

# Achieving Autonomous Power Management Using Reinforcement Learning

HAO SHEN, Syracuse University  
YING TAN and JUN LU, Binghamton University  
QING WU, Air Force Research Laboratory  
QINRU QIU, Syracuse University

System level power management must consider the uncertainty and variability that come from the environment, the application and the hardware. A robust power management technique must be able to learn the optimal decision from past events and improve itself as the environment changes. This article presents a novel on-line power management technique based on model-free constrained reinforcement learning (Q-learning). The proposed learning algorithm requires no prior information of the workload and dynamically adapts to the environment to achieve autonomous power management. We focus on the power management of the peripheral device and the microprocessor, two of the basic components of a computer. Due to their different operating behaviors and performance considerations, these two types of devices require different designs of Q-learning agent. The article discusses system modeling and cost function construction for both types of Q-learning agent. Enhancement techniques are also proposed to speed up the convergence and better maintain the required performance (or power) constraint in a dynamic system with large variations. Compared with the existing machine learning based power management techniques, the Q-learning based power management is more flexible in adapting to different workload and hardware and provides a wider range of power-performance tradeoff.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*

General Terms: Design, Experimentation, Management, Performance

Additional Key Words and Phrases: Power management, thermal management, machine learning, computer

## ACM Reference Format:

Shen, H., Tan, Y., Lu, J., Wu, Q., and Qiu, Q. 2013. Achieving autonomous power management using reinforcement learning. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2, Article 24 (March 2013), 32 pages.  
DOI: <http://dx.doi.org/10.1145/2442087.2442095>

## 1. INTRODUCTION

Power consumption has become a major concern in the design of computing systems today. High power consumption increases cooling cost, degrades the system reliability and also reduces the battery life in portable devices. Modern computing/communication devices support multiple power modes which enable power and performance trade-off. *Dynamic power management (DPM)* has proven to be an effective technique for power

---

This work is supported in part by NSF under grant CNS-0845947.

Author's address: H. Shen, email: [shenao.ee.seu@gmail.com](mailto:shenao.ee.seu@gmail.com).

©2013 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

© 2013 ACM 1084-4309/2013/03-ART24 \$15.00

DOI: <http://dx.doi.org/10.1145/2442087.2442095>

reduction at the system level. It selectively shuts-off or slows-down system components that are idle or underutilized. The power manager needs to make wise decisions on when to put the devices into which power mode. *Dynamic voltage and frequency scaling* (DVFS) is another technique that has been widely used in modern processors for energy reduction or temperature control by dynamically changing the working frequency and voltage of the processor. Both DVFS and DPM provide a set of control knobs for runtime power management. From this perspective, they are fundamentally the same. While the DVFS is usually found as the power control knob for CMOS digital ICs, such as microcontrollers or microprocessors, during the active time; the DPM is usually for the peripheral devices, such as the hard disk drives and network interface, or for microprocessors running interactive applications accompanied with long idle intervals. In this work, we refer to both DPM and DVFS as power management as there is no fundamental difference between these two. The effective use of those power management techniques at run time usually requires application and architecture specific information.

Robust power management must consider the uncertainty and variability that come from the environment, the application and the hardware. For example, the workload of a complex system is usually unpredictable as it strongly depends on the nature of the application, the input data and the user context. The workload variation changes the device usage pattern and has the most significant impact on the system speed and power consumption. The contention of shared resources such as buses or I/Os in an MPSoC also increases the variability of hardware response time for communication and computation. Furthermore, the process, voltage, and temperature (PVT) variation results in a large fluctuation in hardware performance and power consumption. Therefore, statically optimized resource and power management policies are not likely to achieve the best performance when the input characteristics change. The ability to observe, learn and adapt to different hardware systems and different working environments is essential for a power management controller.

In this article, we present a novel approach for system level power management based on online *reinforcement learning* (RL). The proposed power manager learns a new power control policy dynamically at runtime from the information it receives. This is achieved by trying an action in a certain system state, and adjusting the action when this state is re-visited next time, based on the reward/cost received. This is a model-free approach as the power manager learns the policy directly. The technique does not require any prior information of the system or workload. However, if such knowledge is available, it can help to speed up the convergence of the learning algorithm and better track the performance (or power consumption) constraints.

A reinforcement learning model consists of three basic elements: a state space that describes the environment status, an action space that defines the available control knobs and a cost function that evaluates the reward/cost of different actions in different states. How these three elements should be defined is determined by the available environment information, the nature of the system under control, as well as the user objectives and constraints. Therefore, it varies from problem to problem.

In this work, we investigate RL model construction for the power management of two most common types of devices in a computer, the peripheral device and the microprocessor. The peripheral device is an interactive system that processes the I/O requests generated by software applications. Its performance is measured by its response time (which is proportional to the average length of request waiting queue.) It assumes that each I/O request is an atomic task. For such peripheral device, its workload is captured by the distribution of idle intervals and the request generation rate. For the microprocessor, we focus on its power management during the time of batch processing. The performance is measured by the execution time, and the workload characteristic is

captured by its CPU intensiveness. From the power management algorithm design perspective, a microprocessor in batch mode and a peripheral device differ in many ways.

First of all, they have different power management control knobs. The power consumption of a peripheral device can be controlled by configuring the device into discrete power modes. For example, a wireless adaptor usually has 4 power modes: receiving, transmitting, idle and standby modes. However, the power consumption of a microprocessor performing batch processing is usually controlled by dynamic voltage and frequency scaling. For example, the AMD Opteron processor is implemented with 5 levels of voltage and frequencies. This leads to different action spaces in the Q-learning algorithm design.

Secondly, their performances are measured by different criteria and affected by different factors. While the performance of a peripheral device is measured by its response time, which is determined by the request incoming rate and processing speed; the performance of a microprocessor working on batch processing is measured by its execution time, which is affected by the CPU clock frequency and architectural events such as pipeline stalls. Therefore, the two devices should be modeled by different sets of parameters. This leads to different state classification methods in Q-learning model construction.

Furthermore, their optimization objectives are different. The ultimate goal of power management is to minimize the energy dissipation under the given performance constraint. For peripheral devices, an infinite horizon is usually assumed. Therefore, minimizing energy dissipation is the same as minimizing the average power consumption over a long time. However, for a microprocessor working on batch processing, we usually focus on minimizing the energy dissipation over the execution time. While the average power consumption of a peripheral device is a monotonic decreasing function of its response time; such monotonic relation does not always exist between the total energy dissipation and execution time of a modern microprocessors. On one hand, although lowering the voltage and frequency of a CPU effectively reduces the dynamic energy, it also increases the leakage energy because the system has to be kept active for a longer time [Jejurikar et al. 2004]; therefore, the total energy may not necessarily decrease. On the other hand, as the limited memory bandwidth has already become the performance bottleneck for some high performance computers, lowering CPU voltage and frequency to a certain extent may not have significant impact on performance, since the memory subsystem still works under a constant frequency [Langen and Juurlink 2006; Jejurikar et al. 2004; Choi et al. 2004]. As we can see, in addition to different objective functions, the relations between objectives and constraints are also different for the two power management problems. Hence, different cost function must be constructed.

Finally, other constraints sometimes are usually considered in microprocessor power management. For example, temperature has significant impact on the chip performance and reliability, and hence should be considered as another constraint. This requires a cost function design that is flexible enough to incorporate multiple constraints.

These discussions show that different Q-learning models (i.e., different environment state classification methods and different cost function formulations) are needed for peripheral device power management and microprocessor power management. These two power management problems are both interesting and are complementary to each other. It is important to discuss their Q-learning models separately. In this article, we focus on how these Q-learning models are constructed and how effective they will be. In addition to model construction, techniques are developed to enhance the performance of the RL based controller in a power managed system. Novel techniques are proposed that speed up the convergence of learning and better maintain the required performance (or power) constraint in a dynamic system. The following summarizes the main contribution of this work.

- (1) We present a power manager that does not require any prior knowledge of the workload. It learns the policy online with real-time information and adjusts the policy accordingly. After a certain set-up time, the optimal policy can positively be found.
- (2) We propose a set of enhancement techniques that utilize the partial knowledge of the system to accelerate the convergence speed and enable the runtime tracking of the performance (power consumption) constraint.
- (3) We apply the RL based controller to perform power management of peripheral devices and microprocessors. Different model construction approaches are discussed for these two types of devices. The performance of the proposed power management controller is evaluated by either simulation or real measurement.

Compared to the previous works [Tan et al. 2009; Liu et al. 2010], this work offers has the following major contributions.

- (1) In addition to the peripheral devices, we also apply the RL based power management to microprocessors. Model construction and cost function formulation are discussed.
- (2) The RL model construction for peripheral devices is improved to handle real application scenarios with more diversified workload and practical constraints. For example, we improved the state partition techniques to cover workloads with large variations.
- (3) While the traditional stochastic power management is able to satisfy the given constraints on long term average performance (or power consumption), they usually have large performance (or power consumption) variations during short period of time. In this work, a two level controller is proposed to find the weight factor that balances the power-performance trade-off of the learning based power management policy so that it operates at a relatively constant performance (or power consumption) that is close to the given constraint.
- (4) More experimental data are provided. In addition to traces collected from personal PCs, the proposed power management technique is evaluated using the HP hard disk traces that resemble the workload of large data centers. It is also implemented on a Dell Precision T3400 workstation to control the runtime voltage and frequency scaling for simultaneous energy, performance and temperature management.

The rest of the article is organized as follows: Section 2 talks about the related work including the expert-based DPM algorithm, which will be used as a comparison with our algorithm. Section 3 introduces the general RL model for power management. Sections 4 and 5 discuss model construction and enhancement techniques for the power management of peripheral devices and microprocessors, respectively. Section 6 presents the experimental results. Finally Chapter 7 gives the conclusions.

## 2. RELATED WORKS

Based on when it is applied, system level low power techniques can be categorized into design time approaches and run time approaches. The former modifies and optimizes the system and component architecture during design time for a lower power consumption or to facilitate runtime power reduction [Chou and Marculescu 2010; Fei et al. 2007; Agarwal et al. 2010; Smullen et al. 2010]; while the latter performs online to dynamically control the power with the respect of performance constraints. Dynamic power management (DPM) and dynamic voltage frequency scaling (DVFS) belong to the second category.

The simplest and most widely used DPM algorithm is the timeout policy which puts the device into low power mode after it has been idle for certain amount of time.

Though it is easy to implement and relatively effective in many computer systems, the timeout policy is far from optimized because it wastes energy during the timeout period. Furthermore, the traditional timeout policy uses a fixed timeout value which cannot adapt to the change of workload or user context.

In order to best adjust itself to the dynamic system, many DPM works on a system model that is learned from the history information. For example, the predictive DPM [Hwang and Wu 2000] predicts the next idle time based on previous idle time and makes power mode switching decision based on the predicted value. The previous works in stochastic power management [Qiu et al. 2007; Rosing et al. 2001; Tan and Qiu 2008] model the system as a Markov decision process. The model construction requires offline learning. [Thecharous et al. 2006] proposed a user-based adaptive power management technique that considered user annoyance as a performance constraint. Ahmad et al. [2008] convert the scheduling task on multiprocessor into a cooperative game theory problem to minimize the energy consumption and the makespan simultaneously, while maintaining deadline constraints.

Many research works have been proposed to find the optimal DVFS scheduling for energy reduction. Choi et al. [2004] use runtime information on the statistics of the external memory access to perform CPU voltage and frequency scaling. Its goal is to minimize the energy consumption while translucently controlling the performance penalty. Tan et al. [2006] first present a workload prediction model for MPEG decoder and the predicted workload is further used to guide the voltage and frequency scaling. Choudhary and Marculescu [2009] considers processors as producers and consumers and tunes their frequencies in order to minimize the stalls of the request queue while reducing the processors' energy.

Multidimensional constraints sometimes are considered in power management. Performance and temperature are two typical constraints in designing a power management policy for microprocessors. [Coskun et al. 2008; Liu et al. 2009] propose to use mathematical programming to solve voltage and frequency scheduling problem for energy optimization with temperature constraints. Both works assume that the workload is known in advance. Wang et al. [2009] and Zanini et al. [2009] apply model predictive control (MPC) to find the best sequence of voltage and frequency settings for minimum energy under given temperature constraint over a finite horizon. A temperature model is required for the MPC controller to work effectively. Ayoub et al. [2012] control active memory modules and schedules workload between CPU sockets to achieve balanced thermal distribution for energy management with temperature and performance constraints.

All of these works either assume given workload model or require offline model construction and policy optimization, therefore they cannot adapt to the workload changes in real time. Online learning algorithms are natural choices for real-time adaptive power management. Cai et al. [2006] present a method that periodically adjusts the size of physical memory and the timeout value to turn off the hard disk to reduce the average energy consumption. The joint power management predicts the next hardware accesses frequency and idle interval based on previous information. Gniady et al. [2006] use program counters to learn the access patterns of applications and predicts when an I/O device can be shut down to save energy. Weddle et al. [2007] use a skewed striping pattern to adaptively change the number of powered disks according to the system load. They also enhanced the reliability of the storage system by limiting disk power cycles and using different RAID encoding schemes. Martinez and Ipek [2009] and Ipek et al. [2008] propose a machine learning approach for multicore resource management using on-chip hardware agents that are capable of learning, planning, and continuously adapting to changing demands. Those works also use the machine learning technique to perform DRAM bandwidth scheduling for a maximum

throughput. In [Dhiman and Rosing 2009], the authors propose a learning algorithm that dynamically selects different experts to make power management decisions at runtime, where each expert is a pre-designed power management policy. This approach leverages the fact that different experts outperform each other under different workloads and hardware characteristics. The similar approach is applied in [Coskun and Rosing 2008] to perform power management with performance and temperature constraints. There is also a large body of works that learn workload/temperature model online for thermal management [Ayoub and Rosing 2009; Coskun et al. 2009; Yeo et al. 2008, 2011], or uses random policy with temperature aware adaptation [Coskun et al. 2007]. However, these works do not consider energy (or power) minimization.

