Achieving Autonomous Power Management Using Reinforcement Learning

HAO SHEN, Syracuse University
YING TAN, Binghamton University
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 history and improve itself as the environment changes. This paper 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 first consider the problem of power management of a peripheral device and discuss system modeling and algorithm construction of the Q-learning agent. Enhancement techniques are proposed to speed up the convergence and better maintain the required performance (or power) constraint in a dynamic system with large variation. The proposed technique is further extended to dynamically learn the voltage and frequency scaling policy for joint energy and thermal management of a microprocessor. Compared to 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]: Real-time systems and embedded Systems
General Terms: Design, Experimentation, Management, Performance
Additional Key Words and Phrases: Power management, thermal management, machine learning, computer

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 tradeoff. Dynamic power management (DPM) has proven to be an effective technique for power reduction at 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. 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 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 paper, 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/penalty 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 helps to better track the performance (or power consumption) constraints. The uniqueness of the proposed work is summarized as follows:
1. The power manager 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. The partial knowledge of the system is utilized to accelerate the convergence speed and enhance the runtime tracking of the performance (power consumption) constraint.
3. The learning based power management controller is first designed to control the power modes of a peripheral device (such as a hard disk drive) to achieve the power and performance tradeoff. It is then extended to control the voltage and frequency selection of a CPU to achieve tradeoff among energy, performance and temperature.
4. The performance of the proposed power management controller is evaluated by both simulation and real measurement.

Compared to our previous works in [Tan et al. 2009; Liu et al. 2010], this work has the following major extensions.
1. The model construction technique is improved to handle real devices with more diversified workload and practical constraints. For example, we improved the state partition techniques to cover workloads with large variations.
2. 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 tradeoff 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.
3. In addition to personal PCs, the proposed power management technique is evaluated using the HP hard disk traces which resemble the workload of large data centers.
4. The learning based power controller is further applied 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 paper is organized as follows: In Section 2, we will talk about the related work including the expert-based DPM algorithm which will be used as a comparison with our algorithm. In Section 3 and 4, we will introduce the general model construction and enhancement techniques of the proposed learning based power management respectively. In Section 5, we present the simulation results of the learning based power management for peripheral device. In Section 6, we extend the model to control the voltage and frequency of a CPU and present the performance of the controller implemented on a Dell workstation. 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 system architecture and component design 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 later performs online to dynamically control the power with the respect of performance constraints. The DPM and DVFS techniques belong to the second category.

2.1 Related Works in DPM

One of the most widely used runtime low power technique is dynamic power management (DPM). 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 ideal 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 \cite{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. \cite{Theocaris et al. 2006} proposed a user-based adaptive power management technique that considered user annoyance as a performance constraint. \cite{Ahmad et al. 2008} converts the scheduling task on multiprocessor into a cooperative game theory problem to minimize the energy consumption and the makespan simultaneously, while maintaining deadline constraints. All of the above works require offline model construction and policy optimization, therefore they cannot adapt to the workload changes in real-time.

Online learning algorithms are natural choice for real-time adaptive power management. \cite{Cai et al. 2006} presented 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. \cite{Gniady et al. 2006} uses program counters to learn the access patterns of applications and predicts when an I/O device can be shut down to save energy. \cite{Weddle et al. 2007} uses 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. References \cite{Martinez and Ipek 2009} and \cite{Ipek et al. 2008} proposed a machine learning approach for multicore resource management based on-chip hardware agents that are capable of learning, planning, and continuously adapting to changing demands. Those works also used the machine learning technique to perform the DRAM bandwidth scheduling for a maximum throughput. In \cite{Dhiman and Rosing 2009}, the authors proposed a learning algorithm that dynamically selects different experts to make power management decisions at runtime. This approach leverages the fact that different experts outperform each other under different workloads and hardware characteristics.

### 2.2 Related Works in DVFS

Traditionally, reducing the voltage and clock frequency of a digital IC is considered to give cubical reduction in its power consumption and linear reduction in its performance. However this trend has started to change. As semiconductor technology keeps scaling down, leakage power becomes more and more dominant in modern processors \cite{Dhiman and Rosing 2009; ITRS}. Although the DVFS technique effectively reduces the dynamic energy, it also increases the leakage energy because the system has to be kept active for a longer time \cite{Dhiman and Rosing 2009; Jejurikar et al. 2004}. On the other hand, as the CPU speed increases, the limited memory bandwidth has become the performance bottleneck for many applications with intensive memory access. For those memory bound applications, the DVFS technique incurs less performance penalty because the memory subsystem still works under a constant frequency \cite{Dhiman and Rosing 2009; Langen and Juurlink 2006; Jejurikar et al. 2004; Choi et al. 2004}.

