Hierarchical Power Management of a System with Autonomously Power-Managed Components Using Reinforcement Learning

M. Triki [a,*,1], Y. Wang [b], A. C. Ammari [a, c], M. Pedram [b]

[a] *Carthage University, MMA Laboratory, Institut National des Sciences Appliquées et de Technologie, Centre Urbain Nord, B.P. 676, Tunis, Cedex 1080, Tunisia*
[b] *Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA*
[c] *Department of Electrical & Computer Engineering, Faculty of Engineering, King Abdulaziz University, P.O. Box 21589, Jeddah 21589, Saudi Arabia*

ABSTRACT

This paper presents a hierarchical dynamic power management (DPM) framework based on reinforcement learning (RL) technique, which aims at power savings in a computer system with multiple I/O devices running a number of heterogeneous applications. The proposed framework interacts with the CPU scheduler to perform effective application-level scheduling, thereby enabling further power savings. Moreover, it considers non-stationary workloads and differentiates between the service request generation rates of various software application. The online adaptive DPM technique consists of two layers: component-level local power manager and system–level global power manager. The component-level PM policy is pre-specified and fixed whereas the system-level PM employs temporal difference learning on semi-Markov decision process as the model-free RL technique, and it is specifically optimized for a heterogeneous application pool. Experiments show that the proposed approach considerably enhances power savings while maintaining good performance levels. In comparison with other reference systems, the proposed RL-based DPM approach, further enhances power savings, performs well under various workloads, can simultaneously consider power and performance, and achieves wide and deep power-performance tradeoff curves. Experiments conducted with multiple service providers confirm that up to 63% maximum energy saving per service provider can be achieved.

***Keywords**:* power management, reinforcement learning, temporal difference learning, semi-Markov decision process.

1. Introduction

Power consumption in battery operated portable devices is nowadays a major concern. Such systems generally contain many I/O device components, ranging from digital and analog to electro-mechanical and electro-chemical. For these systems, the major energy dissipation is coming from these I/O devices. Dynamic power management (DPM) refers to a set of strategies that achieves efficient power consumption by the selective shut-off or slow-down of I/O components that are idle or underutilized [1]. Such technique has proven to be a particularly effective way of reducing power dissipation at the system level [2]. An effective DPM policy should minimize power consumption while maintaining performance degradation within an acceptable level. The DPM methods proposed in the literature can be broadly classified into three categories: heuristic, stochastic, and learning based methods.

Heuristic methods attempt to predict the length of the next idle time based on the computation history, and then shut the device down if the predicted idle period length justifies the cost. More precisely, a decision to sleep will be made if the prediction indicates that the idle period is longer than the break-even time T_{be}. Among these methods, Srivastava et al. [3] use a regression function to predict the idle period length, while Hwang et al. [4] propose an exponential-weighting average function for predicting the idle period length. Such techniques are simple and easy to implement, and have been adopted in many commercial products. However, they perform well only when the requests are highly correlated and do not take performance constraints into account, and thus, can hardly achieve a desirable trade-off between performance and energy dissipation. The stochastic approaches can take into account both power and performance and are able to derive provably optimal DPM policies, by modeling the request arrival times and device service times as stationary stochastic processes such as Markov Decision Processes (MDP) [5], [6], [7]. The essential shortcoming of these methods 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 for accurate prediction [8]. 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 [9] and must be able to interact with the environment to obtain information which can be processed to produce optimal policies.

In this sense, several previous works use machine learning for adaptive policy optimization. Compared to heuristic policies, 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*. In [10], an online policy selection algorithm is proposed, which generates offline a set of DPM policies to choose from. Tan et al. in [11] propose to use an enhanced Q-learning algorithm for system-level DPM. The Q-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. However, this work is based on a discrete-

[1] * Corresponding author. Tel.: +216 98923252.
E-mail addresses: maryam.triki@gmail.com (M. Triki).

time model of the stochastic process, and thus has large decision making overhead. Wang et al. in [12] extend this work to allow the power manager working in a continuous-time and event-driven manner with faster convergence rate, by exploiting the TD(λ) learning framework for semi-MDP (SMDP) [13].

All of the above-mentioned DPM researches have focused on developing local component-level policies. However, a number of built-in power-management techniques have been already incorporated into various standards and protocols. These techniques cannot be modified because they ensure the correct functionality of the device running the corresponding protocol. In this sense, we consider such a device as an autonomously power-managed component. Even beyond protocol considerations, vendors may already have their appropriate power management methods specifically designed for their products and directly integrated into device drivers.

Based on the above considerations, we define the problem of hierarchical power management (HPM) for an energy managed computer **(**EMC) system with autonomously power-managed components. A few research results have been reported to resolve the HPM problem of a computer with self power-managed components. Reference [14] uses a similar system setup as this paper and derives offline the optimal policies using Continuous-Time Markov Decision process model of the system. In this approach, the request inter-arrival times and system service times are modelled as stationary processes that satisfy exponential probability distributions, which is not always realistic. Reference [15] proposes a hierarchical adaptive DPM, where the term "hierarchical" refers to the manner in which the authors formulate the DPM policy optimization as a problem of seeking an optimal rule that switches policies among a set of pre-computed ones.

In this paper, we develop a novel online adaptive approach for HPM of a computer with autonomously power-managed components. The proposed framework extends the basic HPM policy presented in [16] to multiple I/O devices and heterogeneous applications. The proposed approach consists of two layers: local component-level power manager (LPM) and global system–level power manager (GPM). The LPM policy is pre-specified and fixed whereas, the GPM uses temporal difference learning on SMDP as the model-free reinforcement learning technique to perform power management in a continuous-time and event-driven manner. Moreover, the GPM interacts with the CPU scheduler to perform effective application-level scheduling, thereby enabling even more power savings. This work also extends the work of [14] by considering non-stationary workloads. The proposed RL approach does not assume the request inter-arrival times and system service times are modelled as stationary processes with exponential probability distributions. Thus, the power manager learns the optimal policy under non-stationary workloads associated with multiple application types with different service request generation rates. Moreover, the proposed approach simultaneously learns the optimal policy for non-stationary workloads and uses that policy to control instead of only evaluating one predefined policy. In addition, the proposed framework performs precise power-latency tradeoff of each application type based on a user-defined parameter, whereas reference [14] sets out to meet given performance constraints. The proposed framework is model-free, performs learning and power management in a continuous-time and event-driven manner, has fast convergence rate and less reliance on the Markovian property.