Reinforcement learning has been applied to resource allocation and further extended to microprocessor power management in Tesauro et al. [2006] and Tesauro et al. [2007], respectively. Both works focus on the Web application servers. The environment state is describe by the rate of the incoming request and the policy is optimize offline using the Sarsa(0) algorithm. In order for the offline trained policy to work effectively, the implied assumption is that the workload is highly repeatable. Our approach differs from these two works in that, we adopt different state classification; focus on general purpose computing applications and our policy is optimized online.

### 3. GENERAL ARCHITECTURE OF Q-LEARNING BASED POWER MANAGEMENT

In this section, we will first introduce the principle of Q-learning and then we will discuss how to extend the traditional Q-learning algorithm to solve the dynamic power management problem.

#### 3.1. Q-Learning Algorithm

Reinforcement learning is a machine intelligence approach that has been applied in many different areas. It mimics one of the most common learning styles in natural life. The machine learns to achieve a goal by trial-and-error interaction within a dynamic environment.

The general learning model consists of

- an agent,
- a finite state space  $S$ ,
- a set of available actions  $A$  for the agent,
- a penalty function  $P : S \times A \rightarrow P$ .

The goal of the agent is to minimize its average long-term penalty. It is achieved by learning a policy  $\pi$ , that is, a mapping between the states and the actions.

Q-learning is one of the most popular algorithms in reinforcement learning. At each step of interaction with the environment, the agent observes the environment and issues an action based on the system state. By performing the action, the system moves from one state to another. The new state gives the agent a penalty which indicates the value of the state transition. The agent keeps a value function  $Q^\pi(s, a)$  for each state-action pair, which represents the expected long-term penalty if the system starts from state  $s$ , taking action  $a$ , and thereafter following policy  $\pi$ . Based on this value function, the agent decides which action should be taken in current state to achieve the minimum long-term penalties.

The core of the Q-learning algorithm is a value iteration update of the value function. The Q-value for each state-action pair is initially chosen by the designer and then it will be updated each time an action is issued and a penalty is received based on the

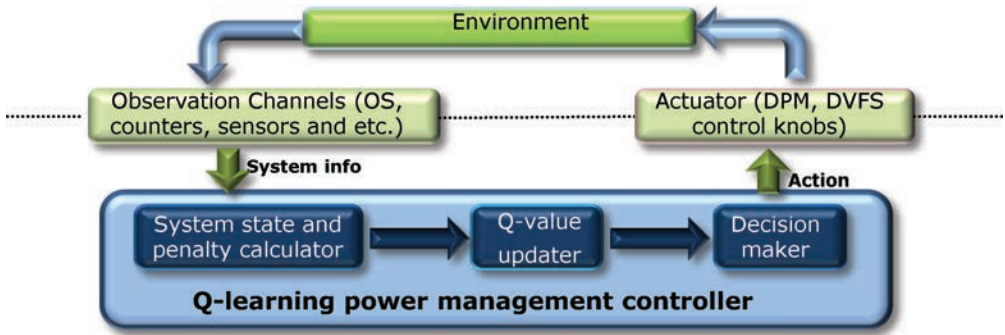


Fig. 1. Illustration of system under power management.

following expression.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\varepsilon_t(s_t, a_t)}_{learning\ rate} \times \left[ \underbrace{p_{t+1}}_{Penalty} + \underbrace{\gamma}_{Discount\ Factor} \underbrace{\min_a Q(s_{t+1}, a)}_{Min\ Future\ Value} - \underbrace{Q(s_t, a_t)}_{Old\ Value} \right]. \quad (1)$$

In this expression,  $s_t$ ,  $a_t$  and  $p_t$  are the state, action and penalty at time  $t$  respectively, and  $\varepsilon_t(s, a) \in (0, 1)$  is the learning rate. The discount factor  $\gamma$  is a value between 0 and 1 which gives more weight to the penalties in the near future than the far future. The next time when state  $s$  is visited again, the action with the minimum Q-value will be chosen, that is,  $\pi(s) = \min_{a \in A} Q(s, a)$ . The value of  $Q(s_t, a_t)$  is updated at the beginning of cycle  $t + 1$ , that is, the Q-value for the state-action pair of the previous cycle is updated at the beginning of current cycle.

As a model-free learning algorithm, it is not necessary for the Q-learning agent to have any prior information about the system, such as the transition probability from one state to another. Thus, it is highly adaptive and flexible.

### 3.2. Q-Learning Model for Power Management

Figure 1 shows the general architecture of a Q-learning based power management system. It consists of two parts, the environment and the controller. The environment can further be divided into hardware and software environments. The hardware environment could be any peripherals device such as hard disk and network card or the microprocessor. The software environment includes OS, application software, user inputs, etc. The controller continuously observes the environment and manages the control knobs (also denoted as the actuators in the figure). The environment information can be obtained through different channels. Some of the I/O requests and software activities can be observed through the operating system, the architecture event can be observed by reading the performance counters, and some of the device physical information (such as temperature) can be obtained by reading the embedded sensors. Based on the environment information, the current system state will be classified and the penalty of current state action pair will be calculated. This penalty information will be used to update the Q-values. The best action (i.e., a setting of the control knobs) that has the lowest Q-value will be selected to control the states of the actuators. A discrete-time slotted model is used throughout this work, which means all the decision making and system state updating occur on a cycle basis. A time slot  $n$  is defined as the time interval  $[nT, (n + 1)T]$ , and the power manager makes decision for this time slot at the beginning of this interval at time  $nT$ .

To construct the Q-learning model for a given control problem, three questions need to be answered: (1) How to classify the environment status into different state space of the Q-learning model? (2) How to formulate cost function from the observed information? (3) Given the set of actuators, what controls are available? In the rest of the article, we will answer these questions to design a Q-learning model for the power management of peripheral devices and microprocessors. Although the objectives of the two problems are similar, because the performance and operation status of peripheral devices and microprocessor are usually characterized by different parameters and requires different control knobs, different Q-learning models must be constructed for these two power management problems. Detailed discussion in model construction will be provided in

### 3.3. Enhanced Q-Learning

Q-learning is originally designed to find the policy for a Markov Decision Process (MDP). It is proved that the Q-learning is able to find the optimal policy when the learning rate  $\alpha$  is reduced to 0 at an appropriate rate, given the condition that the environment is MDP. However, it is important to point out that a computing system for power management is typically non-Mark. First of all, the workload of most computing system exhibits long range similarity [Varatkar and Marculescu 2004] and hence the request pattern generated by the environment in our power management system is most likely to be non-Markovian. Furthermore, even if the underlying system is Markovian, what the power manager observes may not be Markovian due to the noise and disturbance, such as state aggregation, during the observation. As we mentioned earlier, the Q-learning may not be able to find the optimal policy in a non-Markovian environment. Nevertheless we still choose Q-learning to solve this problem because of its simplicity and also because of its robustness to endure noise.

Reinforcement learning in a non-Markovian environment is an open problem. Many research works have investigated the feasibility of applying the traditional RL algorithms to solve the decision problem in a non-Markovian environment or a partially observable Markovian environment [Pendrith 1994; Sikorski and Balch 2001]. Pendrith [1994] applies five RL algorithms in a noisy and non-Markovian environment and compared their performance and convergence speed. Their results show that the Q-learning exhibits the highest robustness at low noise level and medium robustness at high noise level. However, the convergence speed of Q-learning reduces the most drastically when the noise level increases. In Sikorski and Balch [2001] similar results are reported. Q-learning is capable to achieve the same performance as the other two reference learning algorithms at the cost of slower convergence speed. Based on the study we conclude that the major limitation of Q-learning, when being applied in a non-Markovian environment, is its convergence speed.

Traditional Q-learning assumes no prior information of the environment. However, in a power management system, the model of system can be precharacterized. We know exactly how many power modes the system has and how it switches its power mode given a power management command. In other words, we have partial information of the power management system. Based on this information, we are able to design an improved Q-learning algorithm with faster convergence speed. More details are provided in Section 4.2.

## 4. LEARNING-BASED POWER MANAGEMENT FOR PERIPHERAL DEVICES

In this section, we will introduce the details of designing a Q-learning based power management algorithm to achieve the performance and power trade-off for a peripheral device. The peripheral devices, also known as input/output devices, can be considered as a *service provider* (SP). The request to the device is buffered in a *service request*



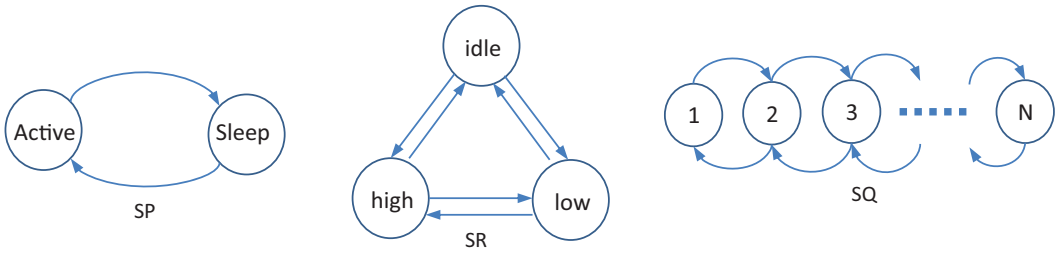


Fig. 2. State transition diagram of SP, SR, and SQ models.

queue (SQ) maintained by the OS. The software application that accesses the device is considered as *service requestor* (SR).

A peripheral device usually has several different working modes and low power modes. The state of SP can be partitioned based on its power modes. The time to transit from one state to another is hardware specific and is assumed to be known. The state of SR can be classified by its request generation rate, which is time varying. The transition rate from one SR state to another is usually unknown. Furthermore, such transitions are usually non-Markovian. SQ is a queuing model and its states are classified based on the number of waiting requests. Obviously, the state transition rate of the SQ is determined by the request generation rate and request processing rate, which can be derived from the status of SP and SR. Finally, the environment of the power management controller is modeled as the composition of the SP, SQ, and SR.

Figure 2 gives an example of SP, SR, and SQ models. The SP has two power modes, active mode and sleep mode. They can switch to each other based on the power management command. The SR has three request generation mode, high speed, low speed and idle mode. They can also transit to each other. The SQ can hold up to  $N$  requests and each time the number of requests can only increment or decrement by one.

#### 4.1. State Partition and Penalty Calculation

The observed power mode of SP can naturally be used to represent its state. SP has two types of states, stable state and transient state. During the stable state (e.g., active states and sleep states), the SP stays at a specific power mode. It processes the request at a certain speed (which could be as low as zero in sleep state). The learning agent observes the environment and issues power management command periodically. During the transient state, the SP switches from one power mode to another. It does not process any request. The learning agent halts during the transient state because the SP does not respond to any power management command.

The state of SR is classified based on the rate of the incoming request. Due to the high variation of the workload, this value is a random variable distributed over a wide range and it can almost be considered as continuous. In order to reduce the state space, it is necessary to discretize the values into fewer states. In order to adapt to different workload intensities, we propose to partition the rate of incoming request based on its exponential moving average (EMA) [Hwang and Wu 2000]. The EMA of current cycle  $i$  is calculated as  $EMA_i = \alpha \cdot EMA_{i-1} + (1 - \alpha) \cdot sr_i$ , where  $EMA_{i-1}$  is the exponential moving average request rate calculated in previous cycle and  $sr_i$  is the observed incoming request rate of current cycle. Let  $N$  denote the total number of states of SR. The SR is in state 0 and  $N-1$  when the incoming request rate is in the range  $[0, 2^{-N/2+1}EMA]$  and  $[2^{N/2-1}EMA, \infty)$  respectively. The SR is in state  $i$ ,  $0 < i < N - 1$  when the incoming request rate is in the range  $[2^{-N/2+i}EMA, 2^{-N/2+i+1}EMA]$ . The state of SQ is classified based on the length of the queue. State aggregation is also adopted to reduce state space.

In order to find the best trade-off between power and performance, we define a Lagrangian cost for each state and action pair  $(s, a)$  that combines the costs of power consumption ( $power(s, a)$ ) and performance penalty ( $q(s, a)$ ):

$$C(s, a; \lambda) = Power(s, a) + \lambda q(s, a). \quad (2)$$

When SP is in a stable state,  $Power(s, a)$  and  $q(s, a)$  represent the system power consumption and the number of waiting request of current state. When SP is in a transient state, because the Q-learning agent will be suspended as mentioned before, we are not able to update the cost until the end of the transient state. Therefore, the accumulated cost during the entire switching period should be calculated. Furthermore, many systems have nonsymmetric penalty for switching into and switching out from a low power mode. Sometime turning off the device may be effortless, but we still need to anticipate the difficulty to turn it back on in the future. Based on these motivations, for a transient state  $s$  where SP switches from power mode A to power mode B, the power cost is calculated as the average of the energy dissipation to switch from A to B and from B to A, that is,  $Power(s, a) = (P_{A2B} * T_{A2B} + P_{B2A} * T_{B2A})/2$ , where  $P_{A2B}$ ,  $P_{B2A}$  are power consumptions during A to B and B to A switch respectively, and  $T_{A2B}$ ,  $T_{B2A}$  are delays of those switches. The performance cost is calculated as the average accumulated request delays during the time the SP switches from A to B and from B to A, that is,  $q(s, a) = (q_{A2B} * T_{A2B} + q_{B2A} * T_{B2A})/2$ , where  $q_{A2B}$ ,  $q_{B2A}$  is the average request incoming rate during the power mode switching along the history. To give an example, consider a hard disk drive. To transit this hard disk from sleep state to active state usually associates with long latency and high power consumption because we have to spin up the disk mechanically. During the transition, all the new incoming requests will be accumulated in SQ. This transition will not be necessary if the disk did not go to sleep state at all in previous decision. With this knowledge, we distribute the penalty evenly between the sleep to active and active to sleep transitions so that SP will not transit to sleep state (normally taking little effort) aggressively. Our experiment shows that such cost function calculation for the transient state leads to better result.

