turns the SP active to process the incoming requests.

In this implementation, the optimal policy is achieved by minimizing the cost function which is a linearly weighted combination of (i) the battery operation time and (ii) the average response time.

B. Framework enhancement with learning of the optimal timeout Value

The duration 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 major difficulty is to accurately define the optimal timeout period (the minimum duration of idle time before entering into the sleep mode). 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 comes 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.

In addition to the learning of the optimal battery threshold value, the proposed DPM framework is enhanced with learning the optimal timeout policy under non-stationary workload. In this case, the PM will automatically learn the optimal timeout policy for the system and then accordingly learn the best threshold voltage output value (*V*) suitable for that timeout policy. A list of voltage threshold values (*V*) will serve as the action set A. In addition, a list of timeout values (T_{out}) will serve as a second action set $\hat{A}$. The PM learns to choose the optimal a = *V* for the optimal $\hat{a}= T_{out}$ values among the action sets A and $\hat{A}$. Details of the finally implemented RL-Based DPM algorithm are provided in Algorithm 1. The cost function for learning the optimal timeout value is a linearly weighted combination of (i) instantaneous power consumption and (ii) the number of requests buffered in the SQ.

V. Simulation Results

In this section, the effectiveness of the DPM framework on extending the battery lifetime is shown. For the experiments we consider a *Lithium-Ion* battery as it is widely utilized for portable system. Table1 lists some of battery characteristics (See [13] for details about extracting the Battery parameters).

TABLE I. Extracted Simulation Parameters of the Battery.

b11	*-0.67*	***b12***	*-16.21*	***b13***	*-0.03*
b14	1.28	***b15***	-0.40	***b16***	7.55
b21	0.10	***b22***	-4.32	***b23***	0.34
b31	0.15	***b32***	-19.60	***b33***	0.19
b41	-72.39	***b42***	-40.83	***b43***	102.80
b51	2.07	***b52***	-190.41	***b53***	0.20
b61	-695.30	***b62***	-110.63	***b63***	611.50
Cinit	0.35				

Algorithm 1:The Enhanced RL-Based DPM Algorithm.

Input: An action set $\hat{A}$ of T_{out} values, an action set A of V values, the battery output voltage V_{out}.

Do for 1,2,3,..,10000 (number of charge/discharge cycle)

Initialization : The battery is fully charged

Repeat

At each decision epoch t_k:

Choose an action a, which corresponds to a specific battery output voltage threshold, from the action set A.

Choose an action $\hat{a}$, which corresponds to the Timeout value, from the action set $\hat{A}$.

Let the PM execute the timeout policy with timeout value $\hat{a}$.

If the SP is in the idle state:

If some request comes before the timeout period (with duration of $\hat{a}$) expires:

The PM turns the SP active for processing requests until the SP becomes idle again. Then we have reached decision epoch t_{k+1}.

Else

The PM keeps SP idle for $\hat{a}$ period of time.

End

Else if the SP is in the sleep state:

If some request comes:

The PM turns the SP active for processing requests until the SP becomes idle again. Then we have reached decision epoch t_{k+1}.

Else

The PM keeps SP in the sleep state.

End

Else (the SP is in the active state)

If$V_{out} < a$:

The PM turns the SP into the Low Performance state for processing requests. Then we have reached decision epoch t_{k+1}.

Else

The PM turns the SP into the High Performance state for processing requests. Then we have reached decision epoch t_{k+1}.

End

End

Evaluate the chosen action $\hat{a}$ using the TD(λ) technique.

Until the battery is fully discharged

Evaluate the chosen action a using the RL technique.

The first set of experiments demonstrates the effectiveness of the basic DPM framework implemented without considering the learning of the optimal timeout value. These experiments are applied to a Hard Disk Drive (HDD) with parameters defined in table 2. The timeout value is fixed to 0.2 T_{be}. We conduct two experiments by assuming different HDD characteristics such as voltage; current and time of service values for both *high performance* state and *low performance* state (see Table 2). At each state, given the SP current and voltage, we calculate the battery voltage and current. Assuming the battery voltage is constant in each state, the actual power drawn from the battery is obtained. The SoC degradation denotes the amount of the consumed charge from the initial battery state of charge. We perform different simulations under multiple battery's charging and discharging cycles. For the workload, we measured a real 6-hour server accessing trace using the *tcpdump* utility in Linux server. Then, we copy it multiple times to simulate multiple charge/discharge cycles of a battery.

TABLE II. THE ASSUMED HDD CHARACTERISTICS FOR THE TWO EXPERIMENTS

	Experiment 1		**Experiment 2**	
	LP mode	***HP mode***	***LP mode***	***HP mode***
Current (Amp)	0.4	0.6	0.4	0.8
Voltage (V)	3.6	3.6	3.6	3.6
Time of service (s)	1.7	1.2	1.7	1.2
Power sleep (W)	0.13			
Power idle (W)	0.2			
Time transition idle to sleep (s)	1.6			
Time transition sleep to active (s)	1.6			

