Online Learning of Timeout Policies for Dynamic Power Management

UMAIR ALI KHAN and BERNHARD RINNER, Alpen-Adria Universität Klagenfurt, Austria

Dynamic power management (DPM) refers to strategies which selectively change the operational states of a device during runtime to reduce the power consumption based on the past usage pattern, the current workload, and the given performance constraint. The power management problem becomes more challenging when the workload exhibits nonstationary behavior which may degrade the performance of any single or static DPM policy.

This article presents a reinforcement learning (RL)-based DPM technique for optimal selection of timeout values in the different device states. Each timeout period determines how long the device will remain in a particular state before the transition decision is taken. The timeout selection is based on workload estimates derived from a Multilayer Artificial Neural Network (ML-ANN) and an objective function given by weighted performance and power parameters. Our DPM approach is further able to adapt the power-performance weights online to meet user-specified power and performance constraints, respectively. We have completely implemented our DPM algorithm on our embedded traffic surveillance platform and performed long-term experiments using real traffic data to demonstrate the effectiveness of the DPM. Our results show that the proposed learning algorithm not only adequately explores the power-performance trade-off with nonstationary workload but can also successfully perform online adjustment of the trade-off parameter in order to meet the user-specified constraint.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*Design studies*; *Performance attributes*; C.5.3 [**Computer System Implementation**]: Microcomputers—*Portable devices*; I.2.6 [**Artificial Intelligence**]: Learning—*Knowledge acquisition*

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Dynamic power management, reinforcement learning, traffic monitoring, online learning, timeout policies

ACM Reference Format:
U. A. Khan and B. Rinner. 2014. Online learning of timeout policies for dynamic power management. ACM Trans. Embedd. Comput. Syst. 13, 4, Article 96 (February 2014), 25 pages.
DOI: http://dx.doi.org/10.1145/2529992

1. INTRODUCTION

Effective power management has become a major concern especially for mobile devices. Power management techniques both at the hardware and software level have recently arisen to achieve an optimal trade-off between power consumption and performance. The power-managed devices offer a set of operational states which provide different performance at different power consumptions. This set includes active (e.g., processing and idle) and inactive states (e.g., sleep, hibernate, and power down). Dynamic power management (DPM) refers to strategies which selectively change the operational states

This work has been sponsored in part by the Austrian Research Promotion Agency under grant 825840.
Authors' addresses: U. A. Khan (corresponding author) and B. Rinner, Institute of Networked and Embedded System, Pervasive Computing Group, Alpen-Adria Universität, 9020-Klagenfurt, Austria; corresponding author's email: umair.khan@aau.at.

DOI: http://dx.doi.org/10.1145/2529992

of a device during runtime to reduce the power consumption based on the past usage pattern, the current workload, and the given performance constraint. The dynamic power manager decides how much time should be spent in a specific operational state and when to make a transition to a different state satisfying certain objectives, including processing requirements, tasks deadlines, and real-time response. The individual decisions follow a DPM policy which trades performance for power consumption. However, the identification of a policy that minimizes power under performance constraints (or maximizes performance under power constraint) represents a constrained policy optimization problem which is very important for all portable systems [Simunic et al. 2000; Ren et al. 2005].

The motivation of this research is based on our previous work on traffic monitoring [Pletzer et al. 2012; Bischof et al. 2010] and the development of a mobile, multi-camera traffic surveillance system [Khan et al. 2011]. In contrast to most of the existing traffic surveillance systems which are mostly based on fixed installations and large sensors, our portable platform can be easily deployed and used for various monitoring tasks, including law enforcement and construction site monitoring. Our embedded platform is comprised of multiple heterogeneous image sensors and executes high-level (stereo) image processing algorithms to perform a number of traffic analysis tasks, including license plate detection, vehicle detection, and classification [Khan et al. 2012]. Since our system runs on batteries and is intended to autonomously operate for longer periods, DPM techniques must be integrated. The nonstationary workload caused by the road traffic represents a dedicated challenge for the DPM implementation.

Since our system senses and analyzes predominantly nonstationary processes, a traditional requirement of such a system is time-dependent models for modeling and understanding their behavior [Bogdan and Marculescu 2011c], which have also been extensively studied in the context of cyberphysical systems (CPS). CPSs are intelligent systems with sensing, embedded computation, and physical processes tightly coupled [Wang et al. 2011a; Bogdan and Marculescu 2011c]. Modeling the workloads for studying the nonstationary and self-similarity nature of physical processes and their implication in CPS is a very hot topic and actively researched [Bogdan and Marculescu 2011c, 2011b; Barrero et al. 2010]. These modeling approaches analyze the temporal characteristic of nonstationary CPS workloads and construct a model for optimization and control. Although accurate workload modeling has profound implications for power and performance optimization in CPS, it requires rigorous analysis, mathematical formulation, and validation. Moreover, the model constructed for a specific workload (e.g., road traffic workload) cannot be used in a general and broader perspective.

Some approaches [Paul 2013; Bhatti et al. 2010] do not consider modeling the nonstationary workloads but rather learn the environment to choose the best DPM policy with respect to the workload from a given set of policies. However, these techniques are limited to and strongly dependent on the chosen DPM policies.

Although the CPS workloads are nonstationary at a granular level, they exhibit similarity over a long period of time [Bogdan and Marculescu 2011a] and hence can be predicted with a certain degree of confidence. In this article, we propose an online machine-learning-based DPM approach for power/performance optimization and control of a system which is intended to operate in a dynamic environment and required to meet the same challenges as faced by a CPS. The proposed DPM approach is a combination of model-predictive and reinforcement learning techniques for predicting the nonstationary workloads, learning the dynamic environment, and adjusting the DPM decisions accordingly. With this integrated approach, the nonstationary CPS workloads can be successfully dealt with, eliminating the need for any workload modeling.

The scientific contribution in this article can be summarized as follows.

—We introduce a reinforcement learning (RL)-based DPM approach for optimal selection of timeout values in the different operational states of a computing system. In contrast to the existing timeout policies that target only the operational (idle) state of a system, our algorithm also works for the non-operational (sleep) states.
—We estimate the workload by a Multilayer Artificial Neural Network (ML-ANN) and incorporate this information to improve the timeout selection of the RL-based DPM. Our approach relies neither on any offline workload data analysis nor on a priori system model; it is able to online explore and (after some learning time) exploit the power-performance trade-off.
—Our learning algorithm is further able to adapt to the given power/performance constraints online.
—We have completely implemented our DPM algorithm on our embedded traffic surveillance platform and performed long-term experiments using real traffic data to demonstrate the effectiveness of the DPM.

The rest of this article is structured as follows. In Section 2, we provide an extensive survey and critical analysis of the existing DPM approaches and argue how a machine-learning-based DPM can outperform other DPM approaches. This section also compares and distinguishes our work with the existing reinforcement-learning-based approaches for power management. Section 3 describes an overview of our system model. Section 4 provides a detailed problem formulation, and Section 5 describes the workload estimation using an ML-ANN. In Section 6, we show how our RL-based algorithm can successfully explore the power-performance trade-off. Section 7 discusses the online adjustment of the power-performance trade-off parameter to meet a specific power or performance constraint. Section 8 concludes this article.

2. RELATED WORK

The related work on DPM can be classified into timeout, predictive, stochastic, machine learning, and control-theoretic approaches. The next sections describe the existing DPM policies, including a critical analysis.

2.1. Timeout Policies

The timeout policy switches a device to low-power state after it has been idle for a certain time period. Timeout policies can be either *static* or *adaptive* [Lu and De Micheli 2001]. A static timeout policy uses a fixed timeout period whereas an adaptive timeout scheme adjusts the timeout period according to the idle periods' history.

Several timeout policies have been studied in the literature. An adaptive timeout policy proposed in Douglis et al. [1995] adjusts the timeout period by the ratio of the current and the previous timeout period. Olsen and Narayanaswarni [2006] propose an OS-directed timing scheme to eliminate the periodic system timer ticks associated with the idle state of the operating system to reduce power consumption. When there are no more tasks to execute in the active state, the system is switched to a low-power state and a timeout value is determined. However, when there are some tasks to be executed in the idle state, the normal periodic system timer is restored.

Shih and Wang [2012] propose an adaptive timeout scheme to adjust the timeout period with the bursty request arrival patterns. The proposed scheme first derives the average idle time of a system in the bursty period and non-bursty period separately. To increase power saving, it uses the average idle time in the bursty period to adjust the timeout value. In the non-bursty period, the average idle time is used to decide which low-power state the system should be switched to.

With a critical analysis of the existing timeout policies for power management, the following conclusions can be drawn.

—Without a proper anticipation of workload, timeout policies result in wasting a significant amount of power waiting for the timeout to expire.
—Timeout policies work at the idle state(s) of a device and do not deal with the non-operational (sleep or power-down) states.
—Timeout policies do not provide mechanisms for controlling the trade-off between power consumption and performance.

2.2. Predictive Policies

Predictive policies work on a system model that is learned from historic information in order to best adjust themselves to the system dynamics. The basic idea of predictive policies is to forecast the length of idle periods and switch the device into a low-power mode when the predicted idle period is longer than a certain threshold. The relevant literature describes several predictive policies for power management.

The first work in predictive policies [Srivastava et al. 1996] uses nonlinear regression over the history of past idle periods to predict the length of the next idle period. The limitation of this approach is that it requires an offline data collection and analysis to construct and fit the regression model [Benini et al. 2000]. Moreover, it does not address the performance penalty.

