

Hierarchical Adaptive Dynamic Power Management

Zhiyuan Ren, *Member, IEEE*, Bruce H. Krogh, *Fellow, IEEE*, and Radu Marculescu, *Member, IEEE*

Abstract—Dynamic power management aims at extending battery life by switching devices to lower-power modes when there is a reduced demand for service. Static power management strategies can lead to poor performance or unnecessary power consumption when there are wide variations in the rate of requests for service. This paper presents a hierarchical scheme for adaptive dynamic power management (DPM) under nonstationary service requests. As the main theoretical contribution, we model the nonstationary request process as a Markov-modulated process with a collection of *modes*, each corresponding to a particular stationary request process. Optimal DPM policies are precalculated offline for selected modes using standard algorithms available for stationary Markov decision processes (MDPs). The power manager then switches online among these policies to accommodate the stochastic mode-switching request dynamics using an adaptive algorithm to determine the optimal switching rule based on the observed sample path. As a target application, we present simulations of hierarchical DPM for hard disk drives where the read/write request arrivals are modeled as a Markov-modulated Poisson process. Simulation results show that the power consumption of our approach under highly nonstationary request arrivals is less than that of a previously proposed heuristic approach and is even comparable to that of the optimal policy under stationary Poisson request process with the same arrival rate as the average arrival rate of the nonstationary request process.

Index Terms—Low-power design, hierarchical modeling, adaptive dynamic power management, nonstationary service requests.

1 INTRODUCTION

MANY computing devices today offer multiple operating states with different levels of power consumption and performance. Dynamic power management (DPM) saves power by selectively switching such a device into lower power states when there is no useful activity. The objective of DPM is to reduce power consumption without severely affecting the overall performance of the device [2], [3]. In recent years, DPM has helped to reduce the power consumption of hard disk drives (HDDs) in mobile computers from 25 percent to less than 10 percent [14].

Fig. 2a illustrates a basic power management system. Service requests arrive and are buffered in a queue if the device cannot serve them immediately. The power manager faces the DPM problem to make decisions about when the device should go to a different operating state and which operating state to go to. In general, the arrival and service processes of requests and power state transition processes are random. If these processes are stationary, the DPM problem can be formulated as a Markov decision process (MDP) problem [1], [8], [13] for which a stationary optimal DPM policy exists and can be computed. Typically, the objective is to optimize performance (e.g., waiting time

and loss probability of requests) subject to a given limit on power consumption or to minimize power consumption subject to a bound on performance. Under the stationary optimal policy, the power manager selects the device power state based on a *system state*. Under the Markovian assumption on the random processes [1], [8], the system state is the number of pending requests and the device power state. A time index is added into the system state in [13] to handle non-Markovian but stationary random processes.

In most applications, however, the service request process, is *not* stationary. For example, as shown in [12], the service requests to a hard disk drive (HDD) can vary significantly as the pattern of application accessing the disk changes. Fig. 1 shows a two-hour long trace of service requests in which five applications (*Windows Explorer*, *Internet Explorer*, *Microsoft Outlook*, *Adobe Photoshop*, and *Windows Media Player*), in a Windows 2000 environment, access the HDD alternatively for read/write operations (transfers). As we can see, the statistics of the arrival times of the transfers vary dramatically among these applications. This wide variation provides opportunities for power savings. The main objective of this paper is to introduce a hierarchical adaptive scheme for DPM that deals effectively with such nonstationary process of service requests.

For nonstationary situations, we model the request process as a *Markov-modulated stochastic process* of stationary workloads. Each workload generates stationary request arrival and service processes for the device. The modulated process is a *multimode* model in which each mode corresponds to a given workload and mode transitions are modeled as a discrete-time Markov chain. The statistics of

• Z. Ren is with the Signal Electronics and Embedded Systems Laboratory, General Electric Global Research Center, One Research Circle, Niskayuna, NY 12309. E-mail: ren@research.ge.com.

• B.H. Krogh and R. Marculescu are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: {krogh, radum}@ece.cmu.edu.

Manuscript received 28 Oct. 2002; revised 24 May 2004; accepted 10 Nov. 2004; published online 15 Feb. 2005.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 117680.

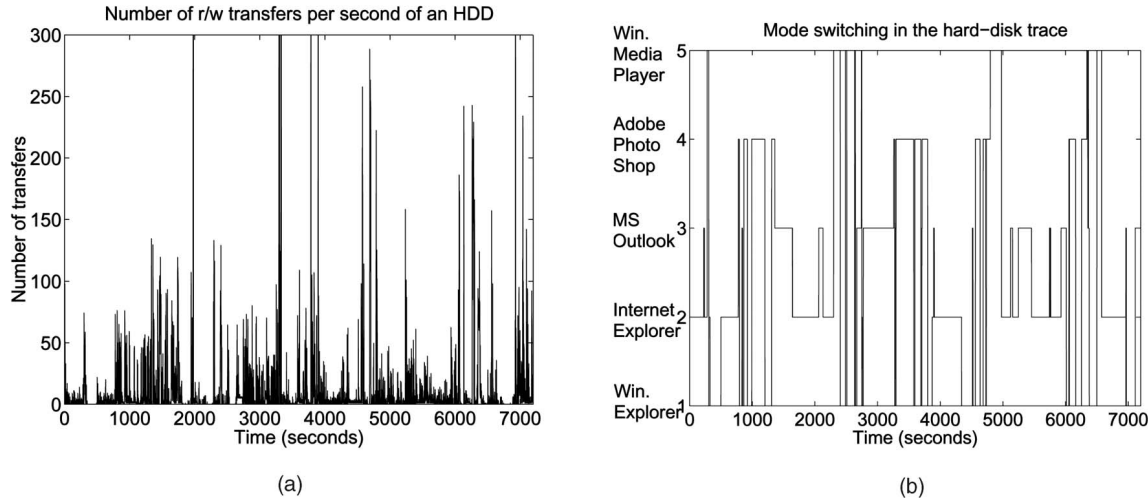


Fig. 1. A two-hour HDD trace: (a) read/write per second and (b) application (mode) switching.

the mode transition process, determined by a Markov chain transition matrix Q , define the *usage pattern* of the device. In this paper, we develop an adaptive algorithm that works hierarchically and does not assume precise knowledge of the parameters in the Markov-modulated model.

The two-level hierarchical scheme for the adaptive DPM follows from the multimode model for the request process. The lower level is comprised of a set of precomputed DPM policies: Each (randomized) policy selects the device power state according to the current system state. The upper level selects the DPM policy to be applied at any time according to the current system state and the current mode of request process (if the mode is known).

From the perspective of the policy-switching Power Manager, the switching problem can again be viewed as an MDP. The decision difficulty arises because of 1) a large set of possible modes and policies, 2) limits on the frequency of policy switching, and 3) some unobservable modes (arrival statistics that cannot always be associated with observable features in the system). All of these issues are due to overhead concerns since the complexity of the policy switching is directly related to the number of modes, switching decision frequency, and obtaining the mode information.