To better understand how the battery lifetime is affected, we report in Fig.4 the SoC degradation and latency tradeoff curve obtained for the HDD using the both experiment 1 and 2 setting parameters in comparison with what it is obtained with the pre-specified timeout policy. It is seen from fig 4 that the developed DPM framework provides a wide range of SoC-latency tradeoffs for both experiments 1 and 2 settings. The SoC and latency tradeoffs are precisely controlled based on a user-defined parameter that is weighing the cost function's linear combination of (i) the battery operation time and (ii) the average response time. Better performances are obtained for the HDD with experiment 1 characteristics particularly for low latency values. Higher average latencies would imply reduced performance at the gain of longer battery duration and higher values of *battery output voltage threshold.* Given that the HDD is running the same LP mode current for both experiments, similar tradeoff curve behaviors are observed for increasing latency values for both experiment 1 and 2 settings. In comparison with the fixed-timeout policy, **15.7%** maximum battery SoC improvement is obtained for low delay constraints using the HDD experiment 1 parameter settings. Using

experiment 2 settings, even better performances are obtained achieving **29.89%** of SoC saving in comparison with the fixed-timeout policy. This even outperforms reference [10] by **35.86%** in terms of battery lifetime extension.

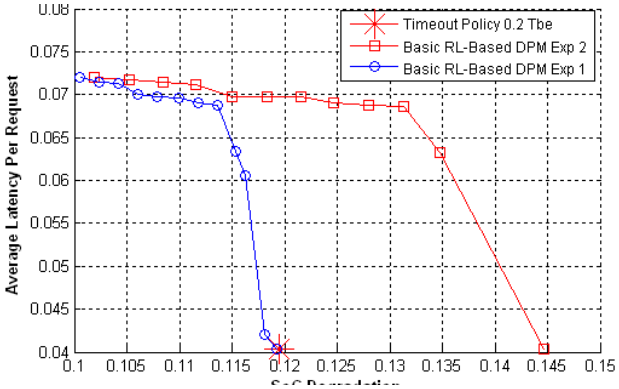

Fig.4. SoC-latency trade off curves of the 0.2 T_{be} fixed timeout policy running the RL-based DPM on HDD for both experiment 1, and 2 sets.

To show the effectiveness of the enhanced RL- battery DPM framework with learning *timeout*, we run a third experiment with the same workload for the HDD using experiment 1 parameter settings. . For this experiment, learning timeout is activated with a second action set $\hat{A}$ timeout values (T_{out}) listed as follows :{ 0.1T_{be}, 0.2 T_{be}, 0.3 T_{be},0. 5T_{be}, 2 T_{be},3T_{be} and 5 T_{be} }. In this case, the PM will simultaneously learn the optimal timeout value among the timeout action set values and, then learn the best battery voltage threshold for the optimal learned timeout policy. The obtained results are given in fig 5.

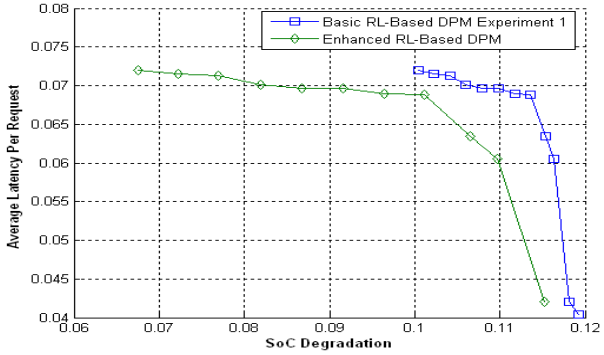

Fig.5. SoC-latency tradeoff curves of the RL-based DPM with and without learning timeout using HDD with experiment 1 parameter settings.

It is clear from figure 5 that incorporating learning the timeout with learning the voltage threshold further helps minimizing the average power consumption. In this case, average power reduction and battery lifetime extension are combined together to higher the effectiveness of the proposed DPM framework.

In comparison with a pre-specified given timeout policy, the Enhanced framework with learning timeout can achieve a "wider and deeper" SoC-latency tradeoff curve and outperforms the basic one under the same latency values. The enhanced RL-based DPM algorithm with *Timeout*-learning can achieve much lower power consumption than the basic RL-based DPM framework particularly when the system has tight latency constraint.

VI. CONCLUSION

In this paper we proposed a closed-loop policy that increases battery lifetime by using reinforcement learning technique to accurately define the best battery voltage threshold value and accordingly set the system active mode. The proposed DPM framework is model-free and requires no prior information of the workload characteristics. Moreover, the PM uses the TD(λ) algorithm for SMDP to define the best timeout value and automatically adjusts the power management policy to further enhance energy savings.

REFERENCES

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

[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] H. Jung, M. Pedram, Dynamic power management under uncertain information, DATE, Apr. 2007, pp. 1060-1065.

[6] G. Dhiman, and T. Simunic Rosing, "Dynamic power management using machine learning", ICCAD, Nov. 2006, pp. 747-754.

[7] 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", DAC, Jun. 2011, pp. 875-878.

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

[9] L. Benini, G. Castelli, A. Macii, and R. Scarsi, "Battery-driven dynamic power management," IEEE Design & Test of Computers, Vol. 18, pp.53-60, 2001.

[10] P. Rong, and M. Pedram, "Battery-Aware Power Management Based on Markovian Decision Processes," IEEE Trans. on Computer Aided Design, Vol. 25, No. 7, Jul. 2006, pp. 1337-1349.

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

[12] M. Chen and G. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and I-V performance," IEEE T. on Energy Conversion, 2006.

[13] D. Shin, Y. Wang, N. Chang, and M. Pedram, "Battery-supercapacitor hybrid system for high-rate pulsed load applications," Proc. of Design Automation and Test in Europe (DATE), Mar. 2011.