Many research works have been proposed to find the optimal DVFS scheduling for energy and temperature reduction. Reference \cite{Choi et al. 2004} uses 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. The authors of \cite{Dhiman and Rosing 2009} take different frequencies of the processor as different experts. These experts are dynamically selected based on their weight, which is a function of the energy dissipation and performance penalty and is updated online. Reference \cite{Tan et al. 2006} first presents a workload prediction model for MPEG decoder and the predicted workload is further used to guide the voltage and frequency scaling. Reference \cite{Coskun et al. 2009} presents a set of new job scheduling and power management policies for chip multiprocessors. Their impact on chip lifetime is evaluated. The authors of reference \cite{Ge and Qiu 2011} use machine learning to adaptively change the frequency of the processor for the thermal management of multimedia applications. Reference \cite{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.

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 micro-controllers or microprocessors; the DPM is usually for the peripheral devices, such as hard disk drives or network interface. In this paper, we
consider reinforcement learning based runtime power management for both types of systems. Although they share the same basic concept, the learning agents that manage DPM and DVFS should be constructed differently.

3. General Architecture of Q-learning based Power Management

In this chapter, 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 \( \pi \), i.e. 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 following expression.

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \varepsilon_t(s_t, a_t) \left( \text{expected discounted penalty} \times \left[ \frac{P_{t+1}}{\text{penalty}} + \gamma \frac{\min_a Q(s_{t+1}, a) - Q(s_t, a_t)}{\text{min future value}} \right] \right)
\]

In the above expression, \( s_t, a_t \) and \( p_t \) are the state, action and penalty at time \( t \) respectively, and \( \varepsilon_t(s_t, a_t) \in (0,1) \) 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, i.e. \( \pi(s) = \min_{a \in A} Q(s, a) \).

The value of \( Q(s_t, a_t) \) is updated at the beginning of cycle \( t+1 \), i.e., 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 Power Management System Model

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 further can be divided into hardware and software environments. The hardware environment could be any peripherals device such as hard disk and network card or the processor itself. 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
Figure 1. Illustration of system under power management

(i.e. a setting of the control knobs) that has the lowest Q-value will be selected to control the status 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 \( nT \).

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-Markovian. 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]. The author of [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] the 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 pre-characterized. 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 tradeoff for a peripheral device. The peripheral device, also known as input/output device, can be considered as a service provider (SP). The
request to the device is buffered in a service request queue (SQ) maintained by the OS. The software application that accesses the device is considered as service requestor (SR). The environment of the power management controller is the composition of the observation of SP, SQ and SR.

4.1 State Partition and Penalty Calculation
The observed power mode of SP can naturally be used to represent its state. The 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 tradeoff between power and performance, we define a Lagrangian cost for each state and action pair \((s, a)\) that combines the costs of power consumption \((\text{power}(s, a))\) and performance penalty \((\text{penalty}(s, a))\):

\[
C(s, a; \lambda) = \text{Power}(s, a) + \lambda \text{penalty}(s, a)
\]  

(2)

When SP is in a stable state, \(\text{Power}(s, a)\) and \(\text{penalty}(s, a)\) 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 non-symmetric 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, i.e.

\[
\text{power}(s, a) = \frac{(P_{A2B} * T_{A2B} + P_{B2A} * T_{B2A})}{2},
\]

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, i.e.

\[
\text{penalty}(s, a) = \frac{(q_{A2B} * T_{A2B} + q_{B2A} * T_{B2A})}{2},
\]

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 didn’t 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 equation (2).

\[
Q(s, a; \lambda) = (1 - \varepsilon(s,a))Q(s, a; \lambda) + \varepsilon(s,a)(C(s, a; \lambda) + \min_{a'}Q(s', a'; \lambda))
\]  

(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 \(s'\), i.e. \(Q(s, a) = (1 - \epsilon)Q(s, a) + \epsilon(Q(s, a) + \min_{a'} Q(s', a'))\). 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 \((sp', sr, sq)\), we will update all Q values corresponding to the state-action pairs \((sp', sr, sq')\), \(\forall sq' \in SQ\) \(\forall a' \in A\) (Here the symbol ‘\(\forall\)’ means virtual states and actions). This requires us to know the next state of a state-action pair \((sp', sr, sq')\), \(a'\) even though it is not currently being visited. More specifically, given the condition that the system was in state \((sp, sr, sq)\) and 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 \((sp', sr, sq')\) and an action \(a'\) \((a' \neq a\) may be the same) taken.