We provide a way to flexibly adjust the decision complexity for the Power Manager. This is done through selection of a set of special “observable” modes. The policy set is chosen as the set of precomputed optimal policies under stationary workloads associated with these special modes. The Power Manager only does policy switching when the system is in a special mode, either a “safe” policy (safe in the sense of acceptable performance delivery) or the previously selected policy stays applied in between switching epochs. In our adaptive algorithm, only an agreement on what modes are observable is needed; we do not assume the arrival statistics under these modes are known and we do not directly estimate these statistics in our scheme. We develop an adaptive algorithm of policy switching for the Power Manager. The decision complexity can be adjusted through selecting different “observable” mode sets. We also

provide some heuristics in selecting mode and policy sets for the policy switching.

We present simulation results for hierarchical DPM in an HDD application. In the simulation, the request arrival process is modeled as a Markov modulated Poisson process [5] and the request service process is assumed stationary Poisson. We compare our policy switching approach to an approach proposed in [4], based on *policy interpolation*, in which the arrival statistics are estimated directly from the request arrival processes. The approach in [4] is similar to our approach in that it uses precalculated policies, but it applies an interpolated DPM policy that matches the current statistics under the assumption that those statistics remain stationary long enough to make it essentially an infinite time horizon. We show that our adaptive hierarchical DPM scheme performs better than the interpolation scheme under the highly nonstationary arrival processes and that it is even comparable to the optimal policy for the stationary Poisson arrival process with the same arrival rate as the average arrival rate of the nonstationary process.

The remainder of the paper is organized as follows: Section 2.1 reviews the DPM problem. Section 2.2 illustrates the formulation of DPM problem as a stochastic optimization problem when the service request process is stationary. In Section 3, we present our hierarchical approach to DPM and the adaptive algorithm for computing an optimal switching rule based on sample-path observations. Section 4 presents several simulation studies for the hierarchical adaptive DPM algorithm. Finally, we summarize our main contribution and suggest a few directions for future work.

2 REVIEW OF THE DPM PROBLEM

This section reviews some background material on DPM. For illustration, we consider a device with two power states, *active* and *idle*, as shown in Fig. 2a. In Fig. 2a, P_a and P_i are power consumed in the two states, respectively; P_{ai} and T_{ai} are the power and time needed for the transition from *active* to *idle* and P_{ia} and T_{ia} are the power and time for the transition in the opposite direction. The Power Manager can

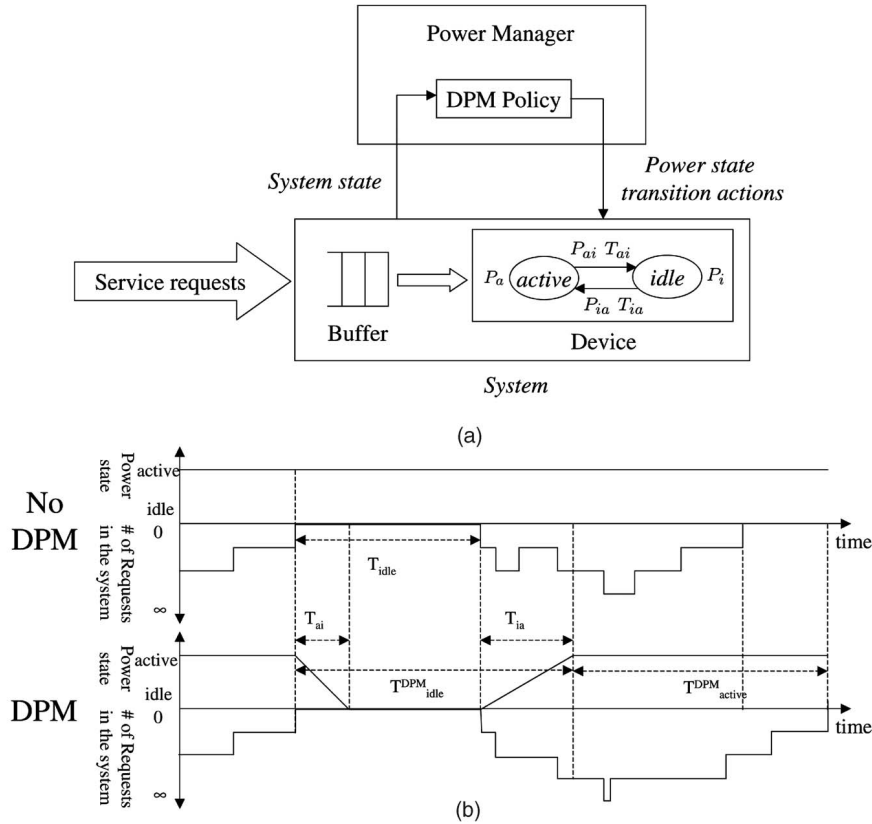


Fig. 2. (a) Dynamic power management of a device with two power states. (b) An operation sample path of the device without DPM and with a greedy DPM policy.

issue two control actions: *GO – ACTIVE* and *GO – IDLE*. When the control action *GO – ACTIVE* (*GO – IDLE*) is issued, the device will start the transition to the *active* (*idle*) state if the current power state is *idle* (*active*) or will stay *active* (*idle*) if the current power state is already *active* (*idle*).

2.1 Heuristic Power Management

Fig. 2b shows an operation sample path of the device without DPM and with a greedy DPM policy. For the greedy DPM policy, the Power Manager commands a power-state transition to *idle* as soon as there is no request in the system and to *active* as soon as there is a new arrival. From Fig. 2b, we see that the service performance (measured by the average number of pending requests) deteriorates after a power-state transition to *idle* is issued.

Under the assumption of unbounded buffer and Poisson arrival and service processes with rates λ and μ , the expected power saving with a greedy policy is calculated as

$$E\{P\} - E\{P^{DPM}\} = \frac{(\mu - \lambda)[\lambda(T_{ai} + T_{ia})(P_a - P_{tran}) + e^{-\lambda T_{ai}}(P_a - P_i)]}{\lambda\mu(T_{ai} + T_{ia}) + \mu e^{-\lambda T_{ai}}}, \quad (1)$$

where transition power is defined as $P_{tran} = \frac{T_{ai}P_{ai} + T_{ia}P_{ia}}{T_{ai} + T_{ia}}$. We have the limiting case in which the time and power consumption of the power-state transitions can be ignored in (1):

$$\lim_{T_{ai} \rightarrow 0} \lim_{T_{ia} \rightarrow 0} \{E\{P\} - E\{P^{DPM}\}\} = (1 - \lambda/\mu)(P_a - P_i). \quad (2)$$

The performance degradation with a greedy policy is calculated as

$$E\{N^{DPM}\} - E\{N\} = \frac{\lambda T}{2} + \frac{\lambda T}{2(1 + \lambda T)}, \quad (3)$$

where $T = T_{ai} + \frac{1}{\lambda}(e^{-\lambda T_{ai}} - 1) + T_{ia}$ and N is the number of requests in the system.

