IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE)
e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 12, Issue 3 Ver. II (May – June 2017), PP 34-46
www.iosrjournals.org

Extending the Service Lifetime of a Battery-Powered System Supporting Multiple Active Modes

Maryam Triki [(1)], Ahmed Chiheb Ammari [(2)]
[1]*Faculty of Computing & Information Technology, Department of Computer Science, King Abdulaziz University, Jeddah 21589, Saudi Arabia*
[2]*Renewable Energy Research Group, Department of Electrical and Computer Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah 21589, Saudi Arabia*

Abstract: *Dynamic power management (DPM) is a well known power optimization technique that aims at reducing the power consumption of a system while maintaining an acceptable performance degradation level. This paper presents a Reinforcement Learning-based dynamic power management framework for extending the battery service lifetime of a system with multiple active modes. In the proposed framework, a Power Manager adapts the system operating mode to the actual battery state of charge based on the observation of the battery output voltage. Moreover, it uses the reinforcement learning technique to accurately define the optimal battery voltage threshold value and use it to control the system active mode. Additionally, the power management policy is automatically adapted to the optimal timeout value. The proposed algorithms are experimented on both single and dual-battery powered systems and the obtained results confirm their excellent performance for defining the optimal power management policy particularly for high latency constrained systems. Experimental results show an improvement in battery State of Charge (SoC) savings up to 42.83% and 55.38 % respectively for single battery and dual battery systems in comparison with classic pre-specified timeout policies using a given preset battery threshold voltage*

Keywords: *Battery-powered system design, Energy savings, Reinforcement Learning; Dynamic Power Management; Extending battery lifetime.*

I. Introduction

Batteries are widely used as the only source of power in many applications. Thus, extending the battery service lifetime of battery-powered systems become a major concern [1], [2], [3]. Dynamic Power Management (DPM) techniques refer to a selective shut-off or slow-down of system components that are idle. Although, such techniques effectively reduce the system power consumption, they are not able to obtain the optimal policy for extending the battery lifetime. This is because the battery characteristics are not properly modeled and exploited in these techniques [4]. Due to the battery characteristics, a minimum power consumption policy does not necessarily result in the longest battery lifetime. Therefore, an effective power management technique must consider the system workload and the battery characteristics to extend the battery lifetime [5], [6]. In this context, authors in [7] proposed three policies: an open-loop policy, a closed-loop policy and their combination. The open-loop policies attempt to reduce the average power consumption independently from the battery state of charge while managing the system power. The closed-loop policy is based on the idea of switching from a high quality factor to a low quality factor of the system state when the output voltage of the battery drops below some voltage threshold. Compared to the open-loop ones, the closed-loop policies are based on the observation of both battery's output voltage and system workload. As a consequence, they help maximize the time of battery operation more effectively by adapting a component's shutdown scheme to the actual battery state of charge. This can be performed by lowering the quality of service when the battery's output voltage falls below a voltage threshold. If the battery is fully charged, the system is kept in a high performance state providing high quality services. The rationale for this policy is to provide graceful degradation of system performance as the battery discharges. In this case, it is clear that the choice of the voltage threshold is critical for trading off the service quality with battery lifetime. For the work presented in [7], the threshold is obtained in a heuristic manner.

Rong et al. in [8] attempt to maximize the battery service lifetime by using a stochastic approach. It can take into account both power and performance and able to derive provably optimal DPM policies, by modeling the request arrival times and device service time as stationary stochastic processes such as Markov Decision Processes (MDP) [9]. Its essential shortcoming is the need of exact knowledge of the MDP state transition probability function. However, the workload of a complex system is usually changing with time and hard to be predicted accurately [10]. The workload variation has a significant impact on the system performance and power consumption. Thus, a robust power management technique must consider the uncertainty and variability that emanate from the environment, hardware and application characteristics [11] and must be able to interact with the environment to obtain information that can be processed to produce optimal policies.

In this context, several previous works have used machine learning for adaptive DPM policy optimization [12]-[16]. Machine learning-based methods such as Reinforcement Learning (RL) can simultaneously consider power and performance, and perform well under various workload conditions when the system model is not known *a priori*. Tan et al. in [17] propose to use an enhanced Q-learning algorithm for system-level DPM. Wang et al. in [18] extend this work to allow a power manager working in a continuous-time and event-driven manner with faster convergence rate, by exploiting the a learning framework for Semi-MDP [19].

This paper presents a reinforcement learning approach for extending battery service lifetime in a portable electronic system with multiple active modes. More precisely, a DPM framework is implemented to perform power management by learning online the optimal battery threshold voltage for a closed-loop policy. This work extends the basic DPM framework presented in [20] to a dual battery powered system and is enhanced to automatically adjust the power management policy by learning the optimal timeout value. This framework has the following main characteristics: i) it can dynamically perform power management in a continuous-time and event-driven manner according to both system workload and battery state of charge; and ii) it has fast convergence rate and less reliance on the Markovian property. Using the learning approach in [18], a power manager simultaneously learns the optimal policy for non-stationary workloads and uses that policy to control instead of only evaluating one predefined policy. Experiments on measured data traces show the power of learning the optimal DPM policy and demonstrate the superior performance of the proposed framework in comparison with prior works [7], [8].