Given the current state \(sp', and action \(a'\), it is not difficult for us to find the next state \(sp_{t+1}'\) as the SP model is pre-characterized. 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 how many requests 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 \(sp'\). Because SP has been pre-characterized, this information can also be estimated fairly accurately. After the next state is determined, the Q values of the state-action pair \((sp_{t+1}', sr_{t+1}', sq_{t+1}')\) that has been virtually visited can easily be calculated. In the rest of the paper, 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 traditional Q-learning, the learning rate \(\epsilon(s, a)\) 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

\[
\epsilon(s, a) = \frac{\mu}{\text{Visit}(s, a)}
\]

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

Figure 2 gives the pseudo code 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 the 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 $(S_t, a_t, \lambda)$ as well as all the virtual state action pairs $(S_t', a_t'; \lambda)$.

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 $\lambda$.

1. Calculate the cost $C(S_{t-1}, a_{t-1}; \lambda)$ using Equation (2);
2. Calculate the Q-value $Q(S_{t-1}, a_{t-1}; \lambda)$ using Equation (3);
3. For each $sp_{t-1}' \in SP$
   - For each action $a_{t-1}' \in A$
     - If $(sp_{t-1} \neq sp_{t-1}' \land sq_{t-1} \neq sq_{t-1} \land a_{t-1} \neq a_{t-1}')$
       - *Do not update the Q-value of the real state action pair twice* /
     - Given the virtual state action pair $(S_t', a_{t-1}; \lambda)$, find the projected next state $S_t' = (sp_t', sr_t', sq_t')$;
     - Calculate the cost $C(S_t', a_{t-1}; \lambda)$ using Equation (2);
     - Calculate the Q-value $Q(S_t', a_{t-1}; \lambda)$ using Equation (3);
   - /* end if*/
   - /* end for*/
     - Choose action $a_t$ with $\min_{a_t} Q(S_t, a_t; \lambda)$.

---

Figure 2. Pseudo code for Q-learning power manager using the VSS technique

---

### 4.3 Power (Performance) Tracking using 2-level Controller

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 tradeoff 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.

The above analysis leads to a 2-level control unit that tunes the value of $\lambda$ to keep the system aligning to the given power (performance) constraint. The proposed 2-level power (performance) tracking unit has a neural network based coarse grain controller in the lower 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 upper 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, i.e. minimizing power consumption for a given performance constraint can be solved in a similar way.
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.

To construct a model that estimates $\hat{\lambda}$ directly from power (performance) constraint is difficult, 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 relation, and therefore is not a function. Fortunately, power (or performance) 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 3. 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 tradeoff 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 tradeoff factor $\lambda$. Figure 4 gives the relation of the simulated power consumption and the value of $1g\lambda$ of a hard drive whose read/write activities follows the HP Cello99 trace [Ruemmler and Wilkes 1993; Open Source software at 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 $t_{in}$ 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 $1/4T_{be}$. Table 1 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 extremely few inputs is not accurate either because it does not have enough sensitivity to detect a workload change.

<table>
<thead>
<tr>
<th>Input selection</th>
<th>$i = 1$</th>
<th>$i = 6$</th>
<th>$i = 12$</th>
<th>$i=1,2,3\ldots15$</th>
<th>$i = 1, 6, 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction error</td>
<td>27%</td>
<td>17%</td>
<td>30%</td>
<td>14%</td>
<td>8%</td>
</tr>
</tbody>
</table>

Considering the fact that $T_{be}$ is $4T$ for our experiment device, the selection of probability distribution for intervals of length longer than $T$, $6T$ and $12T$ is reasonable as they represent the
short idle, medium idle, and long idle intervals, thus forms a relatively complete 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. The range of the actual tradeoff factor is set to be \((\bar{\lambda}/C, \bar{C}\bar{\lambda})\), where \(C > 1\) and \(\bar{\lambda}\) is the tradeoff factor predicted to exactly satisfy the given power (performance) constraint by the neural network. In this way, the value of \(lg\bar{\lambda}\) is confined to the range \((lg\bar{\lambda} - C, lg\bar{\lambda} + C)\).