The remainder of this paper is organized as follows. We provide an overview of reinforcement learning background in Section 2. The proposed hierarchical DPM framework architecture is presented in Section 3, and its implementation details are given in Section 4. Framework improvements targeting multiple I/O devices and multiple heterogeneous applications are explained in Section 5. The experimental results and analysis are presented in Section 6. We conclude our findings in Section 7.

2. Theoretical background

In this section, we provide a brief introduction of the general RL framework and the RL algorithm proposed for the GPM, namely, the TD (λ) learning algorithm for SMDP. One may refer to [13] for more details.

Reinforcement Learning is a machine intelligence approach based on the learning through experience accumulation [17]. The core of the RL technique is the interaction between agent and environment in terms of states, actions, and rewards [18]. 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 any sensory information the agent receives about. Actions refer to the decision the agent will be called to make [19]. State represents the situation the agent can find itself in. More precisely, state is the available information about the agent's environment that helps in decision making.

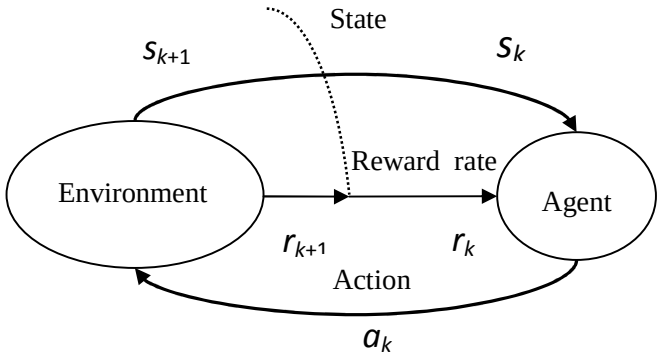

Fig. 1. Agent-environment interaction model.

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. 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 taken action, the agent moves 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 received rewards. More precisely, the agent keeps a value function $Q^{\pi}(s,a)$, for each state-action pair (s,a) initially chosen by the designer and later, it is updated each time an action is taken and a reward is received, based on the following equation.

$$\forall (s,a) \in S \times A:$$
$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha \times [r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)] \quad (1)$$

In the above expression, $\alpha \in (0,1)$ 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.

The value function represents the *expected long-term reward* when starting from state s, choosing action a (according to the policy π), and following π thereafter. Thus, the agent continuously adjusts its policy so as to maximize the total amount of reward received over the long run. For a realistic DPM problem, the power manager has no predefined policy or knowledge about state transition characteristics that are essential for any stochastic DPM approach. Therefore, the power manager has to simultaneously learn the optimal policy, and use it to make decisions. In this case, traditional value methods cannot be applied. Instead, TD learning methods [20] for SMDP[2] may be used. Such a method 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 [19]. 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 evaluating methods. We choose to use the algorithm for SMDP [20] 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:

$$\forall (s,a) \in S \times A:\ Q^{(k+1)}(s,a) = Q^{(k)}(s,a) + \alpha \times e^{(k)}(s,a) \times \begin{bmatrix} \frac{1-e^{-\beta\tau_k}}{\beta} r(s_k,a_k) \\ + \text{MAX}_a,\ e^{-\beta\tau_k} Q^{(k)}(s_{k+1},a') - Q^{(k)}(s_k,a_k) \end{bmatrix} \quad (2)$$

In the above expression, $\tau_k = t_{k+1} - t_k$ is the time 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 Eqn. (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:

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

where $\delta((s,a),(s_k,a_k))$ denotes the delta kronecker function.

3. Hierarchical DPM framework

This paper focuses on reducing energy consumption of a uni-processor computer system with built-in component-level local power managers. Basically, we propose an adaptive approach that regulates the requests service flow of the autonomously power-managed components, thereby effectively reducing component energy consumption.

Timeout[3] policies are assumed to be the local power management policies for the components given their wide usage. Under these policies, a Local Power Manager (LPM) directly controls the state transitions of the device. Meanwhile, the system-level Global Power Manager (GPM) helps the LPM improve the power efficiency by performing service flow regulation and application scheduling.

3.1 Service provider model

One of the key components of the DPM framework is the device providing services to the workload called service provider (SP).

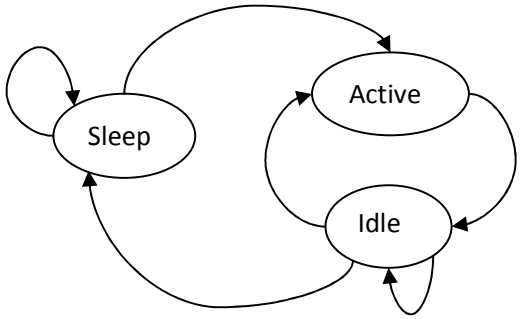

Fig. 2. Service provider model.

In this work, we consider a SP as shown in Fig. 2. It has three main states explained below:

- *Active* state: the SP is fully functional and it is processing some 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

[2] Note that the temporal difference learning for SMDP is named because it can be proved to converge to the optimal policy if the agent-environment interaction system evolves as a stationary SMDP. In fact such learning algorithm is robust and has less reliance on the Markovian assumption [19].

[3] Timeout is the duration the SP is kept in its idle state before entering the sleep state.

from the idle state. The device turns into active as soon as a service request arrives.