The limiting case of (3) is

$$\lim_{T_{ai} \rightarrow 0} \lim_{T_{ia} \rightarrow 0} \{E\{N^{DPM}\} - E\{N\}\} = 0. \quad (4)$$

From (2) and (4), we see that, when the transitions between power states are instantaneous and negligible time and power are paid for performing these transitions, DPM is a trivial task and the optimal solution is the greedy policy.

There are usually nonnegligible performance and power costs for power state transitions, however; that is, in most practical cases, T_{ai} , T_{ia} , P_{ai} , and P_{ia} cannot be ignored. When power state transitions have costs, the power manager needs to decide when it is worthwhile to transit to a low-power state and which state should be chosen if there are multiple low-power states available. The essence of DPM is to find a good trade off between power consumption and performance delivery.

A heuristic power management is based on a simple time-out policy [6]. That is, when there is no activity in the device, the Power Manager waits for a fixed period before putting the device into low-power state. The weakness of this approach is that the time-out value does not use any

models of the service request process for the device. An improved approach implemented for the HDD application by IBM [14] is based on an evaluation of the statistics of idle periods (the time period between two request arrivals) and minimizes the expected power consumption in an idle period with this statistical information. IBM's software, Adaptive Battery Life Extender (A.B.L.E.) [15], measures the request frequency distribution and calculates the probability that the current burst of requests is complete based on the statistics of the distribution for the power-performance trade off. The result of this approach is a time-threshold-based power state transition scheme and its improvement over the static time-out-based approach is that time-out values are now made adaptive to the actual usage pattern. This approach does not, however, take into account the fact that turning the device into the low power state in the current idle period will affect the length of the next idle period.

2.2 Formal Models of Power Management

When the service request process is stationary, a Markovian model can be derived for a power managed systems by properly defining the state in the model. Under the Markovian model, the DPM problem or, essentially, a sequential decision problem under uncertainty (MDP problem) is formulated as follows:

The system is modeled as a controlled discrete-time Markov chain, denoted as $\mathbf{S} = \{S_n, n \geq 0\}$, with system-state space $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_{|\Sigma|}\}$, where $|A|$ denotes the cardinality of the set argument A . At each decision epoch $n \geq 0$, a power state transition action U_n is taken from an action space \mathcal{A} and applied to the Markov chain. $\mathcal{A}(\sigma) \subseteq \mathcal{A}$ is the set of available actions if the system is in state $\sigma \in \Sigma$. If an action $U_n = \alpha \in \mathcal{A}(\sigma)$ is taken at state $S_n = \sigma$, a power consumption of $f(\sigma, \alpha)$ occurs with a set of performance measurements $f^1(\sigma, \alpha), f^2(\sigma, \alpha), \dots, f^I(\sigma, \alpha)$; the system then transits to state $S_{n+1} = \sigma' \in \Sigma$ with probability $p^\alpha(\sigma, \sigma')$.

A *policy* c is a mapping $c: \Sigma \rightarrow \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(A)$ denotes the set of all the probability distributions over a given finite set A . Under a policy c , action $\alpha \in \mathcal{A}(\sigma)$ is applied to the system with probability $[c(\sigma)]_\alpha$ when the system state is $\sigma \in \Sigma$. The long run average power consumption under a policy c is

$$J_f(c) = \lim_{N \rightarrow \infty} E^{S_0, c} \left\{ \frac{1}{N} \sum_{n=0}^{N-1} [f(S_n, U_n)] \right\}.$$

The finite state space guarantees the existence of the limit. Depending on the property of the Markov chain, this long run average can depend on the initial state of the chain in the multichain model. In the case of a unichain, that is, when the chain has a single positive recurrent state class with some possible transient states under every policy, the long run average is the same under a policy no matter what the initial state is. Similar long run average performance measurements can also be defined for every f^1, f^2, \dots, f^I .

Based on the MDP, the Power Manager solves a problem of stochastic optimization under performance constraints, for example, minimization of average power consumption subject to limits on average waiting time and loss probability for service requests. Various algorithms exist for

solving the constrained MDP problem and a good collection of them is given in [9]. The solution of the optimization problem is an optimal DPM policy that initiates power-state transitions based on the current system-state information. More accurate trade offs can then be made with this more complicated decision model. For instance, instead of using a greedy policy, the Power Manager could wait until there is more than one request in the system before turning on the device to save more power if the degradation in (3) is far below the acceptable level.

We now provide simulation results for DPM formulated as an MDP problem. For simplicity, we do not use the time-indexed model in [13] for nonexponential probability distributions. That is, in the simulation, we assume that the request interarrival time, power state transition times, and request service time are all random variables with negative exponential distributions. Under this assumption, the system can be modeled as a continuous-time Markov chain. This continuous-time system is then discretized, by sampling the continuous-time system at times when *request-arrival* events, *service-completion* events, and *power-state transition-completion* events occur, to obtain the discrete-time model.

We simulate DPM on an HDD with the power states in Fig. 2a and a buffer of length one. The parameters used in our simulation are as follows: $P_a = 2.1W$, $P_i = 0.65W$; $P_{ai} = P_{ia} = 1.4W$,¹ $T_{ai} = T_{ia} = 0.4s$ are expected values of power state transition times.

The discrete-time MDP has six states, that is, the state space is $\Sigma = \{(n_{hq}, n_{ps}), n_{hq} = 0, 1, 2, n_{ps} = 1, 2\}$, where n_{hq} is the total number of requests in the buffer and HDD and n_{ps} is the power state of the HDD (1 stands for *active* and 2 for *idle*). The action space is $\mathcal{A} = \{GO - ACTIVE, GO - IDLE\}$. The control objective is to minimize the average power consumption under two performance constraints: the average waiting-time limit of service requests in the buffer b_w and the loss-probability limit of incoming service requests b_l . The performance function f^1 for calculating average waiting time is defined as the number of service requests in the system or $f^1((n_{hq}, n_{ps}), \cdot) = n_{hq}$. The long run average of f^1 is proportional to the average waiting time based on Little's theorem in queuing theory. The performance f^2 associated with loss probability is defined as a Boolean function $f^2((n_{hq}, n_{ps}), \cdot) = 1$ if the buffer is full and zero otherwise.

A policy provides probabilities for choosing actions *GO - ACTIVE* and *GO - IDLE* for the six system states. Clearly, the always-active (always-idle) policy which issues *GO - ACTIVE* (*GO - IDLE*) at all the states consumes maximum (minimum) power of 2.1W (0.65W) and delivers maximum (minimum) performance.

We calculated the optimal DPM policy under five Poisson workloads. A *workload* in this paper is defined as a pair of stationary request arrival and service processes in the DPM setup, Fig. 2a. When both the arrival and service processes are stationary and Poisson, we have a Poisson workload. The five Poisson workloads in our simulation

1. The power-specific parameters are chosen to be comparable to those from [14] for an HDD. Note that the *active* power state in this paper is in fact the combination of *active* and *performance idle* states in [14]; the *idle* state in this paper is the *low power idle* state in [14].