The remainder of this paper is organized as follows. We provide an overview of reinforcement learning background in Section 2. The proposed dynamic power management framework architecture is presented in Section 3, and its implementation details are given in Section 4. The experimental results are analysed in Section 5. We conclude this paper in Section 6.

II. Theoretical Background

Reinforcement learning is a branch of machine learning based on learning through experience accumulation [21] and is widely applied for large optimal decision making problems.

1. Formalism of the agent-environment interaction

A general reinforcement learning (RL) model as illustrated in Figure 1 consists of an agent, a finite state space S, a set of available actions A, and a reward function $Reward: S \times A \rightarrow R$. The core of an RL technique is the interaction between the agent and its environment. The agent is the learner and decision maker. The environment is defined as any sensory information the agent receives. Actions refer to the decisions the agent is intended to make [22]. State represents the situation the agent can find itself in. More precisely, the state is the available information about the agent's environment that helps the agent in decision making.

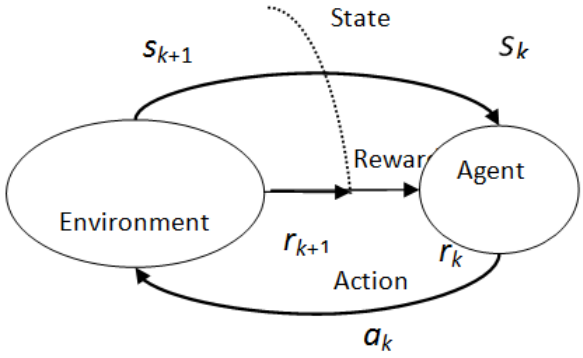

Fig.1. Agent-environment interaction model.

At each step of interaction with the environment, the agent receives a representation of the environment's state $s_t \in S$ and 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 taken action, the agent moves from its current state s_k to a new state s_{k+1} and receives from the environment a reward r_{t+1} (a real or natural number) or a punishment (a negative reward) 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 cumulative received rewards.

We define a policy $\pi = \{(s, a) | a \in A, s \in S\}$ as the set of all possible state-action pairs in the RL framework. It can be interpreted as mapping from states to probabilities of selecting possible actions: $\pi_t(s, a) = Pr\{a_t = a | s_t = s\}$.

The RL agent continuously adjusts its policy so as to maximize the total amount of reward received over the long run. It is worth to state that the RL agent-environment interaction model is abstract, generic and flexible. In fact, the time can be continuous and not defined in fixed intervals. The actions can be different and vary as well. The states can be completely intrinsic and can have different forms (signal readings, symbolic description, etc).

2. Reinforcement learning in discrete-time

The reinforcement learning problem is to determine the optimal policy that maximizes the value functions for all state-action pairs. Generally, the agent has no predefined policy or knowledge about state transition characteristics. Therefore the agent has to simultaneously learn the optimal policy and use that policy to control. 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 $Q^{\pi}(\mathrm{s}, \mathrm{a})$, as the expected return when starting from state s, choosing action a (according to policy π), and following π thereafter.

The agent keeps a value function $Q^{\pi}(s, a)$, for each state-action pair (s, a) initially chosen by the designer and then updated each time an action is taken and a reward is received. The Q value function represents the *expected long-term reward* when starting from state s, choosing action a (according to policy π), and following π thereafter. At decision epoch t_k, the action with the highest Q value is chosen. At decision epoch t_{k+1}, the Q value $Q(s_t, a_t)$ is updated.

The update rule for discrete time RL is based on the following equation:

$\forall\ (s, a) \in S \times A$:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\ [\, r_{t+1} + \gamma \quad max_a\ Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad (1)$$

where $\alpha \in (0,1)$ denotes the *learning rate*; r_{t+1} is the reward received at time t and $\gamma \in (0,1)$ is the *discount factor*. The agent chooses the action with the maximum estimated value $Q(s, a)$ for various actions $a \in A$ next time the state s is visited, and then adjusts its policy so as to maximize the total amount of reward received over the long run.

3. Reinforcement learning in continuous-time

The continuous-time RL technique is implemented based on the *Temporal Difference* learning method [22] for a Semi Markov Decision Process (SMDP) called TD(λ) for short. Since the SMDP is a continuous time dynamic system composed of a countable state set S, and a finite action set *A*, there exists a countable set of *decision epochs* $\{t_0, t_1, t_2, \ldots, t_K, \ldots\}$. More precisely, at decision epoch t_k, the system has just transitioned to state s_k in response to a certain event. The agent selects an action a_k according to some policy. 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+1}.

The temporal difference learning (TD) 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 state s_k is visited at epoch t_k. Then at that epoch the agent chooses an action either with the maximum estimated value $Q^k(s_k, a)$ for various actions $a \in A$, or by using other semi-greedy policies [22]. 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}. We choose to use the algorithm presented in [9] 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 [22] is given as follows:

$$\forall\ (s, a) \in S \times A:\ Q^{(k+1)}(s, a) = Q^{(k)}(s, a) + \alpha\, e^{(k)}(s, a) \left[\begin{array}{c} \frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k) \\ + \max_{a'} e^{-\beta\tau_k} Q^{(k)}(s_{k+1}, a') - Q^{(k)}(s_k, a_k) \end{array} \right] \qquad (2)$$