Given the next state  $s'$  and its Q values, the learning agent updates the Q-values of the state action pair  $(s, a)$  periodically using the following equation.

$$Q(s, a; \lambda) = (1 - \varepsilon_{(s,a)}) Q(s, a; \lambda) + \varepsilon_{(s,a)}(C(s, a; \lambda) + \min_{a'} Q(s', a'; \lambda)) \quad (3)$$

The Q-value of state action pair  $(s, a)$  reflects the expected average power and request delay caused by the action  $a$  taken in state  $s$ . The new action  $a'$  with minimum Q-value  $\min_{a'} Q(s', a'; \lambda)$  will be issued at state  $s'$ .

## 4.2. Accelerating the Speed of Convergence of Q-learning

The convergence of the Q-learning relies on the recurrent visits of all possible state-action pairs. Based on equation (3) we can see, each time a state  $s$  is visited and an action  $a$  is taken, a corresponding Q value  $Q(s, a)$  is updated. It is calculated as the weighted sum of itself and the best Q value of the next state  $s'$ , that is,  $Q(s, a) = (1 - \varepsilon) Q(s, a) + \varepsilon(C(s, a) + \min_{a'} Q(s', a'))$ . The frequency that state  $s'$  occurs after state-action pair  $(s, a)$  reveals the information of the system transition probability. In traditional Q-learning, only the Q value corresponding to the actual visited state-action pair will be updated. This is because the controller has no information of the system dynamics, and it totally relies on the actual execution trace to find out the next state information for a given state-action pair.

In contrast to conventional Q-learning systems, we do have some partial information of our target power management system. The state of a power management system is a composition of the states of SP, SR and SQ. Among these three, only SR has unknown behavior. The state space SP is the set of available power modes and its

power consumption, processing speed and power mode transition overhead are known. We also know that SP and SR are independent to each other, and when SP and SR are given, the behavior of SQ is determined.

Based on the available information on SP and SQ, we propose to update more than one Q values each cycle to speed up convergence. For each visited state-action pair  $((sp, sr, sq), a)$  we will update the Q values for a set of state-action pairs  $\{((sp', sr, sq'), a') | \forall sp' \in SP, \forall sq' \in SQ, \forall a' \in A\}$ . These state actions pairs are referred as virtual states and actions, because we assume that the system has (virtually) visited these state action pairs and will update their Q values. Note that all virtual state has the same SR state which is the actual SR state that the system has recently visited. In order to update the Q values of a virtual state-action pair  $((sp', sr, sq'), a')$ , we need to know, starting from this state action pair, what the next system state will be even though it is not currently being visited. More specifically, given the information that the system was in state  $(sp_t, sr_t, sq_t)$  and it switched to  $(sp_{t+1}, sr_{t+1}, sq_{t+1})$  after action  $a$  is taken, we would like to guess what the next state  $(sp'_{t+1}, sr'_{t+1}, sq'_{t+1})$  will be if the system is currently in a different state  $(sp'_t, sr_t, sq'_t)$  and another action  $a'$  is taken.

Given the current state  $sp'_t$  and action  $a'$ , it is not difficult for us to find the next state  $sp'_{t+1}$  as the SP model is precharacterized. We also know that the SR works independently to the SP. Regardless of the state of SP, the requests are generated in the same way. Therefore,  $sr'_{t+1}$  is the same as  $sr_{t+1}$ . The value of  $sq'_{t+1}$  (i.e., the number of waiting requests) depends on both the number of incoming requests and the number of requests that have been processed. The former is determined by the state of SR and can be measured from the actual system, while the later is determined by the processing speed of SP at current power mode  $sp'_t$ . Because SP has been precharacterized, this information can also be estimated fairly accurately. After the next state is determined, the Q values of the state-action pair  $((sp'_t, sr_t, sq'_t), a')$  that has been virtually visited can easily be calculated. In the rest of the article, we refer to this technique as *Virtual State Switching* (VSS).

Using VSS, the number of Q-values that would be updated in each cycle is  $|SP| \times |SQ| \times |A|$ , where  $|SP|$ ,  $|SQ|$  and  $|A|$  are the cardinality of the  $SP$ ,  $SQ$  and  $A$ . The complexity of the constrained Q-learning is  $O(|SP| \times |SQ| \times |A|)$ . The size of SP state space and action space is fixed for a given hardware. With a carefully controlled SQ state partition, this computation complexity is affordable.

We further improve the convergence speed of the proposed Q-learning algorithm by adopting a variable learning rate. Compared to the traditional Q-learning, the learning rate  $\varepsilon_{(s,a)}$  is not fixed in our algorithm. Instead, it is dependent on the frequency of the visit to the state-action pair  $(s, a)$  and is calculated as:

$$\varepsilon_{(s,a)} = \frac{\mu}{Visit(s, a)}, \quad (4)$$

where  $Visit(s, a)$  is the number of times that the state-action pair  $(s, a)$  has been visited, and  $\mu$  is a given constant.

Figure 3 gives the pseudocode for the power management controller using enhanced Q-learning algorithm with VSS. The algorithm is executed at the beginning of each time slot. Its input is current environment state  $s_t$ , the previous environment state  $s_{t-1}$ , the action  $a_{t-1}$  in last cycle, and the weight coefficient  $\lambda$ . Each time, it updates the Q values of the real state action pair  $(S_{t-1}, a_{t-1})$  as well as all the virtual state action pairs  $(S'_{t-1}, a'_{t-1})$ .

It is important to point out that the more information we know about the system, the more accurate projection we can make about the virtual state switching. If we do not have enough information or cannot find solid reasoning to project the virtual state switching, we may apply VSS only to a small set of state-action pairs.

---

```

Q_Learning_Power_Manager( $S_t, S_{t-1}, a_{t-1}, \lambda$ )
Input: Current state  $S_t = (sp_t, sr_t, sq_t)$ , last state  $S_{t-1} = (sp_{t-1}, sr_{t-1}, sq_{t-1})$ , action  $a_{t-1}$ ,
and weight coefficient  $\lambda$ ;
Calculate the cost  $C(S_{t-1}, a_{t-1}; \lambda)$  using Equation (2);
Calculate the Q-value  $Q(S_{t-1}, a_{t-1}; \lambda)$  using Equation (3);
For each  $sp'_{t-1} \in SP$  {
  For each  $sq'_{t-1} \in SQ$  {
    For each action  $a'_{t-1} \in A$  {
      If ( $sp'_{t-1} \neq sp_{t-1} \parallel sq'_{t-1} \neq sq_{t-1} \parallel a'_{t-1} \neq a_{t-1}$ ) {
        /*Do not update the Q-value of the real state action pair twice*/
        Given the virtual state action pair  $(S'_{t-1}, a'_{t-1}) = ((sp'_{t-1}, sr_{t-1}, sq'_{t-1}), a'_{t-1})$ , find
        the projected next state  $S'_t = (sp'_t, sr'_t, sq'_t)$ ;
        Calculate the cost  $C(S'_{t-1}, a'_{t-1}; \lambda)$  using Equation (2);
        Calculate the Q-value  $Q(S'_{t-1}, a'_{t-1}; \lambda)$  using Equation (3);
      } /* end if*/
    }
  }
}
} /* end for */
Choose action  $a_t$  with  $\min_{a_t} Q(S_t, a_t; \lambda)$ ;

```

---

Fig. 3. Pseudocode for Q-learning power manager using the VSS technique.

### 4.3. Power (Performance) Tracking Using 2-level Controller

For general peripheral devices, power and performance are two metrics inversely proportional to each other in many computing systems. In general, a performance constrained system achieves the lowest power dissipation when delivering just enough performance as required (or vice versa for a power constrained system). The Lagrange cost function defined in equation (2) enables us to find trade-off between power and performance by varying the parameter  $\lambda$ . However, what is the right value of  $\lambda$  that exactly meets the power (or performance) constraint is difficult to find out.

It is known that when the value of  $\lambda$  increases, the Q-learning algorithm will favor policies that have better performance and vice versa. By comparing the actual power consumption (or performance) to the power (or performance) constraint, we can adjust the value of  $\lambda$  using a feedback control. However, without knowing the exact relation among power, performance and  $\lambda$ , the feedback control method will easily generate large overshoot or undershoot in measured output (i.e., power consumption or performance) and hence lead to an unstable system [Abdelzaher et al. 2008]. To limit the overshoot and undershoot, we propose to further confine the value of  $\lambda$  in a predefined range. The upper bound and the lower bound of the range are estimated from the long term average workload characteristics and the given power (performance) constraints using a neural network.

Given analysis leads to a 2-level control unit that tunes the value of  $\lambda$  to keep the system aligning to the given constraint. The proposed 2-level constraint tracking unit has a neural network based coarse grained controller in the first level to set the upper and lower bound of  $\lambda$  based on the long term average workload. It also has a feedback controller in the second level to fine tune the value of  $\lambda$  based on the instantaneous workload variations.

Here we consider the problem of maximizing performance for a given power constraint as an example to illustrate the 2-level controller. Its dual problem, that is,

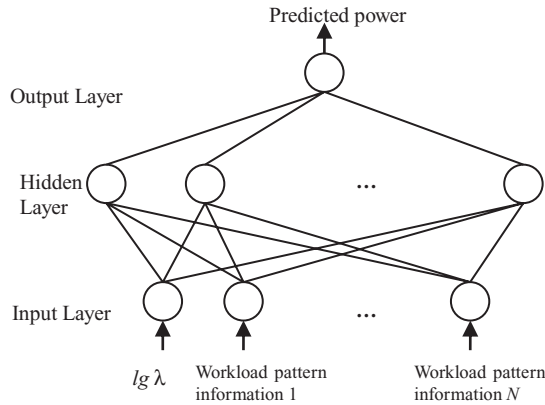


Fig. 4. Level 1 neural network.

minimizing power consumption for a given performance constraint can be solved in a similar way.

The concept of two level controllers has been successfully applied in many power/thermal management works. For example, [Wang et al. 2009] uses online model estimator to predict the workload and generate the desirable trajectory of reference power consumption at lower level and feedback control to keep the actual power consumption close to the trajectory. [Ayoub et al. 2011] performs core level proactive thermal management at lower level and socket level task scheduling at upper level. Here we need to point out that the controller in this section does not directly manage the control knobs. Instead, it controls the value of the Lagrange multiplier ( $\lambda$ ) which is used to realize power-performance trade-off in the Q-learning algorithm.

**4.3.1. Boundary Prediction Using Neural Network Models (Level 1 Controller).** The goal of the first level controller is to estimate the value of  $\lambda$  that exactly meets the performance/power constraint considering only the long term average workload. The estimated value is denoted as  $\hat{\lambda}$ . Using  $\hat{\lambda}$  as a reference, we set the upper and lower bound of the second level controller which fine tunes the value of  $\lambda$  based on the instantaneous workload variation.

We found from the experiments that it is difficult to construct a model that estimates  $\hat{\lambda}$  directly from power (performance) constraint. This is probably because our Q-learning algorithm has discrete behavior and it is very likely that slight change in  $\lambda$  does not make difference in control policy. In other words, the relation from the average power consumption (or performance) to  $\lambda$  is a one to many mapping instead of a properly defined function, and hence it is difficult to obtain. Fortunately, power (or performance) of a peripheral device is a monotonic increasing (or decreasing) function of  $\lambda$ . This means that we can use binary search to find the appropriate value of  $\hat{\lambda}$ , if there is a model that predicts the average achieved power (performance) based on the given  $\lambda$ . A neural network is used for such modeling purpose.

The neural network model adopted in this work has an input layer, an output layer and a hidden layer, as shown in Figure 4. The hidden layer consists of 5 neurons. For a given service provider, the neural network model predicts the average power consumption based on the selected trade-off factor  $\lambda$  and workload information.

In our experiments we observed that, when controlled by the learning based power management unit, the average power consumption of the device has a log-linear relation with the trade-off factor  $\lambda$ . Figure 5 gives the relation of the simulated power consumption and the value of  $lg\lambda$  of a hard drive whose read/write activities follows the HP

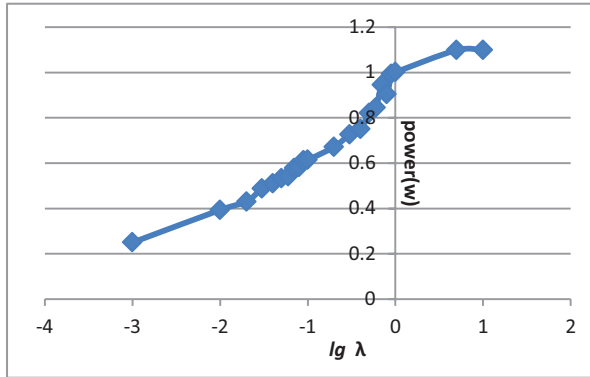


Fig. 5. Relation between power and  $\lg \lambda$  for a given workload.