TABLE 1
Optimal Policies and Power Consumption under Five Poisson Workloads

Policy \ State	State						Power	λ
	(0,1)	(0,2)	(1,1)	(1,2)	(2,1)	(2,2)		
c_1	(0.0 1.0)	(0.0 1.0)	(1.0 0.0)	(0.99 0.01)	(1.0 0.0)	(1.0 0.0)	1.02W	
c_2	(0.1 0.9)	(0.36 0.64)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	1.20W	
c_3	(0.09 0.91)	(0.86 0.14)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	1.48W	
c_4	(0.65 0.35)	(0.05 0.95)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	1.73W	
c_5	(0.86 0.14)	(0.08 0.92)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	1.95W	

have arrival rates λ_i , $i = 1, 2, \dots, 5$ given in the last column of Table 1 and the same service rate of $\mu = 1.0r/s$. Table 1 illustrates the five optimal policies designed under limits $b_w = 0.5$ second and $b_l = 0.05$. Each entry in Table 1 specifies probabilities for choosing actions *GO – ACTIVE* and *GO – IDLE* at a particular system-state. For instance, under the second workload, when the buffer is empty and the HDD is idle, i.e., at state (0, 2), the policy activates the HDD with probability 0.36. The table shows that the major difference between these policies occurs at states (0, 1) and (0, 2), i.e., when there is no service request in the system. The second column from the right in Table 1 gives the power consumption under the five cases.

3 HIERARCHICAL DPM

In the above study, we assumed a stationary workload and, therefore, obtained stationary MDP models. However, each application in the system can generate dramatically different read/write patterns to the disk, which results in nonstationary request arrival and service processes (see Fig. 1). Therefore, the stationary assumption on the MDP model is generally *not* valid in practice. We propose a hierarchical approach to study DPM for systems with nonstationary usage patterns, based on a multimode MDP model.

In the multimode MDP model proposed in [11], a mode variable is used to model the source of the nonstationarity. For instance, the mode can be the application (or process) that generates the read/write requests to the disk. In the multimode MDP model, the state of a controlled Markov process \mathbf{X} has two variables, the *system state*, as defined in the previous section, and the *mode*. The mode influences the system state transition probabilities. If a power state transition action $U_n = \alpha \in \mathcal{A}(\sigma)$ is taken at state $S_n = \sigma$, the incurred power is $f(\sigma, \alpha)$, and the performance measurements are $f^1(\sigma, \alpha)$, $f^2(\sigma, \alpha)$, \dots , $f^I(\sigma, \alpha)$. The system transits to state $S_{n+1} = \sigma' \in \Sigma$ with probability $p_{\mu}^{\alpha}(\sigma, \sigma')$, where $M_n = \mu$ is the mode at time n . The mode transits from $M_n = \mu \in \mathcal{M}$ to $M_{n+1} = \mu' \in \mathcal{M}$ with probability $q(\mu, \mu')$. That is, the mode process is a Markov chain with probability transition matrix $Q = [q(\mu, \mu')]_{\mu, \mu'}$.

The whole system-and-mode model is then given by an MDP: $\mathbf{X} = (\mathbf{S}, \mathbf{M}) = \{X_n = (S_n, M_n), n \geq 0\}$ with state space $\Sigma \times \mathcal{M}$; the action set available for state (σ, μ) is $\mathcal{A}(\sigma)$. Solving

MDP \mathbf{X} means to specify an action $U_n \in \mathcal{A}(S_n)$ when the state is $X_n = (S_n, M_n)$, for every $n = 0, 1, 2, \dots$, based on some optimality criterion (long run average in this paper). This can be difficult to achieve usually because of two reasons: 1) The additional mode variable in the model may result in a much larger state space and computation to obtain the optimal policy; 2) unavailable mode information at some time epochs. For instance, suppose a PC with Windows 2000 has 50 processes running according to the task manager. If we incorporate all 50 processes into our multimode model for the HDD, we will have a state space 50 times larger than that of a regular MDP (300 states compared with six states). Also, to obtain the mode and system state information requires that the Power Manager interfere and pause the disk operation to acquire the process ID associated with each read/write request to decide on the power state transitions, which may cause too big an overhead to be ignored. The decision complexity along the state and time spaces in the model are the main obstacles for achieving the optimal solution.

We provide a decision mechanism to reduce the decision complexity over both the state and time space, based on the idea of *policy switching*. Fig. 3 shows our proposed hierarchical policy switching architecture. The *policy-switching scheme* for this architecture is a triplet $(\mathcal{C}, \mathcal{M}_t, SR)$, where: $\mathcal{C} = \{c_1, c_2, \dots, c_{|\mathcal{C}|}\}$ is a set of available policies;² $\mathcal{M}_t \subseteq \mathcal{M}$ is the set of *observable modes*;³ and $SR: \Sigma \times \mathcal{M}_t \rightarrow \mathcal{P}(\mathcal{C})$ is a (nondeterministic) *switching rule*. The scheme operates as follows: Each time the system mode is in \mathcal{M}_t , the switching rule SR makes the decision on choosing a policy from \mathcal{C} and applies it to the system. This chosen policy is applied to the system until the next time the system mode is in \mathcal{M}_t again. For a given SR , policy $c \in \mathcal{C}$ is applied to the system with probability $[SR(\sigma, \mu)]_c$ when the system state is $\sigma \in \Sigma$ and the mode is $\mu \in \mathcal{M}_t$. By only making decisions at a subset of states in the state space and at a subset of time epochs along the time axis, this

2. Although policies in \mathcal{C} can be general policies, they are usually optimal policies associated with some stationary modes. The rationale is that this setup can at least handle the case of slow mode evolution process well.

3. A basic requirement of \mathcal{M}_t is that it should include at least one of the positive recurrent states in the Markov chain of the mode process. This ensures a finite time interval between switching epochs.

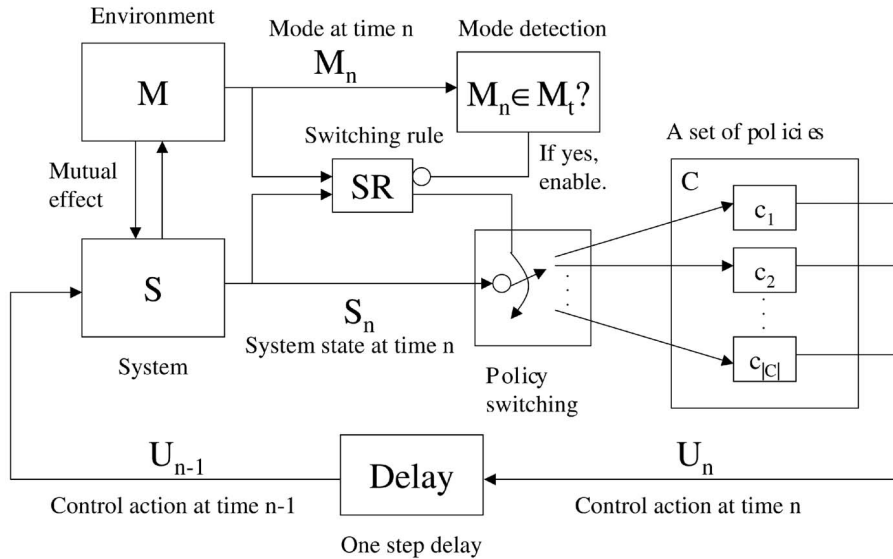


Fig. 3. A hierarchical policy switching architecture for multimode MDPs.

mechanism therefore reduces the decision complexity from solving the entire MDP \mathbf{X} .

