Reinforcement Learning for DPM of Embedded Visual Sensor Nodes

UMAIR ALI KHAN1* FAREED AHMED JOKHIO**, AND INTESAB HUSSAIN SADHAYO*

RECEIVED ON 20.09.2013 ACCEPTED ON 09.01.2014

ABSTRACT

This paper proposes a RL (Reinforcement Learning) based DPM (Dynamic Power Management) technique to learn timeout policies during a visual sensor node's operation which has multiple power/performance states. As opposed to the widely used static timeout policies, our proposed DPM policy which is also referred to as OLTP (Online Learning of Timeout Policies), learns to dynamically change the timeout decisions in the different node states including the non-operational states. The selection of timeout values in different power/performance states of a visual sensing platform is based on the workload estimates derived from a ML-ANN (Multi-Layer Artificial Neural Network) and an objective function given by weighted performance and power parameters. The DPM approach is also able to dynamically adjust the power-performance weights online to satisfy a given constraint of either power consumption or performance. Results show that the proposed learning algorithm explores the power-performance tradeoff with non-stationary workload and outperforms other DPM policies. It also performs the online adjustment of the tradeoff parameters in order to meet a user-specified constraint.

Key Words: **Embedded Visual Sensor Nodes, Dynamic Power Management, Online Learning, Timeout Olicies.**

1. INTRODUCTION

DPM of visual sensor nodes encompasses a set of strategies which dynamically reconfigure a sensor node such that the on-demand exploitation of individual components and different power-performance levels can be leveraged to reduce power consumption. Based on the underlying assumption that all the components of an embedded visual sensing platform are not always operational (or are not serving peak workload), a DPM policy tries to find the period of inactivity for each (or a specific) component of a visual sensor node. As soon as a period of inactivity or a low-workload is detected, the overall sensor node or a specific targeted component of the sensor node can be either turned off or switched to a low-power state. Based on the current environmental conditions, a good DPM policy should anticipate the future usage pattern of a visual sensor node and should take proactive measures for power management. This extends the scope of a DPM from statically optimized resources and power management to an intelligent and adaptive decision-making strategy. For the sake of simplicity, we refer to an embedded visual sensor node as a device or alternatively as a system having

* Assistant Professsor, Department of Computer Systems Engineering, Quaid-e-Awam University of Engineering, Science & Technology, NawabShah.
** Åbo Akademi University, Finland.

(visual) sensing, processing, storage and communication units. Embedded visual sensor nodes mostly run on batteries and are intended to autonomously operate for longer periods. Therefore, the power consumption requierd for sensing, processing and transmission should have known bounds and must be optimized by the implementation of an effective DPM technique which targets to enhance a sensor node's battery lifetime for longer operation. Since embedded visual sensor nodes sense and analyze predominantly non-stationary processes (e.g. road traffic, area surveillance, tracking, environmental monitroing, etc), a static DPM policy cannot cope up with the challenges presented by non-stationary workloads. Therefore, an effective DPM policy is required which can change its decision based on the observation of time-varying characteristics and self-similarity nature of an embedded visual sensor node's workload.

1.1 Existing DPM Approaches

Relevant literature about the DPM of visual sensor nodes (and in general) demonstrates various approaches. The widely used DPM strategies, the timeout policies, switch a sensor node to a low-power state after it has been idle for a certain time period. Timeout policies can be either static or adaptive [1]. A static timeout policy uses a user-selected fixed timeout period. Adaptive timeout policies [2-4] are more efficient because they adjust the timeout period according to the idle periods' history. However, without a proper anticipation of workload, timeout policies result in wasting a significant amount of power waiting for the timeout to expire. Additionally, they do not provide mechanisms for controlling the power-performance tradeoff.

Predictive policies work on a sensor node's model that is learned from historic information in order to best adjust themselves to the sensor node's dynamics. The basic idea of predictive policies is to predict the length of idle periods and switch the sensor node into a low-power mode when the predicted idle period is longer than a certain threshold. Several predictive policies are studied in the literature that include nonlinear regression over the history of past idle periods [5], adaptive learning trees [6], exponential weighted average of previous idle periods [10], and predictive shutdown method based on collecting and analyzing information on the access patterns to I/O devices [7]. Predictive policies do address workload uncertainty, but do not deal with the queuing models [8] where the requests can be queued before processing. Hence, they do not offer a possibility to control the power-performance tradeoff.

Some limitations such as uncertainty, queuing, and power-performance tradeoff of predictive policies are addressed by stochastic policies. These policies make probabilistic assumptions about the usage pattern of a sensor node and exploit the nature of the probability distribution to formulate an optimization problem, which derives an optimal DPM strategy [9]. The node's states and request queues in stochastic policies are generally modeled as MDP (Markov Decision Process). Based on the type of MDP (Discrete, Continuous, Semi-Markov), several stochastic policies are studied in the literature [8,10-12]. These policies either require observing and issuing power commands at regular time-discretized intervals which results in an additional power cost, or assume a specific workload distribution which exclusively fits to a specific model. Additionally, these policies require a rigorous modeling of the system and the system-specific formalization of the optimization problem. Therefore, they are model dependent.

Recently, model-free RL-based DPM approaches have gained increasing attention due to their simplicity. Model-free RL-based DPM techniques learn the system dynamics by interacting with the system, implementing certain actions, evaluating the effects of the implemented actions, and adjusting the actions on the fly. The RL-based DPM approach proposed in [13] uses a set of pre-selected DPM policies for controlling power consumption under different workloads. The algorithm associates and maintains relative weights for each policy which reflect its individual performance. The weights are adjusted online and at any point in time the best performing policy is selected with the highest probability. This approach does lead to an optimal DPM policy, but is dependent on and limited to the chosen experts.

In [14], the authors propose a model-free, continuous-time RL algorithm with workload estimation to guide the learning algorithm for changing workloads. The workload prediction is based on a Bayesian classifier and implemented on the real data of a WLAN card which performs well for this setting due to the regular traffic that has high statistical predictability [15].

1.2 Our DPM Approach

We address the issues found in [14] in our prior work [16-17] by proposing a model-free, RL-based DPM algorithm for non-stationary workloads targeting an embedded visual sensing platform for traffic surveillance. We use a ML-ANN based workload estimator with backpropagation algorithm to provide estimated workload information to the learning algorithm. Based on the estimated workload, the power manager executes certain timeout values in the operational (idle) state and waits for the service queue to be populated with a certain number of requests in the non-operational (sleep) state. Workload estimation using a ML-ANN achieves higher accuracy with the non-stationary data and the algorithm is able to find a tradeoff for the multi-objective optimization of power and performance. However, waiting for a certain number of requests in the queue degrades performance when the workload drops abruptly.

We further address these issues by proposing an online, RL-based timeout policy [18] that learns to use timeout values for both operational and non-operational states of a visual sensing platform. We demonstrate that an online, RL-based DPM policy that learns to use timeout values in each state of a visual sensor node is capable of delivering much improved performance. However, this work is primarily focused on limited number of power/performance states and its viability for higher number of states is not investigated.