In the above expression, $\tau_k = t_{k+1} - t_k$ is the time the system remains in state s_k; $\frac{1-e^{-\beta\tau_k}}{\beta} r(s_k, a_k)$ 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 Eq.(2), $e^{(k)}(s, a)$ denotes the eligibility of each state-action pair in order to facilitate the implementation of the TD(λ) algorithm. Such eligibility reflects the degree to which the state-action pair (s, a) has been chosen in the recent past and it is updated as follows:

$$e^{(k)}(s, a) = \lambda\, e^{-\beta\tau_{k-1}} e^{(k-1)}(s, a) + \delta\big((s, a), (s_k, a_k)\big) \qquad (3)$$

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

III. Indentations And Equations

We target a battery powered system under non-stationary workload. The basic system as shown in Figure 2 consists of a service requestor (SR) generating the requests, a service queue (SQ) to store the requests waiting for processing, and a device providing services to the workload called service provider (SP). Like real systems, the service request's exact generating time instances are not known a priori.

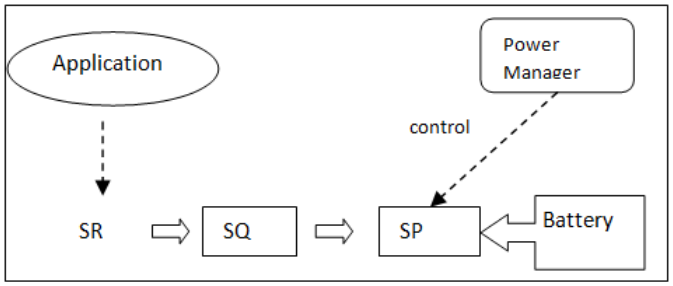

Fig.2. The global system architecture.

3.1. Service Provider Model

In this work, we consider a service provider as shown in Figure 3. It has five main states explained as follows:

- *Active* state: the SP is fully functional and some requests are being processed. The active state includes two different power modes :
  - *High Performance* state (HP) when the system consumes more power but provides high quality services;
  - *Low Performance* state (LP) when the system provides low quality services and consumes less power.

  We use the service provider response time as the service quality metric [23]. In HP state, the SP power consumption is higher but services the service requests more rapidly than in LP state. In contrast, the SP has longer execution time and less energy consumption in LP state compared to its HP state.
- *Idle* state: the system is still operational, but there are no service requests to deal with [24]. 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* once a service request arrives.
- *Sleep* state: The SP moves to the *Sleep* state- where it has reduced power consumption- only from the idle state. The device turns into active as soon as a service request arrives.
- *Off* state refers to a completely turned off system.

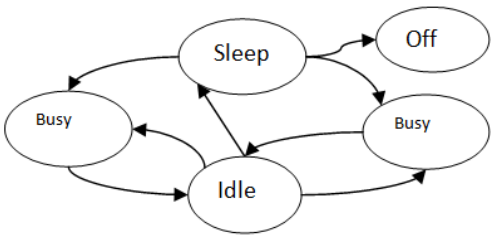

Fig.3. State diagram of a service provider.

3.2. Battery Model

We use *state-of-charge* (SoC) to denote the status of the battery. A battery SoC is usually changing over time depending both on user activities and on battery properties. Ideally, the battery voltage is constant over a complete discharge cycle, and it drops to zero when the battery is fully discharged. In practice, the battery output voltage decreases as a function of the discharge time, and the battery is exhausted when its output voltage falls below a given voltage threshold [7]. Thus, the battery output voltage describes the battery SoC.

Recall that the targeting system has two active modes with different performance and power consumption levels. The SP switching between its two busy HP and busy LP states depends on the status of the power supply, i.e., the battery SoC. More precisely, we use the battery voltage threshold for setting the appropriate active mode. Thus, an efficient DPM framework must identify the best battery output voltage threshold V_{th} to be set in order to make the battery service lifetime optimal.

In addition, two other important factors distinguish the real batteries from ideal ones. These factors are the basis of the proposed DPM framework: i) the total energy capacity of a battery is strongly related to its discharge current rate. More precisely, the deliverable capacity of the battery decreases as the discharge current rate increases. This phenomenon is called the rate-capacity characteristic [25]; and ii) a battery can recover some

of its deliverable capacity when it is given some rest in a state during which no current is drawn. This property is called *the recovery or relaxation characteristic* [26], [27]. This characteristic will be exploited in a dual-battery powered system case.

For this study, we are considering a *Lithium-Ion* battery given its wide use in portable systems. We use the equivalent circuit model of a Li-ion battery given in [28] as shown in Figure 4. This circuit model can be used to predict both the battery runtime and *I–V* performance accurately.

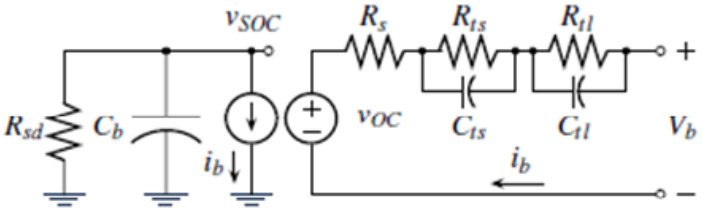

Fig.4. Equivalent circuit model of the Li-ion battery [18].