Another approach [Chung et al. 1999], which is able to control multiple sleep states, uses adaptive learning trees to transform sequences of idle periods into discrete events and stores them into tree nodes. This algorithm predicts idle periods using finite-state machines to select a path which resembles previous idle periods. However, this algorithm also does not consider the system's performance.

Hwang and Wu [2000] propose two improvements over the previous work to deal with workload uncertainty to a certain level: (i) this approach uses online exponential weighted average of previous idle periods to predict the next idle period; (ii) at the end of the predicted idle period, it performs a predictive wakeup to decrease the delay for servicing the first incoming request after an idle period.

A more recent and advanced work [Young et al. 2010] describes a predictive shut down method based on collecting and analyzing information on the access patterns to I/O devices. The trace of function calls is monitored until the processor initiates an access to a certain I/O device. By analyzing the pattern of program counter values, the next I/O access time is predicted and the device is shut down if the predicted time is longer than a certain threshold. When a certain access pattern at an I/O device is observed, the device is woken up (in advance) before the actual I/O access requests take place in order to reduce the performance penalty.

Predictive policies share the following limitations.

—Predictive approaches do address workload uncertainty, but they assume deterministic response and transition times for the system [Benini et al. 2000].
—Predictive policies work at idle state(s) of a device. The only option in a non-operational state is to wake up the device as soon as a request arrives. Hence, predictive policies cannot deal with general system models where multiple incoming requests can be queued before processing when the device is in the off or sleep state.
—Predictive policies do not offer a possibility to control the trade-off between power consumption and performance.

2.3. Stochastic Policies

Some of the limitations (uncertainty, queuing, power-performance trade-off) of predictive policies are addressed by stochastic policies. These policies make probabilistic assumptions about the usage pattern of a device and exploit the nature of the probability distribution to formulate an optimization problem, which derives an optimal DPM strategy [Fallahi and Hossain 2007]. The device states and request queues in stochastic policies are generally modeled as a Markov decision process (MDP).

A discrete-time MDP-based stochastic power management policy [Benini et al. 1999] introduces an abstract system model for the formulation of a linear optimization problem whose solution is computed in polynomial time. Due to the discrete nature of MDP, this approach has two drawbacks: (i) it requires observing and issuing power commands at regular time-discretized intervals which results in an additional power cost; (ii) it assumes a stationary geometric distribution for all state transitions which is not true in many practical cases [Simunic et al. 2000]. These issues are addressed by Qiu and Pedram [1999] with a continuous-time MDP to issue power commands upon event occurrences instead of regular discrete intervals. The state transitions in this work are assumed to follow an exponential distribution which is a better assumption than the geometric distribution, but exclusively fit to the models where the request arrival times are exponentially distributed. This problem is further addressed by Simunic et al. [2001] with a semi-Markov model which can treat a general distribution occurring at the same time with an exponential distribution. However, none of these approaches is able to handle nonstationary request patterns.

In order to adapt to nonstationary requests, some approaches model the request arrival distribution and analyze the history of past request arrival times to estimate the current workload [Eui-Young et al. 2002; Ren et al. 2005]. These models are still based on discrete-time MDP which is not a realistic model for a request process. These models expose two drawbacks: (i) a prior modeling of request arrival distribution is required, and (ii) these models are memory- and computation-expensive [Shih and Wang 2012]. A recent work [Durand et al. 2012] uses continuous-time MDP for the power management of printers. However, this approach also requires modeling the sequence of requests from a statistical point of view.

Stochastic policies do provide a flexible way to control the power-performance tradeoff. However, these policies share the following limitations.

—Stochastic policies require a rigorous modeling of the system and the system-specific formalization of the optimization problem. Therefore, they are model dependent.
—Solving the stochastic optimization problem to find the optimal DPM strategy either in a time-driven or event-driven manner increases the complexity [Irani et al. 2003].
—The selection of a distribution for modeling the request pattern is problem-specific and is not realistic in most of the real-world applications.
—Stochastic policies that can deal with nonstationary requests are memory- and computation-expensive.

2.4. Machine Learning Policies

Several approaches apply machine learning to learn the request arrival patterns for dynamic power management. Fei et al. [2006] use a genetic algorithm to predict idle periods. However, this approach shares the same limitations as the aforementioned predictive policies.

Supervised learning approaches for dynamic power management [Mannor et al. 2006; Hwisung and Pedram 2010] use Bayesian classifiers to predict the system performance state from some readily available features (e.g., sensor measurements, priority of an incoming request, and occupancy state of service queue) and then use this predicted state to look up the optimal power management action from a precomputed policy table. These approaches require offline data collection and training of a classifier.

The effort involved in the offline data preparation and training is comparable to the one needed to model a system in stochastic policies.

Recently, RL-based approaches have gained increasing attention due to their simplicity and interaction to a system in a natural way. RL-based approaches do not require the construction of a system model. Instead, they learn the 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 approaches proposed by Dhiman and Rosing [2006] and Prabha and Monie [2007] use a set of pre-selected DPM policies for controlling power consumption under different workloads. These policies are referred to as *experts*. The algorithm associates and maintains relative weights for each expert which reflect its individual performance. The weights are adjusted online, and at any point in time, the best performing expert is selected with highest probability. These approaches do lead to an optimal DPM policy, but they are dependent on and limited to the chosen experts. Tan et al. [2009] propose a constrained RL algorithm for DPM using the same MDP-based generic system model as proposed in Benini et al. [1999]. This is also a model-free approach that does not require any pre-selected experts. However, the size of the state space used in this algorithm is quite large which results in increased complexity. Additionally, this work is based on a discrete-time model of the stochastic process and thus has a large computational overhead in real implementations. Following the merits of this work, Wang et al. [2011b] propose a modified, continuous-time RL algorithm by incorporating a Bayesian-classifier-based workload estimation to guide the learning algorithm for changing workloads. They implement this prediction approach on the real data of a WLAN card which performs well for this setting due to the regular traffic. Nevertheless, the same workload prediction can not render acceptable accuracy with nonstationary workloads (e.g., road traffic workload).

We address this issue in our prior work [Khan and Rinner 2012a; Khan et al. 2012] by proposing a model-free, RL-based DPM algorithm for nonstationary workloads. We use an 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 idle state and waits for the service queue to be populated with a certain number of requests. The decisions of timeout values and the time spent in sleep state are controlled by an adjustable power-performance trade-off parameter. Workload estimation using an ML-ANN achieves higher accuracy with the nonstationary data and the algorithm is capable of exploring the trade-off in the power-performance design space. However, the upper and lower bounds of average latency in request processing cannot be ascertained in this approach. Moreover, since the algorithm waits for a certain number of requests in the queue, a drawback of this approach is the high latency in request processing when the workload drops abruptly.

RL-based DPM approaches have the following advantages.

—These approaches are simple, easy to implement, and computationally efficient.
—No (a priori) system model is required.
—The power-performance trade-off can be flexibly controlled.
—Nonstationary workloads can be efficiently dealt with.
—Online learning and policy implementation can take place in parallel.

However, RL-based approaches do face the following generic challenges.

—The large state space results in large computational overhead. Therefore, the state space should be designed carefully and kept as small as possible.
—The convergence (speed) of the RL algorithm is an issue in many problems. However, this problem can be overcome by optimal design of state space, balanced selection of

random versus deterministic actions, and/or multiple states update with a slightly increased overhead [Khan and Rinner 2012b].

2.5. Control-Theoretic and Model Predictive Approaches

Several approaches apply predictive and machine learning techniques with the combination of control theory and model predictive control (MPC) for the DPM of multiprocessor systems-on-chip (SoC) and embedded systems. Moser et al. [2010] studied the application of MPC in designing three controllers for the DPM of an energy-harvesting embedded system: (i) application rate controller; (ii) service-level allocator, and (iii) real-time transmission scheduler. The approach targets the DPM of wireless sensor networks powered by solar cells. A predictive technique is used to estimate the future energy harvesting to optimize the system performance. This estimation is further incorporated into an online scheduler to control the application. Although this technique outperforms other types of controllers used in the same perspective, it is based on modeling a large variety of application scenarios, constraints, and optimization objectives.

Several control-theoretic approaches use MPC for dynamic voltage and frequency scaling (DVFS). Kandasamy and Abdelwahed [2004] propose a control-theoretic approach for the DPM of embedded processors. The proposed technique is based on an online MPC which predicts the future behavior of a processor over a finite look-ahead interval and derives the lowest possible frequency-voltage settings subjected to several QoS constraints. This technique also requires a mathematical model of the processor operation.

Another technique proposed in Ghasemazar et al. [2012] addresses the problem of DVFS for chip multiprocessors (CMPs) for performance optimization of heterogeneous multiprocessor systems subjected to a total power budget and the die temperature estimated by a moving-average-based predictive method. A convex optimization-based algorithm is formulated and solved online for core consolidation, application assignment, and producing optimal DVFS settings. This approach is based on several models and assumptions that do capture the first-order effects which are important to the problem but are not the most accurate and realistic models and may ignore some second-order effects.

Some approaches use voltage-frequency island (VFI) partitioning for the DPM of multiprocessor and multicore embedded platforms. These techniques combine DVFS on voltage islands to further exploit temporal workload variations of each island. Bogdan et al. [2012] propose an optimal control method based on fractional calculus to manage the power consumption of VFI-based systems. This approach enables the system to capture fractality and nonstationary characteristics of a workload. It considers both processing elements and routers in the control scheme for voltage-frequency settings and allows for directly optimizing a certain performance metric subjected to some fractional derivatives state equations (i.e., for queue utilization) and bounded control signals (i.e., operating frequencies). However, this approach does not provide a flexibility to control power-performance trade-off.