The concept of OLTP/OAPP proposed in this paper is an incremental approach of our aforementioned work on reinforcement learning based DPM policies for embedded visual sensor nodes. We propose a generic framework of a RL-based timeout policy for the DPM of a sensing platform with multiple power/performance states and targeted to operate in a highly dynamic environment. The motivation behind our proposed DPM policy is extending the scope of traditional timeout policies which are still the state-of-the-art approach for solving the power management problems but do have the aforementioned limitations. The work proposed in this paper is mainly focused on two major aspects of dynamic power management: (i) exploring the power-performance tradeoff in a highly dynamic DPM environment and allowing the flexibility to obtain every possible solution corresponding to a certain power-performance level, and (ii) dynamically reconfiguring the sensing system during the operation so that a user-specified constraint (or level) of power consumption or performance is achieved. Our proposed DPM technique is an application-level software framework that targets the DPM of a sensing system having multiple operational and non-operational states that can be leveraged to find a better power-performance tradeoff and to satisfy a user-specified constraint of power consumption or performance.

The remainder of the paper is organized as follows. Section 2, presents the generic background of reinforcement learning and its implementation on a DPM problem. Section 3 provides an in-depth problem formulation for the OLTP framework. In Section 4, we discuss the overall algorithm and present some results to demonstrate the effectiveness of our algorithm. In Section 5, we compare the OLTP with some existing DPM policies. Section 6 presents the integration of OAPP (Online Adaptation of Power/Performance) controller to the OLTP framework. Section 7 concludes the paper.

2. REINFORCEMENT LEARNING

RL is a machine learning paradigm concerned with learning to control a system so as to maximize (or minimize) a numerical performance (or penalty) measure that expresses a long-term objective. RL assumes that the system dynamics follow the Markov property, i.e. the next state $s \in S$ is reached by taking an action $a \in A$, and the immediate reward r depends only on the current state $s \in S$ and action a. In the simplest form of RL, the Q-learning, policies and the state-action values are represented by a 2-dimensional lookup table indexed by state-action pairs. For all $(s,a) \in S \mathrm{x} A$, the Q-learning principle is given in Equation (1) [19-20]:

$$Q^{(t+1)}(s_t, a_t) = Q^t(s_t, a_t) + \alpha_t(s_t, a_t)[r_{t+1} + \gamma \max_{a'} Q^{(t+1)}(s_{t+1}, a') - Q^t(s_t, a_t)] \quad (1)$$

where $\alpha_t(s_t,a_t) \in (0,1)$ is the learning rate. $Q^t(s_t,a_t)$ represents the accumulated reward for each state s at time t where action a is taken under the policy learned so far. The positive constant γ is called discount factor which determines the importance of the reward such that $\gamma \in (0,1)$. The reward r_{t+1}, obtained after taking action a_t at time t, is added to the best (maximum) discounted value of the next state $\max_{a'} Q^{(t+1)}(S_{t+1},a')$. The previous state estimate $Q^t(s_t,a_t)$ is subtracted from this sum. This subtraction gives an error estimate. The overall aim of the RL algorithm is to reduce the error estimate for each state-action pair in order to identify appropriate actions for each state of the system. The description of all the related notations is given in Table 1.

3. PROBLEM FORMULATION

To preserve generality, we introduce the term PMS (Power Managed System) in our working definition of a single component, device or a complete (visual) sensing platform/ node which has multiple power states each having different power/performance levels. The PMS may represent a functional sub-system, e.g. visual sensor, communication unit, processor, graphics processing unit, memory, etc, or it may represent a complex computing system as a whole such as a SBC (Single Board Computer). The underlying assumption about a PMS is the availability of multiple power modes that can be exploited to control the power-performance tradeoff. Based on the real-life observations, our definition of PMS encompasses following characteristics: (i) Switching the power modes results in an additional transition cost measured in terms of power consumption and latency; (ii) The power states of the PMS can be controlled by a software framework, the PM (Power Manager), that either runs on the (operating system) kernel level or the application layer.

We adopt the same generic system model as proposed in [8]. This generic model allows to control the power-performance tradeoff and supports queueing of requests before processing. A modified version of this model that fits to our learning based timeout policy is depicted in

TABLE 1. DESCRIPTION OF THE NOTATIONS USED IN Q-LEARNING PRINCIPLE (EQUATION (1))

Notation	Description
$\alpha_t(S_t,\alpha_t)$	The learning rate which is decreased slowly during the learning process.
S	Set of environment states
A	Set of available actions in each state $s \in S$
γ	Discount factor to describe the importance of rewards in immediate or long-term sense
r_{t+1}	The reward received in the next state S_{t+1} by taking action a_t in current state S_t
$Q^{(t+1)}(S_t,a_t)$	State-action pair estimate of the next state S_{t+1}
$Q^t(S_t,a_t)$	State-action pair estimate of the current state S_t
$\max_{a'} Q^{(t+1)}S(s_{t+1},a')$	Best discounted value of the next state S_{t+1}

Fig. 1. In this abstract model, the SR (Service Requestor) is the (software) application that generates requests which are buffered in the SQ (Service Queue) before processing. The SP (Service Provider) is the request processing device which can be in any of the available power states (e.g. busy, idle, sleep, etc.) at any point in time. The PM is the dynamic power management algorithm that decides which power mode the SP should be switched to. In our case, the PM is based on the OLTP. The PMS has multiple power/performance states, $S_1,...,S_n$, which can be switched by the PM. The DPM actions taken by the PM include selecting an appropriate timeout in a specific state of the PMS and (at the expiration of the timeout or the occurrence of an event) changing the power state. At each decision time, the PM receives an observation that includes the estimate of the SR from the ML-ANN (low/high workload), the state of the SQ (number of requests waiting in the queue), and the state of the SP (power mode of the processing element of the PMS). Based on these observations, the PM selects an appropriate DPM action.

3.1 State Space

In our learning model, each system state has a composite form $S=(SR,SQ,SP)$, where $SR=\{0,1\}$is the current workload estimate (low, high), $SQ=\{0,1,2,...,N\}$ is the number of requests in the service queue, and $SP=\{S_1,S_2,...S_n\}$ is the set of power states of the SP. We perform a state aggregation in the design space to limit the values of the SQ to a state having no requests in the queue and the one having some requests, i.e. $\forall sq \in SQ, sq=\{0,N \mid N \in \mathbf{N}\}$. State aggregation not only reduces the search space, but also contributes to speed up the convergence of the learning algorithm.

In each composite state, the available actions comprise a set of pre-selected timeout values t^k_{out}. Since using continuous timeout values increases the complexity and also severely affects the convergence speed of the algorithm, we use discretized timeout values in the action set. The set of timeout values A depends on a selected threshold T_{thr} and is defined as:

$$A = \{t^k_{out}\} = \{\varepsilon_k T_{thr}\}, \ \varepsilon_k \in \Re^+, \ k = 1,2,..., n \tag{2}$$

where ε_k is a positive weight for discretization and T_{thr} is a break even time [21] which can compensate the overhead and cost involved in switching the PMS from one power state to another. The discretization step and the size of action set can be selected by the user. In general, the action set can comprise timeout values such as:

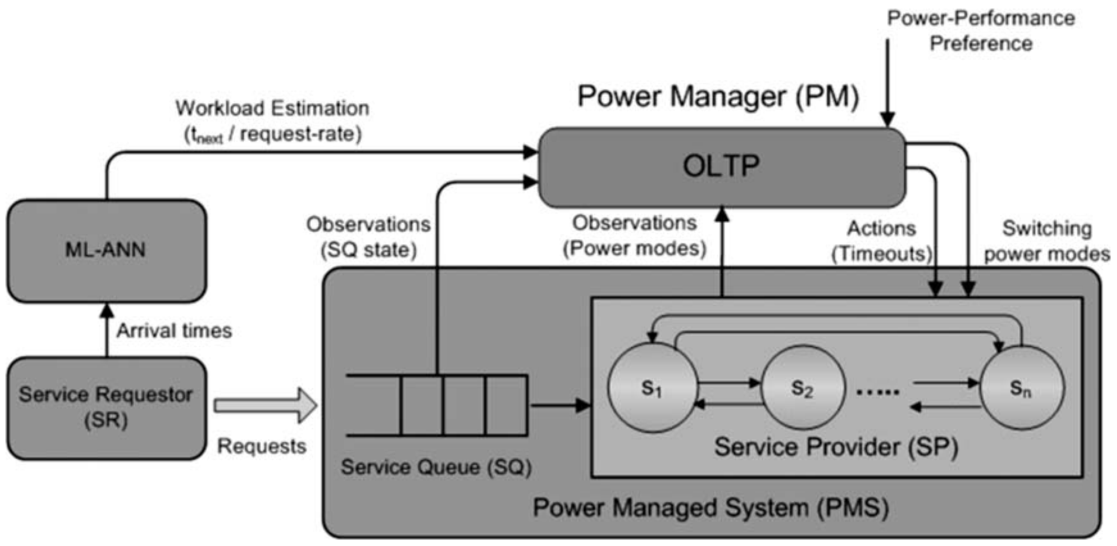

FIG. 1. ABSTRACTION OF THE POWER MANAGED SYSTEM

$$A = \{0, 0.1T_{thr}, 0.2T_{thr}, ..., T_{rhr}, 1.1T_{thr}, ...\varepsilon T_{thr}) \quad (3)$$

3.2 Dynamics of the OLTP

From our description of the PMS, we can derive a hierarchical model of the power-performance states. The highest power consuming mode is the processing state P in which the PMS is processing requests buffered in the SQ. From the PM's perspective, this is a non-preemptive state in which the PM has no action to select until all the requests have been processed and the PM takes over to make a DPM decision. The idle state is the second highest power consuming state in which the PMS is waiting for the requests. The idle state may have a number of power/performance states. In order to process the requests, the PMS has to transit from a higher idle state I_k to lower idle state I_1, and then to the processing state P. The deeper the idle state, the higher the power consumption and delay caused by the transition.

When the PMS transits to non-operational state ($SQ=0$), it first enters the lowest sleep state S_1. Based on the observations ($SQ=0$) and expiration of timeouts, the PMS may transit to higher sleep states S_k. When an event is detected ($SQ>0$), the PMS has to transit from the sleep state S_k to the idle state I_1 at the expiration of timeout. The deeper the sleep state, the higher the power consumption and latency involved in transiting from sleep state to the idle state I_1. However, the transitions from idle state I_1 to processing state P (and vice versa) are assumed to be autonomous and instantaneous with negligible cost.

In any of the higher idle state I_k, the PMS transits to the idle state I_1 as soon as the SQ transits from 0 to 1. Whereas, the transitions from any of the sleep states to the idle state I_1 take place when the SQ transits from 0 to 1 and the timeout in the sleep state expires. The decisions of selecting timeout periods and switching the power states of the PMS are taken by the PM.

If t_{out} represents the timeout in any state, t_{S_k} is the time spent in sleep state S_k and t_{I_k} is the time spent in an idle state I_k, we can define the following 9 transitions:

$$C_1 \rightarrow (SP = S_k \wedge SQ = 0 \wedge t_{S\_k} > t_{out})$$
$$C_2 \rightarrow (SP = S_k \wedge SQ > 0 \wedge t_{S\_k} > t_{out})$$
$$C_3 \rightarrow (SP = I_k (k > 1) \wedge SQ > 0)$$
$$C_4 \rightarrow (SP = I_k \wedge SQ = 0 \wedge t_{I\_k} > t_{out})$$
$$C_5 \rightarrow (SP = I_1 \wedge SQ > 0)$$
$$C_6 \rightarrow (SP = P \wedge SQ = 0)$$
$$C_7 \rightarrow (SP = S_n \wedge SQ = 0 \wedge t_{S\_n} > t_{out})$$
$$C_8 \rightarrow (SP = P \wedge SQ > 0)$$
$$C_9 \rightarrow (SP = I_n \wedge SQ = 0 \wedge t_{I\_n} > t_{out})$$

Fig. 2 depicts the OLTP dynamics.

3.3 Cost Function

The selection of timeout values in sleep and idle states is subjected to two objectives, i.e. minimizing power consumption and maximizing performance. We consider

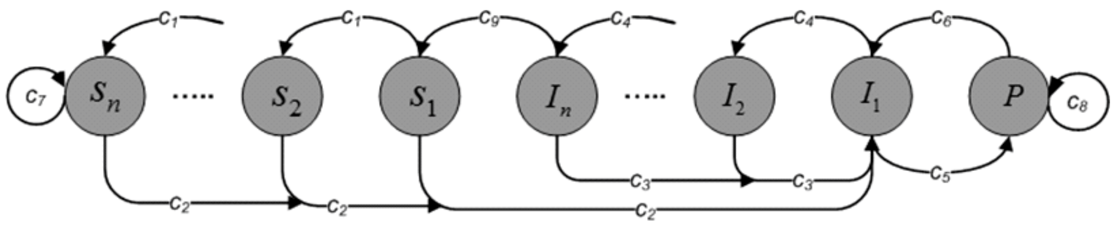

FIG. 2. STATE-TRANSITION DIAGRAM OF THE OLTP

the average latency per request caused by an action as the performance measure. The average latency includes the average queuing time plus the average execution time. In a multi-objective RL problem, it is convenient to combine the multiple objectives into a single objective function via the process of linear scalarization [22]. The primary advantage of linear scalarization is that the user can assign a relative weight/preference to each objective and can direct the algorithm to a specific part of the objective space by favoring one objective over another. Varying the relative weight between the objectives provides a set of solutions on the Pareto front. In our RL algorithm, we combine the average energy consumption and average latency caused by an action taken in a specific state into a single objective function and call it a cost function which can be treated in the same way as the reward. In each state, the learning algorithm evaluates the effectiveness of a selected action by the measure of the cost computed after executing the action. The objective of the RL algorithm is to minimize the cost function with respect to the relative weights assigned to power consumption and latency. The cost function is defined in Equation (4):

$$c_t(s,a,\omega) = (1-\omega)\, p_t(s,a) + \omega l_t(s,a) \tag{4}$$

where $p_t(s,a)$ and $l_t(s,a)$ are the average power and average latency in state s at time t after taking an action a. The relative weight $\omega \in (0,1)$ between the two objectives serves as a tradeoff parameter.