On the right of Figure 4, the used battery model consists of a series resistor R_S and two RC parallel networks (R_{TS}, C_{TS}) and (R_{TL}, C_{TL}). These components are modeling for the battery transient response to load events at a particular open-circuit voltage V_{OC}. For example, in a step load current event, R_S is responsible for the instantaneous voltage drop of the step response. (R_{TS}, C_{TS}) and (R_{TL}, C_{TL}) are respectively modeling for the short time constant and long time constant responses.

On the left, a capacitor C_b and a current-controlled current source i_b model the capacity, State of Charge (SoC), and runtime of the battery. C_b represents the whole charge stored in the battery, i.e., SoC. i_b is used to charge or discharge C_b so that the SoC changes dynamically and the battery runtime is obtained when the battery voltage reaches the end-of-discharge voltage. R_{sd} is used to characterize the self-discharge energy loss when batteries are stored for a long time. Theoretically, all the parameters of the proposed model are multivariable functions of SoC, discharge current, temperature, and cycle count. However, within certain error tolerance, the model parameters' dependence on temperature, discharge current and cycle number can be simplified given their negligible effects [29]. Finally, the circuit model component values are obtained according to the following non-linear single variable equations [29].

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

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

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

$$C_{ts} = b_{41}\, e^{b_{42}\, v_{soc}} + b_{43}$$

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

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

$$C_b = 3600.\, C_{init}$$

where

b_{ij} :is an empirically-extracted regression coefficient. Its detailed extraction methods can be found in [30].

C_{init}: represents the nominal energy capacity of the battery (SoC) in Ahr

v_{oc} : denotes the open circuit voltage and

v_{soc} : denotes for the initial voltage across C_b that quantitatively represents the battery SoC.

Table 1 lists the empirically-extracted regression coefficients as detailed in [30].

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				

IV. Dynamic Power Management Framework Using Reinforcement Learning

This paper focuses on extending the service lifetime of a battery-powered electronic system. The goal is to develop a Power Manager (PM) that implements the optimal policy for minimizing the energy delivered from the battery while maintaining an acceptable average number of waiting requests in a service queue. Basically, the first problem we are addressing in this paper is to identify the battery output voltage threshold V_{th} to be set in order to make the battery service lifetime optimal. For this aim, reinforcement learning techniques are used to learn the optimal V_{th} that guarantees the best tradeoffs between the battery lifetime and the system-performance. A system with a given pre-specified timeout policy is first used and an optimal voltage threshold value for that fixed timeout value is obtained by reinforcement learning. Then, the framework is enhanced with an optimal timeout-policy that integrates a learned voltage threshold policy to get best system performance.

Learning optimal battery output voltage threshold

The Battery SoC is regularly changing over time and the SoC values are sampled at regular intervals regardless of the system events. Therefore, discrete time RL is adopted for learning the optimal threshold voltage V_{th} according to Eq. (1).

Policy evaluation

The goal of the PM is to extend battery lifetime by reducing the system power consumption while maintaining an acceptable performance level. In this problem, the SP power value is known for each power state and the average delay is used as a performance metric. The average delay stands for the average number of waiting requests in the SQ. This is reasonable because as indicated in [9], the average number of requests in the SQ is proportional to the average latency per request that is defined by the average waiting time for each request to be processed. The average service time represents the time between the moment the request is generated and the moment the SP finishes processing it i.e., it includes queuing time plus execution time.

In this work, we use "cost rate" instead of "reward rate" in RL algorithms. The cost rate is a linearly weighted combination of the SoC degradation starting from the initial battery charge state, and the number of requests buffered in the SQ. The SoC degradation is directly related to the battery discharge current which is a function of the instantaneous SP power consumption that is obtained from its power state observation. In this way, the value function $Q(s,a)$ for each state-action pair (s,a) is a linear combination of the expected total discounted SoC degradation and latency. The relative weight between the SoC degradation and latency can be changed to get a SoC-latency tradeoff curve. More precisely, upon selection of action a from state s, the cost function $C(s,a;\mu)$ is defined as follows:

$$C(s,a;\mu) = \mu\,\theta\,(s,a) + (1-\mu)\,\tau\,(s,a) \tag{11}$$

where $\tau\,(s,a)$ represents the drawn charge from the battery; $\theta(s,a)$ denotes the caused delay and μ is a user-defined parameter enabling the SoC-delay tradeoff.

Battery output voltage threshold learning algorithm

A timeout policy is assumed to be the SP policy due to its wide usage in many devices. Under such policy, the SP moves into the sleep state if it remains idle for more than a specified timeout period. Fist, a system with a given pre-specified timeout policy is used. The optimal voltage threshold for a fixed timeout value is obtained by applying reinforcement learning. Let V_{th} denote the threshold value of the battery output voltage. In the RL technique, a list of V_{th} serves as the PM action set for controlling the SP switching between its two busy HP and busy LP states. The PM learns to choose the optimal action $a \in A$, which corresponds to the optimal V_{th} value, by using the 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. The PM continuously observes the following parameters (i) the SP power state: busy, idle and sleep, (ii) the SQ state: number of waiting requests, and (iii) the battery output voltage that is nonlinearly related to the SoC as represented in the battery model. Based on these observable parameters, the PM makes a decision and issues commands to the SP in 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. Depending on the battery output voltage, the PM decides either to turn the SP into the low performance (Busy LP) state or to keep it into the high performance (Busy HP) state. More precisely, if the battery output voltage falls below V_{th}, the SP is operated 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 drawn charge from the battery and (ii) the caused delay as explained above. Details of the proposed RL-based algorithm are provided in Algorithm 1.