The techniques proposed in David et al. [2012] and Ogras et al. [2009] use online algorithms for DVFS. These techniques solve a feedback control problem by monitoring the communication between different islands and adjusting the cores voltage and frequency levels to save power while maintaining the required performance imposed by application constraints. These approaches provide different solutions of power versus performance and runtime versus accuracy.

2.6. Our Work in Comparison with Related Work

A qualitative comparison of the DPM approaches is summarized in Table I. The DPM approach proposed in this article is a continuous-time, event-driven, RL-based timeout

Table I. Comparison of DPM Techniques

	Performan Con-straint	Power-Performan Tradeoff	Workload Han-dling	Queuing Model	Model-Free	Non-Stationar Work-load	Offline Analy-sis	Complexity
Static Timeout	✖	✖	✖	✖	✔	✖	✖	Low
Adaptive Timeout	✖	✖	✔	✖	✔	✖	✖	Low
Predictive Shutdown	✖	✖	✔	✖	✔	✔	✖	Low
Predictive Wakeup	✔	✖	✔	✖	✔	✔	✖	Low
Stochastic	✔	✔	✔	✔	✖	✔	✔	High
Evolutionary	✖	✖	✔	✖	✔	✖	✖	Low
Supervised Learning	✔	✔	✔	✖	✔	✔	✔	Medium
Online RL	✔	✔	✔	✔	✔	✔	✖	Medium

approach. We explore a deeper trade-off between the power consumption and performance. Therefore, in contrast to the existing DPM algorithms (timeout, predictive) which aim for an immediate response as soon as a request arrives in sleep state our DPM algorithm uses timeout values both in idle and sleep states so that the computing device does not necessarily wake up immediately to process a request if the request arrives before the sleep timeout expires. This allows delaying the request processing to save more energy and avoid more frequent state transitions. To the best of our knowledge, using timeout periods both in operational and non-operational states of a computing device has never been studied in the literature.

Our DPM approach has the following advantages.

—The proposed approach does not require any offline data analysis or system modeling.
—The wakeup decisions of the aforementioned DPM approaches that can deal with queuing models [Benini et al. 1999] are based on the queue occupancy. These approaches result in poor system response in the case of low workload. However, using timeout values in sleep states delivers an improved performance.
—We use an ML-ANN to estimate the workload and incorporate this information to the learning algorithm in order to use optimal timeouts in both idle and sleep states.
—The power/performance trade-off can be adjusted by a selectable parameter.
—Our online RL-based algorithm is able to adapt to a given constraint of power consumption or latency.

3. IMPLEMENTATION PLATFORM

Our traffic surveillance system has a heterogeneous setup with different types of sensors, each having different capabilities. The two main components of the sensing platform comprise visual sensors (smart RGB, grayscale, infrared, high-resolution RGB) and an Intel ATOM-based processing platform for image processing. This heterogeneous setup serves many purposes: (i) the tasks can be distributed among different sensors; (ii) different light spectrums can be efficiently covered; (iii) different focal

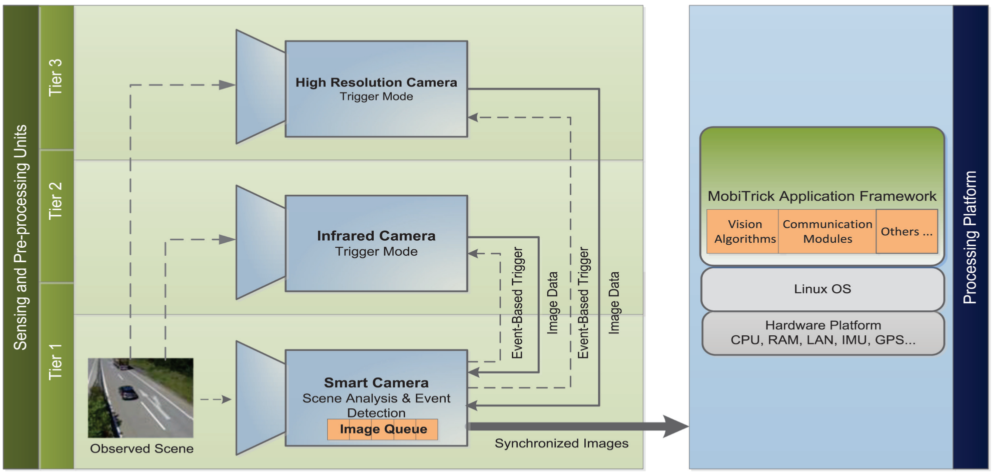


Fig. 1. High-level architecture of the traffic surveillance system.

lengths provide different views of the scene (e.g., while one sensor gives an overview of the scene, the other delivers an enlarged view of the object); (iv) the redundant information from multiple sensors helps to increase reliability. With the combination of these advantages, the setup enables full 3D reconstruction to perform measurements on the objects under surveillance. Figure 1 provides a high-level view of our sensing platform. The sensing platform has a multitier architecture where the sensors reside at different levels based on their energy consumption and capabilities. A multitier framework leverages the heterogeneity of sensors to allow their on-demand exploitation. In such a setup, high-power and more capable sensors can operate at higher levels in an event-driven manner. These higher-level sensors can be triggered to sense an event by the power-efficient sensors running at lower levels in continuous mode. In our setup a smart, low-power color camera that runs on-board algorithms for event detection operates at the third (i.e., lowest) level. The smart camera triggers other cameras at the higher levels in case of detected events. When triggered, the higher-level cameras capture and transfer synchronized images to the smart camera where they are temporarily buffered in a queue. The queued images from all cameras are then periodically sent to the processing platform for processing.

All sensing and processing elements have been deployed with the perspective of low-power hardware design. While one of the sensors operates in continuous mode, the other (higher-level) sensors only operate in trigger mode, where they capture and transfer images after reception of a trigger signal or stay in the waiting state otherwise. The waiting state of these sensors is the only configurable energy-saving state which has less power consumption than the image-capture and the image-transfer states. A complete shutdown of these sensors and waking them up at the detection of an event causes huge latency in initialization and establishing network connections. Therefore, such a configuration is not appropriate for capturing events in real time. Hence, our aforementioned sensor configuration is already power- and latency-efficient and does not provide a mandate for additional power savings with a software framework. However, the processing platform, which is the major contributor to the overall power consumption in the entire system, does have different power saving modes. Due to the power-efficient hardware design, targeting individual components (processor, memories, buses, etc.) of the processing platform will be of limited use. Instead, we are committed to enable aggressive power management strategies that encompass the

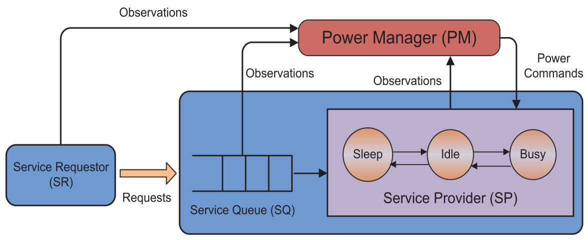


Fig. 2. Generic DPM model.

entire system. Therefore, the contribution of an efficient DPM policy in this platform is to determine at what point in time it is appropriate to switch the processing platform to a low-power state, considering the power-latency cost associated to the state transitions and current rate of event occurrence. Likewise, the DPM policy must also determine when the processing platform should be put to the operational state to process the queued images so that both the power consumption and performance remain at an optimal level. Since the performance can be compromised for higher energy savings (or vice versa), the aforementioned DPM decisions should also reflect a user-specified bias for the two objectives.

4. ONLINE LEARNING OF TIMEOUT POLICIES

Our platform adopts a generic system model for DPM which was originally proposed for stochastic policies [Benini et al. 1999]. This model (Figure 2) provides a generalized way to control the power-performance trade-off and deals with systems where the requests can be queued before processing. In this abstract model, the *service requestor* (SR) is the (software) application that generates requests which are buffered in the *service queue* (SQ) before processing. The *service provider* (SP) is the request processing device which can be in any of the available power states (e.g., busy, idle, sleep, etc.) in any point in time. The *power manager* (PM) is the dynamic power management algorithm that decides which power mode the SP should be switched to.

In our implementation platform, the event detector serves as the SR, the image queue maintained in the smart camera represents the SQ, the processing platform represents the SP, and the PM (i.e., the learning agent that issues appropriate power commands to the SP) resides on the smart camera. The workload estimator, running on the smart camera, is based on an ML-ANN which receives the detected events as input from the SR. In this setup, the smart camera works as a controller on the sensing platform that issues control signals (trigger, power, and data transfer commands) to other components of the system. Figure 3 shows a low-level view of the sensing platform.

The processing platform has several power modes, including busy (i.e., when some requests are being processed), idle (i.e., waiting for requests), sleep (i.e., standby), hibernate, and off. Since hibernate and off states have a long latency to an operational state, we consider only busy, idle, and sleep states in the learning algorithm. However, our algorithm can be easily extended to a processing platform having more power states. The power and state-transition delay characteristics of the processing platform are given in Table II. 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 board from sleep to idle state and vice versa, respectively. The transitions from busy to idle (and vice versa) are assumed to be instantaneous and autonomous. Given the characterization of the processing platform and an estimation of the workload, we can design an RL-based DPM algorithm which converges to an optimal DPM policy based on a selected power-performance criteria.