3.4 Estimation of State-Actions Values

A 1-step Q-learning algorithm, as described in Equation (1), can be used to estimate the state-action values with the evaluated cost. However, we must incorporate the time spent in each state to the update rule. Moreover, the selection of timeout values in idle and sleep states follows an opposite behavior. If the power saving is given more preference over latency ($\omega \rightarrow 0$), the learning algorithm should select higher timeouts in sleep states and smaller timeouts in idle states to save power. But a higher preference to the latency should follow an opposite behavior. Therefore, the calculated cost of an action and the minimum accumulated cost of the next state must be discounted with appropriate factors to learn the timeout values subjected to a selected preference. We introduce two discount factors for the Q-learning update rule defined in Equation (1):

$$d_1 = (1 - e^{-\beta t_{out}}), \quad \beta \in (0,1) \tag{5}$$

$$d_2 = (1 - d_1) \tag{6}$$

In Equations (5-6), t_{out} is the time (selected timeout period) spent in a state. For the composite state $S=(SR,SQ,SP)$, action set $A=\{t^1_{out},t^2_{out},...t^n_{out}\}$, and $\forall(s,a)\in S\text{x}A$, Equation (1) can be modified with the discount factors d_1 and d_2 as follows in Equation (7):

$$\begin{aligned} Q^{(t+1)}(s_t,a_t) &= Q^t(s_t,a_t) + \alpha_t(s_t,a_t)[d_1 c_{(t+1)}(s,a,\omega) + \\ & d_2 \min_{a'} Q^{(t+1)}(s_{(t+1)},a') - Q^t(s_t,a_t)] \end{aligned} \tag{7}$$

3.5 Workload Estimation

Although a dynamic environment has a high degree of uncertainty in request patterns, the inter-arrival times of requests (or the workload) can be predicted with a certain level of confidence [23]. Based on the underlying assumption that there is a significant correlation between the recent inter-arrival times of events/requests and the events occurring in near future [24], the workload estimation can be performed by analyzing the recent history of input requests. For incorporating workload estimation in OLTP, we use a ML-ANN which operates on the recent history of events and estimates the inter-arrival time of the next event [18]. We use a fix-sized moving window on the history of previous inter-arrival periods and input these inter-arrival periods (normalized between 0 and 1) to the ML-ANN. The ML-ANN estimates the length of the next inter-arrival period (de-normalized).

4. FINDING THE PARETO-FRONT

In each decision time, the PM selects a timeout value and relinquishes the control until the timeout expires or some requests arrive during the timeout period in idle state. At the end of the timeout period (or when the timeout is forced to terminate by the incoming requests), the PM regains the control and evaluates the last action (Equation (4)) and updates the Q-value of the last visited state-action pair (Equation (7)). The PM then selects another action (timeout) in the new state based on the probabilities of the individual actions. The output policy π obtained from Algorithm-1 is $\pi = min_a Q(s,a), \forall s \in S, a \in A$. The learning rate $\alpha_t(s,a)$ is decreased slowly (Step 9, Algorithm-1) such that it reflects the degree to which a state-action pair has been chosen in the recent past. It is calculated in Equation (8) as:

$$\alpha_t(s,a) = \frac{\xi}{visited(s,a)} \tag{8}$$

where $\zeta \in (0,1)$ is a positive constant. Every time a state-action pair (s,a) is visited with this learning rate, the difference between its estimated Q-value $Q^{(t+1)}(s,a)$ and the current Q-value $Q^t(s,a)$ reduces (Step 11, Algorithm-1). Hence, for all state-action pairs, the algorithm converges to a (timeout) DPM policy.

The learning algorithm should not only explore the state-space adequately, but should also exploit the experience hitherto gained. We use a semi-greedy exploration policy which starts out with selecting random actions (exploration) which are equally distributed (Step 6, Algorithm-1). When the algorithm acquires more knowledge about the system, the probabilities of actions with minimum cost begin to increase. Eventually, the policy becomes greedy by selecting minimum-cost actions due to their high probabilities (exploitation). The action probabilities, p^k_r, in a state are given by Equation (9):

$$p^k_r(s,a) = \frac{e^{\frac{Q^t(s,a_k)}{\tau}}}{\sum_{k=1}^{n} e^{\frac{Q^t(s,a_k)}{\tau}}}, \quad \forall a \in A, n = |A| \tag{9}$$

where τ is initialized with a high value which gives equal weights (probabilities) to all actions (exploring). τ is then decayed over time which increases the probability for low-cost actions. Thus, the behavior of the learning changes towards exploitation. For simulations purpose, we use a synthetic PMS model having 1 processing state, 2 idle states I_1, I_2, and two non-operational (sleep) states S_1, S_2.

ALGORITHM-1 ONLINE LEARNING OF TIMEOUT POLICIES

```
1.  Define the power-performance preference ω
2.  Initialize $Q,T$ and probability matrix $p_r$ from uniform distribution repeat
      (a) Get the current information of workload from the ML-ANN estimator (section 3.5)
      (b) Obtain the current composite state (Section 3.1)
      (c) Calculate the action probabilities (Equation (9))
      (d) Select the current action based on the calculated probabilities (exploration/exploitation)
      (e) Execute the selected action (a timeout policy)
      (f) Calculate the cost of the last action (Equation (4))
      (g) Update the learning rate (Equation (8))
      (h) Update $T$ with the new state-action pair
      (i) Update Q-value of the visited state-action pair (Equation (7))
until
    Specific number of iterations or a certain threshold of error estimate is achieved
```

The power and state-transition delay characteristics of this synthetic model are given in Table 2. In order to demonstrate the effectiveness of the OLTP with a dynamic environment, we use road traffic workload which represents a non-stationary request pattern. We collected the road traffic data by taking several test recordings at different locations, where we recorded the inter-arrival times of vehicles by a vehicle detection algorithm [25]. The test recordings were carried out on different highways where the traffic intensity is usually higher and provides adequate data in terms of changing workload to evaluate the performance of our DPM algorithm. We use the vehicles' inter-arrival times as the request inter-arrival times in our algorithm. The characteristics of different workloads are given in Table 3.

For each workload, we vary the relative weight ω between power and latency in the cost function (Equation (4)) to obtain a number of Pareto-optimal solutions. Fig. 3 shows the power-performance Pareto-fronts of workloads 1-5.

Note that the power-performance tradeoff curves of all workloads follow the same trend. However, the power profile of each workload depends on the MIAT (Mean Inter Arrival Time) of the requests. In case of shorter MIATs, the PMS has to spend more time in idle and busy states. Since workload 2, 3 and 5 do not have significantly different MIATs, their curves overlap with each other. On the other hand, workloads 1 and 4 have relatively (shorter) MIATs as compared to workload 2, 3 and 5. Therefore, the power profiles of these workloads are shifted upward.

Fig. 4 shows the states occupancy (time spent in each state) of the PMS with respect to ω. For $\omega \rightarrow 0$, the PMS spends more time (hours) in sleep states to achieve higher power savings. When the value of ω is increased, the PMS shortens the time in sleep states and prefers to spend more time in idle states. This also increases the overall transition times. When ω approaches 1, the PMS spends more time in idle states and less time in sleep states to increase the performance. This again reduces the overall transition time. The time in processing state is the same for all cases due to same number of requests.