3.2 The global system architecture

The architecture of the proposed hierarchical DPM framework is presented in Fig. 3. The framework contains two service providers, e.g. two I/O devices.

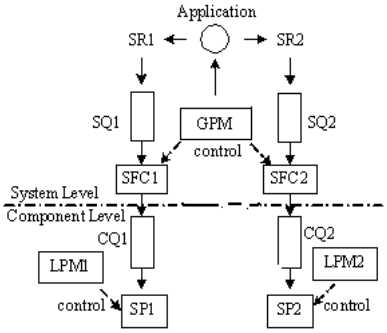

Fig. 3. Block diagram of a hierarchical DPM structure [14].

The service requesters (SR1, SR2) generate requests to be processed by two types of SP, respectively. The requests are buffered in the service queues (SQ1, SQ2) before being processed.

The proposed architecture decomposes the power management task into two layers: local component-level power manager (LPM) and global system-level power manager (GPM). Each service provider is controlled by a LPM whose timeout policy is pre-specified and cannot be changed. The LPM is monitoring the number of waiting service requests in the component queue (CQ) and consequently adjusts the state of the service provider.

3.3 The global power manager

At the system-level, the GPM cannot overwrite the LPM policy or directly control the state transition of a service provider. Thus, a Service Flow Controller (SFC) is incorporated to control the service request traffic that reaches the SP. The GPM monitors the SQ and employs TD(λ) to guide accordingly SFC actions and interact with the CPU scheduler to perform effective application-level scheduling for further reducing the power consumption.

The main functions of the SFC are detailed as follows:

Fake SR generation: In the case of expecting high activity in the near future, the SFC generates a fake service request[4] in order to wake up the SP and prevent it from entering a deep sleep state.

SR blocking: To reduce the wake-up frequency of the SP and extend its sleep time, the SFC blocks all incoming requests from entering its component queue. Thus, the blocked requests remain in the SQ.

SR transfer: To wake up the SP, the SFC continuously moves the stored requests in the SQ to its component queue.

3.4 Application model

Different applications may be executed on the energy-managed computer system. These applications can however be classified into different types based on their device usage correlations and *workload characteristics*, i.e., their SR generation rates and the target SPs. A pre-characterization technique is used to obtain statistical information about each application program. Applications with the same device dependency list and similar request generation rate are grouped into one application type. Notice that applications which are hard to characterize separately may be grouped into a special class where the request generation rate of that class is equal to the average rate for all the applications belonging to this type.

For example, as shown in Fig. 4, application type 1 generates SR1 with a rate of $\lambda_{1a}^{(1)}$ and SR2 with a rate of $\lambda_{1a}^{(2)}$ in state r_{1a}. Similarly, in state r_{1b}, the generation rates for these two SRs become $\lambda_{1b}^{(1)}$ and $\lambda_{1b}^{(2)}$, respectively. In state r_{1a}, application type 1 transits to state r_{1b} with an average rate of $\nu_{1,ab}$, which also implies that the average time for application type 1 to stay in state r_{1a} is $\frac{1}{\nu_{1,ab}}$.

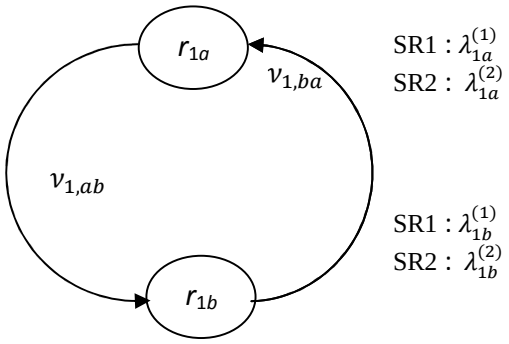

Fig. 4. Model of application type transitions.

In the proposed framework, the exact generating time instances of service requests are considered to be non-stationary, none of the applications is considered to be critical, and therefore, all the heterogeneous applications have the same priority. Moreover, we assume that a resource can handle one request at a time, and a request can be executed by only one device.

4. Reinforcement learning for hierarchical DPM

In this section, the system architecture is limited to a single application type executed on the energy-managed computer. The SR workload characteristics are non-stationary and the SP is controlled by a pre-specified LPM policy. The GPM learns the optimal policy under non-stationary workloads and issues commands accordingly to the SFC to determine whether to block or forward the service requests to the local component queue or to generate a fake request to keep the SP awake.

4.1 Policy learning

The goal of the GPM is to reduce power consumption while maintaining an acceptable performance level. In this study, we assume the SP power model known for each power state and we consider the average delay as the performance metric. The average number of waiting requests in the SQ is

[4] Fake Service Request is handled in the same way as a regular SR by the SP, but requires no service from the SP.

used as an estimation of the average delay. This is reasonable because as outlined in reference [6], the average number of requests in the SQ is proportional to the average latency for each request defined as the average waiting time for each request to be processed. The average waiting 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, the cost of a taken action in the RL algorithm is measured by both the instantaneous power consumption and the caused number of requests buffered in the SQ. The instantaneous power consumption is estimated by the SP model based on 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 energy consumption and latency. Since the total execution time and number of requests are fixed, the value function is equivalent to a linear combination of the average discounted power consumption and average latency per request. The relative weight between power and latency can be changed to get a power-latency tradeoff curve. More precisely, upon selection of action a from a state s, the cost function $Cost(s,a;\mu)$ is defined as follows:

$$Cost(s,a;\mu) = \mu \times delay\,(s,a) + (1-\mu) \times pwr\,(s,a) \qquad (12)$$

where $pwr\,(s,a)$ is the consumed power amount; $delay(s,a)$denotes the caused delay and μ is a user defined parameter enabling for setting a power-delay tradeoff.

4.2 The local power management policy (LPM)

Timeout policy is assumed to be the pre-specified LPM policy due to its wide usage in many built-in power-management policies and has been already incorporated into various standards and protocols. Using such policy, the SP moves into the sleep state if it remains in its idle state for more than a specified timeout period. The timeout determines the tradeoff between the service latency and power dissipation of the SP. An undesirable degenerate 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.