Table I. Input selection vs. prediction error

Input selection	$i = 1$	$i = 6$	$i = 12$	$i = 1,2,3 \dots 15$	$i = 1, 6, 12$
Prediction error	27%	17%	30%	14%	8%

Cello99 trace [Ruemmler and Wilkes 1993]<sup>1</sup> Open Source software at [tesla.hpl.hp.com](http://tesla.hpl.hp.com)]. As we can see that their relation is approximately linear. To reduce the nonlinearity of the neural network model, we choose  $\lg \lambda$  instead of  $\lambda$  as one of its inputs.

What input variables should be selected for the neural network to represent the average workload characteristics is a nontrivial problem. For those peripheral devices where service speed is much faster than the request incoming speed, (for example, in general a hard disk drive can process all accumulated read/write request in very short time after the disk has been spun up), the input variables could be the probability distribution of the request inter-arrival time which reflects current workload pattern.

The probability distribution of the request inter-arrival time is represented by a set of variables. The  $i$ th variable gives the probability that the inter-arrival time  $t_{int}$  is greater than or equal to  $iT$ , where  $T$  is a user defined time period. Similar to many other estimation models, an accurate power estimator needs to have both good specificity and high sensitivity. Selecting too few input variables may lead to low sensitivity of the model as it misses much useful information. However, including too many input variables may cause low specificity because useful features will be covered by noises. We propose to use the greedy feature selection method [Caruana and Freitag 1994] to select only those variables that give the most information to the prediction of average power consumption.

In our experiment, a neural network is constructed to predict the power consumption of a hard disk drive under learning based power management. We select  $T$  as  $1/4T_{be}$ , where  $T_{be}$  is the break-even time which is the minimum amount of time that a device must stay in low power mode for the energy saving to equal the overhead of power mode switching.

Table I gives the prediction error for different input selections for the HP Cello99 workload. For more details of the hard disk drive model and the Cello99 workload, please refer to Section 5. As we can see, including too many features does not help to increase the accuracy of the model because this introduces more noise in the input and will actually decrease the specificity of the model. On the other hand, a model based on

<sup>1</sup>Open Source software at [tesla.hpl.hp.com](http://tesla.hpl.hp.com). <http://tesla.hpl.hp.com/opensource/>.

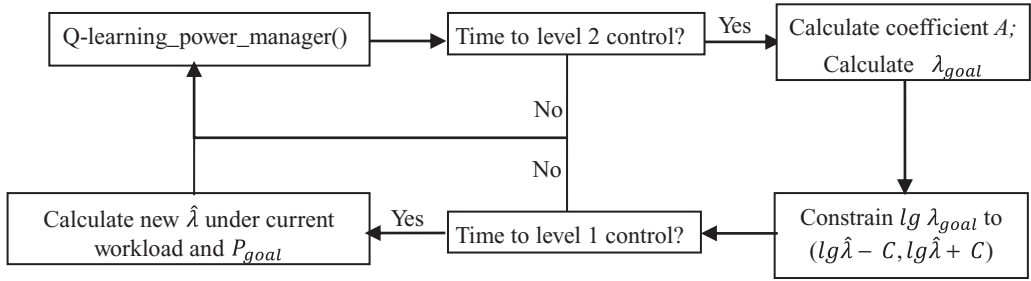


Fig. 6. Block diagram of the power control flow of the Q-learning power manager.

extremely few inputs is not accurate either because it does not have enough sensitivity to detect a workload change.

Considering the fact that  $T_{be}$  is  $4T$  for our experiment device, the selection of probabilities for idle intervals longer than  $T$ ,  $6T$  and  $12T$  is reasonable as they represent the short idle, medium idle, and long idle intervals, thus form relatively complete spectrum of idle interval information of the workload.

The training of the neural network relies on recorded operation information of the system. For better accuracy, different models may be constructed for different types of workload if they can be classified.

With the neural network, we predict the trade-off factor that exactly satisfies the given power (performance) constraint and denote the value as  $\hat{\lambda}$ . We confine the range of the trade-off factor to be  $(\hat{\lambda}/C, C\hat{\lambda})$ , where  $C$  is a constant that is greater than 1. Consequently, the value of  $lg\lambda$  is confined to the range  $(lg\hat{\lambda} - C, lg\hat{\lambda} + C)$ .

**4.3.2. Fine Adjustment Using Feedback Control (Level 2 Controller).** In order to fine-tune the value of  $\lambda$ , we use a linear model to approximate the relation between  $lg\lambda$  and the power consumption  $P$  for a given workload, that is,  $P = A * lg(\lambda) + B$ , where  $A$  and  $B$  are unknown coefficients. Such linear relationship has been observed in our experiment, as shown in Figure 5. The values of  $A$  and  $B$  are assumed to be constant when workload does not change abruptly and  $\lambda$  is confined to a limited range. Let  $P_{curr}$  and  $\lambda_{curr}$  be the current power consumption and current value of  $\lambda$ , also let  $P_{goal}$  and  $\lambda_{goal}$  be the power constraint and the corresponding value of  $\lambda$  that exactly achieves this power constraint. If  $\lambda_{curr}$  and  $\lambda_{goal}$  are not too far from each other, we will have Equation (5) and (6):

$$P_{curr} = A * lg \lambda_{curr} + B \quad (5)$$

$$P_{goal} = A * lg \lambda_{goal} + B. \quad (6)$$

Combining Equation (5) and (6), the goal value of  $\lambda$  can be calculated as the following:

$$\lambda_{goal} = \lambda_{curr} * 10^{\frac{P_{goal} - P_{curr}}{A}}. \quad (7)$$

The value of  $A$  can be obtained by observing the average power consumption of the system under different  $\lambda$ 's. Let  $P_1$  and  $P_2$  be the average power consumption of the system using  $\lambda_1$  and  $\lambda_2$ ,  $A$  can be calculated using Equation (8).

$$A = (P_1 - P_2) / (lg \lambda_1 - lg \lambda_2). \quad (8)$$

**4.3.3. Overall Flow.** Figure 6. gives the block diagram of the overall flow of the Q-learning power manager with constraint tracking. The function *Q-learning\_power\_manager()* is the basic Q-learning function shown in Figure 3 Both

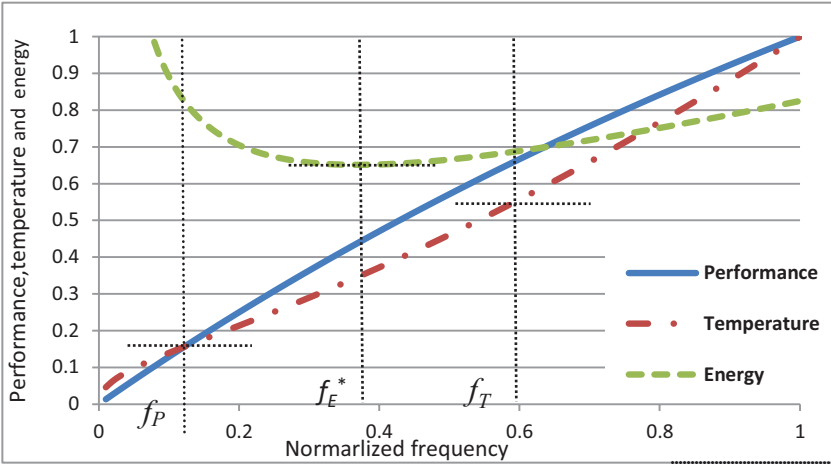


Fig. 7. Qualitative illustration of the relation between CPU temperature, performance, energy and clock frequency.

level 1 and level 2 controllers are triggered periodically. The level 2 controller is triggered at a higher frequency than the level 1 controller. In our experiment, the periods are set to 1000 and 200 cycles for level 1 and level 2 controllers respectively. When level 2 controller is triggered,  $A$  and  $\lambda_{goal}$  will be calculated using Equation (8) and (7). If  $lg\lambda_{goal}$  is out of the range  $(lg\hat{\lambda} - C, lg\hat{\lambda} + C)$ , it would be rounded to  $lg\hat{\lambda} - C$  or  $lg\hat{\lambda} + C$ . When level 1 controller is invoked, a new  $\hat{\lambda}$  will be predicted and the allowed range of  $\lambda$  will be adjusted accordingly. The learning rate factor (i.e.,  $Visit(s, a)$  in Equation (4)) will be reset every time when level 1 or level 2 controller is triggered because a new trade-off factor is found.

## 5. LEARNING-BASED CPU POWER MANAGEMENT

As explained in Section 1, a microprocessor working in batch mode has very different characteristic compared to the peripheral devices and hence requires different Q-learning model. In this section, we discuss how to apply the Q-learning algorithm to the power management of such microprocessor. Our goal is to minimize the CPU energy dissipation for the batch processing under the given execution time constraint. In order to demonstrate the Q-learning's capability of handling multidimensional constraints, we add the average die temperature as the second constraint. Please note that user may select any other measurement as the second constraint, because the construction of the Q-learning algorithm does not rely on any temperature mode or thermal analysis. The relationship of the CPU frequency and its energy, performance (i.e., inverse of the execution time), and temperature is qualitatively shown in Figure 7. The energy first decrease as the frequency reduces. If we further reduce the frequency, the energy will increase as the leakage power becomes dominant and power reduction is slower than the runtime increase. The CPU frequency that gives the minimum energy dissipation is denoted as  $f_E^*$ . The performance and temperature increases as the CPU frequency rises. However, since the clock speed for the memory subsystem does not change, the performance gain due to fast CPU will gradually slow down. Therefore, the performance is a concave function of CPU frequency.

We use  $f$  and  $v$  to denote the scaling ratios of the CPU voltage and frequency. They are calculated as  $f = F/F_{max}$  and  $v = V/V_{max}$ , where  $V_{max}$  ( $F_{max}$ ) and  $V$  ( $F$ ) represent the maximum voltage (frequency) and the scaled voltage (frequency) of the processor



respectively. We assume that  $f$  and  $v$  have one to one correspondence, that is, for each CPU frequency there is a matching supply voltage level. We use  $\mu$  to represent the percentage of time the application is processed on CPU and cache. It is referred as *CPU intensiveness*. It is calculated as the following [Dhiman and Rosing 2009]:

$$\mu = 1 - \frac{\text{cycles\_lli\_stalled} + \text{cycles\_ld\_stalled}}{\text{total number of cycles}}, \quad (9)$$

where *cycles\_lli\_stalled* and *cycles\_ld\_stalled* are the number of cycles during which the CPU is stalled for instruction and data fetches. They can be recorded periodically in many commercial processors. Though there are other architectural events related to  $\mu$ , such as the cycle of stalls due to TLB miss, branch prediction miss and etc., they are less dominant than the cache miss event and usually cannot be monitored at the same time with the cache miss events. Hence, they will be ignored in this formula.

The CPU intensiveness varies from application to applications or even inside the same application. Its value affects how the energy and execution time change with voltage and frequency scaling. When the value of  $\mu$  reduces, the CPU spends more time waiting for the memory reads/writes. Less performance gain can be achieved by increasing the CPU frequency. On the other hand, because most of the time is spent on memory subsystem, reducing the clock frequency will cause less increase of the execution time. Therefore, the energy optimal frequency  $f_E^*(\mu)$  is lower. In this work, we assume that the CPU has been characterized so that for specific  $\mu$ , the minimum energy frequency  $f_E^*(\mu)$  is known.

At the end of each time slot, three cost variables are updated, which include energy cost ( $C_E$ ), performance cost ( $C_P$ ) and temperature cost ( $C_T$ ). While the die temperature can be read from on-chip temperature, the other two cannot be obtained directly. This is because both of them depend on the execution time, which is unknown during the runtime as we assume no prior knowledge of the workload. To overcome this limitation, in this work we define the energy cost at cycle  $t$  as the normalized deviation from the energy minimum frequency of current workload (i.e.,  $f_E^*(\mu_t)$ ) to the energy cost, that is,  $C_{E,t} = |f_t - f_E^*(\mu_t)| / (f_{\max} - f_{\min})$ , where  $f_t$  and  $\mu_t$  are frequency and CPU intensiveness during cycle  $t$ ,  $f_{\max}$  and  $f_{\min}$  are the maximum and minimum frequency of the CPU. The similar energy cost definition is also used in Dhiman and Rosing [2009]. We also define the performance as the normalized deviation from the maximum clock frequency, that is  $C_{P,t} = (f_{\max} - f_t) / (f_{\max} - f_{\min}) * \mu_t$ . Finally, the temperature cost of cycle  $t$  is defined as the temperature increase from cycle  $t-1$ , that is,  $C_T = (T_t - T_{t-1}) / T\_range\_intervals$  where  $T\_range\_intervals$  is the maximum temperature change in two adjacent time intervals. It is about 2°C in our experiment system.

We partition the environment state so that the cost functions remain relatively constant during the same state. Based on this criterion, the state is classified based on four parameters: ( $f$ ,  $T$ ,  $IPS$ ,  $\mu$ ). They represent the clock frequency, the temperature, the instructions per second ( $IPS$ ) and the workload CPU intensiveness respectively. Let  $N$  be the total number of clock frequencies supported by the processor, we use  $f_i$  to denote the  $i$ th clock frequency, with  $f_0$  representing the minimum frequency. We discretize the possible range of temperature into  $M$  levels, with  $T_0$  representing the ambient temperature and  $T_{M-1}$  representing the maximum temperature threshold.