TABLE 3. CHARACTERISTICS OF DIFFERENT TRAFFIC WORKLOADS

Workload	MIAT (Seconds)	Number of Requests	Duration (Hours)
Workload 1	6.79	11649	22
Workload 2	11.13	7762	24
Workload 3	11.07	7803	24
Workload 4	9.06	9502	24
Workload 5	12.05	7155	2

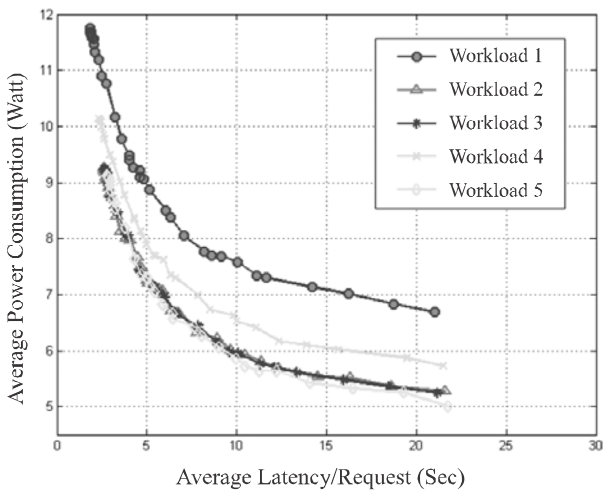

FIG. 3. POWER-PERFORMANCE PARETO-FRONTS OF WORKLOAD 1-5

TABLE 2. POWER AND DELAY CHARACTERISTICS OF THE SYNTHETIC PMS

	Busy	I_1	I_2	S_t	S_2	$I_1 \rightarrow I_2$	$I_2 \rightarrow I_1$	$I_2 \rightarrow S_1$	$S_1 \rightarrow I_1$	$S_1 \rightarrow S_2$	$S_2 \rightarrow I_1$
Power (Watt)	16	13	10	4	1	11	11	7	7	3	8
Delay (Sec)	-	-	-	-	-	1	1	4	5	2	7

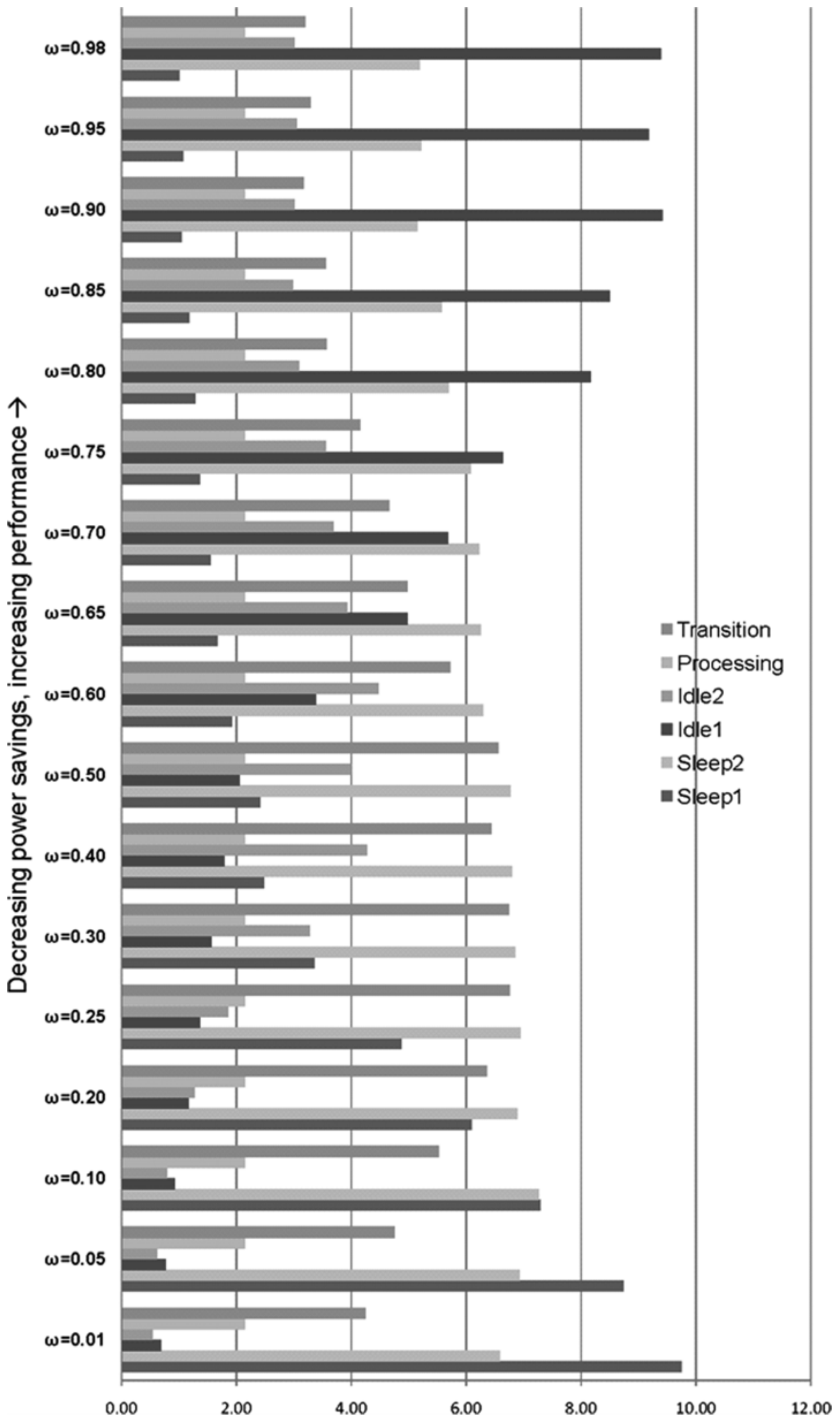

FIG. 4. STATES OCCUPANCY WITH RESPECT TO POWER-PERFORMANCE TRADEOFF (WORKLOAD 2)

5. EVALUATION OF THE OLTP

We compare the performance of OLTP based power manager with some of the power managers based on the existing DPM policies including fixed timeout, adaptive timeout [2], predictive (ML-ANN) [18], exponential predictive [26], and online Timeout/N policy [17]. For demonstrating the effectiveness of the OLTP, we implement it on a real PMS and compare its performance with the aforementioned DPM policies. The real PMS is our embedded visual sensing platform with multiple visual sensors for traffic surveillance [18]. We target the processing platform of the real PMS which has 1 processing, 1 idle and 1 sleep state. The power and delay charateristics of the real PMS are given in Table 4. The comparison is performed on Workload 1. The time threshold T_{thr} for all the policies is set to 10 seconds. In adaptive timeout policy, if the ratio of the current timeout and previous request inter-arrival time is less than a threshold of 0.8, the timeout is incremented by 0.1 (or decreased otherwise).

In Table 4, P_{sleep}, P_{idle}, P_{busy}, and P_{trans} represent the power consumptions in sleep, idle, busy and transition (sleep to idle, idle to sleep) states, respectively. t_{s2i} and t_{i2s} represent the time required to switch the PMS from sleep to idle state and vice versa, respectively.

Fig. 5 shows the comparison of the OLTP with the above mentioned DPM policies for workload 1. From Fig. 5 it is evident that the OLTP is capable of finding a better power-performance tradeoff than other DPM policies. The online Timeout/N policy does allow to control the power-performance tradeoff, but results in higher latency as compared to the OLTP. The OLTP provides a much deeper power-performance tradeoff curve with the same level of power consumption and a significantly reduced latency corresponding to each solution on the pareto-front. The fixed timeout policy results in the highest power consumption. The predictive policies perform better than the fixed timeout policies with almost the same latency level. The ML-ANN predictive policy gives slightly higher power savings than the exponential predictive policy. The adaptive timeout policy, due to its intrinsic similarity to the OLTP, matches with one of the solutions of the OLTP. Fig. 5 also shows the power-performance set without any DPM policy. It is evident that the OLTP provides significant power savings without substantially degrading the performance.