4.3 The global power management policy (GPM)

The GPM continuously observes the following parameters (i) the running application type, (ii) the SP power state: busy, idle, sleep, etc, (iii) the SQ state: number of waiting requests, and (iiii) the current application-specific service request generation rate: high, low, medium, etc. Based on these observable parameters the GPM makes decision and issues commands to the SFC in the following four cases:

1. The SP is in the sleep state and the number of requests in the SQ is less than a given threshold value.
2. The SP is in the sleep state and the number of requests in the SQ is equal to the given threshold value.
3. The SP is in the idle state and the timeout has not expired yet.
4. The SP is in the idle state and the timeout has expired.

Let N denote the threshold value of the number of requests in the SQ. A list of N values will serve as the action set A for the GPM in controlling the SFC. The GPM learns to choose the optimal action $a \in A$, which corresponds to the optimal N value, by using an RL technique.

The proposed RL framework operates as follows. At each decision epoch, the GPM finds itself in one of the four aforesaid conditions; it will issue commands to the SFC to implement the decision according to the following four cases:

1. The SP is in the sleep state and SQ contains less than N requests. In this case, the GPM decides to keep the SP in the sleep state and issues '*SR blocking'* commands to SFC to block all the incoming requests from entering the component queue CQ.
2. The SP is in the sleep state and SQ contains N requests. In this case, the GPM decides to turn on the SP for processing requests. It will issue '*SR transfer'* command to SFC *to transfer* SRs from the SQ to the CQ, and the SP will wake up to service the service requests.
3. The SP is in the idle state and the timeout has not yet expired:
   a. If some request comes during that period of time. The GPM will issue '*SR transfer'* command to SFC to transfer the incoming SRs from the SQ to the CQ so that the SP goes to the active state for processing requests according to the LPM policy.
   b. If (i) the timeout is about to expire and (ii) the GPM predicts a group of N requests to come within ε amount of time after the timeout expires, the GPM will decide to prevent the SP from entering the sleep state. More precisely, the SFC generates a '*Fake SR'* and the SP is kept in the idle state. ε is a small extra-time to extend the LPM pre-specified timeout value.
4. The SP is in the idle state and the timeout has expired, then the SP goes to the sleep state according to its LPM policy.

Details of the proposed RL-based GPM algorithm is provided in algorithm 1 in Appendix.

4.4 Multiple power-latency tradeoff curves with different ε values

As stated above, when (i) the SP is in the idle state and the timeout is about to expire and (ii) if the GPM predicts high activity within ε amount of time after the timeout expires, '*Fake SR*' is generated to prevent the SP from entering the sleep state according to the proposed policy. Actually, such action makes the SP wait for requests till timeout + ε. This is somewhat equivalent to establishing a new timeout value equal to the sum of ε and the original pre-specified timeout in the LPM. For a fixed ε value, the relative weight between power and latency can be changed to get a power-latency tradeoff curve. With different ε values, multiple power-latency tradeoff curves are obtained. The best-suited ε value is obtained by trial-and-error experiments to derive the desirable power-latency tradeoff curve.

4.5 Framework enhancement with learning the optimal ε

The use of multiple ε values enables different power-latency tradeoff curves. However, instead of manually varying ε to get the desired value, the GPM policy can be enhanced to make it automatically learn the best ε value for a given tradeoff between power and latency. The RL algorithm has then two action sets: (i) a list of N values that serves as the threshold value of the number of requests in the SQ and (ii) second list of ε values. In that case, the GPM learns the best threshold number of requests to turn on the SP from the sleep to the active state and also the best ε value to prevent within it the SP from entering the sleep state. Both N and ε learned values are obtained for a fixed power-latency tradeoff setting.

A list of N values will still serve as the action set $\tilde{A}$. In addition, a list of ε values will serve as a second action set $\ddot{A}$. The GPM learns to choose the optimal $\tilde{a} = N$ *and* $\ddot{a} = \varepsilon$ values among the action sets $\tilde{A}$ and $\ddot{A}$. In Appendix, Algorithm 2 provides the details of the modified RL learning algorithm with a learning ε.

5. Framework enhancement for multi-type application

This section presents improvements of the previous HPM framework with handling heterogeneous types of user applications. The GPM has to interact with the CPU scheduler to perform effective application-level scheduling in addition to SFC controlling, thereby, enabling the LPM to further component power optimization.

For this work, we are considering the CPU scheduling only for non critical applications. Thus, all the applications are running with the same priority. This assumption is particularly valid for our target of interactive battery operated portable devices for which an acceptable average system performance is generally required for a reduced power consumption

We start this section by introducing the fairness issue CPU scheduler, and then we demonstrate how application scheduling can further reduce the system performance. Implementation details of the enhanced framework are also presented.

5.1 CPU scheduling considering fairness issue

For a multi-type application framework, a system scheduler is needed for distributing the CPU among the different software applications. The CPU system scheduler changes from Operating System (OS) to OS. For our case, we are targeting non critical applications and we are considering a CPU system scheduler that uses a fairness issue to distribute execution times among various software applications. We use **APPL** = {1, 2,…, N} to denote the various application pool. For the i^{th} ($i \in$ **APPL**) type of application, we use CPU_{Time} to denote the total CPU occupying time of such an application type starting from time t_0 (i.e., the system starting time) until the current time t. The fairness constraint states that each i^{th} ($i \in$ **APPL**) application type cannot, on average, occupy more than C_i percentage of the total CPU execution time.

In Appendix, Algorithm 3 presents the used system CPU scheduling algorithm considering the fairness issue among different types of applications.

Application scheduling effectiveness

To point out the potential effectiveness of performing application type scheduling as part of the GPM by the following motivation example. Let us consider a system in which there are two types of applications A1 and A2, generating service requests at rates of one request per time unit and two requests per time unit, respectively. The SP wakes up as soon as a request is generated and sleeps when all requests have been serviced.