![Diagram of Level 1 neural network](image)

**Figure 3.** Level 1 neural network

### 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 \(lg\bar{\lambda}\) and the power consumption \(P\) for a given workload, i.e. \(P = A \ast lg(\lambda) + B\), where \(A\) and \(B\) are unknown coefficients. Such linear relationship has been observed in our experiment as shown in Figure 4. 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 \(P_{curt}\) and \(\bar{\lambda}_{curt}\) be the current power consumption and current value of \(\lambda\), also let \(P_{goal}\) and \(\bar{\lambda}_{goal}\) be the power constraint and the corresponding value of \(\lambda\) that exactly achieves this power constraint. If \(\bar{\lambda}_{curt}\) and \(\bar{\lambda}_{goal}\) are not too far from each other, we will have equation (5) and (6):

![Graph showing relation between power and lg\lambda](image)

**Figure 4.** Relation between power and \(lg\lambda\) for a given workload

\[
P_{curt} = A \ast lg \bar{\lambda}_{curt} + B \quad (5)
\]

\[
P_{goal} = A \ast lg \bar{\lambda}_{goal} + B \quad (6)
\]
Combining Equation (5) and (6), the goal value of $\lambda$ can be calculated as the following:

$$\lambda_{\text{goal}} = \lambda_{\text{curr}} \times 10^{\frac{r_{\text{goal}} - r_{\text{curr}}}{A}}$$

(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).

$$A = \frac{(P_1 - P_2)}{(lg\lambda_1 - lg\lambda_2)}.$$  

(8)

Figure 5 gives the block diagram of the power control flow of the Q-learning power manager. The function $Q$-

learning\_power\_manager() is the basic Q-learning function shown in Figure 2. Both level 1 and level 2 controllers are triggered periodically. The level 2 controller has 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 $\lambda_{\text{goal}}$ will be calculated using Equation (8) and (7). If $lg\lambda_{\text{goal}}$ is out of the range $(lg\lambda - C, lg\lambda + C)$, it would be rounded to $lg\lambda - C$ or $lg\lambda + C$. When level 1 controller is invoked, a new $\lambda$ will be predicted and the allowed range of $\lambda$ will be adjusted accordingly. The learning rate factor $(Visit(s,a)$ in Equation (4)) will be reset every time when level 1 or level 2 controller is triggered because a new tradeoff factor is found.

![Figure 5. Block diagram of the power control flow of the Q-learning power manager](image)

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1 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 2 summarizes the power and performance of the hard disk drive. These parameters are obtained from real hard disk datasheet [TOSHIBA Hard Disk Drive Specification]. The $T_{be}$ value is round up to the nearest integer for the convenience of simulation. In the table, the $T_{trans}$ and $T_{tran}$ are power and performance overhead of sleep to active transition. The active to sleep transition is mentioned in the datasheet hence will be ignored in the model.

<table>
<thead>
<tr>
<th>Table 2: Characteristics of Service Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{sleep(W)}$</td>
</tr>
<tr>
<td>1.1</td>
</tr>
</tbody>
</table>

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++ [OMNeT+]. 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 3 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 tradeoff [Dhiman and Rosing 2009].
5.2 Q-learning based Power Management 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.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Timeout(1~5)</td>
<td>Timeout = 1* $T_{be}$ ~ 5* $T_{be}$</td>
</tr>
</tbody>
</table>
| Adaptive Timeout       | Initial timeout = 3 * $T_{be}$
|                        | Adjustment = +/-1 * $T_{be}$                                                    |
| Exponential Predictive | $l_{i+1} = \beta l_i + (1 - \beta)l_i$, $\beta = 0.5$                          |
| Expert-based Learning  | Uses the above seven policies as experts.                                       |

In Figure 6, the blue dots represent the state of SR. It is categorized into 2 states, with 0 represents zero incoming rate and 1 represents non-zero 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 $T_{be}$. 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:

Figure 6. Response of Q-learning power manager to synthetic trace 1

Figure 7. 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 7, 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 the Figure 7, 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 the policy that is most suitable to this workload pattern, that 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 above mentioned seven heuristic policies classifies SR into different states; hence they are not able to detect the workload pattern in this example.

5.3 **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 [Ruemmmer 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 8 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 reference [Ruemmmer 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 subsection 4.3. We extracted 3 workloads (i.e. HP-1, HP-2 and HP-3) at different time of the day.

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

![Figure 8. Three consecutive days' requests from HP hard disk traces](image-url)
Both Q-learning algorithm and expert-based algorithm can achieve different power-performance tradeoff by controlling the tradeoff factors $\lambda$ and $\alpha$ respectively. Figure 9 shows the power latency tradeoff curves for these two learning based power management algorithms tested using 5 real workload traces. Here the latency is the average queue length of SQ. The results for power management using the traditional Q-learning algorithm without VSS enhancement are also shown in those figures. Note in this set of experiment, learning rate $\epsilon_{(0,a)}$ in Equation (4) is reset to 1 periodically to adapt to the change of the workload patterns.