In Section 4, we solve the performance constrained power optimization problem by dynamically adjusting the weight coefficient of the Lagrange cost function to find minimum power policy that exactly meets the performance constraint. The rationale of this approach is that power is a decreasing function of response time for the peripheral device. Such relation no longer exists between energy and performance for a batch mode CPU as shown in Figure 7 Multidimensional constraints make things even more

Table II. Characteristics of Service Provider

$P_{active}(W)$	$P_{sleep}(W)$	$P_{tran}(W)$	$T_{tran}(s)$	$T_{be}(s)$	Speed (MB/s)
1.1	0.1	1.42	3	4	16.6

complicated. For example, as shown in Figure 7, assume the minimum frequency that satisfies the performance constraint is  $f_P$  and the maximum frequency that satisfies the temperature constraint is  $f_T$ . Our goal is to constrain the performance and temperature, while at the same time minimizing the energy. As we can see, it is not possible to find a frequency that satisfies both performance and temperature constraints exactly. Hence we have to modify the cost function of the Q-learning algorithm to decouple these two constraints.

We denote the performance and temperature constraints as  $con_P$  and  $con_T$ . We also use  $\Delta_P$ , and  $\Delta_T$  to represent the difference between the constraint and the actual average penalty during a history window for performance and temperature respectively. The value of  $\Delta$  will be positive if the system outperforms the user constraint during the history window, otherwise it will be negative. Because we are interested in constraining only the average performance and average temperature, we consider the system to be performance and temperature *bounded* when  $C_P \leq con_P + \Delta_P$  and  $C_T \leq con_T + \Delta_T$ , otherwise, the system is *unbounded*. In this way, if the system has been outperforming the user constraint during the past, it will be considered performance (or temperature) bounded even if the cost of the current cycle is a little higher than the constraint.

The modified cost function considers 3 scenarios:

$$C = \begin{cases} C_E & \text{if } C_P \leq con_P + \Delta_P \ \& \ C_T \leq con_T + \Delta_T \\ C_E + \alpha \cdot C_P & \text{if } C_P > con_P + \Delta_P \ \text{and } C_T \leq con_T + \Delta_T \\ C_E + \alpha \cdot C_T & \text{if } C_P \leq con_P + \Delta_P \ \text{and } C_T > con_T + \Delta_T. \end{cases}$$

In this equation,  $\alpha$  is a large positive number. Based on the modified cost function, when the system is bounded in both performance and temperature, the Q-learning algorithm will search for policies that minimize the energy cost. As soon as the system becomes unbounded in either performance or temperature, the cost function will be modified and the Q-learning algorithm will put more emphasis on improving the performance or temperature that has violated the constraint. It can be proved that as long as the performance and temperature constraints are feasible, they will not be violated at the same time.

We need to point out that the proposed approach manages the voltage and frequency of a single CPU. For a multicore system, this approach is viable if cores work independently to each other. It can be extended to manage multiple cores with interactions simultaneously by augmenting the state space of the Q-learning model to consider the joint state of different cores.

## 6. EXPERIMENTAL RESULTS AND ANALYSIS

### 6.1. Experimental Results for Peripheral Device Power Management

*6.1.1. Experimental Setup.* In this section, we will present the simulation results of learning based power management for peripheral devices. The target SP in the experiment is a hard disk drive (HDD). Table II summaries the power and performance of the hard disk drive. These parameters are obtained from real hard disk datasheet.<sup>2</sup> The  $T_{be}$  value is round up to the nearest integer for the simplicity of calculation. In the table, the  $P_{tran}$  and  $T_{tran}$  are power and performance overhead of sleep to active

<sup>2</sup>TOSHIBA Hard Disk Drive Specification 1.8 inch Hard Disk DriveMK6006GAH/MK4006GAH/MK3006GAL.

Table III. Characteristics of Different Reference Policies

Policy	Characteristics
Fixed Timeout(1 ~ 5)	Timeout = $1 * T_{be} \sim 5 * T_{be}$
Adaptive Timeout	Initial timeout = $3 * T_{be}$ Adjustment = $+/- 1 * T_{be}$
Exponential Predictive	$I_{n+1} = \beta i_n + (1 - \beta) I_n, \beta = 0.5$
Expert-based Learning	Uses the above seven policies as experts.

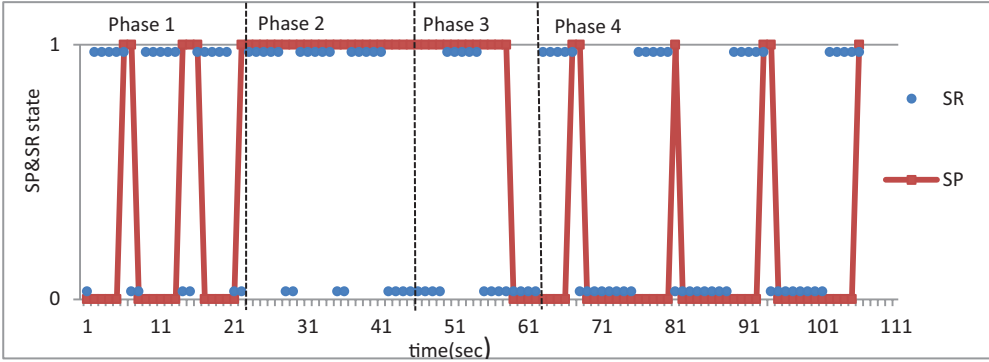


Fig. 8. Response of Q-learning power manager to synthetic trace 1.

transition. The active to sleep transition is not mentioned in the datasheet and hence will be ignored in the model.

In order to evaluate the performance of our learning based power management policy, we developed a fast performance evaluation framework of the HDD using OMNeT++.<sup>3</sup> OMNeT++ is a discrete event simulation environment written in C++.

The performance of the Q-learning based power management is compared with the expert based learning algorithm proposed in [Dhiman and Rosing 2009]. Table III lists five fixed timeout policies, an adaptive timeout policy, and an exponential predictive policy. These 7 heuristic policies form the set of experts for the expert-based learning algorithm. Hence, the expert-based learning algorithm overcomes the limitation of any of these single heuristics by dynamically selecting one of them to adapt with the changing workload. A control knob factor  $\alpha$  is provided for power performance trade-off [Dhiman and Rosing 2009].

**6.1.2. Results for Synthetic Workload.** In this experiment, we use two synthetic workload to intuitively illustrate how the Q-learning based power manager is able to adapt to the workload change.

In Figure 8, the blue dots represent the state of SR. It is categorized into 2 states, with 0 represents zero incoming rate and 1 represents nonzero incoming rate. We assume that when there are incoming request, they come in at a constant rate. The red solid line represents the state of SP, with 0 representing sleep mode and 1 representing active mode. The SP is controlled by a Q-learning based power manager. The synthetic workload trace we created shows a changing pattern during the time. At the beginning of the experiment, the SR's idle time is always 2 seconds, which is smaller than the system  $T_{be}$ , hence the system should not go to sleep during the idle interval. While later in the experiment, the SR's idle time is increased to 8 seconds which is longer than  $T_{be}$ . From the behavior of the SP we can see that the power manager undergoes 4 phases.

<sup>3</sup>OMNeT+. <http://www.omnetpp.org/>.

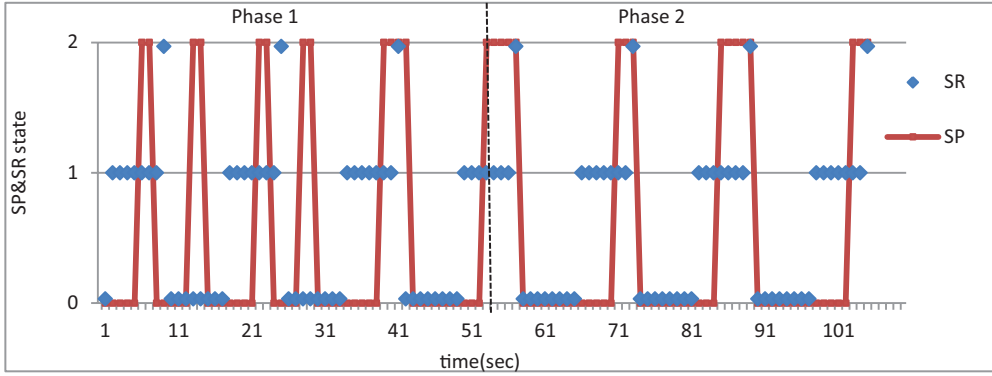


Fig. 9. Response of Q-learning power manager to synthetic trace 2.

- (1) Phase 1. The power manager learns the pattern of the workload.
- (2) Phase 2. The pattern has been learnt and the power manager decides to keep the SP active during the short idle period.
- (3) Phase 3. After the workload changed, the power manager start learning again.
- (4) Phase 4. The new pattern has been learnt and SP will go to sleep during long idle period.

Note in our system, the SP service rate is always much higher than the request incoming rate. The SP only takes a short time to process the accumulated requests after activated.

In the second example shown in Figure 9, the SR has 2 different incoming rates, and hence overall 3 states. States 0, 1 and 2 represent idle, low incoming rate and high incoming rate respectively. The workload has a clear pattern which always starts with a long idle period followed by a long period of low incoming rate and then a short period of high incoming rate. After that the pattern repeats itself. As we can see in Figure 9, during the learning phase (i.e., phase 1) the power manager tried different control policies by turning the SP on and off at different time. Eventually, it found that the best policy for this workload is to turn on the device in the middle of the low rate incoming period and turn it off immediately after the high incoming rate period is over. Note that none of the seven heuristic policies in Table III classifies SR into different states; hence they are not able to detect the workload pattern in this example.

**6.1.3. Q-learning Power Management for Real Workload.** In this experiment, we evaluate the performance of the Q-learning based power manager using two different types of workloads.

- (1) Workloads extracted from HP cello99 traces [Ruemmler and Wilkes 1993; Open Source software at tesla.hpl.hp.com]. Cello99 trace records file system read write activities of HP data center. All the requests with the same PID within one microsecond are merged into one large request. One interesting observation we have found is that hourly request incoming rate has strong correlation to the time of a day. Figure 10 shows the hourly request incoming rate for 3 days. As we can see, the peak and bottom occurs at approximately the same time. This observation agrees with Ruemmler and Wilkes [1993] and it indicates that similar applications are running at the same period of time on different days. Such property can be used to gather training data to construct the neural network based power (performance) prediction model presented in Section 4.3. We extracted 3 workloads (i.e., HP-1, HP-2 and HP-3) at different time of the day.

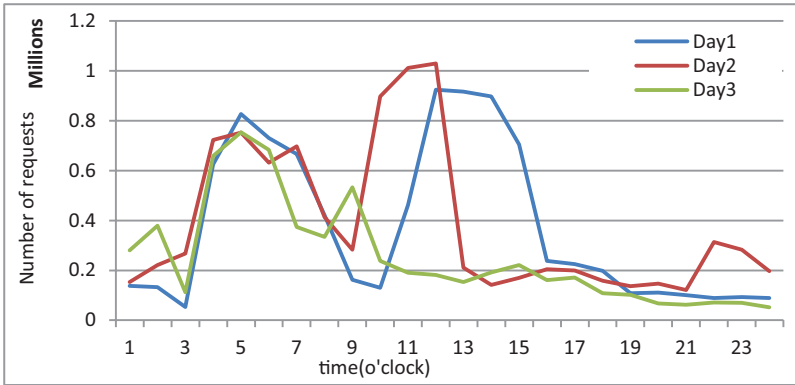


Fig. 10. Three consecutive days' requests from HP hard disk traces.

Table IV. Characteristics of Workload Traces

Trace name	Duration(sec)	No. of total requests after merging	No. of idle time $\geq T_{be}(4 \text{ sec})$
HP-1	14322	14994	1127
HP-2	14375	44468	332
HP-3	14387	151404	742
Desktop-1	21634	18036	1166
Desktop-2	1026	27782	43

- (2) Workloads collected from the desktop computer [Tan et al. 2009]. Using Windows Performance Monitor, we collected hard disk read/write request sequences from two different desktop workstations whose hard disk usage level differs significantly. We stopped collection when the generated file size reaches 5MB, which is equivalent to 70,000 read/write requests in the sequence. The first trace was collected in the afternoon when a set of applications were running simultaneously with high disk I/O activities, resulting in a short collection time (i.e., about 1000 seconds). The other trace was collected at night when only two applications were running and it takes more than 20000 seconds to complete the collection.

Table IV summaries the characteristics of the HP and desktop workload traces that we use.

Both Q-learning algorithm and expert-based algorithm can achieve different power-performance trade-off by controlling the trade-off factors  $\lambda$  and  $\alpha$  respectively. By varying these trade-off factors, we generate multiple power management policies with different power/latency trade-offs. Figure 11 shows these power latency trade-off points for these two learning based power management algorithms tested using 5 real workload traces. The results for power management using the traditional Q-learning algorithm without VSS enhancement are also shown in those figures. To better show the trend of power/latency trade-off, we use solid line to sequence the power/latency points following the decreasing order of the value of corresponding trade-off factors. Note in this set of experiment, learning rate  $\varepsilon_{(o,a)}$  in Equation (4) is reset to 1 periodically to adapt to the change of the workload patterns.

From Figure 11, four observations can be made.