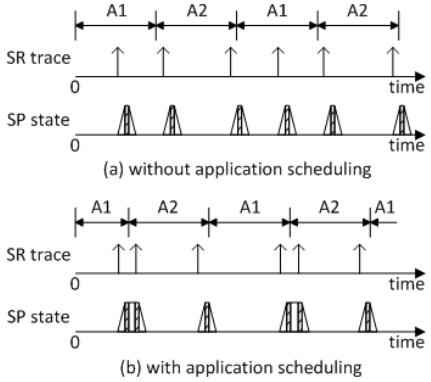

Fig. 5. Example of effectiveness of application level scheduling [14].

Two execution sequences have been considered as shown in Fig. 5. In the first sequence, there is no application scheduling implemented as a part of the GPM (Fig 5.a). Each application is alternatively executed for exactly one unit of time. In the second sequence, the GPM performs application scheduling as soon as a request of application type A1 is generated, switching the system to application type A2. We switch back to A1 when the second request of application type A2 has been generated and serviced (Fig 5.b). In this case service requests targeting one SP are "grouped together" ensuring that the component sleeping time is maximized. Assuming fixed wakeup and sleep transition times and energy dissipation values, both the power efficiency and performance are improved [14]. This motivation example clearly shows that a well designed GPM application scheduling can maximize the SP sleep time, and may help further reducing the total energy consumption.

5.2 The enhanced HPM considering application scheduling as part of the GPM

For a multi-type application framework running on the top of the system scheduler, the GPM is enhanced with application scheduling for some particular situations to improve the fairness and reduce the SP on/off switches, thereby, enabling more component power optimizations and further reduction of the total power consumption. The application scheduling is based on the global system state, i.e., states of the SP and the SQ. The use of global system state instead of the device state makes scheduling decisions based on a more precise view of the system and enables the consideration of any correlation between applications and devices. Basically, we implemented multiple run queues each one associated with different application type. Every application run queue is fed into the appropriate service provider queue.

Let N denote the threshold number of waiting requests in the SQ. The particular situation cases where the GPM decides to force for an application scheduling are given as follows:

a) The SP is in the sleep state and the current application run queue contains less than N requests. In this case, and if another application run queue contains the N waiting requests, the PM makes application switching, therefore the SP is waked up to process the requests. Application scheduling is beneficial for maintaining a good performance level in this case.

b) The SP is in the idle state and the run queue of the running application contains no requests before the timeout expires. The PM may decide to switch to another application if it predicts the arrival of N requests in its run queue in the near future.

In Appendix, Algorithm 4 provides the details of the enhanced RL learning algorithm with application-type scheduling.

6. Experimental results

In this section, we present the experimental results of the proposed HPM framework with real traces measured by using the *tcpdump* utility in Linux. First, we conduct a set of experiments with a single SP, i.e., a Wireless Local Area Network adapter card (WLAN). Then, we consider the case of a system with two SPs: a hard disk drive (HDD) and a WLAN card.

6.1 Experimental results on a single service provider

The SP used in the first set of experiments is a WLAN card. Table 1 lists its power consumption and switching time. T_{tr} is the time taken in transitioning to and from the sleep state while E_{tr} is the energy consumed for waking up the device. T_{be} refers to the break-even time.

TABLE 1
Power and delay characteristics of the considered WLAN card [12].

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.

We set the timeout value for the LPM to 0.3 T_{be}. Under this policy, the SP starts in the "idle" state. If no service request comes during the timeout period, the SP enters its "sleep" state. The action set of the GPM corresponds to the possible threshold values of waiting requests in the SQ and is defined by {1, 2, 3, 4 and 5}.

To assess the effectiveness of the basic HPM framework without considering neither ε-learning nor application-level scheduling, we started with the evaluation of its outcome in reference to only using the fixed LPM timeout policy. Simulations are based on a single type of application with a service request trace given by 6-hour combined web surfing, online chatting and server accessing trace. We perform various simulations using different ε and power-latency weight values. More precisely, we evaluate three HPM policies with fixed ε value set to 0.1 T_{be}, 0.6 T_{be} and 0.9 T_{be} respectively. For each aforesaid ε value, we vary the relative weight between power and latency to obtain a new power-latency tradeoff. A low weight assigned for power, i.e., 0.01 denotes high-constrained latency system. High power weights are related to low-constrained latency systems.

Experimental details and results are presented in Table 2. We in this table report (i) the average SP power consumption (Watt), (ii) the average latency per request (seconds) and (iii) the power savings percentage in comparison with the fixed-timeout policy under the same latency value. One can see that the basic HPM framework can achieve a wide range of power-latency tradeoffs. The shorter average latency we have, the higher power consumption is obtained for all used ε values. However, when a reduced power consumption is a priority, a higher tradeoff weight is set for power, i.e., 0.99 and as it is presented in table 2, the lowest average power values are obtained for the different ε values. In comparison with the pre-fixed LPM timeout policy for the same LPM latency value, the proposed framework achieves up to 24.25% power savings For low-constrained latency systems, we can achieve up to 72% maximum power savings.

These results outperform largely reference [14] where only 25 % of maximum power savings has been obtained. In addition, the proposed framework performs power-latency tradeoff based on a user-defined parameter, whereas reference [14] sets out to meet given performance constraints.

TABLE 2
Simulation results of the basic HPM algorithm and baseline policies on the WLAN card.