Let $g : (\Sigma \times \mathcal{M}) \times \mathcal{A} \rightarrow \mathcal{R}$ be a given function characterizing the power or performance measures for each state-action pair. $J_g(SR)$ is the long run average of g under switching rule SR , that is, for $g = f, f^1, f^2, \dots, f^I$,

$$J_g(SR) = \frac{1}{|\Sigma \times \mathcal{M}_t|} \sum_{S_0, M_0} \lim_{N \rightarrow \infty} E^{SR, S_0, M_0} \left\{ \frac{1}{N} \sum_{n=0}^{N-1} [g(S_n, U_n)] \right\}.$$

We define the long run average in a way that accommodates the general multichain model. In a multichain, the average depends on the probability distribution of the initial state. This definition assumes equal probabilities of all possible initial states.

For the hierarchical switching architecture in Fig. 3, the *optimal switching rule problem* can be formulated as: Given policy set \mathcal{C} and mode set \mathcal{M}_t , find an SR with minimum long run average power under the performance constraints, that is:

$$\min_{SR} J_f(SR) \text{ s.t. } J_{f_i}(SR) \leq b^i, \quad i = 1, 2, \dots, I, \quad (5)$$

where b^i for $i = 1, 2, \dots, I$ are performance bounds on the performance measurements.

The process \mathbf{X} running under an SR is not generally Markovian since knowing the current system mode and state does not imply knowing the current policy. However, it can be shown that a process embedded in the switching epochs (times when the system mode is in \mathcal{M}_t) is Markovian. Consider a sample path of \mathbf{X} : $\omega = \{X_0, X_1, X_2, \dots\}$ running under an SR . Without loss of generality, we assume that $M_0 \in \mathcal{M}_t$. Let $T_0 = 0$ and $T_i = \min_t \{t > T_{i-1}, M_t \in \mathcal{M}_t\}$, $i = 1, 2, \dots$. $\{Y_i = X_{T_i}, i = 0, 1, 2, \dots\}$ forms an embedded Markov chain.

A new MDP can be formulated on process \mathbf{Y} . In this new MDP, at each state $(\sigma, \mu) \in \Sigma \times \mathcal{M}_t$, an action is the selection of a policy in \mathcal{C} . When policy c is selected and

applied to the system, \mathbf{Y} transits to state $(\sigma', \mu') \in \Sigma \times \mathcal{M}_t$ with probability

$$\tilde{p}^c[(\sigma, \mu), (\sigma', \mu')] = \sum_{\Phi_1} [c(S_0)]_{U_0} \times p_{M_0}^{U_0}(S_0, S_1) q(M_0, M_1) \times \dots \times [c(S_{n-1})]_{U_{n-1}} \times p_{M_{n-1}}^{U_{n-1}}(S_{n-1}, S_n) \times q(M_{n-1}, M_n),$$

where

$$\Phi_1 = \{S_0, M_0, U_0, S_1, M_1, U_1, \dots, S_n, M_n \mid S_0 = \sigma, M_0 = \mu, M_1 \notin \mathcal{M}_t, \dots, M_{n-1} \notin \mathcal{M}_t, S_n = \sigma', M_n = \mu'\}.$$

For a (power or performance) function g , $h_g[(\sigma, \mu), c]$ is the expected accumulated g from the time when \mathbf{X} is at state (σ, μ) to the time right before \mathbf{X} reaches a state in $\Sigma \times \mathcal{M}_t$, that is:

$$h_g[(\sigma, \mu), c] = \sum_{\Phi_2} \{g[S_0, U_0] + \dots + g[S_{n-1}, U_{n-1}]\} \times [c(S_0)]_{U_0} \times p_{M_0}^{U_0}(S_0, S_1) \times q(M_0, M_1) \times \dots \times [c(S_{n-1})]_{U_{n-1}} \times p_{M_{n-1}}^{U_{n-1}}(S_{n-1}, S_n) \times q(M_{n-1}, M_n), \quad (6)$$

with

$$\Phi_2 = \{S_0, M_0, U_0, \dots, S_n, M_n \mid S_0 = \sigma, M_0 = \mu, M_1 \notin \mathcal{M}_t, \dots, M_{n-1} \notin \mathcal{M}_t, M_n \in \mathcal{M}_t\}.$$

Under a given SR , we have a fractional structure of the long run average

$$J_g(SR) = \frac{\sum_{S_0, M_0} \lim_{N \rightarrow \infty} E^{SR, S_0, M_0} \left\{ \frac{1}{N} \sum_{n=0}^{N-1} [h_g(Y_n, C_n)] \right\}}{\sum_{S_0, M_0} \lim_{N \rightarrow \infty} E^{SR, S_0, M_0} \left\{ \frac{1}{N} \sum_{n=0}^{N-1} [h_t(Y_n, C_n)] \right\}}, \quad (7)$$

where h_t is obtained by letting $g(\cdot) = t(\cdot) \equiv 1$ in (6), that is, $h_t[(\sigma, \mu), c]$ is the expected accumulated time from the time

when \mathbf{X} is at state (σ, μ) to the time right before \mathbf{X} reaches a state in $\Sigma \times \mathcal{M}_t$ after policy c is applied. We have $h_t(\cdot, \cdot) > 0$.

To derive an adaptive policy switching algorithm, we first consider the case when all the model parameters are known. Similar to the approach used to solve Markov decision problems via linear programming, the problem in (5) can be turned into a linear fractional programming problem from (7) and solved by the following algorithm.

Algorithm 1 (Optimal Switching Rule for Known Parameters)

1. Choose $\delta > 0$, $r_0 > 0$ and set $k := 0$.
2. At the k th step, solve

$$\begin{aligned} & \text{maximize} \sum_{\sigma, \mu, c} \{h_f[(\sigma, \mu), c] - r_k h_t[(\sigma, \mu), c]\} z[(\sigma, \mu), c], \\ & \text{subject to} \sum_{\sigma, \mu, c} \{\delta_{[(\sigma, \mu)(\sigma', \mu')]} - \tilde{p}^c[(\sigma, \mu), (\sigma', \mu')]\} \\ & \quad z[(\sigma, \mu), c] = 0, \end{aligned} \quad (8)$$

$$\sum_c z[(\sigma', \mu'), c] + \sum_{\sigma, \mu, c} \{\delta_{[(\sigma, \mu)(\sigma', \mu')]} - \tilde{p}^c[(\sigma, \mu), (\sigma', \mu')]\} \times$$

$$w[(\sigma, \mu), c] = \frac{1}{|\Sigma \times \mathcal{M}_t|}$$

for every $(\sigma', \mu') \in \Sigma \times \mathcal{M}_t$,

$$\sum_{\sigma, \mu, c} \{h_{f_i}[(\sigma, \mu), c] - b^i h_t[(\sigma, \mu), c]\} z[(\sigma, \mu), c] \leq 0,$$

for $i = 1, 2, \dots, I$,

$$z[(\sigma, \mu), c] \geq 0, w[(\sigma, \mu), c] \geq 0$$

for every $\sigma \in \Sigma, \mu \in \mathcal{M}_t, c \in \mathcal{C}$.