<table>
<thead>
<tr>
<th>Trace name</th>
<th>Duration(sec)</th>
<th>No. of total requests after merging</th>
<th>No. of idle time $\geq T_{be}(4$ sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-1</td>
<td>14322</td>
<td>14994</td>
<td>1127</td>
</tr>
<tr>
<td>HP-2</td>
<td>14375</td>
<td>44468</td>
<td>332</td>
</tr>
<tr>
<td>HP-3</td>
<td>14387</td>
<td>151404</td>
<td>742</td>
</tr>
<tr>
<td>Desktop-1</td>
<td>21634</td>
<td>18036</td>
<td>1166</td>
</tr>
<tr>
<td>Desktop-2</td>
<td>1026</td>
<td>27782</td>
<td>43</td>
</tr>
</tbody>
</table>
Figure 9. Power/Latency tradeoff curves for workload (a)HP-1; (b)HP-2; (c)HP-3; (d)Desktop-1; (e)Desktop-2

From Figure 9, 3 observations can be obtained:

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 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 tradeoff. The tradeoff 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) Our proposed VSS technique in Section 4.2 significantly improves the power latency tradeoff curves due to the faster speed of Q-learning convergence. Figure 10 compares their convergence speed.

As we mentioned earlier, two enhancement techniques are used to speed up the convergence. First, the learning rate $\varepsilon$ 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 10 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 \textit{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.

5.4 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 11 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 tradeoff 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, 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].
Tracking the Power (Latency) Constraint

In this section, we will demonstrate the effectiveness of our proposed power (latency) constraint tracking algorithm. The effectiveness of the constraint tracking algorithm is measured by the difference between the actual average power consumption (or latency) and the power (or latency) 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. The results are presented in Table 5. The first row in the table specifies the power constraint. For each workload trace, the first 2 rows present the actual average power consumption of the power managed system and the percentage difference compared to the constraint when the 2-level controller presented in Section 4.4 is applied with the Q-learning algorithm. The next 2 rows give the same results when only the level-1 controller (i.e. the value of \( \hat{\lambda} \) is set exactly equal to \( \lambda \) predicted by the neural network) is used. 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 results shown in Table 5.

As we can see from the table, only using level-1 controller already leads to fairly good tracking results. And adding level-2 controllers can reduce the constraint tracking error from 4.58% to 2.15%, which stands for approximately 50% accuracy improvement. The capability to accurately reaching the required power constraint and always able to find tradeoff between power and performance enables the Q-learning based power manager to maximize the system performance under the given power constraint. Such capability will be 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 Table 6, the variation of power consumption is reported as the percentage mean square errors (MSE) between the instantaneous power consumption and the power constraint. Here we use the average power consumption over 200 cycles to represent the instantaneous power. We compare the percentage MSE for systems with or without level-1 controller. As we can see, including level-1 controller reduces the variation of the power by 15.6% in average.

Figure 12 illustrates the instantaneous power consumption at different time with and without level-1 controller for HP-1 workload. The power constraint is set to 0.8W. From the figure we can clearly see that the level-1 controller reduces the vibration of the power consumption and keeps it close to the goal.
In the second set of experiments, we consider the problem of power minimization with performance constraint. Unfortunately, the rate of incoming request to a hard disk drive has very large variation. It could vary from 0 request to more than 200 requests in a second. So in a relatively short workload trace (HP traces we use are typically 4 hours each), it is more difficult to track the average latency constraint than to track the average power constraint. It is also difficult to train a neural network model that could accurately predict the average latency. Therefore, we use only the level 2 controller which tracks the latency constraint using feedback control. And 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. Table 7 shows the actual average latency of the power managed system and the percentage difference to the constraint.