ALGORITHM 1. The basic RL-Based DPM Algorithm.

Input: the timeout $\boldsymbol{T_{out}}$ value, the action set $\boldsymbol{A}$ (a set of V_{th} values), the battery output voltage $\boldsymbol{V_{out}}$.
Do for 1,2,3,..,80 (number of charge/discharge cycles)
Initialization : The battery is fully charged
Choose an action $\boldsymbol{a}$, which corresponds to a specific battery output voltage threshold, from the action set $\boldsymbol{A}$.
repeat
At each decision epoch $\boldsymbol{t_k}$:
Let the PM execute the timeout policy with timeout value $\boldsymbol{T_{out}}$.
if the SP is in the idle state **then**
if some request comes before the timeout period expires **then**
The PM turns the SP active for processing requests until the SP becomes idle again.
Then we have reached decision epoch $\boldsymbol{t_{k+1}}$.
else
The PM keeps SP idle for $\boldsymbol{T_{out}}$ period of time.
end
else if the SP is in the sleep state **then**
if some request comes **then**
The PM turns the SP active for processing requests until the SP becomes idle again.
Then we have reached decision epoch $\boldsymbol{t_{k+1}}$.
else
The PM keeps SP in the sleep state.
end
else (the SP is in the active state)
if $\boldsymbol{V_{out} < a}$ **then**
The PM turns the SP into the Low Performance state for processing requests.
Then we have reached decision epoch $\boldsymbol{t_{k+1}}$.
else
The PM turns the SP into the High Performance state for processing requests.
Then we have reached decision epoch $\boldsymbol{t_{k+1}}$.
end
end
until the battery is fully discharged
Evaluate the chosen action $\boldsymbol{a}$ using the RL technique (Equation 1).

Learning the optimal timeout policy

In addition to learning the optimal battery output voltage threshold, we propose to enhance the framework to automatically adjust the power management policy by learning the optimal timeout value under non-stationary workloads. Recall that the timeout policy is assumed to be the service provider policy due to its wide usage in many devices. The timeout determines the tradeoff between the service latency and power dissipation of the SP. The major difficulty is to accurately define the optimal timeout period. An undesirable situation is where the SP is put to sleep too fast, only to be awakened immediately. Thus, the system would suffer from extra energy consumption and latency of waking up the SP and bringing it to the active state.

On the other hand, if the timeout is set to be too long and meanwhile no service requests arrives, the SP has unnecessarily wasted energy by waiting in the idle state. Predicting the optimal timeout value under a given system performance constraint is a major concern for us in this case.

For learning the optimal timeout value, since the system functions in an event driven manner, continuous-time learning based on TD(λ) learning method for SMDP is used. The update rules for the SMDP TD(λ) algorithm are defined according to Eqs. (2) and (3). In this case, the PM simultaneously learns the optimal timeout policy and adapts the suitable output voltage threshold for that timeout policy.

Policy evaluation

To learning the optimal timeout value, we use the cost function $C'(s,a';\mu')$ which is a linearly weighted combination of (i) instantaneous power consumption and, (ii) the number of requests buffered in the SQ. The cost function is defined as follows:

$$C'(s,a';\mu') = \mu' delay\,(s,a') + \quad (1-\mu')\ p\,(s,a') \qquad (12)$$

where $p\,(s,a')$ is the service provider absorbed power for state s; $delay\,(s,a')$ is the delay and μ' is a user-defined parameter enabling the power-delay tradeoff.

Timeout value learning algorithm

For this aim, the RL algorithm uses two action sets: (i) a list of output voltage values that serve for the selection of the battery output voltage threshold, and (ii) a second list of timeout values to be used when the SP is idle. At a given decision epoch t_k , the SP is idle and SQ contains no requests, the PM chooses an action from the timeout action set (list of timeout values) and then executes the learning of the optimal battery output threshold voltage based on the chosen timeout value as described in the previous section. At the next decision epoch t_{k+1}, 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). The system evaluates the chosen action using the continuous-time TD(λ) algorithm for SMDP presented before. Details of the optimal timeout policy enhancement are shown in Algorithm 2.

Dual-battery Policy

Modern battery powered systems can accommodate two batteries or more, e.g. portable computers and mobile phones where batteries are generally used in sequence one after the other. The second battery starts operating only when the first battery is totally discharged. However, as it was discussed earlier, a battery can recover some amount of its deliverable capacity when it is given some rest after some period of current discharge. This is due to its electro-chemical characteristics [7] that can be fruitfully exploited in a dual-battery system by enabling the PM to alternate the use of the two batteries. In this way, one battery powers the system and the other can recover while it is temporarily disconnected from the load. An effective power management scheme must consider such a recovery effect to extend the battery lifetime.