	Weight	HDD	WLAN card
Power	0.01	1.6122	1.2831
Latency	0.99	0.8117	1.4923
Power	0.83	1.3051	0.9863
Latency	0.17	1.2660	1.7891
Power	0.99	0.9863	0.7108
Latency	0.01	2.9260	3.2608
Power saving compared with the LPM (with the same latencies)		12.42%	13.61%
Maximum Power saving		58.67%	63.17%

To better clarify the outcome of the HPM framework, we represented in Figure 6 the power- latency tradeoff curves using: (i) the pre-specified timeout policy and (ii) the basic framework with two different ε values (0.1 T_{be} and 0.6 T_{be}). Using these curves, one can see that similar component power consumption are obtained at high latency values. However, under lower latencies and high ε values, the HPM framework further minimizes the power consumption in reference to simple LPM timeout.

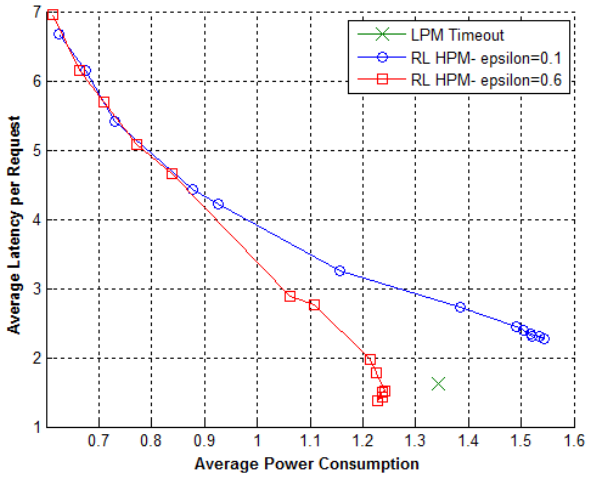

Fig.6. Power-latency tradeoffs curves of the fixed timeout policy and the basic framework on the WLAN card with $\varepsilon = 0.1\ T_{be}$ and $\varepsilon = 0.6\ T_{be}$

To demonstrate the effectiveness of the enhanced RL-based HPM algorithm with ε-learning compared with the basic HPM algorithm, we perform a second experiment and provide the experimental results in Figure 7. We use the same WLAN card and run the same application type. Fig. 7 shows the power-latency tradeoff curves achieved by (i) the enhanced framework with ε-learning and (ii) the basic framework for ε equal to 0.1 T_{be} and ε equal to 0.6 T_{be} values. We can see that the enhanced RL-based HPM algorithm with ε-learning can achieve much lower power consumption than the basic RL-based HPM algorithm particularly when the system has tight latency constraints.

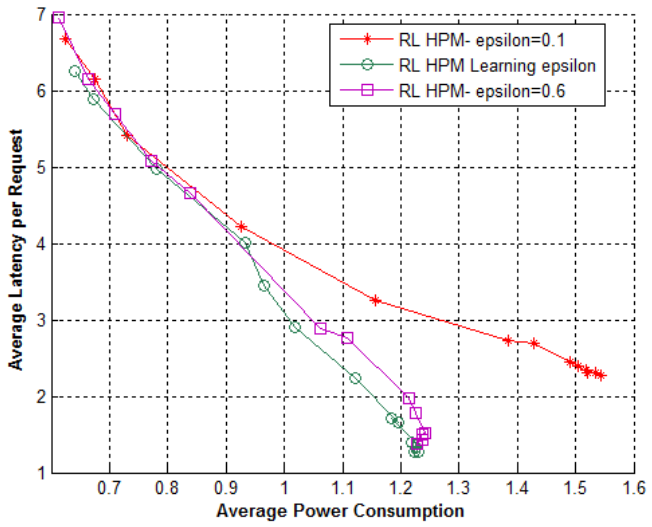

Fig. 7. Effectiveness of the enhanced framework with ε-learning.

For a third set of simulations, we consider the same service provider and we run two types of applications with two different service request traces. We use for the first application type the combined trace of web surfing, online chatting and server accessing and for the second application a 4-hour duplicated web surfing trace. First, we run the RL-based HPM algorithm with ε-learning, using the basic fair scheduler to distribute execution times among the two applications types. Then, we run the enhanced RL-based HPM algorithm with application-type scheduling. The experimental results are given in Figure 8. This figure illustrates the power-latency trade-off curves compared with the performance of the fixed-timeout LPM policy. It is clearly shown that the enhanced RL-based HPM algorithm with application-type scheduling can even further enhance performance in a multi-type application environment. For example, the improved HPM framework with application scheduling can further minimize the power consumption by up to 13.4% under the same latency value.

Moreover, one can clearly observe that the proposed HPM framework achieves further power consumption reduction in comparison with reference [14] in addition to allowing for the power and latency trading- offs.

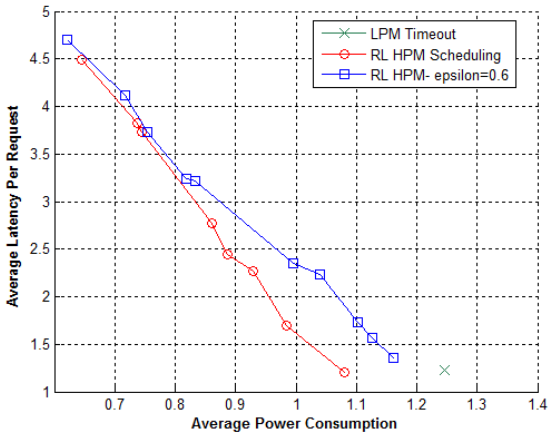

Fig. 8. Power-latency tradeoffs curves of the enhanced framework with application type scheduling.

6.2 Experimental results on a system with two service providers

In a second set of experiments, we considered an EMC system with two service providers: a hard disk drive (HDD) and the previously used wireless WLAN card. The power and delay characteristics of the HDD are given in Table 3. Non-stationary service request workloads are applied to all these service providers.

TABLE 3

Power and delay characteristics of the hard disk drive [12].