With the solution (z^*, w^*) to (8), a switching rule SR_k^* can be constructed as follows: At state $(\sigma, \mu) \in \Sigma \times \mathcal{M}_t$ if $\sum_{c \in \mathcal{C}} z^*[(\sigma, \mu), c] > 0$, policy c_i is applied with probability

$$\frac{z^*[(\sigma, \mu), c_i]}{\sum_{c \in \mathcal{C}} z^*[(\sigma, \mu), c]};$$

if $\sum_{c \in \mathcal{C}} z^*[(\sigma, \mu), c] = 0$, policy c_i is applied with probability

$$\frac{w^*[(\sigma, \mu), c_i]}{\sum_{c \in \mathcal{C}} w^*[(\sigma, \mu), c]}.$$

Also with z^* , calculate

$$r_{k+1} = \frac{\sum_{\sigma, \mu, c} h_f[(\sigma, \mu), c] z^*[(\sigma, \mu), c]}{\sum_{\sigma, \mu, c} h_t[(\sigma, \mu), c] z^*[(\sigma, \mu), c]}.$$

3. If $|r_{k+1} - r_k| \leq \delta$, exit; otherwise, set $r_k := r_{k+1}$ and $k := k + 1$, go to Step 2.

This algorithm is essentially the extension of linear programming for solving multichain model [9, p. 462] for solving MDP with fractional objective function. It solves a sequence of linear programming problems to obtain the solution to the original fractional programming problem. The following theorem, proven in [11], says that this

algorithm will lead to an SR arbitrarily close to the optimal switching rule.

Theorem 1. *Algorithm 1 terminates in a finite number of steps. Suppose it terminates at SR_K^* . Then, SR_K^* is feasible and*

$$|J_f(SR_K^*) - J_f(SR^*)| \leq o(\delta).$$

For DPM applications, the parameters \tilde{p} and h in (8) are usually unknown. They can be estimated along the sample path of the system, however. Along a segment of sample path of \mathbf{X} ,

$$\begin{aligned} & \{X_{T_0}, X_{T_0+1}, \dots, X_{T_1}, X_{T_1+1}, \dots, X_{T_N}\} = \\ & \{(S_{T_0}, M_{T_0}), (S_{T_0+1}, M_{T_0+1}), \dots, (S_{T_1}, M_{T_1}), (S_{T_1+1}, M_{T_1+1}), \\ & \dots, (S_{T_N}, M_{T_N})\}, \end{aligned}$$

we can observe the system state all the time and we can observe the mode state at T_n , $n = 0, 1, \dots, N$. Let $\{Y_0, Y_1, \dots, Y_N\} = \{(S_{T_0}, M_{T_0}), (S_{T_1}, M_{T_1}), \dots, (S_{T_N}, M_{T_N})\}$ be the full-state observations at switching times and let $\{C_0, C_1, \dots, C_{N-1}\}$ be the policies chosen at switching times. We use capitalized notations for the policies because they are random variables assuming values in \mathcal{C} . We have the following maximum-likelihood estimation for the \tilde{p} -quantities in (8),

$$\begin{aligned} \hat{\tilde{p}}^c[(\sigma, \mu), (\sigma', \mu')] &= \frac{1}{\sum_{n=0}^{N-1} \mathbf{1}_{\{Y_n=(\sigma, \mu)\}} \mathbf{1}_{\{C_n=c\}}} \times \\ & \sum_{n=0}^{N-1} [\mathbf{1}_{\{Y_n=(\sigma, \mu)\}} \mathbf{1}_{\{C_n=c\}} \mathbf{1}_{\{Y_{n+1}=(\sigma', \mu')\}}], \end{aligned} \quad (9)$$

where indicator function $\mathbf{1}_{\{\cdot\}}$ equals one when the condition in $\{\cdot\}$ is true, zero otherwise. For a (power or performance) function g , let the observed power or performance along the sample path be $\{F_{T_0}, F_{T_0+1}, \dots, F_{T_1}, F_{T_1+1}, \dots, F_{T_N}\}$. We have the following maximum-likelihood estimation for the h -quantities in (8)

$$\begin{aligned} \hat{h}_g[(\sigma, \mu), c] &= \frac{1}{\sum_{n=0}^{N-1} \mathbf{1}_{\{Y_n=(\sigma, \mu)\}} \mathbf{1}_{\{C_n=c\}}} \times \\ & \left[\mathbf{1}_{\{Y_n=(\sigma, \mu)\}} \mathbf{1}_{\{C_n=c\}} \sum_{m=T_n}^{T_{n+1}-1} F_m \right]. \end{aligned} \quad (10)$$

The adaptive implementation of Algorithm 1 is given as follows:

Algorithm 2 (Adaptive Switching Rule)

1. Estimate $|\mathcal{C}| |\Sigma \times \mathcal{M}_t|^2$ \tilde{p} -quantities and $(I+2)|\mathcal{C}| |\Sigma \times \mathcal{M}_t|$ h -quantities in (8) from a sample path of the system by using (9) and (10). There are many ways to generate a sample path for the estimation. The basic requirement is to force each state-policy pair $[(\sigma, \mu), c] \in (\Sigma \times \mathcal{M}_t) \times \mathcal{C}$ a sufficient number of times along the sample path so that the parameter estimates are adequate.⁴

4. Forcing maybe involves resetting the state and policy to the desired pair. If such a forcing is not allowed, we need a recurrent condition on the embedded chain to guarantee the infinitely often appearance of a state on the sample path. This is not a problem in our simulation since the recurrent condition holds.

2. Implement Algorithm 1 with h and \tilde{p} -quantities replaced by \hat{h} and \tilde{p} -estimates from Step 1 and inequality constraints in (8) replaced by the following:

$$\sum_{\sigma, \mu \in \mathcal{M}_t, c} \left\{ \hat{h}_f[(\sigma, \mu), c] - b^i \hat{h}_t[(\sigma, \mu), c] \right\} z[(\sigma, \mu), c] \leq -B_m, i = 1, 2, \dots, I,$$

where $-B_m < 0$ is a margin introduced to ensure the feasibility of the switching rule to the original inequality constraints in (8).

The following theorem, proven in [11], provides confidence interval results for the adaptive switching rule.