In this context, a dual-battery system (B1 and B2) is considered where batteries are alternated in operating the system. As performed for a single battery system, RL technique is used to learn the optimal threshold voltages for both the two batteries (V_{th1} and V_{th2}) that guarantee the best tradeoffs between battery lifetime and system-performance. When the output voltages of the two batteries exceeds their corresponding optimal threshold voltages, the system operates in an HP mode and we perform the switching between the two batteries with a fixed frequency *f*. Using this policy, the lifetime of the system is expected to increase. However, this strongly depends on *f*. For very low *f*, each battery is drained for a long time with full current load. In the limiting case of $f = 0$, two batteries are discharged in sequence. In the Experiment section, a suitable procedure for locating the most suitable value of f is implemented. When the output voltage of the currently operated battery, e.g., B1's voltage falls below V_{th1} while B2's voltage exceeds V_{th2}, B1 is disconnected from the load and it is given some rest. In this case, no switching between the batteries is performed and the system is kept powered only B2 such that the SP continues to operate in an HP until B2 becomes exhausted.

When both batteries are exhausted, the PM turns the SP into the LP mode and the fixed switching policy between B1 and B2 is again performed until both are fully discharged.

ALGORITHM 2. The Enhanced RL-Based DPM Algorithm.

Input: the action set $\hat{A}$ (a set of T_{out} values), the action set A(a set of *Vth* values), the battery output voltage V_{out}.
Do for 1,2,3,..,80 (number of charge/discharge cycle)
Initialization : The battery is fully charged
Choose an action a, which corresponds to a specific battery output voltage threshold, from the action set A.
repeat
At each decision epoch t_k:
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 **then**
if some request comes before the timeout period (with duration of $\hat{a}$) expires **then**
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 **then**
if some request comes **then**
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 $V_{out} < a$ **then**
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 (Equation 2 and Equation 3).
until the battery is fully discharged
Evaluate the chosen action a using the RL technique (Equation 1).

V. Experimental Results

In this section, we present the effectiveness of the proposed DPM framework on extending the battery lifetime. The proposed algorithms are experimented on both single and dual-battery powered systems.

5.1. Experiments on a Single-Battery Powered System

The first set of experiments demonstrates the effectiveness of the basic DPM framework implemented with a given pre-specified timeout policy fixed to 0.2 T_{be} (T_{be} refers for the break-event time defined in [4]). We have conducted two experiments by assuming different Hard Disk drive (HDD) characteristics defined in Table II such as voltage; current and time of service values for both *HP* and *LP* states.

Table II. The HDD characteristics in two experiments [16].

	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			

At each state, the battery voltage and current are calculated. Assuming that 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 SoC. We perform different experiments under multiple battery's charging and discharging cycles. For the workload, we measure a real 6-hour server accessing trace using the *tcpdump* utility in a Linux server. Then, this real trace has been reproduced and utilized several times to simulate multiple charge/discharge cycles of a battery.

To assess the effectiveness of the basic DPM framework in improving the battery lifetime, various experiments have been performed by targeting different SoC-latency tradeoffs. The SoC degradation 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 SoC degradation and (ii) the average response time. A low weight for the SoC degradation implies a highly-constrained latency system. High SoC degradation weights are related to lowly-constrained latency systems. Figure 5 reports the obtained results for an HDD having different characteristics as defined in Table II (experiments 1 and 2 settings). These results are compared to what can be obtained if only a 0.2 T_{be} pre-specified timeout policy is used with a preset battery threshold voltage V_{th} = 3.7V.

First, as shown in Figure 5, the proposed DPM framework provides a wide range of SoC-latency tradeoffs for both experiments 1 and 2. Better performances are obtained for HDD with characteristics of experiment 1 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. As shown in Table II, the HDDs are characterized by the same busy LP mode current for both experiments 1 and 2. In this case, similar tradeoff curve behaviors are observed for increasing latency values for both experiments. In comparison with the fixed-timeout policy with 3.7 V preset threshold voltage, **15.7%** maximum battery SoC improvement is obtained for high latency value systems for the HDD with parameter settings of experiment 1.

This is shown in Figure 5, when only the fixed timeout policy with the fixed preset threshold voltage is used, the SoC degradation is about *11.93*% for a latency value equal to *4 %*. Using our framework, only *10.06 %* of battery SoC degradation is obtained for low performance constrained systems that tolerate up to *7.2* % latency.

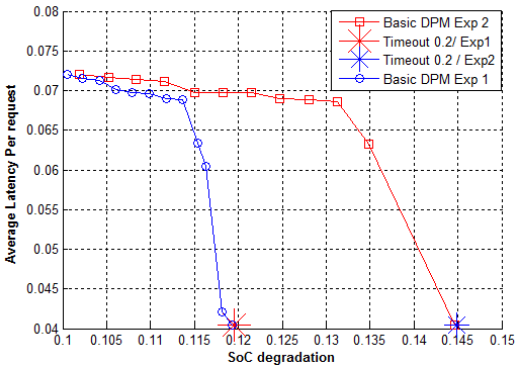

Fig. 5. SoC-latency trade off curves of the 0.2 Tbe fixed timeout policy running the RL-based DPM on HDD for both experiments 1 and 2

For experiment 2 settings, even better performances are achieved with up to **29.89%** of SoC saving obtained in comparison with the fixed-timeout policy with 3.7 V preset threshold voltage. In this case, similar SoC/latency tradeoff behavior is observed for high latencies constrained systems. However, when only the fixed timeout policy with the fixed preset threshold voltage is used, the SoC degradation with experiment 2 settings is *14.34 %* with *4 %* latency. This even outperforms the method in [8] by **35.86%** in terms of battery service lifetime extension.