Table II. Power and Delay Characteristics of the Processing Platform

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

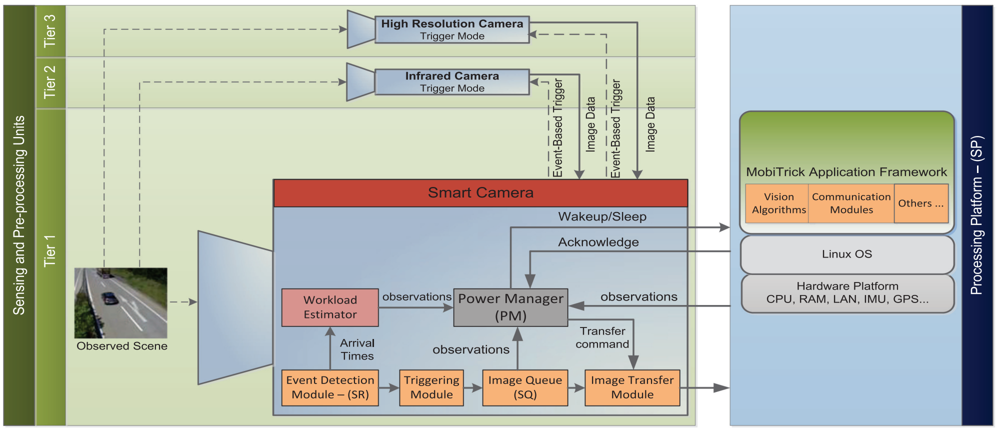


Fig. 3. Architecture of the DPM on the processing platform.

At each decision time, the PM receives an observation that includes the estimate of the SR (i.e., low/high workload), the state of the SQ (i.e., number of requests waiting in the image queue), and the state of the SP (i.e., power state of the processing platform). Based on these observations, the PM issues an appropriate DPM action. Therefore, each system state, S, has a composite form $S = (SR, SQ, SP)$, where $SR = \{0, 1\}$ is the current workload estimate (low, high), $SQ = \{0, 1, 2, \ldots, q\}$ is the number of requests in the image queue, and $SP = \{sleep, idle, busy\}$ is the power state of the processing platform. We apply a state aggregation in the design space for limiting the values of the SQ to a state having no requests in the queue and the one having some requests, that is, $\forall sq \in SQ, sq = \{0, q | q \in \mathbb{N}\}$. Aggregating SQ states not only reduces search space but also contributes to speed up the convergence of the learning algorithm.

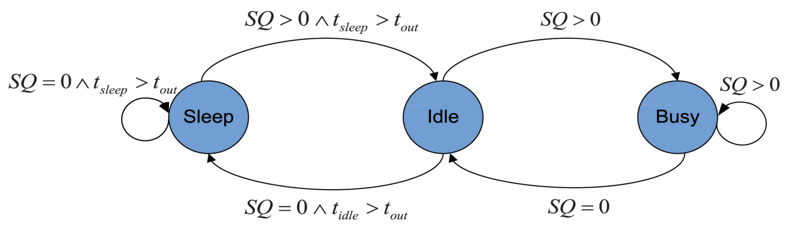
In each (composite) state, the available actions comprise a set A of pre-selected discretized timeout values t^k_{out} which depend on a selected threshold T_{thr} and are defined as

$$A = \left\{t^k_{out}\right\} = \{\epsilon_k T_{thr}\}, \quad \epsilon_k \in \mathbb{R}^+, \quad k = 1, 2, \ldots, n, \tag{1}$$

where ϵ_k is a positive weight and T_{thr} represents a threshold time period (break-even time) which can amortize the power-performance cost associated to the state transitions. The break-even time for a system depends on its power model. The same threshold period is used for predicting the next request arrival time by the ML-ANN workload estimator. For our implementation platform and traffic workload, we use $T_{thr} = 10$ seconds to classify the predicted workload as high if the next interarrival time is predicted to be less than T_{thr} (or low otherwise). However, adapting T_{thr} during operation would be an interesting area for future research.

The discretization step and the size of the action set can be selected by the user. In general, the action set can comprise timeout values, such as

$$A = \{0, 0.1T_{thr}, 0.2T_{thr}, \ldots, T_{thr}, 1.1T_{thr}, \ldots, nT_{thr}\}. \tag{2}$$



$SQ > 0 \wedge t_{sleep} > t_{out}$
$SQ > 0$
$SQ = 0 \wedge t_{sleep} > t_{out}$
Sleep
Idle
Busy
$SQ > 0$
$SQ = 0 \wedge t_{idle} > t_{out}$
$SQ = 0$

Fig. 4. System state-transition diagram.

The problem of selecting timeouts can be formulated as follows. When the SP enters the idle state (after it has processed all the requests in the image queue) or the sleep state, it waits for a certain timeout before the next state transition occurs. There are two cases possible.

Case 1. If there are some requests in the SQ in sleep state, the SP is woken up at the end of the timeout period to process the requests. Otherwise, the SP stays in sleep state for another timeout period.
Case 2. The SP enters the idle state and waits for requests during the timeout period. If some requests arrive, they are immediately executed followed by another timeout period if the SQ is empty. If no requests arrive during the timeout period, the SP is switched to sleep state.

The decisions of selecting timeout periods and switching the power state of the SP are taken by the PM. These decisions are taken when the last selected timeout expires or the state of the SQ changes. Figure 4 shows the state-transition diagram of the system. t_{out}, t_{sleep}, and t_{idle} in the diagram represent the timeout period and the time spent in the sleep and the idle states, respectively.

The selection of timeout values in sleep and idle states is subjected to two objectives, that is, minimizing power consumption and maximizing performance. We consider 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 [Natarajan and Tadepalli 2005]. 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. Our multi-objective optimization problem involves the simultaneous optimization of two conflicting objectives such that any improvement in one objective results in a possible degradation of the other objective. Therefore, we do not have a unique optimal solution, but rather a set of non-inferior, alternative solutions which can be obtained by varying the relative weight between the objectives. The solutions thus obtained are called Pareto-optimal solutions, and a plot of the entire Pareto set in the design objective space represents a Pareto-optimal trade-off.

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 (3).

$$c_t(s, a, \omega) = (1 - \omega)p_t(s, a) + \omega l_t(s, a). \tag{3}$$

$p_t(s,a)$ 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 trade-off parameter. The value of ω can be varied from 0 to 1 to obtain the Pareto-optimal trade-off curve. ω also determines the preference/precedence of the two objectives. If the power consumption is assigned more weight, performance takes precedence (or vice versa).

The decisions of selecting timeout values and/or changing power states are taken in the following scenarios.

idle -> idle. The SP was in idle state and the SQ transits from 0 to N. In this case, the SP immediately processes the new requests and enters the idle state again.
idle -> sleep. The SP was in idle state and no request arrived during the timeout period.
sleep -> sleep. The SP was in sleep state and no request arrived during the timeout period.
sleep -> idle. The SP was in sleep state and the SQ transits from 0 to N. At the end of the timeout period, the SP changes to idle state.

Based on these scenarios, the cost function has the following structure.

$$c_{i\to i}(s,a,\omega) = (1-\omega)p_t(s,a) + \omega l_t(s,a), \tag{4a}$$
$$c_{i\to s}(s,a,\omega) = (1-\omega)p_t(s,a) + \omega l_{avg}^2 e^{-\beta t_{out}}, \tag{4b}$$
$$c_{s\to s}(s,a,\omega) = (1-\omega)p_t(s,a), \tag{4c}$$
$$c_{s\to i}(s,a,\omega) = (1-\omega)p_t(s,a) + \omega l_t(s,a). \tag{4d}$$

Equations (4a) and (4d) represent the standard cost function as defined in Equation (3). Note that in self-transition *sleep -> sleep* case, we do not have an immediate latency value to include in the cost function (Equation (4c)). Similarly, when the SP transits from idle to sleep state, there are no requests processed in the last state. For this case, we incorporate the quadratic value of overall average latency, l_{avg}, in the cost function (to make this cost latency aware). Since the quadratic value of overall average latency in Equation (4b) takes a higher value, we relax it by the factor $e^{-\beta t_{out}}$, where $\beta \in (0,1)$ and t_{out} is the timeout period spent in the idle state before transiting to sleep state. The constant β just normalizes the timeout value so that the relaxing factor may not be too large or too small due to the negative exponential function. Based on our experiments, we select $\beta = 0.5$ which adequately normalizes the value of the relaxing factor.

A 1-step Q-learning algorithm [Watkins and Dayan 2011] can be used to approximate the state-action values with the evaluated cost. The approximated state-action values provide a long-term judgement about selecting an action in a specific state. In Q-learning, policies and the value function are represented by a two-dimensional lookup table indexed by state-action pairs. For all $(s,a) \in S \times A$, the Q-learning principle is given in Equation (5),

$$Q^{(t+1)}(s_t,a_t) = Q^t(s_t,a_t) + \alpha_t(s_t,a_t)\Big[r_{t+1} + \gamma \max_{a'} Q^{(t+1)}(s_{t+1},a') - Q^t(s_t,a_t)\Big] \tag{5}$$

where $\alpha_t(s_t,a_t) \in (0,1)$ is the learning rate which is adjusted properly (i.e., slowly decreased) during the learning process. $Q^\pi(s,a)$ for each state-action pair represents the expected long-term reward if the system starts from state s, takes action a, and thereafter follows policy π. The positive constant $\gamma \in (0,1)$ is called discount factor, which determines the importance of the reward.