Theorem 2. Given $\epsilon > 0$ and $\rho > 0$, for any sample path used in Step 1 in Algorithm 2 along which the event $\{Y_n = (\sigma, \mu), Y_{n+1} = (\sigma', \mu'), C_n = c\}$ is observed at least $O(\frac{1}{\epsilon^2} \ln \frac{1}{\rho})$ times for all $c \in \mathcal{C}$ and $(\sigma, \mu) \in \Sigma \times \mathcal{M}_t, (\sigma', \mu') \in \Sigma \times \mathcal{M}_t$, Step 2 in Algorithm 2 will result in a feasible switching rule \bar{SR} with

$$\text{Prob}\{|J_f(\bar{SR}) - J_f(SR^*)| \leq \epsilon\} \geq \rho.$$

Theorem 2 states that, in order to obtain a feasible approximation to the optimal switching rule in Step 2 of Algorithm 2 with confidence ρ and accuracy ϵ , an estimation time in the order of $\frac{1}{\epsilon^2} \ln \frac{1}{\rho}$ is sufficient in Step 1.

4 SIMULATION RESULTS

In this section, we apply our adaptive hierarchical scheme to solve the DPM problem for an HDD operating under nonstationary service requests. In the HDD scenario, a mode (or workload) is defined as the application associated with a service request and is assumed to generate stationary Poisson request arrival and service processes. An *observable* mode corresponds to an important application for which the Power Manager decides if there is a better DPM policy than the currently applied policy when this application is accessing the disk. The Power Manager faces an *unobservable mode* when an application deemed unimportant is accessing the disk. In the case of an unobservable mode, either a “safe” policy (safe in the sense of acceptable performance delivery) or the previously selected policy stays applied and the Power Manager does not make policy-switching decisions.

We compare our results to the adaptive approach proposed in [4], where the arrival statistics are estimated directly from the request arrival processes. The DPM policy under the current arrival statistics is obtained by linearly interpolating a set of precharacterized policies designed for a set of preselected statistical values. Directly applying such a DPM policy (either preselected or interpolated) matching the current statistics implies the assumption that those statistics remain stationary for a long time because each applied DPM policy is calculated to optimize over an infinite time horizon. For instance, the experiment in [4] uses concatenation of workload traces of variable lengths (ranging from 40,000 to 60,000 time steps) and a sliding window of size 50 time steps to estimate the statistics. The ratio between the average trace length and sliding window

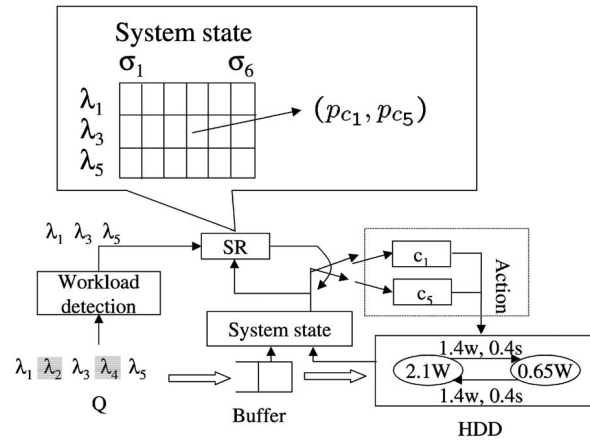


Fig. 4. Our adaptive approach based on hierarchical switching.

size is around 1,000 and demonstrates a slow mode (workload) evolution process. It is therefore not surprising to find that the policy interpolation scheme performs well in this setup. In fact, if there exists alternative information (such as the ID of the process or thread generating each workload trace) for the Power Manager to learn about the workload change, it can be switch more quickly to the right policy for the new workload. This is actually a motivation for us to develop a DPM scheme based on hierarchical switching.

In our simulation, we use Markov-modulated workloads to model the nonstationary service requests to the disk. We use the five workloads given in Section 2.2. Since the request service process is the same, the service process remains stationary Poisson for the modulated workloads. The modulated workloads are the result of a jumping process among the five workloads. That is, the workload index at the sampling times (when request-arrival, service-completion, and power-state transition completion events occur) forms a discrete-time Markov chain with transition probability matrix Q . We call the modulated process with a matrix Q a *usage pattern*. When Q is an identity matrix, we have a stationary usage pattern. The five policies given in Table 1 are therefore the stationary optimal policies corresponding to the five stationary usage patterns, which we use for the policy switching.

We show in this section that our hierarchical adaptive DPM approach performs better than the policy interpolation scheme. When comparing with two approaches, we focus on the intrinsic difference of the two switching methodologies, i.e., the optimal switching versus interpolation. The overhead brought by estimation of parameters is ignored, that is, when calculating the optimal switching rule, we implement Algorithm 2 along a sample path long enough to achieve high confidence ($\rho \approx 1$) and accuracy ($\epsilon \approx 0$) as described in Theorem 2.

For a usage pattern Q , where $Q(i, j)$ is 0.9 when $i = j$ and 0.025 otherwise, our Adaptive Switching Rule is illustrated in Fig. 4 (the set of observable modes is $\mathcal{M}_t = \{\lambda_1, \lambda_3, \lambda_5\}$ and the policy set is $\mathcal{C} = \{c_1, c_5\}$). Note that Q implies the average dwell time of a workload is 10 (compared with 1,000 in [4]), which demonstrates a highly nonstationary request process. Table 2 gives (a) the policy interpolation switching rule and

TABLE 2
(a) Mode-Matching Policy Interpolation and (b) Optimal Switching Rules under a Usage Pattern

Mode	λ_1	λ_1	λ_1	λ_1	λ_1	λ_1
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)	(1.0 0.0)
Mode	λ_3	λ_3	λ_3	λ_3	λ_3	λ_3
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(0.5 0.5)	(0.5 0.5)	(0.5 0.5)	(0.5 0.5)	(0.5 0.5)	(0.5 0.5)
Mode	λ_5	λ_5	λ_5	λ_5	λ_5	λ_5
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(0.0 1.0)	(0.0 1.0)	(0.0 1.0)	(0.0 1.0)	(0.0 1.0)	(0.0 1.0)

(a)

Mode	λ_1	λ_1	λ_1	λ_1	λ_1	λ_1
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(0.0 1.0)*	(1.0 0.0)	(1.0 0.0)	(0.0 1.0)*	(1.0 0.0)	(0.35 0.65)*
Mode	λ_3	λ_3	λ_3	λ_3	λ_3	λ_3
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(1.0 0.0)	(0.0 1.0)	(1.0 0.0)	(0.0 1.0)	(1.0 0.0)	(0.96 0.04)
Mode	λ_5	λ_5	λ_5	λ_5	λ_5	λ_5
System-state	(0,1)	(0,2)	(1 1)	(1 2)	(2 1)	(2 2)
Probabilities	(1.0 0.0)*	(0.0 1.0)	(1.0 0.0)*	(0.0 1.0)	(1.0 0.0)*	(0.96 0.04)*