- (1) Expert-based algorithm generally outperforms Q-learning algorithm for low-latency high performance scenario. This is because all the experts used in the expert-based algorithm are designed for high performance and they will turn on

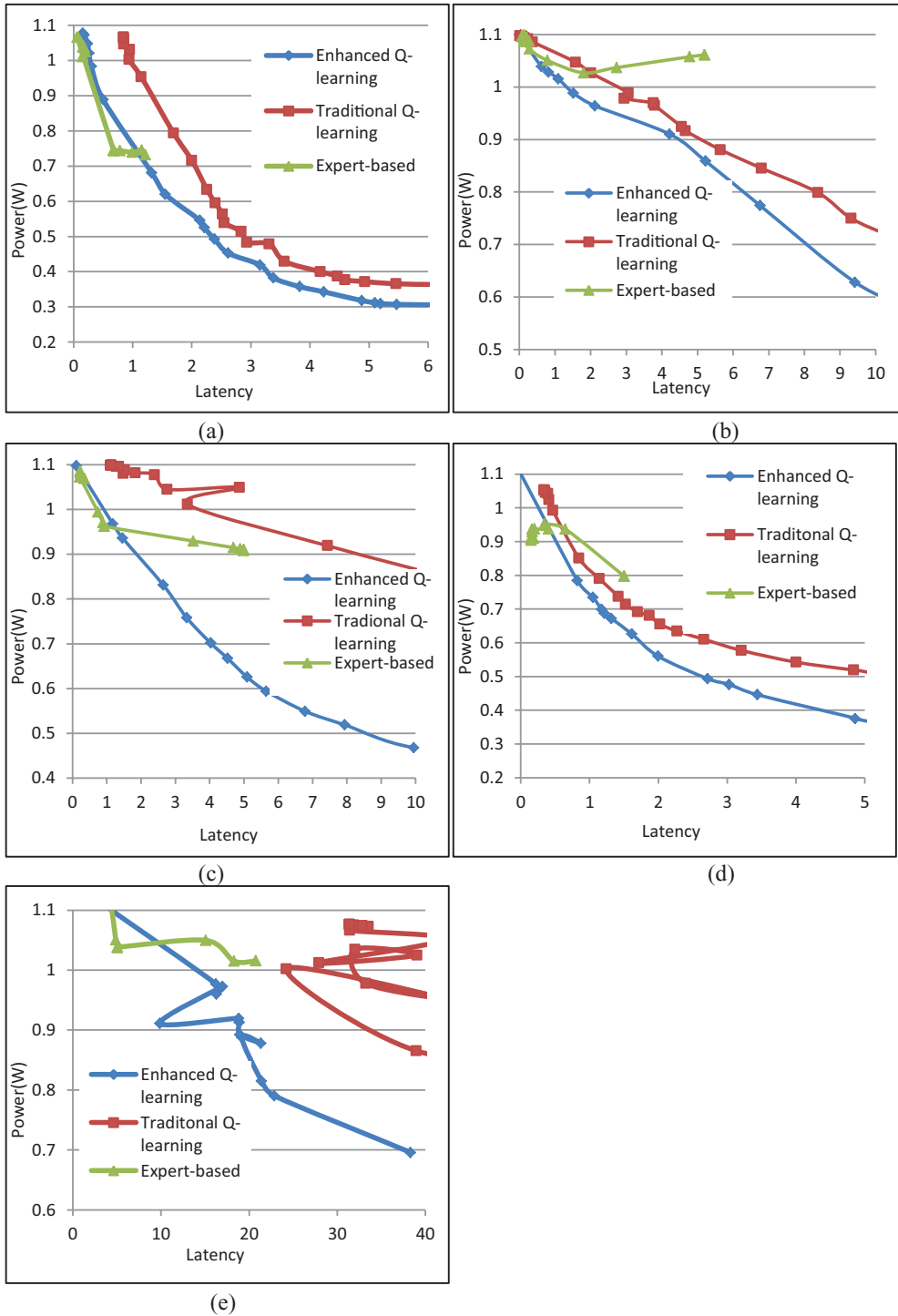


Fig. 11. Power/Latency trade-off curves for workload. (a) HP-1; (b) HP-2; (c) HP-3; (d) Desktop-1; (e) Desktop-2

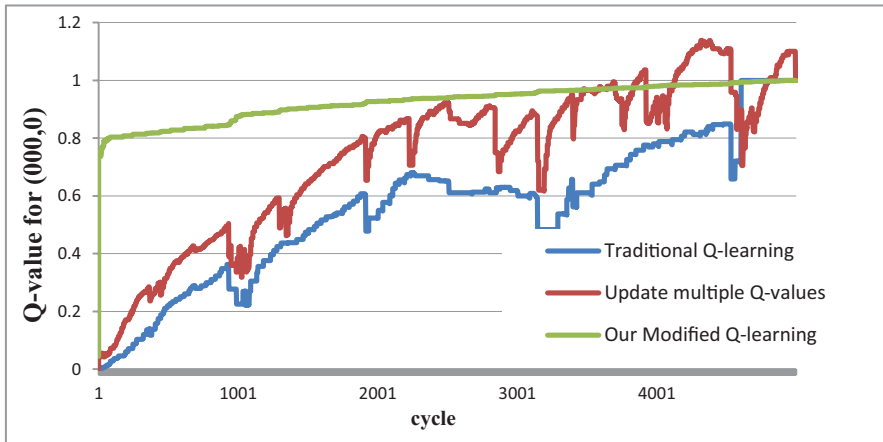


Fig. 12. Q-value for observation-action pair(000,0).

the device as soon as a request comes in. In contrast to the expert based algorithm, the Q-learning algorithm allows the device to buffer the requests.

- (2) The Q-learning outperforms the expert based policy when the performance is relatively less important than the power consumption and it provides wider range of power-performance trade-off. The trade-off curve for Q-learning based power management is also much smoother than the curve for expert based power management. For Q-learning based management, power is a decreasing function of performance in all cases except the last one (i.e., desktop workload 2). While for expert-based power management, such monotonic relation is not obvious for several test cases (i.e., HP-1, HP-2, Desktop-1 and Desktop-2).
- (3) For workload Desktop-2, the red curve moves forward and backward. This means the latency (and the power) of the device does not have a monotonic relation with the trade-off factor. This is probably because the workload is very intensive and changes so rapidly, the traditional Q-learning algorithm does not have enough time to find the best policy before the workload changes. Our enhanced Q-learning algorithm exhibit better monotonic relation between power/latency and the trade-off factor, due to its fast convergence speed.
- (4) Our proposed VSS technique in Section 4.2 significantly improves the power latency trade-off curves due to the faster speed of Q-learning convergence. Figure 12 compares their convergence speed.

As we mentioned earlier, two enhancement techniques are used to speed up the convergence. First, the learning rate  $\epsilon$  is modified as an adaptive factor associated with the observation-action pair. Second, we update multiple Q-values instead of only one Q-value in each learning step using the VSS technique. Figure 12 shows the change of the Q-value of state-action pair (000, 0) for 3 different Q-learning algorithms: the traditional Q-learning (without variable learning rate and multiple Q-value update), the Q-learning algorithm with multiple Q-value update (but no variable learning rate), and our enhanced Q-learning algorithm. The state action pair (000, 0) represents the scenario when there are no incoming requests, no waiting requests in queue, and HDD is in *sleep* mode, and the power management command is to continue sleeping. As we can see, comparing to the other two learning algorithms, the changes of Q-value for the proposed modified Q-learning is smoother. Moreover, it converges much faster to the stable state. The similar trend can be found with all other state action pairs.

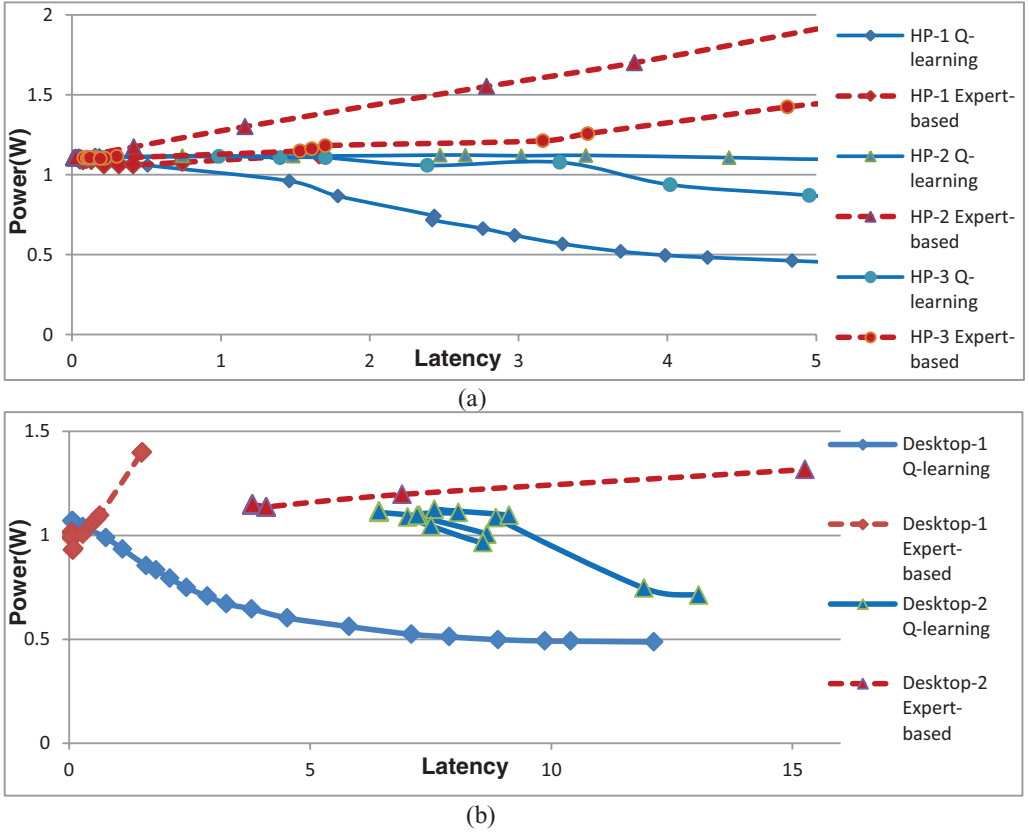


Fig. 13. Power/Latency trade-off curves for (a) HP workloads (b) desktop workloads when  $T_{be} = 8$  seconds.

In terms of complexity, as we mentioned before, the enhanced Q-learning is  $O(|SP| \times |SQ| \times |A|)$ , the expert-based algorithm is  $O(n)$  where  $n$  is the number of simple experts used, and the traditional Q-learning is  $O(1)$ .

**6.1.4. Adaptivity of the Learning Based Power Management to Different Hardware.** In the third experiment, we consider power management of systems with special hardware that has a large penalty to go to sleep mode. The purpose of this experiment is to test the robustness of the Q-learning algorithm in working with different types of service provider. Different devices will have different power and transition characteristics. For example, the server's hard disk or the CD-ROM will always have longer  $T_{be}$  than personal computer's hard disk.

In this experiment, we increase the  $T_{be}$  of the HDD from 4 seconds to 8 seconds by increasing the  $P_{tran}$  and run the simulation again. Figure 13 shows the results for 3 HP workload traces and 2 desktop traces respectively. As we can see, the policies found by the expert-based algorithm do not give proper trade-off between power and performance. When the latency increases, the power consumption of the system increases too. The policies found by the Q-learning based power management are still capable of trading performance for power reduction. This is because the expert-based algorithm is restricted by the selected experts, which are a set of time-out policies whose time out values are multiples of  $T_{be}$ . When the value of  $T_{be}$  gets larger, the flexibility of these time-out policies reduces. Compared to the idle intervals in the request pattern,



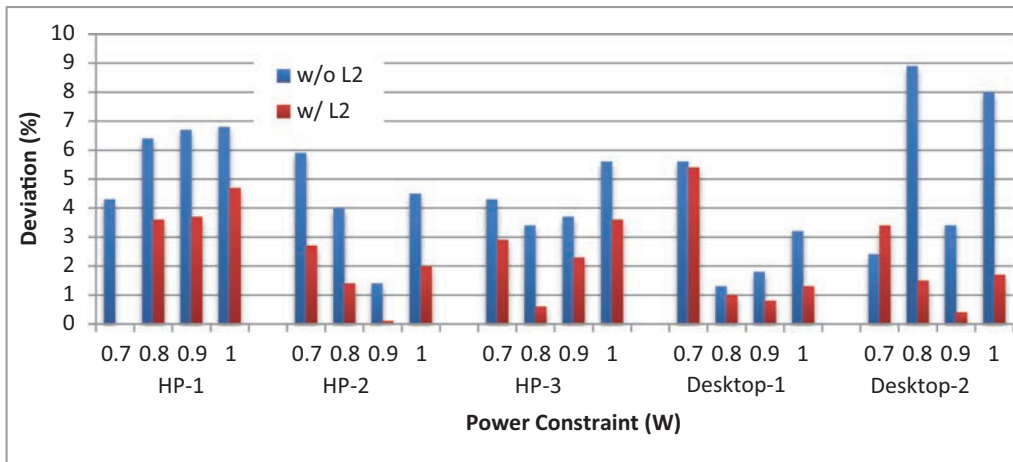


Fig. 14. Relative average power deviation from user constraints.

these timeout values are either too small or too large. When the performance requirement reduces, the power manager will put the device into sleep mode more frequently. However, without proper timeout threshold, there will be lots of mistakes and frequent on-off switches. Hence, not only latency, the power will also increase. This problem can be solved if more timeout policies with finer resolution of timeout threshold are added as experts. However, this means higher complexity. This experiment also shows that with different workload patterns and different hardware devices, the performance of expert-based algorithm depends highly on the right selection of different experts.