Learning the optimal timeout

To evaluate the improvements of the enhanced RL-battery DPM framework with learning *the optimal timeout*, we run experiment 3 with the same workload as used for the HDD with experiment 1 settings. For this experiment, learning the timeout is activated and a second action set of timeout values (T_{out}), denoted by $\hat{A}$, is used as follows: $\hat{A}$ = {0.1Tbe, 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 also learns the optimal timeout value among the timeout action set values and then accordingly learns the best battery voltage threshold for the resulting timeout policy. The obtained results are given in Figure 6.

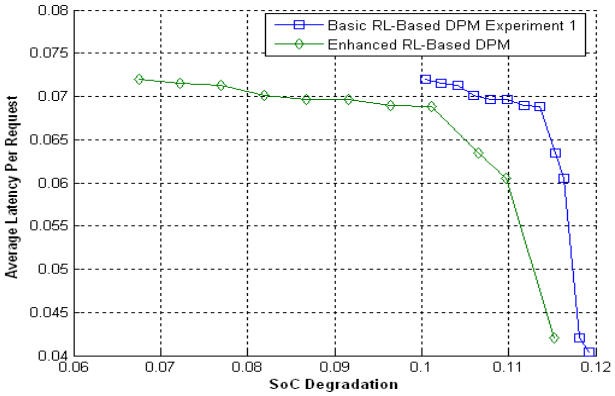

Fig.6. SoC-latency tradeoff curves of the RL-based DPM with and without learning timeout using HDD with experiment 1 parameter settings.

It is clear from Figure 6 that learning both timeout and voltage threshold further helps minimizing the average power consumption leading to more battery service lifetime extension and higher effectiveness of the proposed DPM framework. Indeed, in comparison with the basic RL framework using a pre-specified 0.2 T_{be} timeout value, the enhanced framework with timeout learning can achieve "wider and deeper" SoC-latency tradeoffs. In addition, the obtained results outperform the basic framework under the same latency values. The enhanced RL-based DPM with timeout learning can achieve lower battery SoC degradation in comparison with the basic framework particularly when the system has tight latency constraints.

For systems that can tolerate high latency values up to 7.2%, the battery SoC degradation is enhanced to *6.82* % compared to *10.06 %* previously obtained with the basic framework. In reference with the 0.2 T_{be} pre-specified timeout policy using 3.7V preset battery threshold V_{th} voltage, the enhanced framework is capable of achieving almost **42.83 %** of battery SoC savings.

5.2. Experiments using a Dual-Battery Powered System

HDD is used with parameters settings of experiment 1 and two identical batteries are considered with the same characteristics represented through the extracted simulation parameters given in Table I. The best-suitable frequency value is obtained by trial-and-error experiments to derive the appropriate switching between the two batteries. In the first set of experiment, we run multiple battery charging/discharging cycles and use the RL technique to learn the best battery threshold voltage among a list of voltage thresholds for both batteries with only a fixed timeout value system of 0.2 T_{be}(basic framework). The obtained results are compared to a reference 0.2 T_{be} pre-specified timeout policy system with a preset threshold voltage V_{th} equals to 3.7V, and using the same batteries with the same switching frequency value. In this case, the SoC degradation/latency tradeoff is obtained and shown by the red curve in Figure 7. When comparing the proposed DPM framework with the reference timeout system with a fixed V_{th} equals to 3.7V, we can see again the improvements achieved using the proposed approach especially for low-delay constrained system where up to **27.33%** of SoC saving can be achieved.

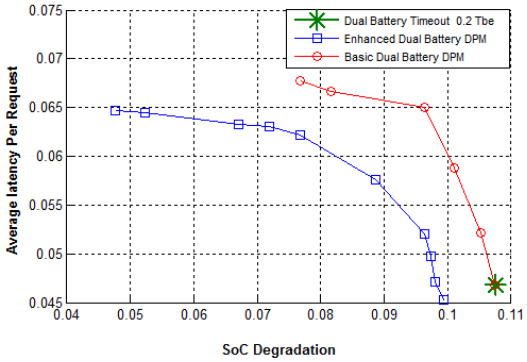

Fig.7. SoC-latency tradeoff curves of the Dual-Battery DPM with and without learning timeout using HDD with experiment 1 parameter settings compared to the reference timeout system.

In a second set of experiment represented by the blue solid curve in Figure 7, we use the reinforcement learning technique to define both optimal timeout value and threshold values while a fixed frequency switching between the two batteries is implemented. The outcome of this enhancement is clear as shown in Figure 7. In comparison with the basic dual battery DPM experiment (red dotted curve).the enhanced framework performs

better for both low and high latencies constrained system under the same performance level. For example, for a system with *6.5%* latency constraint, the SoC saving obtained with the enhanced dual battery with a timeout learning DPM framework is about **46.73%** higher than the dual battery basic DPM using the 0.2 T_{be} pre-specified timeout policy. Now, and in reference to 0.2 T_{be} pre-specified timeout policy using the 3.7V preset battery threshold V_{th} voltage, the dual battery DPM framework enhanced with timeout learning enables up to **55.38%** maximum battery SoC savings. This maximum value is calculated from Figure 7 by using the *10.67%* SoC degradation obtained when the fixed timeout policy with the fixed preset threshold voltage is used in comparison with the *4.76 %* minimum SoC degradation obtained for a system that tolerates up to *6.5%* latency.