Table 5. Power Constraints with and without Level 2

<table>
<thead>
<tr>
<th>Power Constraint(W)</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-1</td>
<td>Level 1&amp;2(W)</td>
<td>0.700</td>
<td>0.771</td>
<td>0.867</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>0</td>
<td>3.6</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>Level 1(W)</td>
<td>0.670</td>
<td>0.749</td>
<td>0.840</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>4.3</td>
<td>6.4</td>
<td>6.7</td>
</tr>
<tr>
<td>HP-2</td>
<td>Level 1&amp;2(W)</td>
<td>0.681</td>
<td>0.789</td>
<td>0.899</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>2.7</td>
<td>1.4</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>Level 1(W)</td>
<td>0.659</td>
<td>0.768</td>
<td>0.887</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>5.9</td>
<td>4.0</td>
<td>1.4</td>
</tr>
<tr>
<td>HP-3</td>
<td>Level 1&amp;2(W)</td>
<td>0.720</td>
<td>0.795</td>
<td>0.879</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>2.9</td>
<td>0.6</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>Level 1(W)</td>
<td>0.730</td>
<td>0.773</td>
<td>0.867</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>4.3</td>
<td>3.4</td>
<td>3.7</td>
</tr>
<tr>
<td>Desktop-1</td>
<td>Level 1&amp;2(W)</td>
<td>0.738</td>
<td>0.808</td>
<td>0.893</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>5.4</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Level 1(W)</td>
<td>0.739</td>
<td>0.810</td>
<td>0.884</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>5.6</td>
<td>1.3</td>
<td>1.8</td>
</tr>
<tr>
<td>Desktop-2</td>
<td>Level 1&amp;2(W)</td>
<td>0.724</td>
<td>0.812</td>
<td>0.896</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>3.4</td>
<td>1.5</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>Level 1(W)</td>
<td>0.717</td>
<td>0.871</td>
<td>0.931</td>
</tr>
<tr>
<td></td>
<td>Error (%)</td>
<td>2.4</td>
<td>8.9</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 6. Power Constraints with and without Level 1

<table>
<thead>
<tr>
<th>Power Constraint(W)</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-1</td>
<td>W/o level 1 MSE (%)</td>
<td>26.5</td>
<td>26.5</td>
<td>27.2</td>
</tr>
<tr>
<td></td>
<td>W/ level 1 MSE (%)</td>
<td>22.0</td>
<td>24.4</td>
<td>22.3</td>
</tr>
<tr>
<td></td>
<td>Improvement</td>
<td>17.0</td>
<td>7.9</td>
<td>18.0</td>
</tr>
<tr>
<td>HP-2</td>
<td>W/o level 1 MSE (%)</td>
<td>17.1</td>
<td>18.5</td>
<td>19.1</td>
</tr>
<tr>
<td></td>
<td>W/ level 1 MSE (%)</td>
<td>16.8</td>
<td>15.8</td>
<td>14.7</td>
</tr>
<tr>
<td></td>
<td>Improvement (%)</td>
<td>1.8</td>
<td>14.6</td>
<td>23.0</td>
</tr>
<tr>
<td>HP-3</td>
<td>W/o level 1 MSE (%)</td>
<td>27.4</td>
<td>28.9</td>
<td>24.8</td>
</tr>
<tr>
<td></td>
<td>W/ level 1 MSE (%)</td>
<td>24.3</td>
<td>24.3</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Improvement (%)</td>
<td>11.3</td>
<td>15.9</td>
<td>19.4</td>
</tr>
<tr>
<td>Desktop-1</td>
<td>W/o level 1 MSE (%)</td>
<td>23.2</td>
<td>23.0</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>W/ level 1 MSE (%)</td>
<td>21.1</td>
<td>21.2</td>
<td>20.7</td>
</tr>
<tr>
<td></td>
<td>Improvement (%)</td>
<td>9.1</td>
<td>7.8</td>
<td>13.8</td>
</tr>
<tr>
<td>Desktop-2</td>
<td>W/o level 1 MSE (%)</td>
<td>33.2</td>
<td>31.0</td>
<td>31.6</td>
</tr>
<tr>
<td></td>
<td>W/ level 1 MSE (%)</td>
<td>36.8</td>
<td>28.3</td>
<td>24.9</td>
</tr>
<tr>
<td></td>
<td>Improvement (%)</td>
<td>-10.8</td>
<td>8.7</td>
<td>21.2</td>
</tr>
</tbody>
</table>
Table 7. Latency constraints for (a) HP and Desktop-1 workloads (b) Desktop-2 workload

(a)

<table>
<thead>
<tr>
<th>Latency Constraint</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-1 Actual</td>
<td>1.27</td>
<td>2.13</td>
<td>3.11</td>
<td>3.62</td>
</tr>
<tr>
<td>Error (%)</td>
<td>27.0</td>
<td>6.5</td>
<td>3.7</td>
<td>9.5</td>
</tr>
<tr>
<td>HP-2 Actual</td>
<td>1.19</td>
<td>1.99</td>
<td>2.94</td>
<td>3.95</td>
</tr>
<tr>
<td>Error (%)</td>
<td>19.0</td>
<td>0.5</td>
<td>2.0</td>
<td>1.3</td>
</tr>
<tr>
<td>HP-3 Actual</td>
<td>1.51</td>
<td>2.41</td>
<td>3.22</td>
<td>3.96</td>
</tr>
<tr>
<td>Error (%)</td>
<td>51</td>
<td>20.5</td>
<td>7.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Desktop-1 Actual</td>
<td>1.04</td>
<td>1.89</td>
<td>2.75</td>
<td>3.85</td>
</tr>
<tr>
<td>Error (%)</td>
<td>4.0</td>
<td>5.5</td>
<td>8.3</td>
<td>3.8</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Latency Constraint</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop-2 Actual</td>
<td>11.9</td>
<td>15.9</td>
<td>19.8</td>
</tr>
<tr>
<td>Error (%)</td>
<td>19</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that a different set of latency constraints are used for trace Desktop-2. This is because it is extremely intensive and no power management policy 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.