TABLE 4. POWER AND DELAY CHARACTERISTICS OF THE REAL PMS

P_{sleep} (Watt)	P_{idle} (Watt)	P_{busy} (Watt)	P_{trans} (Watt)	T_{s2i} (Seconds)	t_{i2s} (Seconds)
3	25	32	15	6	4

6. ONLINE ADAPTATION OF POWER/ PERFORMANCE

Online adjustment of the weights assigned to different objectives in a multi-criteria optimization problem is a challenging issue. Since online learning is time-constrained, long delays in updating ω will result in undesired behavior of the system. Likewise, if ω is updated too rapidly, the system may not get sufficient time to adapt to the new learning parameters and will again result in an unwanted behavior. For a system having a dynamic workload, as in our case, a small online adjustment of the

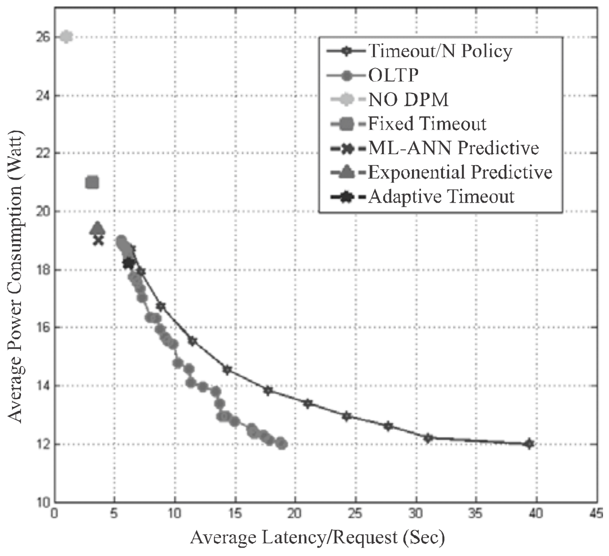

FIG. 5. OLTP COMPARED WITH DIFFERENT DPM POLICIES

weights may lead to a significant divergence (large overshoot or undershoot) of the policy.

For adapting to a given power (or latency) constraint, we integrate a discrete OAPP (Online Adaptation of Power/ Performance) controller in the OLTP framework to perform the online adjustment of ω. The OAPP controller is based on an iterative approach that compares the actual power consumption (or performance) to the given constraint over equally distributed time intervals, or after certain number of RL updates. If the average power and average latency of a timeout policy in the update interval *i* are represented by ϕ_P and ϕ_L, then for a given power constraint C_P or latency constraint C_L, the OAPP controller uses the following rule to update the value of ω:

$$\omega_{i+1} = \omega_i + \kappa(C_P - \phi_P) \tag{10}$$

$$\omega_{i+1} = \omega_i - \kappa(C_L - \phi_L) \tag{11}$$

In Equations (10-11), κ is an adapting coefficient. From the cost function (Equation 4), it is clear that varying the value of ω has opposite effects on power and latency. Therefore, we use Equation (10) to update ω for power constraint and Equation (11) for latency constraint. Fig. 6 depicts the framework of OAPP in conjunction with OLTP. The average power ϕ_P and average latency ω_L in Fig. 6 are unified by ϕ. The constraints are represented by *C*.

Fig. 7 shows the overall flow chart of OLTP used in combination with OAPP. The value of ω is initialized randomly. After a certain number of RL updates, the OLTP decides if the value of ω needs to be updated. For this purpose, the OLTP communicates with the OAPP controller which compares the relative difference between the average power (or latency) ϕ during the last *N* RL updates and the given constraint *C* with a threshold e_1C to check the deviation from the constraint. The positive constant e_1 represents the criteria (threshold) of the amount of deviation between the values of ω and *C* to update the value of ω. If the relative difference $|C-\phi|$ is greater than e_1C, the OAPP controller updates ω according to Equation (10-11) and sets a flag converge to 0 to indicate that the value of ω is in adaptation phase. The OLTP resets the learning rate α so that the learning can start gaining new knowledge with the updated ω value. On the other hand, if the OLTP finds the flag converge already set to 1 (ω has already converged), it refers to OAPP to confirm if the service request pattern has not changed. The OAPP again

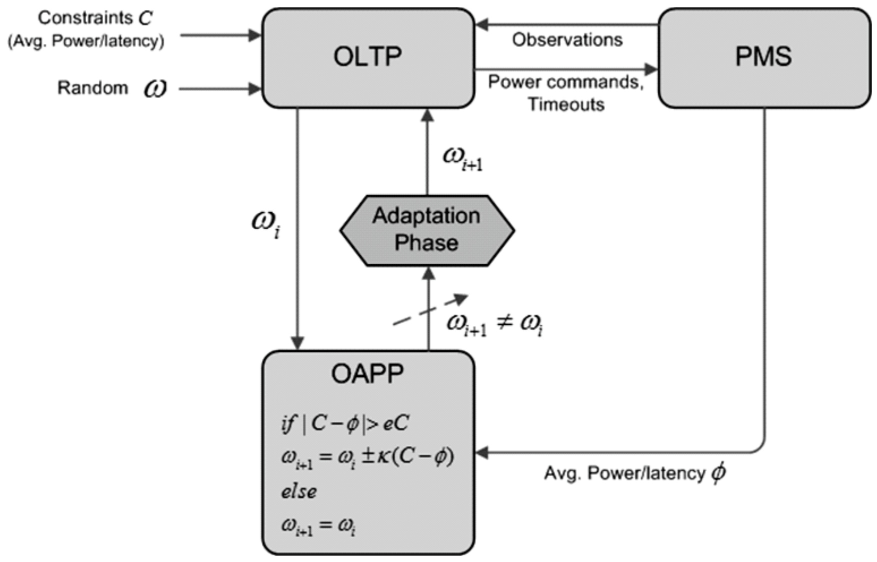

FIG. 6. ONLINE ADAPTATION OF POWER/PERFORMANCE (OAPP)

compares the relative difference $|C-\phi|$ with $e_2C(e_2 > e_1)$. In this case, the current value of ϕ may have a larger deviation from the constraint. If the relative difference $|C-\phi|$ is found to be greater than e_2C, the OAPP resets the flag converge and starts updating ω again.

In our experiments with the OLTP/OAPP framework, we set e_1 and e_2 to 0.05 and 0.1 respectively which represent a deviation of 5% from the given constraint when the flag converge is 0, and a deviation of 10% when the flag converge is set to 1. Fig. 8 shows the the convergence of the OLTP/OAPP to the latency and power constraints of synthetic and real PMS respectively for Workload 1.

We tested the OLTP/OAPP for a number of power and latency constraints for different workloads. The results presented in Tables 5-8 show that the OLTP/OAPP framework can satisfactorily fulfill the given constraints by the online adjustment of power-performance weight.

7. CONCLUSIONS

In this paper, we proposed a generic framework of a RL-based DPM approach that learns selection of timeout values for a sensor node having multiple power/performance states. The timeout selection is based on workload estimates derived from a ML-ANN and an objective function given by weighted performance and power parameters. Our approach relies neither on any offline workload data analysis nor on a priori system model; it is able to explore and (after some learning time) exploit the power-performance tradeoff. Our DPM approach is further able to adapt the power-performance weights online to meet user-specified power and performance constraints, respectively. Our experiments show that a Pareto-optimal tradeoff between average power consumption and average latency per request is achieved. Furthermore, the OLTP/OAPP framework promptly converges to different user-specified power and latency constraints.