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 ($\omega \to 0$), the learning algorithm should select higher timeouts in sleep state and smaller timeouts in idle state to save power, but a higher preference to the latency should follow an opposite behavior. Therefore, the calculated cost of an action and the best value of the next state must be discounted with appropriate factors to learn the optimal timeout values subjected to a selected preference. We introduce two discount factors for the Q-learning update rule defined in Equation (5).

$$d_1 = (1 - e^{-\beta t_{out}}), \quad \beta \in (0, 1), \tag{6a}$$

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

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

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

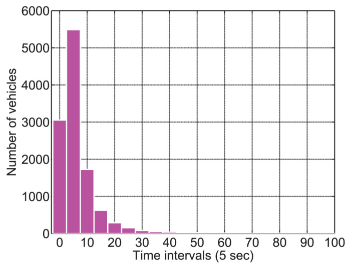
The best next-state value in Equation (7) is the one with the minimum cost. The two discount factors d_1 and d_2 account for the opposite behavior of timeout selection in idle and sleep states.

5. WORKLOAD ESTIMATION

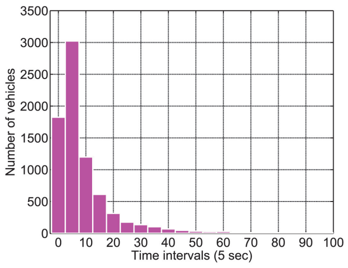
In our system model, the workload (or the request rate) comprises the rate of images captured by all the cameras at the detection of events (vehicle detection). Therefore, the workload reflects the arrival rate of the vehicles. We used real workloads for the learning algorithm by taking several test recordings on different highways, where we recorded the interarrival times of vehicles by a vehicle detection algorithm [Pletzer et al. 2012]. Figure 5 shows the interarrival times distribution (spaced by an interval of 5 seconds) of four different test recordings on different locations. The collected data shows that the interarrival time has a maximum probability within the interval of 5 to 10 seconds. Therefore, it is appropriate to set the threshold T_{thr} to 10 seconds to classify the current workload as low or high. If the interarrival time of the next request is estimated to be less than or equal to 10 seconds, the workload is classified as high (or low otherwise).

It is worth mentioning that the workloads depicted in Figure 5 represent the long-term accumulated sum of interarrival times in different intervals. The long-term and self-similar accumulated behavior is an intrinsic property of all CPS workloads [Bogdan and Marculescu 2011a]. However, at a granular level, the CPS workloads do appear differently and exhibit nonstationarity, which is the major addressable issue in our proposed DPM technique. Disregarding the type of distribution, our proposed DPM technique addresses the nonstationarity of the workloads, which can be exhibited by any workload distribution.

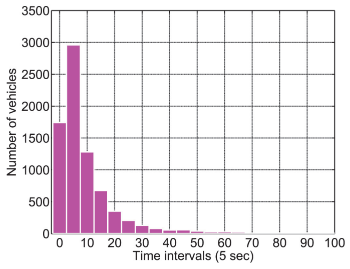
For workload estimation, we use a fix-sized moving window on the history of previous interarrival periods and input these interarrival periods (normalized between 0 and 1) to the ML-ANN. The ML-ANN estimates the length of the next interarrival period (denormalized). Figure 6 shows the 3-layer ANN-based workload estimator, where v_{ij} denotes the weight of the connection between the jth neuron from the input layer and the ith neuron of the hidden layer. Whereas, w_i represents the weight of the connection between the i_{th} neuron from the hidden layer and the neuron of the output layer. The



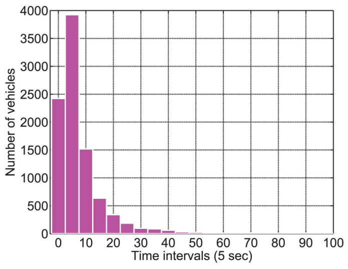
(a) Workload 1.



(b) Workload 2.



(c) Workload 3.



(d) Workload 4.

Fig. 5. Distribution of interarrival times from different test recordings.

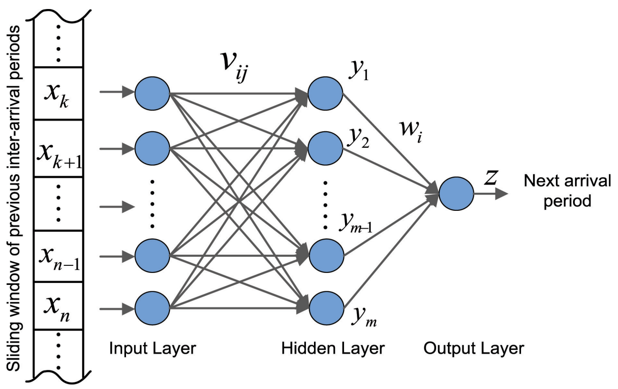


Fig. 6. ML-ANN-based workload estimator.

outputs of the input layer, hidden layer, and output layer, represented by I_i, h, and z, are calculated as follows:

$$I_i = \sum_{j=1}^{n} v_{ij} x_j, \quad i = 1, \ldots, m, \tag{8a}$$

$$h = \sum_{i=1}^{m} w_i f(I_i), \tag{8b}$$

$$z = f(h), \tag{8c}$$

where f is the sigmoid function. The weights are adjusted according to the following back-propagation rule [Phit and Abe 2006]:

$$\Delta v_{ij}(t) = \eta(u - z)w_i x_j f^{'}(h) f^{'}(I_i) + \mu \Delta v_{ij}(t-1), \tag{9a}$$

$$\Delta w_i(t) = \eta(u - z)y_i f^{'}(h) + \mu \Delta w_i(t-1), \tag{9b}$$

$$f^{'} = f(1 - f), \tag{9c}$$

where u is the observed interarrival period, $\eta = 0.05$ is the learning rate, and $\mu = 0.5$ is a positive constant which is determined experimentally. The derivative of sigmoid function f' determines the direction of gradient descent to minimize the estimation error. The experimental size of the moving window (and the number of input neurons) is selected to be $n = 8$. The number of hidden layer neuron is taken as $m = 13$. The prediction accuracy of the online ML-ANN workload estimator, which is defined as the ratio of correctly predicted intervals to the total number of predicted intervals [Zafra et al. 2011], is 81.24%.

The values of n and m in the ML-ANN do determine a desired level of accuracy. If the value of n is too small, the decisions of ML-ANN are biased towards the short-term variations in the workload which can be caused only by a single interarrival time. On the other hand, if the value of n is too large, the workload estimation focuses on a relatively longer history of interarrival times, and hence a real workload change cannot be captured immediately. The values of n and m selected in our learning framework are based on our experiments. The achieved average prediction accuracy of 81.24% is sufficient for our learning framework.

6. EXPLORING POWER-PERFORMANCE TRADE-OFF

In each decision time, the PM finds the system in sleep, idle, or busy states. In sleep and idle states, 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 by assigning it a cost (Equation (3)) 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 overall learning algorithm is described in Algorithm 1. T represents the state-transition matrix which keeps track of the recent history of visited (composite) states, actions, respective costs, queue occupancy, and other parameters. The learning matrix Q, transition matrix T, and the probability

ALGORITHM 1: RL-Based Timeout Policy

```
Input: Power-performance parameter $\omega \in (0, 1)$
Initialize $Q$, $T$ and probability matrix $p_r$ randomly;
repeat
    Obtain the current workload estimation;
    Get the current observations: $(SR, SQ, SP)$;
    Calculate action probabilities: $p_r^k(s, a_k)$;
    Select an action, $a$, with probability $p_r$;
    Execute the selected action;
    Calculate the cost of the last action: $c_{t+1}(s, a)$;
    Update the learning rate: $\alpha_t(s, a)$;
    Update $T$ with the new state-action pair;
    Update Q-value of the visited state-action pair: $Q^{t+1}(s, a)$;
until specific number of iterations or a certain threshold of error estimate is achieved;
Output: The policy: $\pi = min_a Q(s, a), \forall s \in S, a \in A$
```

Table III. Characteristics of Different Traffic Workloads

Workload	Mean Inter-Arrival Time	No. of Requests	Duration
Workload 1	6.79 sec	11,649	22 hours
Workload 2	11.13 sec	7,762	24 hours
Workload 3	11.07 sec	7,803	24 hours
Workload 4	9.06 sec	9,502	24 hours
Workload 5	12.05 sec	7,155	24 hours

matrix p_r are randomly initialized derived from a uniform distribution. The sizes of these matrices depend on the size of the state space. In our implementation framework, the sizes of matrix Q and p_r are 6×20 (i.e., 6 composite states and 20 actions), respectively. The size of matrix T is 8×8. For the larger and more complex systems, the sizes of these matrices will increase linearly with the size of the state space which will only (slightly) affect the convergence speed of the algorithm due to the larger state space. The larger state space and the increased sizes of these matrices will also require slightly larger amounts of memory. Nevertheless, these matrices are not affected by the traffic complexity.