6 Learning based CPU Power Management

In this section, we extend the Q-learning algorithm and use it to dynamically select the voltage and frequency of a CPU that is running in batch mode. The goal is to minimize the energy dissipation ($E$) while satisfying performance ($P$) and temperature ($T$) constraint. The control knobs are the set of DVFS settings.

In the following analysis, we use $V_{max}$ ($F_{max}$) and $V$ ($F$) to represent the maximum voltage (frequency) and the scaled voltage (frequency) of the processor. We use $v$ and $f$ to represent normalized voltage and frequency, i.e. $v = V/V_{max}$ and $f = F/F_{max}$. Obviously, $v$ and $f$ are also the scaling ratios of CPU voltage and frequency. It can be proved that the performance of the CPU is a concave and monotonically increasing function of $f$, i.e. its first derivative is a positive and decreasing function of $f$. The energy dissipation of the CPU is a convex function of normalized frequency $f$. It first decreases and then increases as the frequency increases based on different CPUs’ leaking power characteristics. The frequency that gives the minimum energy is denoted as...
The average temperature of the CPU is proportional to the power consumption of the CPU [Intel® Core™2 Duo Processor E8000 and E7000 Series], therefore, it is an increasing function of \( f \). Figure 13 gives a qualitative illustration of how energy, performance and temperature changes as clock frequency scales. It can also be proved that when the workload gets more CPU intensive, the value of \( f_E^* \) becomes larger, the performance curve shifts to right while the temperature curve shifts to left.

![Figure 13. Qualitative illustration of the relation between CPU temperature, performance, energy and clock frequency](image)

The relation among temperature, performance and energy determines the possible tradeoff space of the power management. The user imposed performance and temperature constraints determine if the optimal frequency \( f_E^* \) is achievable. We use \( f_P \) to denote the minimum frequency that satisfies the given performance constraint, and use \( f_T \) to denote the maximum frequency that satisfies the given temperature constraint. A valid clock frequency \( f \) must be greater than \( f_P \) and less than \( f_T \), i.e. \( f_P \leq f \leq f_T \). Based on the relation among \( f_P \), \( f_T \) and \( f_E^* \), the following 4 management scenarios may happen.

1. \( f_P > f_T \): In this scenario, the performance and temperature constraints could never be fulfilled.
2. \( f_P \leq f_T \leq f_E^* \): In this scenario, selecting \( f = f_T \) gives the minimum energy while satisfying the performance and temperature constraints.
3. \( f_E^* \leq f_P \leq f_T \): In this scenario, selecting \( f = f_P \) gives the minimum energy while satisfying the performance and temperature constraints.
4. \( f_P \leq f_E^* \leq f_T \): In this scenario, selecting \( f = f_E^* \) gives the minimum energy while satisfying the performance and temperature constraints.

The above analysis shows that the relative position of \( f_P \), \( f_T \) and \( f_E^* \) determines how the optimal frequency will be selected. For different workload, as the CPU and memory intensiveness varies, the relative position of \( f_P \), \( f_T \) and \( f_E^* \) will change. Therefore, it is important to have an adaptive technique that automatically searches for the optimal clock frequency.

We model the state of a CPU using a vector of four components, \((f, T, IPS, \mu)\). 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. The value of IPS (instructions per second) can be obtained by the performance counter of many commercial processors. The value of \( \mu \) can be calculated as the following [Dhiman and Rosing 2009]:

\[
\mu = 1 - \frac{cycles_{l1i\_stalled} + cycles_{l1d\_stalled}}{total\_number\_of\_cycles}
\] (9)
where cycles_L1i_stalled and cycles_L1d_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 available actions for the power management controller are the set of voltages and frequencies supported by the CPU. 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 ($C_T$). We assume that the CPU has been characterized so that for different workload CPU intensiveness $\mu$, the minimum energy frequency $f_E^\ast(\mu)$ is known. The energy cost during cycle $t$ is defined as the normalized difference between the actual frequency and the energy optimal frequency, i.e.

$$C_{E,t} = \frac{|f_t - f_E^\ast(\mu_t)|}{(f_{\text{max}} - f_{\text{min}})}$$

where $f_t$ and $\mu_t$ are frequency and workload CPU intensiveness during cycle $t$. The performance cost of cycle $t$ is defined as the normalized difference between the actual frequency and the maximum frequency weighted by the workload CPU intensiveness during this cycle, i.e. $C_p = (f_{\text{max}} - f_t)/(f_{\text{max}} - f_{\text{min}}) * \mu_t$.