In contrast to the expert based policy, the Q-learning power management algorithm not only learns and adapts to different workloads, but also adapt to different hardware, both of which are the requirements of a good power management algorithm [Pettis and Lu 2009].

**6.1.5. Tracking the Power (Latency) Constraint.** In this section, we will demonstrate the effectiveness of our proposed power (latency) constraint tracking algorithm. It is measured by the difference between the actual average power consumption (or latency) and the user specified constraint. The closer the actual value and the constraint are, the more effective the constraint tracking algorithm is.

In the first set of experiments, we consider the problem of performance optimization with power constraint and show the effectiveness of level 1 and level 2 controllers. First we compare our Q-learning algorithm with the same algorithm that has level 2 constraint tracking controller disabled (i.e., the value of  $\lambda$  is set exactly equal to  $\hat{\lambda}$  predicted by the neural network). Please note that we divide each workload trace into 2 segments, a training sequence and a testing sequence. The neural network is trained using the training sequence, and then applied to the testing sequence to collect the comparison results.

As we mentioned in Section 4.3, the function of level 2 controller is to keep the average power consumption close to the power constraint using feedback control. In this experiment, we focus on how much the average power consumption deviates from the power constraint. We vary the power constraint from 0.7 to 1. The average power over the entire simulation time is measured and the relative deviation of the average power is calculated which is the relative difference between actual average power and the power constraint. The comparison results are shown in Figure 14. As we can see, adding level-2 controllers can reduce the constraint tracking error from 4.58% to

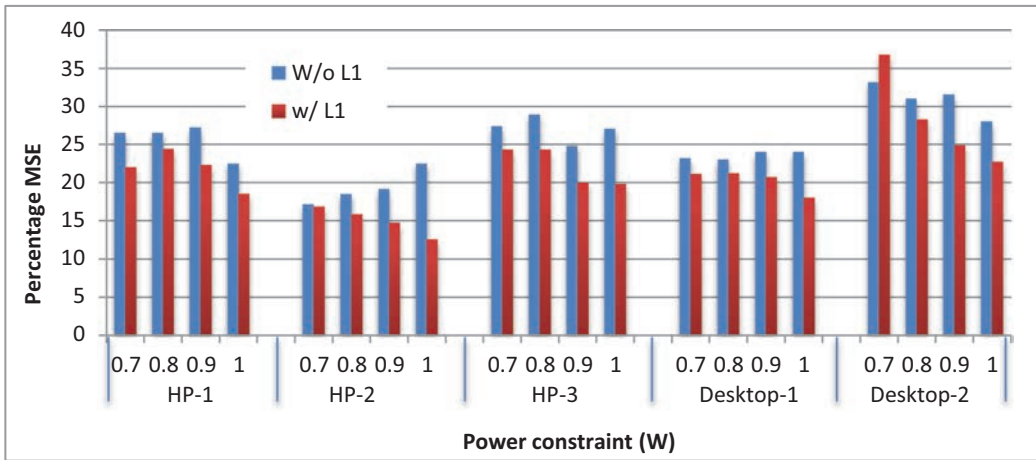


Fig. 15. Percentage MSE of instant power versus user constraints.

2.15%, which stands for approximately 50% improvement. The capability of being able to stay close to the power constraint is very useful for systems powered by battery [Fei et al. 2008] or energy harvesting units [Kansal et al. 2007], where the budget of available power is predefined.

While using level-2 controller helps to keep the average power consumption close to the constraint, using level-1 controller helps to reduce the variation of the instantaneous power consumption. In next experiment, we compare out Q-learning algorithm with the same algorithm that has level 1 controller disabled (i.e., the value of  $\lambda$  is controlled only by the feedback controller.) The percentage mean square errors (MSE) between the instantaneous power consumption and the power constraint is calculated. Here we use the average power consumption over 200 cycles to represent the instantaneous power. The comparison results are given in Figure 15. As we can see, including level-1 controller reduces the variation of the power by 15.6% in average.

The previous experiment shows that including a level 2 controller could reduce the average power deviation from 4.58% to 2.15%. Although this represents 50% relative improvement, the absolute improvement is only 2%, which is quite small. This is because our level 1 predictor is already accurate in predicting the average power for the given  $\lambda$ . Therefore, using level 1 control we can find the trade-off factor close to the right value. However, level 2 controller is especially important when we couldn't construct a good model to predict  $\lambda$  in level 1. In the next set of experiments, we consider the problem of power minimization with performance constraint.

Because the rate of incoming request to a hard disk drive has very large variation, it is difficult to train a neural network model that could accurately predict the average latency. Therefore, the level 1 control can only provide a very loose bound and the search for the appropriate trade-off factor largely depends on the level 2 controller. Instead of confining  $\lambda$  around  $\hat{\lambda}$  which is predicted by the neural network, we constrain it within the range  $(C\lambda_{curr}, \lambda_{curr}/C)$  where  $\lambda_{curr}$  is value of  $\lambda$  that have recently been used. By doing this, we prevent  $\lambda$  from changing too abruptly and stabilize the latency change through the time.

Figure 16. shows the percentage deviation of the average latency compared to the latency constraint. We vary the latency constraint from 1 to 4 for the three HP traces as well as Desktop-1. Note that a different set of latency constraints is used for trace Desktop-2. This is because it is extremely intensive and no power management policy

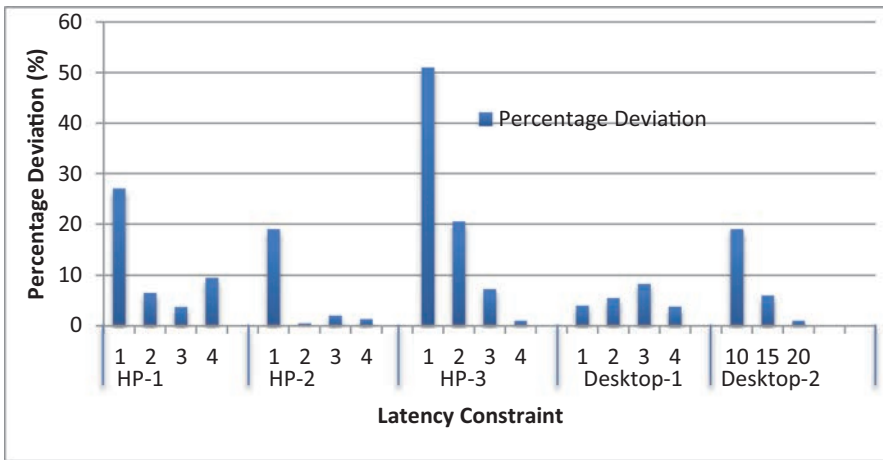


Fig. 16. Relative average latency deviation for latency constrained power management.

except the “always on” policy can meet the same latency constraints as we use for the other 4 traces. The experimental results show that in average the Q-learning based power manager can maintain the system performance within about 10% of the given constraint. Furthermore, it is much easier to track a loose performance constraint than a tight performance constraint. Note that the data shown in Figure 16. are relative deviation. When the latency constraint is tight, although the relative deviation is large, its absolute value is still small.

## 6.2. Q-Learning Based Microprocessor Power Management

In the second set of experiments, we evaluated our Q-learning algorithm for microprocessor power management. We implemented the Q-learning based DVFS controller on a Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor.<sup>4</sup> The processor supports 4 frequency levels: 2GHz, 2.33GHz, 2.67GHz, 3GHz. The Linux kernel we use is version 2.6.29.

We used *coretemp* driver in the Linux kernel to read the temperature sensor of the processors. The default driver updates temperature readings once every second and we modified it to be every 10ms to achieve our required granularity. We used *cpufreq* driver in Linux based on Enhanced SpeedStep technology<sup>5</sup> of Intel Core 2 processor to adjust the processor’s frequency. We used Perform2 tool<sup>6</sup> to monitor performance events of the processors. We ran the experiments on one core and fixed the frequency of the other core to be the minimum. The Q-learning controller was triggered every 20ms. Empirically, this interval will not exert too much overhead to the processor while still capable of tracking the change of workload. The overhead of frequency change is only about 20us. We use the option “`-print-interval=20`” provided by *pfmon* to control its sampling period to also be 20ms.

<sup>4</sup>Intel® Core™2 Duo Processor E8000 and E7000 Series: <http://download.intel.com/design/processor/datashts/318732.pdf>

<sup>5</sup>Enhanced Intel SpeedStep® Technology - How To Document. <http://www.intel.com/cd/channel/reseller/asm-na/eng/203838.htm>

<sup>6</sup>Perfmon2: <http://perfmon2.sourceforge.net/>

Table V. Constraining Performance and Temperature.

Performance \ Temperature	0.34 (constraint)	0.67 (constraint)	1 (constraint)
0.34(constraint)	0.33	0.58	0.78
0.67(constraint)	0.46	0.34	0.30
1 (constraint)	0.24	0.38	0.62
	0.58	0.51	0.44
	0.23	0.37	0.60
	0.61	0.55	0.43

We use benchmarks from MiBench<sup>7</sup> and MediaBench<sup>8</sup> to form the workload of the evaluation system. Our goal is to generate workloads with changing CPU intensiveness. The benchmarks we selected are: bitcount\_small, basicmath\_small, qsort\_large, tiff2rgba, mpeg4dec, and jpeg200dec together with a simple custom application with only CPU busy loops. Their CPU intensiveness varies from 11% to almost 100% with an average of 82% according to our measurement. Each benchmark running a little more than 0.2s under minimum frequency is a running unit. We serialized 100 running units of different benchmarks in 4 different random orders to construct 4 different “workloads”. Every experiment result reported here is the average of the 4 “workloads”. We need to point out that MiBench is a benchmark mainly designed for embedded systems, while our experiment is done on a Core 2 Duo workstation. However, it is our goal is to test how well the power management controller adapts to different workload. Our objective in selecting the benchmark is to create a variety of workload with different CPU intensiveness. We found that the two programs from MediaBench (i.e., mpeg4dec and jpeg200dec) has relatively lower CPU intensiveness (in the range of 70% to 80%) while the program in MiBench has much higher CPU intensiveness which is above 95%. So the combination of MiBench and MediaBench gives us such variety.

Since we have 4 frequency levels (i.e.,  $f_0 \sim f_3$ ) on our platform, we partition the workload CPU intensiveness  $\mu$  into 4 states, so that  $f_i$  is corresponding to the ideal frequency  $f_E^*$  when  $\mu = \mu_i$ ,  $0 \leq i \leq 3$ . Such partition enables us to measure the energy penalty using the deviations from the ideal frequency. The temperature and *IPS* are also empirically partitioned into 4 states.

As discussed in Section 5, the performance is measured by the deviation from the maximum frequency; the energy is measured by the deviation from the energy optimal frequency. The temperature is the average normalized temperature of the CPU observed by the temperature sensor.

We run the Q-learning based power management for minimum energy under different performance and temperature constraints. The results are shown in Table V. Each column in the table represents a performance constraint and each row represents a temperature constraint. Because our platform only supports 4 frequency levels and the frequency increases linearly at an equal step from level 0 to level 3, the corresponding normalized temperature and performance for those frequency levels should also change roughly at an equal step. To better show our results, we set the constraints to be 0.34, 0.67 and 1 as shown in the tables. Each entry gives the actual performance and temperature of the system under the power management. For example, the cell

<sup>7</sup>Mibench: <http://www.eecs.umich.edu/mibench/>

<sup>8</sup>MediaBench: <http://euler.slu.edu/~fritts/mediabench/>

in row 1 and column 1 of Table V shows that the actual normalized temperature and performance of the system is 0.46 and 0.33 respectively when the performance and temperature constraint are both set to 0.34. The entries are shaded differently according to the energy dissipation of the system. The lighter the cell is, the lower energy dissipation we achieve. As we can see, the upper left cell has the darkest color because it corresponds to the most stringent user constraints and hence leaves almost no room for the optimization of the 3<sup>rd</sup> metrics. On the contrary, the bottom right cell has the lightest color because it corresponds to the most relaxed constraints.

We can see that sometimes the Q-learning controller cannot find a policy that satisfies both user constraints. For example, the entry (0.34, 0.34) in Table V has constraint violation. Sometime, the controller finds policies that exactly satisfies one of the constraints and outperforms the other (e.g., entry (0.34, 0.67) in Table V. For the rest of times, the controller finds policies that outperform both user constraints. This clearly shows that the relation among T, P and E are not monotonic. We cannot optimize one metric by setting the other (one or two) metrics exactly to the given user constraints. For example, consider cell (0.67, 0.67) in Table V. The user set a loose performance and temperature constraint ( $con_P = con_T = 0.67$ ) in order to optimize the energy. However the result shows that the policy that minimizes the energy actually does not have to work so slowly and will not generate so much heat. Clearly in this test case, we have  $f_P \leq f_E^* \leq f_T$  for the average  $\mu$  of the workloads, where  $f_E^*$  is the energy optimal frequency,  $f_P$  and  $f_T$  are the frequencies that exactly satisfy the performance and temperature constraints respectively. However, we need to point out that the data reported here is the average of 4 different workloads over 80 seconds simulation. Although in average the CPU intensiveness satisfies the condition  $f_P \leq f_E^* \leq f_T$ , the instantaneous value of  $\mu$  for each individual workload may not always satisfy this condition. That is why the entry (0.67, 0.67) has a darker shade than the cell (1.0, 1.0), which indicates a higher energy. The later, due to the extremely loose performance and temperature constraints, can always reach the energy optimal point  $f_E^*$ .