VI. Conclusions

This paper presents an efficient reinforcement learning-based DPM framework that increases battery lifetime by accurately defining the best battery voltage threshold value and sets accordingly the system active mode. 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. The proposed DPM framework is model-free and requires no prior information of the workload characteristics. The experimental results provide strong evidences that reinforcement learning performs well for defining the optimal power management policy particularly for high latency constrained systems. For a single-battery system, the results implemented on an HDD device show an improvement in battery SoC savings up to **42.83%** in reference with a pre-specified timeout system using a given preset battery threshold voltage. The maximum battery SoC savings are further improved to **55.38%** when using our proposed dual-battery DPM framework enhanced with timeout learning values. In addition, wider and deeper SoC-delay tradeoffs are obtained with extended battery service lifetime in comparison with prior works.

References

[1] S.-H. Yi and S.-B. Cho, A battery-aware energy-efficient android phone with bayesian networks, In Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), pp. 204-209, Sept 2012.

[2] I. Konig, A. Q. Memon, and K. David, Energy consumption of the sensors of smartphones, In Wireless communication Systems (ISWCS 2013), pp. 1-5.VDE, 2013.

[3] W. Song, N. Sung, B.-G. Chun, and J. Kim, Reducing energy consumption of smartphones using user-perceived response time analysis, In Proceedings of the 15th Workshop on Mobile Computing Systems and Applications, HotMobile '14, pp 20:1-20:6, NY, USA, 2014.

[4] L. Benini, A. Bogliolo, and 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.

[5] D. Corujo, M. Lebre, D. Gomes, and R. L. Aguiar, Sensor context information for energy-efficient optimization of wireless procedures, In International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), pp. 1010-1014, 2011.

[6] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda,Characterizing and modeling user activity on smartphones: Summary, In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, pp. 375-376, NY,USA,2010.

[7] 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.

[8] 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.

[9] Q. Qiu, M. Pedram, Dynamic Power Management Based on Continuous-Time Markov Decision Processes, DAC, 1999.

[10] Y.Siyu, D. Zhu, Y. Wang, M. Pedram, Reinforcement learning based dynamic power management with a hybrid power supply, IEEE Computer Design (ICCD), Sept 2012, pp.81-86.

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

[12] A. Paul, Real-Time Power Management for Embedded M2M using Intelligent Learning Methods, ACM Transaction on Embedded Computing Systems, Article No:148, Vol.13, Issue 5s, July 2014.

[13] A. Paul, C.Bo-wei, J. Jeong, and J. Wang, Video Search and Indexing with Reinforcement Agent for Interactive Multimedia Services, ACM Transaction on Embedded Computing Systems, Vol.12, Issue 2, Article no:25, February 2013.

[14] A. Paul, C. Bo-Wei, J. Jeong, J. Wang, Dynamic power management for embedded ubiquitous systems, International Conference on Orange Technologies (ICOT), 2013, pp. 67 – 71.

[15] U.A. Khan, B. Rinner, A Reinforcement Learning Framework for Dynamic Power Management of a Portable, Multi-camera Traffic Monitoring System, Green Computing and Communications (GreenCom), 2012, pp. 557 – 564.

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

[17] Y. Tan, W. Liu, Q. Qiu, Adaptive Power Management Using Reinforcement Learning, ICCAD, Nov. 2009, pp. 461-467.

[18] Y. Wang, Q. Xie, A.C. Ammari, M. Pedram, Deriving a near-optimal power management policy using model-free reinforcement learning and Bayesian classification, DAC, Jun. 2011, pp. 875-878.

[19] 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.

[20] M. Triki, Y. Wang, A.C. Ammari, and M. Pedram, Reinforcement Learning-Based Dynamic Power Management of a Battery-Powered System Supplying Multiple Active Modes, IEEE European Modelling Symposium, 2013.

[21] 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.

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

[23] M. Triki, Y. Wang, A.C. Ammari, M. Pedram, Dynamic power management of a computer with self power-managed components, Lecture Notes in Computer Science (LCNS), Volume 7606, pp 215-224, 2013.

[24] M. Triki, Y. Wang, A. C. Ammari, and M. Pedram, 'Hierarchical power management of a system with autonomously power-managed components using reinforcement learning', Integration, the VLSI Journal, Volume 48, January 2015, Pages 10–20
[25] M. Doyle, J. Newman, "Analysis of capacity-rate data for lithium batteries using simplified models of the discharge process," Journal of Applied Electrochemistry, Vol. 27, Iss. 7, 846-856, 1997.
[26] T.F. Fuller, M. Doyle and J. Newman, "Relaxation phenomena in lithium-ion-insertion cells," Journal of Electrochemical Society, Vol. 141, No. 4, Apr. 1994.
[27] D.W. Dennis,V. S. Battaglia, and A.Belanger, "Electrochemical modeling of lithium polymer batteries," *J. Power Source*, vol. 110, no. 2, pp. 310– 320, Aug. 2002.
[28] M. Chen and G. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and I-V performance," IEEE T. on Energy Conversion, 2006.
[29] 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.
[30] B. Schweighofer, K. M. Raab, and G. Brasseur, "Modeling of high power automotive batteries by the use of an automated test system,"*IEEE Trans. Instrum. Meas.*, vol. 52, no. 4, pp. 1087–1091, Aug. 2003.