Finally, the temperature cost of cycle $t$ is defined as the temperature increase from cycle $t-1$, i.e.

$$C_T = (T_t - T_{t-1})/T_{\text{range}}$$

where $T_{\text{range}}$ is the maximum temperature change in two adjacent time intervals. It is about 2°C in our experiment system.

In section 4 and 5, 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 latency. However, this is not true for the DVFS based CPU power management due to the increasing leakage power. Furthermore, this approach will not work if we have more than one constraint. For example, as shown in Figure 13, 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. 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 the above 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 implemented the Q-learning based DVFS controller on a Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor [Intel® Core™2 Duo Processor E8000 and E7000 Series]. 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 [Enhanced Intel SpeedStep® Technology] of Intel Core 2 processor to adjust the processor’s frequency. We used Perform2 tool [Perfmon2] to monitor performance events of the processors.

We use benchmarks from MiBench [Mibench] and MediaBench [MediaBench] 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”.

Since the proposed algorithm considers the temperature, performance and energy (TPE) management of only one core, 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.

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 \( f_\mu^* \) 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. Our goal is to evaluate our Q-learning algorithm ability of tracking the CPU intensiveness for the energy reduction. The temperature and IPS are also empirically partitioned into 4 states.

Below is the table that shows the results of constraining performance and temperature:

<table>
<thead>
<tr>
<th>Temperature</th>
<th>0.34 (constraint)</th>
<th>0.67 (constraint)</th>
<th>1 (constraint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.34(constraint)</td>
<td>0.33</td>
<td>0.58</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>0.46</td>
<td>0.34</td>
<td>0.30</td>
</tr>
<tr>
<td>0.67(constraint)</td>
<td>0.24</td>
<td>0.38</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>0.58</td>
<td>0.51</td>
<td>0.44</td>
</tr>
<tr>
<td>1 (constraint)</td>
<td>0.23</td>
<td>0.37</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>0.61</td>
<td>0.55</td>
<td>0.43</td>
</tr>
</tbody>
</table>

We measure the performance as the average normalized performance penalty defined by Equation (11). A large number corresponds to poor performance. We measure the energy as the average normalized energy penalty defined by Equation (10). 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 8. 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 in row 1 and column 1 of Table 8 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 3rd 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 TPE controller cannot find a policy that satisfies both user constraints. For example, the entry (0.34, 0.34) in Table 8 has constraint violation. Sometime, the TPE controller finds policies that exactly satisfies one of the constraints and outperforms the other (e.g. entry (0.34, 0.67) in Table 8). For the rest of times, the TPE 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 8. The user set a loose performance and temperature constraint ($\text{con}_P = \text{con}_T = 0.67$) 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 which corresponds to the last category presented in those 4 management scenarios discussed earlier in this section. 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 TPE controller can correctly learn the TPE tradeoff 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 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-art approach, we modified the expert-based algorithm in [Dhiman and Rosing 2009] based on our best understanding to achieve tradeoff among T, P and E. By sweeping the tradeoff factor of the expert-based algorithm, we obtain a set of data points in the energy, performance and temperature space corresponding to different TPE tradeoff. They are shown in Figure 14 (a) and (b). Please note that these data points are obtained without any effort to meet the performance or temperature constraint. In the same figure, we also plot those data points reported in Table 8. They are the TPE values of the Q-learning power management algorithm under the given temperature and
performance constraint. As we can see, our Q-learning algorithm is not only capable to meet the performance and temperature constraint, but also achieves lower energy dissipation.

7. Conclusions
In this paper, 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 tradeoff 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
Intel® Core™2 Duo Processor E8000 and E7000 Series: http://download.intel.com/design/processor/dataSheets/318732.pdf
ITRS. http://www.itrs.net/
LANGEN, P., AND JUURLINK, B. 2006. Leak-aware multiprocessor scheduling for low power. IPDPS’06, pp.60.
MediaBench: http://euler.slu.edu/~fritts/mediabench/
Mibench: http://www.eecs.umich.edu/mibench/
TOSHIBA Hard Disk Drive Specification 1.8 inch Hard Disk Drive MK6006GAL/MK4006GAL/MK3006GAL.