The learning rate $\alpha_t(s, a)$ reflects the degree to which a state-action pair has been chosen in the recent past. For each state-action pair, it is decreased slowly and calculated as

$$\alpha_t(s, a) = \frac{\xi}{visited(s, a)}, \tag{10}$$

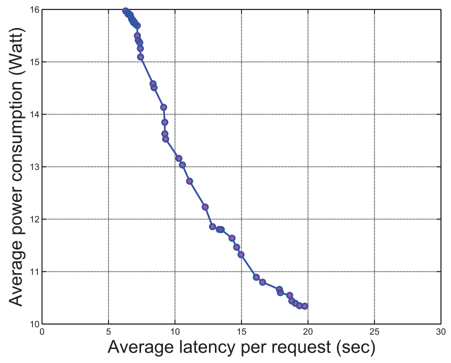
where $\xi \in (0, 1)$ is a positive constant. Every time a state-action pair (s, a) is visited with this learning rate, the difference between its estimated Q-value $Q^{(t+1)}(s, a)$ and the current Q-value $Q^t(s, a)$ reduces. Hence, for all state-action pairs, the algorithm converges to a timeout policy. The value of ξ determines the learning rate. If this value is too large, the learning rate is relatively slow. On the other hand, a too-small value makes the learning rate faster, which cannot fully capture the dynamics of the system, thus leading to a local minimum for a state-action pair. We select $\xi = 0.25$ and find that based on the frequency of visiting a state-action pair, this value provides a moderate learning rate.

In order to balance exploration and exploitation, we use a semi-greedy exploration policy [Sutton 1990] in our learning algorithm. This policy starts out with selecting random actions (exploration) which are equally distributed. 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_r^k, in a state are given by Equation (11):

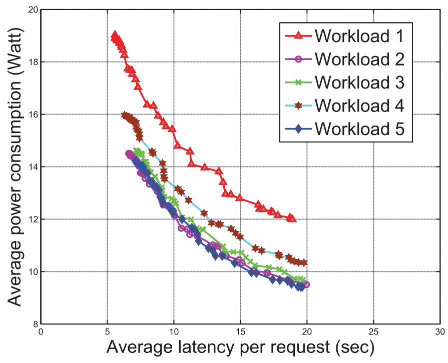
$$p_r^k(s, a) = \frac{e^{\frac{Q^t(s,a_k)}{\tau}}}{\sum_{k=1}^{n} e^{\frac{Q^t(s,a_k)}{\tau}}}, \quad \forall a \in A, \; n = |A|, \tag{11}$$

where τ is called a temperature coefficient. It 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 exploiting.

We tested the algorithm for different workloads of road traffic to explore the power-performance trade-off. The characteristics of these workloads are given in Table III. For each workload, we vary the relative weight ω between power and latency in the cost function (Equation (3)) from 0 to 1 and obtain a number of Pareto-optimal solutions. Figure 7(a) shows the power-performance trade-off curve of workload 4, and



(a) Power-performance trade-off curve (workload 4).



(b) A comparison: workloads 1–5.

Fig. 7. Pareto-optimal trade-off curves of different workloads.

Figure 7(b) depicts its comparison with the power-performance curves of other workloads mentioned in Table III. Note that the power-performance trade-off curves of all workloads follow the same trend. However, the power profile of each workload depends on the mean interarrival time (MIAT) of the requests. In the case of shorter MIATs, the SP has to spend more time in idle and busy states. Since workloads 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 workloads 2, 3, and 5. Therefore, the power profiles of these workloads are shifted upward.

It is worth mentioning that the power profile does not have a short-term correlation with the interarrival times, but rather depends on the MIAT measured over a long period. In environments where the workload is higher, the average power consumption increases because the algorithm has to select timeout values such that the performance is not significantly degraded and the incoming requests (arriving at a higher rate) may be processed with a reduced latency.

Our proposed algorithm also ascertains the upper and lower bounds of the latency in request processing, which are given as follows.

$$[l_{min}, l_{max}] = [t_p, (t_{i2s} + t_{sleep} + t_{s2i} + t_p)]. \tag{12}$$

In this equation, the minimum latency l_{min} can be as small as the processing time t_p of the request. On the other hand, suppose that the PM issues a sleep command to the SP and a request arrives at the same time. The maximum latency l_{max} in this case can be as high as the sum of the transition time from idle to sleep t_{i2s}, the timeout period in sleep state t_{sleep}, the transition time from sleep to idle t_{s2i}, and the processing time of the request.

Figure 8 shows the influence of ω on the time (hours) spent in each state and during transitions. For a higher preference for power consumption ($\omega \rightarrow 0$), the SP stays in sleep state for a larger amount of time. When ω is increased ($\omega \rightarrow 1$), the SP spends more time in the idle state for an immediate request service. The time in the processing (busy) state is the same for all cases due to the same number of requests.

The state-transition overhead is an important aspect in DPM. Our approach considers this overhead, that is, the additional and nonnegligible power consumption and latency caused by any state transition. The state transition overhead is incorporated in the cost function of our learning algorithms (cp. Equation (3) and Table II). Except issuing a state-transition command based on the expiration of timeout and/or the arrival of new requests in the service queue, the power manager does not have to implement/process any additional logic/command or computation. After the state

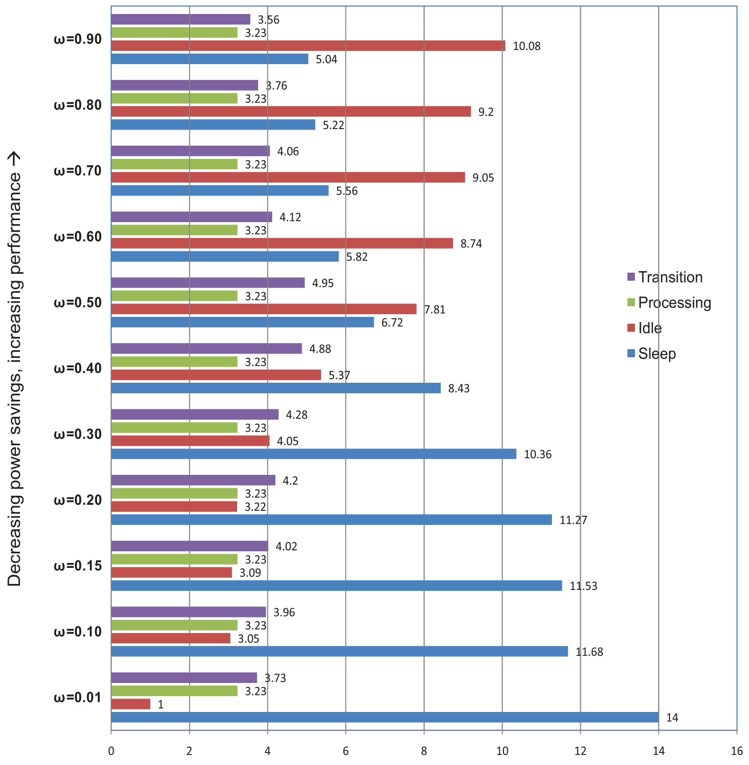


Fig. 8. Influence of ω on states occupancy (in hours) for workload 1.

transition, the cost of the previous action is determined and the state-transition value is calculated by Equation (7).

The computational overhead in our learning framework involves calculating the cost of the implemented action (Equation (3)), calculating the state action values by the update rule defined by Equation (7), and selecting the next action by the exploration-exploitation policy (Equation (11)). The computational complexity/overhead of the cost function and the value function per state-action update is independent of the size of the state-space and is given by $O(1)$. Disregarding the number of states and actions in the search space, the cost and the state-action values are updated with the same computational complexity and execution time. The computational complexity of the exploration-exploitation policy is a function of the state-space size and grows linearly with the size of the action set (timeout values) and is given by $O(|A|)$, where $|A|$ denotes the size of the action set. In our DPM implementation on the smart camera, the overhead for the state-action update and the exploration-exploitation policy is between 2–3 ms for each iteration.

We compare the performance of our proposed DPM technique with some of the techniques found in the literature, including fixed timeout, adaptive timeout [Douglis et al. 1995], ML-ANN predictive policy based only on our workload estimation, exponential predictive [Hwang and Wu 2000], and online Timeout/N policy [Khan and Rinner 2012a] which learns timeout policies only for the idle state and takes wakeup decisions based on the queue occupancy. The comparison was performed on the same workload (workload 1). The break-even time threshold T_{thr} for all the policies is set to 10 seconds.

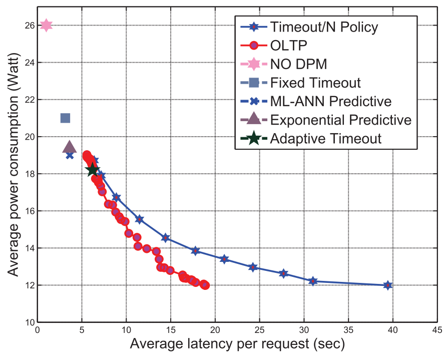


Fig. 9. Comparison of the OLTP with different DPM policies.

Figure 9 shows the comparison of our proposed DPM techniqe, Online Learning of Timeout Policies (OLTP), with the previously mentioned DPM policies for workload 1. From the figure, it is evident that our power manager is capable of finding a better power-performance trade-off than the other DPM policies. The online Timeout/N policy does allow for controling the power-performance trade-off but results in higher latency. Our power manager provides a much deeper power-performance trade-off 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 our online timeout policy, matches with one of its solutions.

7. ONLINE ADAPTATION OF POWER/PERFORMANCE WEIGHT

In online adaptation of power/performance weight, the aim is to define a constraint of either power consumption or latency while optimizing the other. This can be achieved by online adjusting the value of the power/performance weight ω. Online adjustment of the weights assigned to different objectives in a multicriteria optimization problem is a challenging issue. For a system having a dynamic workload, as in our case, a small online adjustment of the weights may lead to a significant divergence (large overshoot or undershoot) of the policy. A possible way is to train multiple RL agents concurrently with different sets of objective weights and then select the agent which satisfies the constraint while minimizing/maximizing the other criteria. However, this approach is not viable, as the system performance during online learning with multiple agents will be deteriorated.