The experimental results also show that, generally without the prior knowledge of hardware and software, our Q-learning based controller can correctly learn the trade-off space and give effective control policies. The only information we need to know related to the hardware is the mapping of different workload CPU intensiveness to the ideal working frequency  $f_E^*$  for the energy optimization purpose. This requirement can be removed if the processor's power consumption can be measured during the runtime.

In order to compare the performance of the proposed learning algorithm with the state-of-the-art approach, we modified the expert-based algorithm in Dhiman and Rosing [2009] for energy management with the consideration of performance and temperature. In our modified expert based approach, we choose different voltage and frequency configurations as experts. The cost function is weighted sum of energy cost, performance cost and temperature cost defined in Section 5, that is,  $C = \alpha C_E + \beta C_P + \gamma C_T$ . The modified expert based approach is actually a reduced version of the expert based controller proposed in Coskun et al. [2008]. While their work considers energy and thermal management for multicore system running interactive applications, our problem is a little different. If we remove some control knobs that are specific to multicore system (e.g., task migration) and interactive applications (e.g., adaptive random and DPM) from their work, and also remove thermal gradient and thermal cycle from their cost function, then it will be reduced to our modified expert based controller.

As we mentioned previously, because the energy and performance no longer have monotonic relation, and also because there are two constraints (i.e., performance and temperature), it is very difficult to find a set of weight factors that minimizes the energy while satisfying the given constraints. Therefore, we sweep the weight factors  $\alpha$ ,  $\beta$ , and  $\gamma$  to generate a set of power management policies that gives different energy/performance/temperature trade-offs. Their corresponding energy, performance

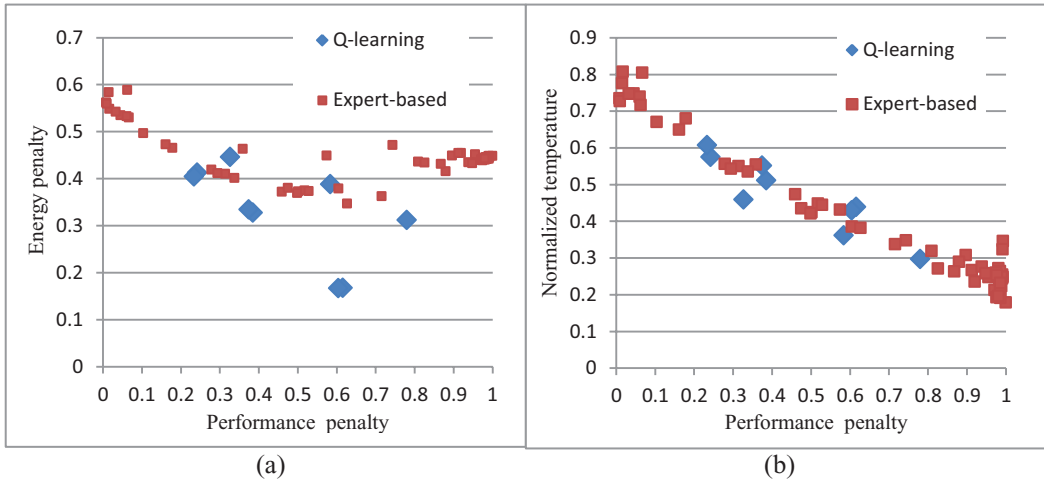


Fig. 17. Energy, temperature and performance results of Q-learning algorithm with constraints and expert-based algorithm without constraints: (a) energy versus performance; (b) temperature versus performance

and temperature values are plotted in Figure 17, represented by the red dots. These policies do not try to meet any performance/temperature constraints. They provide the best effort to minimize the weighted sum of energy, performance and temperature costs.

Our modified expert based control is a reduced version of the controller proposed in Coskun et al. [2008], which is a comprehensive work on multicore system energy and thermal management using expert based framework. The scope of their problem is a little different from ours. They consider multicore system running interactive applications (i.e., Web server), while we consider single core management for CPU running batch processing. In addition to energy, performance and temperature, they also consider thermal gradient and thermal cycles in the optimization. If we remove those control knobs that are specific to multicore system (e.g., task migration) and interactive applications (e.g., adaptive random and DPM) from their work, and also remove thermal gradient and thermal cycle from their cost function, then it will be reduced to the same modified expert based controller implemented in our experiment.

In Table V we have already shown that our approach can meet the performance and temperature constraints. Figure 17. also shows that, for the same performance level, our approach results in the same or even less energy and similar temperature, compared to the expert based approach, which does not guarantee the performance and temperature constraints. Therefore, the Q-learning based approach performs better for this problem.

## 7. CONCLUSIONS

In this article, we propose a general model solving the dynamic power management problem using Q-learning. The Q-learning power manager does not require any prior knowledge of the workload or the system model while it can learn the policy online with real-time incoming tasks and adjusts the policy accordingly. Convergence speed acceleration techniques are proposed that make the Q-learning algorithm more efficient in non-Markovian environment. A 2-level power or performance control model is proposed to accurately keep the system at the given power (or performance) constraint, to achieve maximum performance (or minimum power consumption). Simulation results prove that our Q-learning power management algorithm is able to achieve better power performance trade-off than the existing expert-based power management algorithm.

The Q-learning algorithm is also extended for the CPU power management by controlling its DVFS settings. The control algorithm is capable to achieve minimum energy while meeting the user constraints in performance and temperature.

## REFERENCES

- ABDELZAHER, T., DIAO, Y., HELLERSTEIN, J. L., LU, C., AND ZHU, X. 2008. Introduction to control theory and its application to computing systems. SIGMETRICS Tutorial, Annapolis, MD.
- AGARWAL, Y., SAVAGE, S., AND GUPTA, R. 2010. SleepServer: a software-only approach for reducing the energy consumption of PCs within enterprise environments. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '10)*. 22.
- AHMAD, I., RANKA, S., AND KHAM, S. U. 2008. Using game theory for scheduling tasks on multi-core processor for simultaneous optimization of performance and energy. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*. 1–6.
- AYOUB, R. AND ROSING, T. 2009. Predict and act: Dynamic thermal management for multi-core processors. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '09)*. 99–104.
- AYOUB, R., INDUKURI, K., AND ROSING, T. S. 2011. Temperature aware dynamic workload scheduling in multi-socket CPU servers. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 30, 9, 1359–1372.
- AYOUB, R., NATH, R., AND ROSING, T. 2012. JETC: Joint energy thermal and cooling management for memory and CPU subsystems in servers. In *Proceedings of the IEEE 18th International Symposium on High Performance Computer Architecture (HPCA '12)*. 1–12.
- CAI, L., PETTIS, N., AND LU, Y. 2006. Joint power management of memory and disk under performance constraints. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 25, 2697–2711.
- CARUANA, R. AND FREITAG, D. 1994. Greedy attribute selection. In *Proceedings of the 11th International Conference on Machine Learning, 1994*.
- CHOI, K., SOMA, R., AND PEDRAM, M. 2004. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ration of off-chip access to on-chip computation times. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 24, 18–28.
- CHOU, C.L. AND MARCULESCU, R. 2010. Designing heterogeneous embedded network-on-chip platforms with users in mind. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 29, 1301–1314.
- CHOUHDARY, P., AND MARCULESCU, D. 2009. Power management of voltage/frequency island-based system using hardware-based methods. *IEEE Trans. VLSI Syst.* 17, 3, 427–438.
- COSKUN, A. K., ROSING, T. S., AND WHISNANT, K. 2007. Temperature aware task scheduling in MPSoCs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*. 1659–1664.
- COSKUN, A. K., ROSING, T. S., WHISNANT, K., AND GROSS, K. 2008. Temperature-aware MPSoC scheduling for reducing hot spots and gradients. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC '08)*. 49–54.
- COSKUN, A. K., ROSING, T. S., AND GROSS, K. C. 2008. Temperature management in multiprocessor SoCs using online learning. In *Proceedings of the 45th Annual Design Automation Conference (DAC '08)*. 890–893.
- COSKUN, A. K., ROSING, T. S., AND GROSS, K. C. 2009. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 28, 10, 1503–1516.
- DHIMAN, G. AND ROSING, T. S. 2009. System-level power management using online learning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 28, 676–689.
- FEL, Y., RAVI, S., RAGHUNNATHAN, A., AND JHA, N. K. 2007. Energy-optimizing source code transformation for operating system-driven embedded software. *ACM Trans. Embed. Comput. Syst.* 7.
- FEL, Y., ZHONG, L., AND JHA, N. K. 2008. An energy-aware framework for dynamic software management in mobile computing systems. *ACM Trans. Embed. Comput. Syst.* 7.
- GNADY, C., BUTT, A. R., CHARLIE, Y., AND LU, Y. 2006. Program counter-based prediction techniques for dynamic power management. *IEEE Trans. Comput.* 55, 641–658.
- HWANG, C.-H., AND WU, A. 2000. A predictive system shutdown method for energy saving of event-driven computation. *ACM Trans. Des. Autom. Electron. Syst.* 5, 226–241.
- IPEK, E., MUTLU, O., MARTINEZ, J. F., AND CARUANA, R. 2008. Self-optimizing memory controllers: a reinforcement learning approach. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA '08)*. 39–50.
- JEJURIKAR, R., PEREIRA, C., AND GUPTA, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st Annual Design Automation Conference (DAC '04)*. 275–280.

- KANSAL, A., HSU, J., ZAHEDI, S., AND SRIVASTAVA, M. B. 2007. Power management in energy harvesting sensor networks. *ACM Trans. Embed. Comput. Syst.* 6.
- LANGEN, P. AND JUURLINK, B. 2006. Leakage-aware multiprocessor scheduling for low power. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '06)*. 60.
- LIU, Y., YANG, H., DICK, R. P., WANG, H., AND SHANG, L. 2007. Thermal vs energy optimization for DVFS-enabled processors in embedded systems. In *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED '07)*. 204–209.
- LIU, W., TAN, Y., AND QIU, Q. 2010. Enhanced q-learning algorithm for dynamic power management with performance constraint. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*. 602–605.
- MARTINEZ, J. F. AND IPEK, E. 2009. Dynamic multicore resource management: a machine learning approach. *IEEE Micro* 29, 8–17.
- PENDRITH, M. 1994. On reinforcement learning of control actions in noisy and nonMarkovian domains. Tech. rep. NSW-CSE-TR-9410, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia.
- PETTIS, N. AND LU, Y. 2009. A homogeneous architecture for power policy integration in operating systems. *IEEE Trans. Comput.* 58, 945–955.
- QIU, Q., TAN, Y., AND WU, Q. 2007. Stochastic modeling and optimization for robust power management in a partially observable system. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*. 779–784.
- ROSIK, T. S., BENINI, L., GLYNN, P., AND DE MICHELI, G. 2001. Event-driven power management. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 20, 840–857.
- RUEMMLER, C. AND WILKES, J. 1993. Unix disk access patterns. In *Proceedings of the USENIX Winter Conference*. 405–420.
- SIKORSKI, K., AND BALCH, T. 2001. Model-based and model-free learning in Markovian and non-Markovian environments. In *Proceedings of the Agents-2001 Workshop on Learning Agents*.
- SMULLEN, C. W., COFFMAN, J., AND GURUMURTHI, S. 2010. Accelerating enterprise solid-state disks with non-volatile merge caching. In *Proceedings of the International Green Computing Conference (IGCC '10)*. 203–214.
- TAN, Y., LIU, W., AND QIU, Q. 2009. Adaptive power management using reinforcement learning. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '09)*. 461–467.
- TAN, Y. AND QIU, Q. 2008. A framework of stochastic power management using hidden Markov model. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '08)*. 92–97.
- TAN, Y., MALANI, P., QIU, Q., AND WU, Q. 2006. Workload prediction and dynamic voltage scaling for MPEG decoding. In *Proceedings of the Asia and South Pacific Design Automation (ASP-DAC '06)*. 911–916.
- TESAURO, G., DAS, R., JONG, N., AND BENNANI, M. 2006. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proceedings of 3rd IEEE International Conference on Autonomic Computing (ICAC '06)*. 65–73.
- TESAURO, G., DAS, R., CHAN, H., KEPHART, J., LEVINE, D., RAWSON, F., AND LEFURGY, C. 2007. Managing power consumption and performance of computing systems using reinforcement learning. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS '07)*.
- THEOCHAROUS, G., MANNOR, S., SHAH, N., GANDHI, P., KVETON, B., SIDDIQI, B., AND YU, C.-H. 2006. Machine learning for adaptive power management. *Intel Tech J.* 10, 299–312.
- VARATKAR, G. V., AND MARCULESCU, R. 2004. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Trans. VLSI Syst.* 12, 1.
- WANG, Y., MA, K., AND WANG, X. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. 314–324.
- WEDDLE, C., OLDHAM, M., QIAN, J., AND WANG, A. A. 2007. PARAID: A gear-shifting power-aware RAID. *ACM Trans. Storage* 3, 13.
- YEO, I., LIU, C., AND KIM, E. 2008. Predictive dynamic thermal management for multicore systems. In *Proceedings of the 45th Annual Design Automation Conference (DAC '08)*. 734–739.
- ZANINI, F., ATIENZA, D., BENINI, L., AND DE MICHELI, G. 2009. Multicore thermal management with model predictive control. In *Proceedings of the 19th European Conference on Circuit Theory and Design (ECCTD '09)*. 90–95.

Received January 2012; revised July 2012; accepted October 2012