P_{sleep}	P_{busy}	P_{idle}	E_{tr}	T_{tr}
0.13 W	2.15 W	0.9 W	7.0 J	1.6 s

We run two types of applications, with two different service request traces. We use for the first application type the combined trace of web surfing, online chatting and server accessing. This first application type generates service requests for the WLAN service provider. For the HDD, a second 4-hour duplicated server accessing workload is used. We run the enhanced RL-based HPM with application-type

scheduling. The obtained results are presented in Table 4. In this table, the obtained average power consumption and average latency are listed for each service provider.

TABLE 4
Simulation results of the RL-HPM on the WLAN card and the HDD.

	Weight	HDD	WLAN card
Power values	0.01	1.6122	1.2831
Latency values	0.99	0.8117	1.4923
Power values	0.83	1.3051	0.9863
Latency values	0.17	1.2660	1.7891
Power values	0.99	0.9863	0.7108
Latency values	0.01	2.9260	3.2608
Power saving under LPM same latencies		12.42%	13.61%
Maximum Power saving		58.67%	63.17%

Results in Table 4 confirm that, similar to the case of a single SP, the Enhanced RL-HPM with application-type Scheduling achieves considerable energy savings when applied for different SPs running multiple applications. In comparison to the fixed timeout-based policy, the developed RL framework can minimize power consumption by almost 13% per service provider when using the same latencies obtained using fixed timeout-based policies. For low constrained latency systems, the maximum of energy savings attain 63% per service provider with reference to the fixed timeout-based LPM policy. This considerably improves the total system energy consumption while maintaining acceptable performance levels.

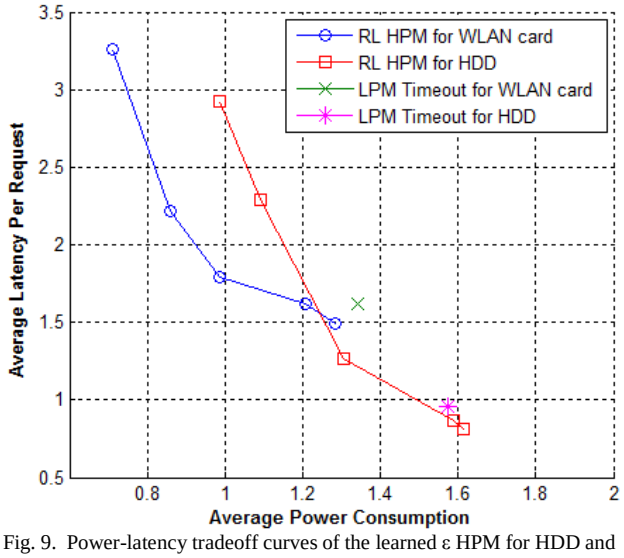

Fig. 9. Power-latency tradeoff curves of the learned ε HPM for HDD and WLAN card.

The power-latency trade-off curves obtained for both the HDD and the WLAN card are shown in Fig 9. This figure confirms the results previously obtained for a single service provider. We can see from this figure that the Enhanced RL-HPM with application-type Scheduling can considerably enhance performance for a system with multi-SP and multi-type applications. Finally, in addition to achieving good power and latency tradeoffs, the proposed HPM framework achieves, for the same latencies values, considerable reduction of power consumption with comparison to the fixed timeout-based LPM policy.

7. Conclusions

In this paper, a novel online adaptive RL-based HPM framework is proposed for an energy managed computer system with autonomously power-managed components. The proposed framework decomposes the power management task into two layers: component-level LPM and system-level GPM. Timeout policy is assumed as the predefined policy for the LPM. The GPM directly controls the service request traffic that reaches the power-managed components through an SFC. The GPM performs power management in a continuous-time and event-driven manner using temporal difference learning on SMDP for model-free RL. We choose to use the TD(λ) algorithm for SMDP due to a joint consideration of effectiveness, robustness and convergence rate.

Experiments show that the proposed HPM approach considerably enhances power savings while maintaining a good performance level. In comparison with other reference systems, the proposed RL-based HPM approach performs well under various workloads, can simultaneously consider power and performance, and achieves wide and deep power-performance tradeoff curves.

For a multi-type application framework, the GPM is enhanced to interact with the CPU scheduler to perform effective application-level scheduling, thereby enabling even more component power savings. We considered application scheduling only for non-critical applications and we use a scheduler that allocates execution times fairly among all types of applications. In comparison to the 'Fairness Issue' based scheduler, it is shown that the improved HPM framework with GPM scheduling can even further enhance performance in a multi-type application environment. The improved HPM framework can further reduce up to 13.4% of power consumption without latency increase. Apart from the 'fairness Issue' based CPU scheduler, the Enhanced RL-HPM with application-type Scheduling can optimally adapt to any system scheduler for non-critical applications.

Experiments conducted on an EMC system with multiple service providers confirm that the proposed HPM framework performs as well as the EMC system with one service provider under various workloads. The maximum achievable energy saving can reach 63% per service provider. This will considerably improve the overall system energy efficiency while maintaining acceptable performance levels.

Appendix

Abbreviations	
Dynamic Power Management	DPM
Reinforcement Learning	RL
Hierarchical Power Management	HPM
Energy Managed Component	EMC
Local Power Manager	LPM
Global Power Manager	GPM
Service Provider	SP
Service Queue	SQ
Component Queue	CQ
Service Flow Control	SFC
Application Pool	APPL
Operating System	OS

Algorithm 1: The RL-Based GPM Algorithm.

Input: the timeout value T_{out} in the LPM, the action set A (a set of N values), the parameter ε.