We can perform the online adjustment of ω by comparing the actual power consumption (or performance) to the power (or performance) constraint over equally distributed RL decision intervals. For a given power or performance constraint, we use an iterative approach for online adaptation of ω. If the average power and average latency of an optimal 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, we use the following rule to update the value of ω per fixed number of RL decisions.

$$\omega_{i+1} = \omega_i + \kappa(C_P - \phi_P), \tag{13a}$$
$$\omega_{i+1} = \omega_i - \kappa(C_L - \phi_L). \tag{13b}$$

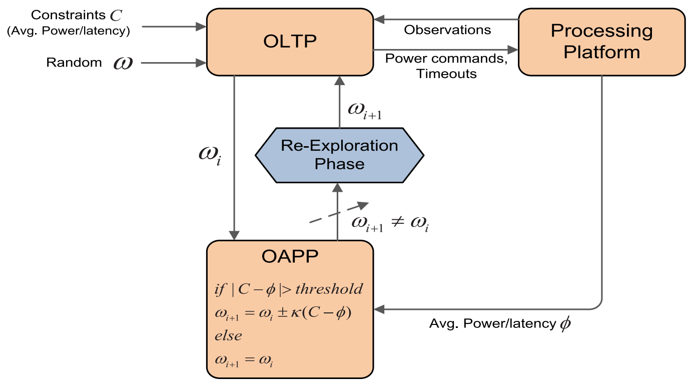


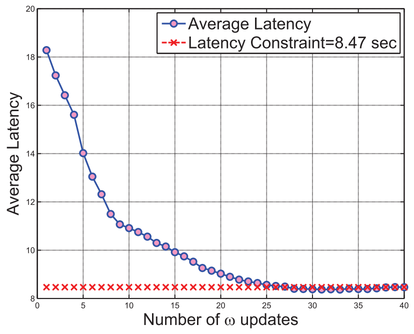
Fig. 10. Online adaptation of power-performance weight.

In Equations (13a)–(13b), $\kappa = 0.01$ is an adapting coefficient. From the cost function (Equation (3)), it is clear that varying the value of ω has opposite effects on power and latency. Therefore, we use Equation (13a) to update ω for power constraint and Equation (13b) for latency constraint. Since $\omega \in (0, 1)$, the updated value of ω based on the deviation from the constraint must be in the interval (0,1). Therefore the deviation of the average power (or latency) with the given constraint must be normalized within this range. Since the average power consumption or average latency is bounded, we find that assigning the value of 0.01 to the adapting coefficient κ adequately normalizes the adjustment quantity to the interval (0,1). Even if the given constraint is inaccessible by the online adaptation rule (i.e., if it is far away from the [min, max] bounds of average power or latency), the online adaptation sets the value of ω to either minimum or maximum (0 or 1) to get as close to the given constraint as possible. Figure 10 provides a high-level view of the online adaptation algorithm with a simple explanation of the logic expressed in Equations (13a) and (13b). It shows the integration of the discrete Online Adaptation of Power/Performance (OAPP) controller to the main OLTP framework. In Figure 10, the average power ϕ_P and the average latency ϕ_L are unified by ϕ. The constraints are represented by C.

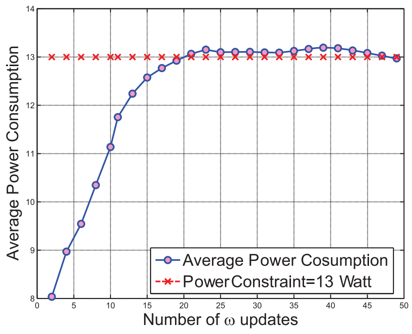
The decision to check and update the value of ω is taken after each N RL updates. The length of the decision interval influences the performance of the online adaptation. Using a large value of N will delay the decision making and the value of ω may get a larger deviation. Likewise, using a (too) small value of N will result in a more opportunistic approach by rapidly updating the value of ω, and the algorithm may not get sufficient time for adaptation. Based on our experiments, we find that the decision making after every 50 RL updates ($N = 50$) and then waiting for an additional 50 RL updates for adaptation provide the algorithm sufficient time for adaptation.

The value of ω is initialized randomly. After 50 RL updates, the OLTP triggers the OAPP controller to take the update decision. If the OAPP controller finds a significant deviation between the average power (or latency) and the given constraint, it updates the value of ω accordingly. After the update phase, the algorithm enters into the re-exploration phase where the learning parameters are reset and the OLTP starts learning the environment with the updated value of ω. The re-exploration phase further spans 50 RL updates after which we start keeping trace of ϕ and make the next update decision after 50 RL updates. For a more detailed explanation of our online adaptation algorithm, please refer to Khan [2013].

Figure 11 shows the convergence of the algorithm to two constraints for workload 1: (i) average latency constraint $C = 8.47$ *sec*, and (ii) average power constraint



(a) Convergence to a latency constraint.



(b) Convergence to a power constraint.

Fig. 11. Power-performance weight adaptation for latency and power constraints.

Table IV. Latency Constraint Adaptation for Different Workloads

Latency Constraint C_L (sec)	**Actual Latency ϕ_L (sec)**	**Relative Difference (%)**	ω
17.40	17.30	0.57	0.16
15.38	15.11	1.74	0.21
13.54	13.30	1.71	0.28
9.00	8.97	0.33	0.40
8.47	8.50	0.27	0.51

Table V. Power Constraint Adaptation for Different Workloads

Power Constraint C_P (Watt)	**Actual Power ϕ_P (Watt)**	**Relative Difference (%)**	ω
12.23	12.14	0.70	0.16
13.53	13.83	2.21	0.32
14.73	15.01	1.93	0.40
16.00	15.96	0.27	0.52
17.83	17.58	1.40	0.62

$C = 13\ Watt$. We tested the algorithm on workloads 1–5 for a number of power and latency constraints. Tables IV and V show the results of ω adaptation for different constraints of latency and power consumption, respectively. The first columns of the two tables show the given constraints (C_L and C_P), the second columns show the actual average latency and average power (ϕ_L and ϕ_P) achieved by the algorithm, the third columns show the percentage of deviation from the given constraint, and the fourth columns show the converged value of ω. The results show that our algorithm can satisfactorily fulfill the given constraints by the online adjustment of power-performance weight.

The interarrival times do not have an immediate effect on the value of omega. If the workload changes, the average power consumption (or average latency) significantly deviates from the defined constraint. During the update interval, the value of omega is adjusted accordingly which reflects the change in the workload. For example, suppose that the online algorithm has to achieve a constraint of an average latency. If the interarrival times begin to decrease (workload increases), the current average latency gradually increases and becomes larger than the given constraint. This is due to the fact that the timeout values selected and implemented by the online algorithm to deal with the (relatively) lower workload and to achieve the given constraints are no longer appropriate for a higher workload. If this change is detected during the update interval, the value of omega is increased with a margin, as described in Equation (13b).

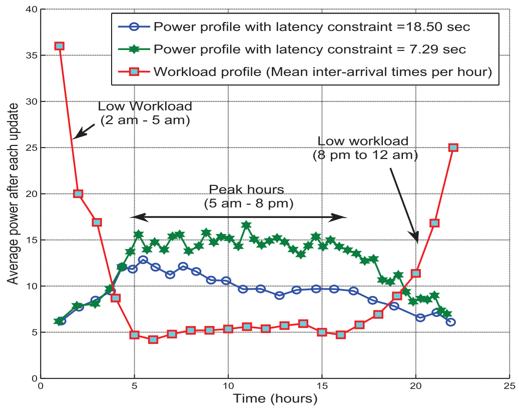


Fig. 12. Power profile with changing workload for two different latency constraints.

Figure 12 shows the power consumption profile of ω adaptation for workload 1 with two different latency constraints: $C_L = 18.50\ sec$ and $C_L = 7.29\ sec$. The test recordings start at 2AM when the workload was relatively low and the algorithm kept the power consumption low. When the traffic intensity increases (at 5AM), the power profile for the two constraints changes. When the traffic intensity declines again (at 8 PM), the power consumption reduces and both profiles converge. For achieving lower latency constraint $C_L = 7.29\ sec$, the power consumption is higher than the one for achieving higher latency constraint $C_L = 18.50\ sec$. This experiment demonstrates that our algorithm is not only able to keep different power (or latency) profiles for different constraints, but also dynamically changes the power (or latency) with the changing workload by adjusting the value of power-performance weight ω.

8. CONCLUSION

In this article, we proposed an RL-based DPM approach for optimal selection of timeout values in the different device states. The timeout selection is based on workload estimates derived from a Multilayer Artificial Neural Network (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 trade-off. Our DPM approach is further able to adapt the given power/performance constraints. We have completely implemented our DPM algorithm on our embedded traffic surveillance platform and performed long-term experiments using real traffic data. These experiments show that a Pareto-optimal trade-off between average power consumption and average latency per request is achieved. The adaptive DPM algorithm promptly converges to different power/performance constraints.