Our future work includes migrating the OLTP/OAPP framework from application level to (operating systems) kernel level and incorporating it into ACPI framework. We are confident that our online, RL-based DPM algorithm running at the OS level and addressing multiple idle and sleep states is able to outperform static ACPI policies and to further improve the power-performance tradeoff.

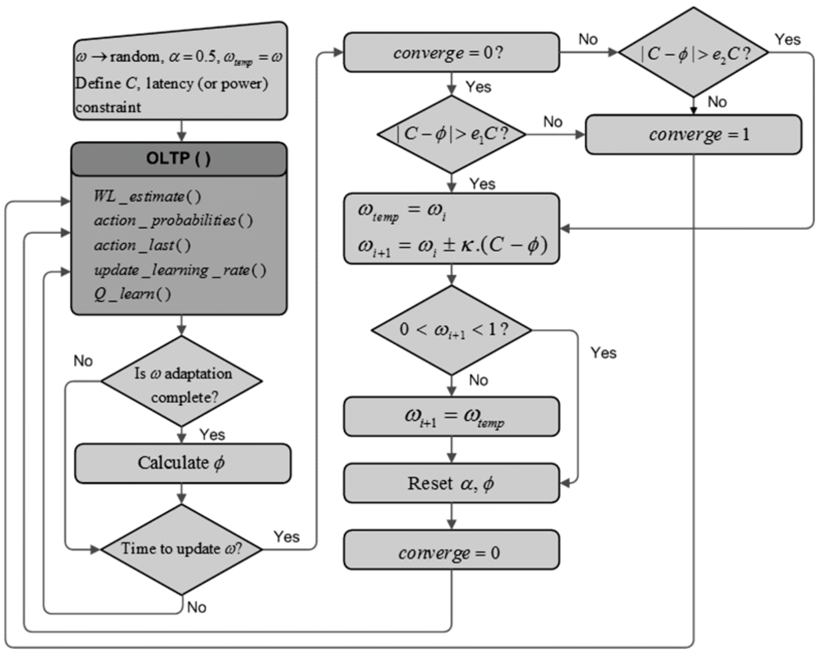
$\omega \rightarrow$ random, $\alpha = 0.5$, $\omega_{temp} = \omega$
Define C, latency (or power) constraint
OLTP ( )
$WL\_estimate()$
$action\_probabilities()$
$action\_last()$
$update\_learning\_rate()$
$Q\_learn()$
Is ω adaptation complete?
No
Yes
Calculate ϕ
Time to update ω?
Yes
No
$converge = 0?$
No
Yes
$|C-\phi| > e_2C?$
Yes
No
$|C-\phi| > e_1C?$
No
Yes
$converge = 1$
$\omega_{temp} = \omega_i$
$\omega_{i+1} = \omega_i \pm \kappa.(C-\phi)$
$0 < \omega_{i+1} < 1?$
Yes
No
$\omega_{i+1} = \omega_{temp}$
Reset α, ϕ
$converge = 0$

FIG. 7. OLTP/OAPP FRAMEWORK FOR POWER/PERFORMANCE ADAPTATION

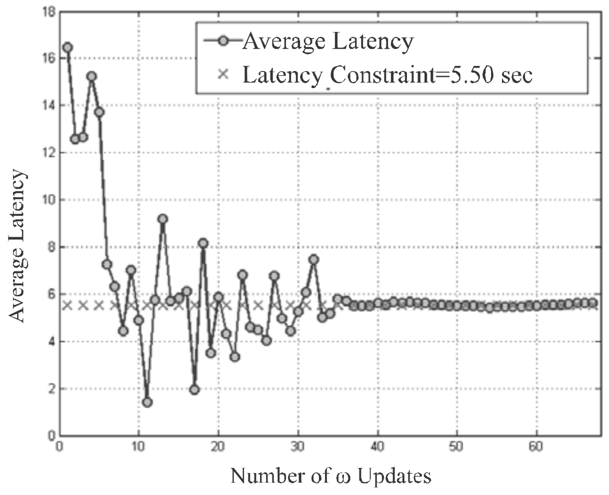

FIG. 8(a). LATENCY CONSTRAINT (SYNTHETIC PMS)

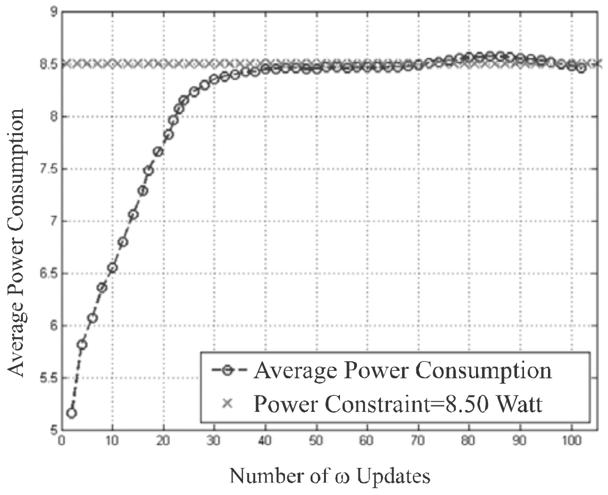

FIG. 8(b). POWER CONSTRAINT (SYNTHETIC PMS)

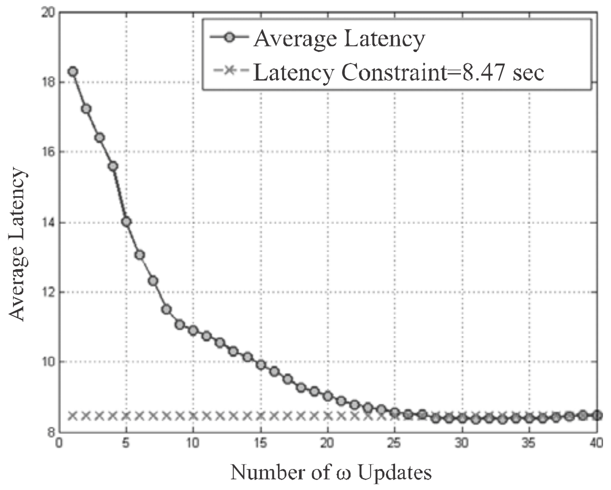

FIG. 8(c). LATENCY CONSTRAINT (REAL PMS)

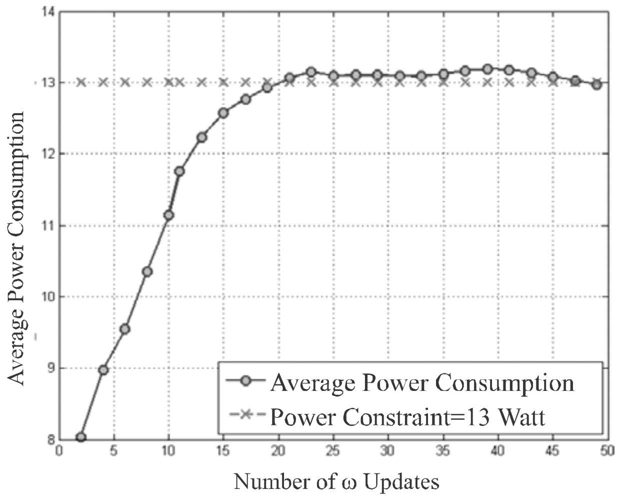

FIG. 8(d). POWER CONSTRAINT (REAL PMS)