At each decision epoch t_k:

- Choose an action a, which corresponds to a specific threshold number of waiting requests in the SQ, from the action set A.
- Let the LPM execute the timeout policy with timeout value T_{out}.
- **If** the SP is in the idle state:
  - **If** some request comes before the timeout period (with duration of T_{out}) expires:
    - The LPM turns the SP active for processing requests until the SP becomes idle again. Then we have reached decision epoch t_{k+1}.
  - **Else**
    - The LPM keeps SP idle for T_{out} period of time.
    - **If** the GPM predicts the coming of a requests within ε amount of time after timeout expires:
      - The SFC generates a fake request in order to prevent the SP from entering the sleep state.
    - **Else**
      - The LPM turns the SP into the sleep state. Then we reach decision epoch t_{k+1}.
- **Else if** the SP is in the sleep state:
  - **If** SQ contains less than a requests:
    - Block all incoming requests using the SFC so that the SP keeps in the sleep state until the SQ accumulates a requests.
    - Then we reach decision epoch t_{k+1}.
  - **Else**
    - Transfer the requests buffered in the SQ to CQ so that the SP will turn on to process requests until it becomes idle again.
    - Then we reach decision epoch t_{k+1}.
- Evaluate the chosen action a using the TD(λ) technique.

Algorithm 2: Modified RL based GPM with learning ε

Input: the timeout value T_{out} in the LPM, the action set A (a set of N values), the action set $\hat{A}$ (a set of ε values.)

At each decision epoch t_k:

- Choose an action a, which corresponds to a specific threshold number of waiting requests in the SQ, from the action set A.
- Choose an action $\hat{a}$, which corresponds to the near future time, from the action set $\hat{A}$.
- Let the LPM execute the timeout policy with timeout value T_{out}.
- **If** the SP is in the idle state:
  - **If** some request comes before the timeout period (with duration of T_{out}) expires:
    - The LPM turns the SP active for processing requests until the SP becomes idle again. Then we have reached decision epoch t_{k+1}.
  - **Else**
    - The LPM keeps SP idle for T_{out} period of time.
    - **If** the GPM predicts the coming of a requests within $\hat{a}$ amount of time after timeout expires:
      - The SFC generates a fake request in order to prevent the SP from entering the sleep state.
    - **Else**
      - The LPM turns the SP into the sleep state. Then we reach decision epoch t_{k+1}.
- **Else if** the SP is in the sleep state:
  - **If** SQ contains less than a requests:
    - Block all incoming requests using the SFC so that the SP keeps in the sleep state until the SQ accumulates a requests.
    - Then we reach decision epoch t_{k+1}.
  - **Else**
    - Transfer the requests buffered in the SQ to CQ so that the SP will turn on to process requests until it becomes idle again.
    - Then we reach decision epoch t_{k+1}.
  - Evaluate the chosen action $\hat{a}$ using the TD(λ) technique.
- Evaluate the chosen action a using the TD(λ) technique.

Algorithm 3: ***CPU Scheduling using the Fairness Issue***

Input: the starting time t_0, the percentage constraint C_i
The running application type is i, $i \in APPL$.
The SP is in idle.
Update $CPU_{Time}(i)\ \forall\ i \in APPL(i)$
At time t:
If $\frac{CPU_{Time}(i)}{t-t_0} > C_i$
A. Find application type $i' \in APPL$ with the minimum $\frac{CPU_{Time}(i)}{t-t_0}$ value.
B. Perform application type switch from type i to type i'
Else
Continue running the application type i.

End

Algorithm 4: The Enhanced RL-HPM with application-type Scheduling.

Input: the timeout value T_{out} in the LPM, the action set A (a set of N values), the action set $\hat{A}$ (a set of ε values), list of application type={i, i',...}

At each decision epoch t_k:
Choose an action a, which corresponds to a specific threshold number of waiting requests in the SQ, from the action set A.
Choose an action $\hat{a}$, which corresponds to the near future time, from the action set $\hat{A}$.
Let the LPM execute the timeout policy with timeout value T_{out}.
If the SP is in the idle state:
If some request comes before the timeout period (with duration of T_{out}) expires:
The LPM turns the SP active for processing requests until the SP becomes idle again. Then we have reached decision epoch t_{k+1}.
Else
The LPM keeps SP idle for T_{out} period of time.
If the GPM predicts the coming of a requests within $\hat{a}$ amount of time after timeout expires:
The SFC generates a fake request in order to prevent the SP from entering the sleep state.
Else If it exists application type i', such as RQ(i') contains the a requests
Perform application type switch from i to i'.
The LPM turns the SP into the active state for processing requests.
Else
The LPM turns the SP into the sleep state. Then we reach decision epoch t_{k+1}.
Else if the SP is in the sleep state:
If SQ contains less than a requests:
If it exists application type i', such as RQ(i') contains the a requests
Perform application type switch from i to i'.
The LPM turns the SP into the active state for processing requests.
Else
Block all incoming requests using the SFC so that the SP keeps in the sleep state until the RQ(i) accumulates a requests.
Then we reach decision epoch t_{k+1}.
Else
Transfer the requests buffered in the SQ to CQ so that the SP will turn on to process requests until it becomes idle again.
Then we reach decision epoch t_{k+1}.
Evaluate the chosen action $\hat{a}$ using the TD(λ) technique.
Evaluate the chosen action a using the TD(λ) technique.

References

[1] 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.

[2] 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.

[3] M. Srivatava, A. Chandrakasan, R. Brodersen, Predictive system shutdown and other architectural techniques for energy efficient programmable computation, IEEE Trans. on VLSI, 1996.

[4] C. H. Hwang, A. C. Wu, A predictive system shutdown method for energy saving of event-driven computation, ICCAD, 1997.

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

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

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

[8] 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.

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

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

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

[12] 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.

[13] S. Bradtke , 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.

[14] P. Rong, M. Pedram, A Stochastic Framework for Hierarchical System-Level Power Management in Proc. of Symp. on Low Power Electronics and Design, Aug. 2005, pp. 269-274.

[15] Z. Ren, B.H. Krogh, R. Marculescu, Hierarchical adaptive dynamic power management, IEEE Trans. Computers**,** Vol. 54, No. 4, April 2005, pp. 409–420.

[16] M. Triki, Y. Wang, A.C. Ammari, M. Pedram, Dynamic power management of a computer with self power-managed components in Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS 2012), Newcastle University, 4-6 September 2012.

[17]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.

[18]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.

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

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