Our future work includes migrating our DPM algorithm from application level to (OS) kernel level and targeting an embedded platform having more numbers of operational and sleep states, as defined by the Advanced Configuration & Power Interface (ACPI) framework [ACPI 2011]. 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 trade-off. Additionally, our proposed DPM framework may be applied at a lower level for the DPM of a multiprocessor system-on-chip, which is an active area of research [Ogras et al. 2009; Bogdan et al. 2012]. The DPM approaches proposed in this context consider the network-on-chip architectures partitioned into several voltage-frequency islands (VFIs) which can be represented as states in our model-predictive and machine-learning-based approach to learn timeout policies for each VFI.

REFERENCES

ACPI. 2011. Advanced Configuration and Power Interface Specification (ACPI). *ACPI Specification Document* 5. http://www.acpi.info.

F. Barrero, S. Toral, M. Vargas, F. Cortés, and J. Milla. 2010. Internet in the development of future road-traffic control systems. *Int. Res.* 20, 2, 154–168.

L. Benini, A. Bogliolo, and G. De Micheli. 1999. Policy optimization for dynamic power management. *IEEE Trans. Comput.-Aid. Des. Integ. Circuits Syst.* 18, 6, 813–833.

L. Benini, A. Bogliolo, and G. De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst.* 8, 3, 299–316.

K. Bhatti, C. Belleudy, and M. Auguin. 2010. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *Proceedings of the IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC)*. 184–191.

H. Bischof, M. Godec, C. Leistner, B. Rinner, and A. Starzacher. 2010. Autonomous audio-supported learning of visual classifiers for traffic monitoring. *IEEE Intell. Syst.* 25, 3, 15–23.

P. Bogdan and R. Marculescu. 2011a. Cyberphysical systems: Workload modeling and design optimization. *IEEE Trans. Des. Test Comput.* 28, 4, 78–87.

P. Bogdan and R. Marculescu. 2011b. Non-stationary traffic analysis and its implications on multicore platform design. *IEEE Trans. Comput.-Aid. Des. Integr. Circuits Syst.* 30, 4, 508–519.

P. Bogdan and R. Marculescu. 2011c. Towards a science of cyber-physical systems design. In *Proceedings of the IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*. 99–108.

P. Bogdan, R. Marculescu, S. Jain, and R. T. Gavila. 2012. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In *Proceedings of the IEEE/ACM 6th International Symposium on Networks on Chip (NoCS)*. 35–42.

E. Y. Chung, L. Benini, and G. De Micheli. 1999. Dynamic power management using adaptive learning tree. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 274–279.

R. David, P. Bogdan, and R. Marculescu. 2012. Dynamic power management for multicores: Case study using the intel SCC. In *Proceedings of the IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*. 147–152.

G. Dhiman and T. S. Rosing. 2006. Dynamic power management using machine learning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 747–754.

F. Douglis, P. Krishnan, and B. Bershad. 1995. Adaptive disk spin-down policies for mobile computers. *Comput. Syst.* 8, 4.

J. Durand, S. Girard, and V. Ciriza. 2012. Optimization of power consumption and device availability based on point process modelling of the request sequence. *J. Royal Stat. Soci.* 3.

C. Eui-Young, L. Benini, A. Bogliolo, L. Yung-Hsiang, and G. De-Micheli. 2002. Dynamic power management for nonstationary service requests. *IEEE Trans. Comput.* 51, 11, 1345–1361.

A. Fallahi and E. Hossain. 2007. QoS provisioning in wireless video sensor networks: A dynamic power management framework. *IEEE Trans. Wirel. Commun.* 14, 6, 40–49.

K. Fei, T. Pin, and Q. Shi. 2006. Genetic algorithm based idle length prediction scheme for dynamic power management. In *Proceedings of the IMACS Multiconference on Computational Engineering in Systems Applications*. 1437–1443.

M. Ghasemazar, H. Goudarzi, and M. Pedram. 2012. Robust optimization of a chip multiprocessor's performance under power and thermal constraints. In *Proceedings of the IEEE 30th Conference on Computer Design (ICCD)*. IEEE, 108–114.

C. H. Hwang and A. C. H. Wu. 2000. A predictive system shutdown method for energy saving of event-driven computation. *ACM Trans. Des. Autom. Electron. Syst.* 5, 2, 226–241.

J. Hwisung and M. Pedram. 2010. Supervised learning based power management for multicore processors. *IEEE Trans. Comput.-Aid. Des. Integ. Circuits Syst.* 29, 9, 1395–1408.

S. Irani, S. Shukla, and R. Gupta. 2003. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embed. Comput. Syst.* 2, 3, 325–346.

N. Kandasamy and S. Abdelwahed. 2004. A control-theoretic approach to power management in embedded processors. In *Proceedings of Informatics 9,* 1–12.

U. A. Khan. 2013. Online learning of timeout policies for dynamic power management. Ph.D. dissertation. Alpen-Adria Universität Klagenfurt, Austria.

U. A. Khan, M. Godec, M. Quaritsch, M. Hennecke, H. Bischof, and B. Rinner. 2012. MobiTrick–Mobile traffic checker. In *Proceedings of the ITS World Congress*.

U. A. Khan, M. Quaritsch, and B. Rinner. 2011. Design of a heterogeneous, energy-aware, stereo-vision based sensing platform for traffic surveillance. In *Proceedings of the 9th Workshop on Intelligent Solutions in Embedded Systems*. 47–52.

U. A. Khan and B. Rinner. 2012a. Dynamic power management for portable, multi-camera traffic monitoring. In *Proceedings of the IEEE Real Time and Embedded Technology and Applications Symposium*.

U. A. Khan and B. Rinner. 2012b. A reinforcement learning framework for dynamic power management of a portable, multi-camera traffic monitoring system. In *Proceedings of the IEEE Conference on Green Computing and Communications*.

Y. H. Lu and G. De Micheli. 2001. Comparing system-level power management policies. *IEEE Trans. Des. Test Comput.* 18, 2, 10–19.

S. Mannor, B. Kveton, S. Siddiqi, and C. H. Yu. 2006. Machine learning for adaptive power management. *Auton. Comput.* 10, 4, 299–312.

C. Moser, J. Chen, and L. Thiele. 2010. Dynamic power management in environmentally powered systems. In *Proceedings of the Design Automation Conference (ASP-DAC)*. IEEE, 81–88.

S. Natarajan and P. Tadepalli. 2005. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. 601–608.

U. Y. Ogras, R. Marculescu, D. Marculescu, and E. G. Jung. 2009. Design and management of voltage-frequency island partitioned networks-on-chip. *IEEE Trans. VLSI Syst.* 17, 3, 330–341.

C. M. Olsen and C. Narayanaswarni. 2006. PowerNap: An efficient power management scheme for mobile devices. *IEEE Trans. Mobile Comput.* 5, 7, 816–828.

A. Paul. 2013. Dynamic power management for ubiquitous network devices. *Adv. Sci. Lett.* 19, 7, 2046–2049.

T. Phit and K. Abe. 2006. Packet inter-arrival time estimation using neural network models. In *Proceedings of the Internet Conference*.

F. Pletzer, R. Tusch, L. Böszörmenyi, and B. Rinner. 2012. Robust traffic state estimation on smart cameras. In *Proceedings of the IEEE Conference on Advanced Video and Signal-Based Surveillance*. 434–439.

V. L. Prabha and E. C. Monie. 2007. Hardware architecture of reinforcement learning scheme for dynamic power management in embedded systems. *EURASIP J. Embed. Syst.* 07, 1, 1–6.

Q. Qiu and M. Pedram. 1999. Dynamic power management based on continuous-time Markov decision processes. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*. 555–561.

Z. Ren, B. H. Krogh, and R. Marculescu. 2005. Hierarchical adaptive dynamic power management. *IEEE Trans. Comput.* 54, 4, 409–420.

H. C. Shih and K. Wang. 2012. An adaptive hybrid dynamic power management algorithm for mobile devices. *Comput. Netw.* 56, 2, 548–565.

T. Simunic, L. Benini, P. Glynn, and G. De Micheli. 2000. Dynamic power management for portable systems. In *Proceedings of the 6th International Conference on Mobile Computing and Networking*. 11–19.

T. Simunic, L. Benini, P. Glynn, and G. De Micheli. 2001. Event-driven power management. *IEEE Trans. Comput.-Aid. Des. Integ. Circuits Syst.* 20, 7, 840 –857.

M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen. 1996. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Trans. Integ. VLSI Syst.* 4, 1, 42–55.

R. S. Sutton. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning*. 216–224.

Y. Tan, W. Liu, and Q. Qiu. 2009. Adaptive power management using reinforcement learning. In *Proceedings of the International Conference on Computer-Aided Design*. 461–467.

Y. Wang, Q. Xie, A. Ammari, and M. Pedram. 2011b. Deriving a near-optimal power management policy using model-free reinforcement learning and Bayesian classification. In *Proceedings of the Design Automation Conference*. 41–46.

Z. Wang, H. Wang, X. Chen, and J. Lin. 2011a. Cyber physical systems. *J. Chinese Comput. Syst.* 32, 5, 881–886.

C. J. C. H. Watkins and P. Dayan. 2011. Q-Learning. *Machine Learn.* 8, 3–4, 279–292.

H. Young, K. Sung, and C. Ki-Seok. 2010. A predictive dynamic power management technique for embedded mobile devices. *IEEE Trans. Consumer Electron.* 56, 2, 713–719.

A. Zafra, E. L. Gibaja, and S. Ventura. 2011. Multiple instance learning with multiple objective genetic programming for Web mining. *Appl. Soft Comput.* 11, 1, 93–102.

Received February 2013; revised June 2013; accepted August 2013