FIG. 8. OLTP/OAPP BASED CONSTRAINT ADAPTATION

TABLE 5. LATENCY CONSTRAINT ADAPTATION (SYNTHETIC PMS)

C_L (sec)	ϕ_L (sec)	Relative Difference (%)	ω
0.50	5.65	2.70	0.45
0.00	7.07	1.00	0.30
0.00	7.92	1.00	0.23
0.00	9.66	3.40	0.15
0.50	10.21	2.80	0.12

TABLE 6. POWER CONSTRAINT ADAPTATION (SYNTHETIC PMS)

C_P (Watt)	ϕ_P (Watt)	Relative Difference (%)	ω
0.50	7.81	4.10	0.25
0.00	8.05	0.60	0.30
0.00	8.90	1.10	0.43
0.00	9.70	2.90	0.60
0.00	10.94	0.50	0.78

TABLE 7. LATENCY CONSTRAINT ADAPTATION (REAL PMS)

C_L (sec)	ϕ_L (sec)	Relative Difference (%)	ω
0.40	17.30	0.57	0.16
0.38	15.11	1.74	0.21
0.54	13.30	1.71	0.28
0.80	12.87	0.62	0.31
0.47	8.50	0.27	0.51

TABLE 8. POWER CONSTRAINT ADAPTATION (REAL PMS)

C_P (Watt)	ϕ_P(Watt)	Relative (%)	Difference ω
0.23	12.14	0.70	0.16
0.00	14.25	1.75	0.35
0.00	15.96	0.27	0.52
0.83	17.58	1.40	0.62
0.85	18.66	0.98	0.70

ACKNOWLEDGEMENT

This work has been sponsored in part by the Austrian Research Promotion Agency.

REFERENCES

[1] Lu, Y.H., and DeMicheli, G., "Comparing System-Level Power Management Policies", IEEE Transactions on Design and Test of Computers, Volume 18, No. 2, pp. 10-19, 2001.

[2] Douglis, F., Krishnan, P., Bershad, B., et al. "Adaptive Disk Spin-Down Policies for Mobile Computers", Computing Systems, Volume 8, No. 4, pp. 381-413, 1995.

[3] Olsen, C.M., and Narayanaswarni, C., "Powernap: An Efficient Power Management Scheme for Mobile Devices", IEEE Transactions on Mobile Computing, Volume 5, No. 7, pp. 816-828, 2006.

[4] Shih, H.C., and Wang, K., "An Adaptive Hybrid Dynamic Power Management Algorithm for Mobile Devices", Computer Networks, Volume 56, No. 2, pp. 548-565, 2012.

[5] Srivastava, M.B., Chandrakasan, A.P., and Brodersen, R.W., "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation", IEEE Transactions on Very Large Scale Integration Systems, Volume 4, No. 1, pp. 42-55, 1996.

[6] Chung, E.Y., Benini, L., and DeMicheli, G., "Dynamic Power Management Using Adaptive Learning Tree", Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 274-279, CA, USA,1999.

[7] Young, H., Sung, K., and Ki-Seok, C., "A Predictive Dynamic Power Management Technique for Embedded Mobile Devices", IEEE Transactions on Consumer Electronics, Volume 56, No. 2, pp. 713-719, 2010.

[8] Benini, L., Bogliolo, A., and DeMicheli, G., "Policy Optimization for Dynamic Power Management", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 18, No. 6, pp. 813-833, 1999.

[9] Fallahi, A., and Hossain, E.. "QoS Provisioning in Wireless Video Sensor Networks: A Dynamic Power Management Framework", IEEE Transactions on Wireless Communications, Volume 14, No. 6, pp. 40-49, 2007.

[10] Simunic, T., Benini, L., Glynn, P., and DeMicheli, G., "Event-Driven Power Management", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 20, No. 7, pp. 840-857, 2001.

[11] Eui-Young, C., Benini, L., Bogliolo, A., Yung-Hsiang, L., and DeMicheli, G., "Dynamic Power Management for Nonstationary Service Requests", IEEE Transactions on Computers, Volume 51, No. 11, pp. 1345-1361, 2002.

[12] Ren, Z., Krogh, B.H., and Marculescu, R., "Hierarchical Adaptive Dynamic Power Management", IEEE Transactions on Computers, Volume 54, No. 4, pp. 409-420, 2005.

13] Dhiman, G., and Rosing, T.S., "Dynamic Power Management Using Machine Learning", Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 747-754, CA, USA, 2006.

[14] Wang, Y., Xie, Q., Ammari, A., and Pedram, M., "Deriving A Near-Optimal Power Management Policy Using Model-Free Reinforcement Learning and Bayesian Classification", Proceedings of Design Automation Conference, pp. 41-46, CA, USA, 2011.

[15] Dovrolis, C., Thayer, B., and Ramanathan, P., "HIP: Hybrid Interrupt-Polling for the Network Interface", ACM SIGOPS Operating Systems Review, Volumne 35, No. 4, pp. 50-60, 2001.

[16] Khan, U.A., Godec, M., Quaritsch, M., Hennecke, M., Bischof, H., and Rinner, B., "Mobitrick-Mobile Traffic Checker", Proceedings of ITS World Congress, Vienna, Austria, pp. 1-10, 2012.

[17] Khan, U.A., and Rinner, B., "Dynamic Power Management for Portable, Multi-Camera Traffic Monitoring", Proceedings of 18th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 37-40, Beijing, China, 2012.

[18] Khan, U.A., and Rinner, B., "A Reinforcement Learning Framework for Dynamic Power Management of A Portable, Multi-Camera Traffic Monitoring System", Proceedings of IEEE International Conference on Green Computing and Communications, pp. 557-564, Besancon, France, 2012.

[19] Watkins, C.J.C.H., and Dayan, P., "Q-Learning", Machine Learning, Volume 8, Nos. 3-4, pp. 279-292, 1992.

[20] Riedmiller, M., "Neural Fitted Q Iteration - First Experiences With a Data Efficient Neural Reinforcement Learning Method", Machine Learning, Volume 3720, No.1, pp. 317-328, 2005.

[21] Lu, Y., Chung, E., Simunic, T., Benini, T., and DeMicheli, G., "Quantitative Comparison of Power Management Algorithms", Proceedings of IEEE Design, Automation and Test in Europe Conference and Exhibition, pp. 20-26, Paris, France, 2000.

[22] Natarajan, S., and Tadepalli, P., "Dynamic Preferences in Multi-Criteria Reinforcement Learning", Proceedings of International Conference on Machine Learning, pp. 601-608, Bonn, Germany, 2005.

[23] Douglis, F., Marsh, B.D., and Krishnan, P., "Method for Managing the Power Distributed To A Disk Drive in A Laptop Computer", US Patent No. 5,481,733, 1996.

[24] Ramanathan, D., Irani, S., and Gupta, R., "Latency Effects of System Level Power Management Algorithms", Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 350-356, CA, USA, 2000.

[25] Pletzer, F., Tusch, R., Böszörmenyi, L., and Rinner, B., "Robust Traffic State Estimation on Smart Cameras", Proceedings of IEEE Conference on Advanced Video and Signal-Based Surveillance, pp. 434-439, Beijing, China, 2012.

[26] Hwang, C., and W, A.C.H., "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", ACM Transactions on Design Automation of Electronic Systems, Volume 5, No. 2, pp. 226-241, 2000.