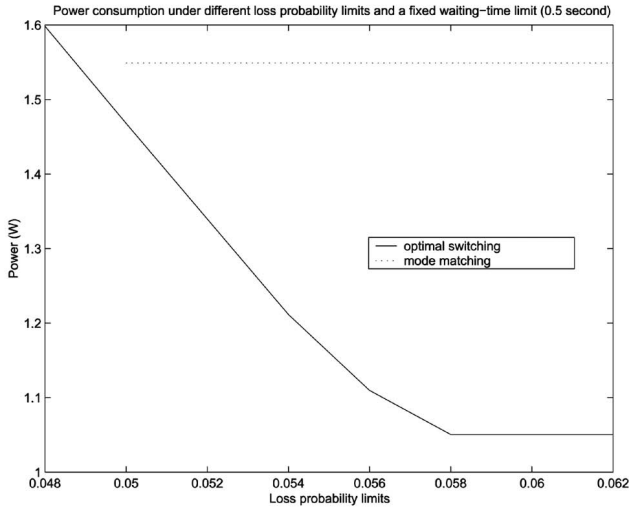
(b)

(b) the optimal switching rule for waiting-time and loss-probability limits $b_w = 0.5s$, $b_l = 0.05$. Each entry in Table 2 gives the mode, system-state, and probabilities for choosing policies c_1 and c_5 for that mode and system-state pair. Under the heuristic approach, the Power Manager applies the preselected policy c_1 (c_5) when the mode is λ_1 (λ_5); since $\lambda_3 = \frac{1}{2}\lambda_1 + \frac{1}{2}\lambda_5$, the Power Manager interpolates c_1 and c_5 evenly to obtain the policy for λ_3 . The power consumption is $1.55W$ under the policy interpolation rule and $1.47W$ under the optimal switching rule.

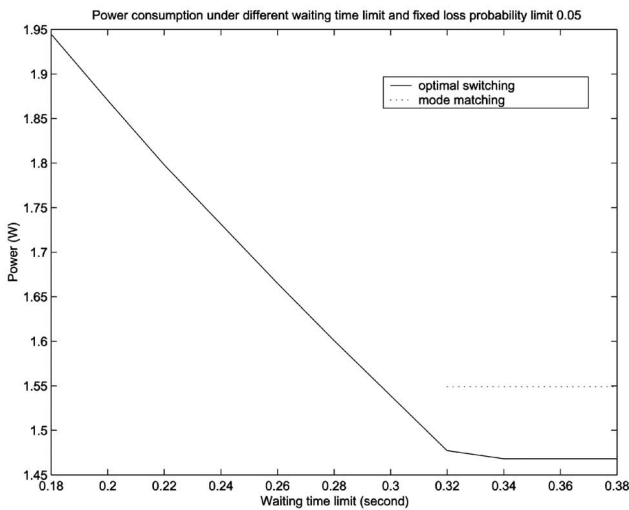
The asterisks in Table 2b indicate switching decisions that do *not* correspond to a policy interpolation rule, that is, the policies selected by the optimal switching rule for the cases with an asterisk are not the stationary policies that were designed for those modes. This demonstrates that policy interpolation to match the current statistics is not

always the best thing to do. The reason is that the policies are designed for *stationary* processes, i.e., under the assumption that the mode will remain constant. When the mode switching dynamics are taken into account, it turns out that, for some cases, it is better to use one of the policies designed for a different mode. (Conditions for which mode matching policy interpolation is, in fact, the optimal switching rule are given in [10].)

For the same usage pattern as that for Table 2, Fig. 5 shows the power consumption of different optimal switching rules obtained for different loss probability and waiting time limits. The figure shows that, with looser performance bounds, more power can be saved by the optimal rule. For a comparison, Fig. 5 also shows the power consumption of the policy interpolation power management scheme. The power consumption of the mode-matching switching rule is a constant



(a)

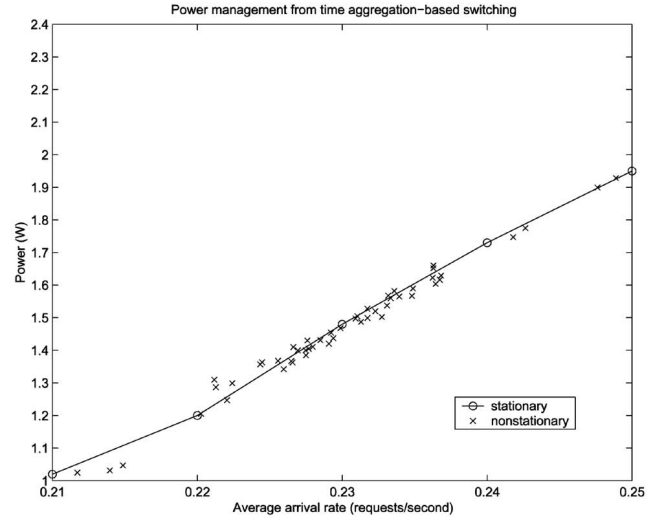


(b)

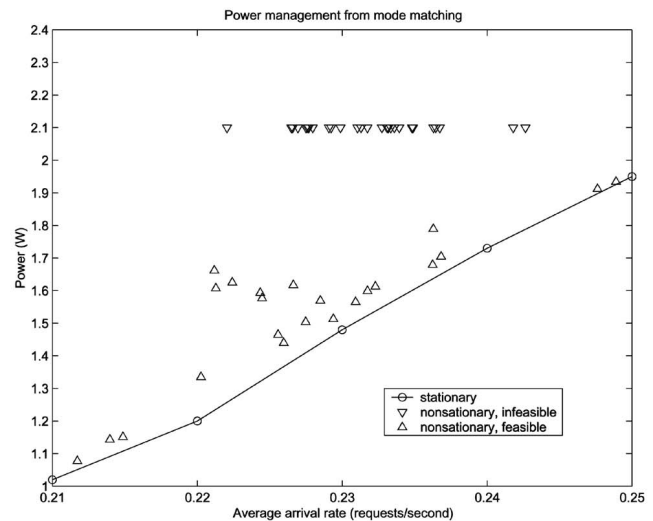
Fig. 5. Power consumption under different (a) loss probability limits and (b) waiting time limits.

because it does not change under different performance limits. Note that the policy interpolation scheme becomes infeasible for a loss probability limit less than 0.05 (Fig. 5a) and cannot achieve performance bounds stricter than a waiting time limit less than 0.32 seconds (Fig. 5b).

To further demonstrate the effectiveness of our scheme, Fig. 6 gives the power consumption of our scheme for 50 usage patterns. The 50 Q matrices for the 50 usage patterns were generated randomly. Fig. 6 shows that our adaptive switching rule performs well for the nonstationary usage pattern compared to the constant-mode optimal policy for the stationary usage pattern with the same arrival rate as the average arrival rate of nonstationary usage pattern. Fig. 6 also shows the application of policy interpolation power management under the same 50 usage patterns. For 27 out of the 50 patterns, the policy interpolation scheme is infeasible or cannot achieve the 0.5 second waiting time and the 0.05 loss probability limits.



(a)



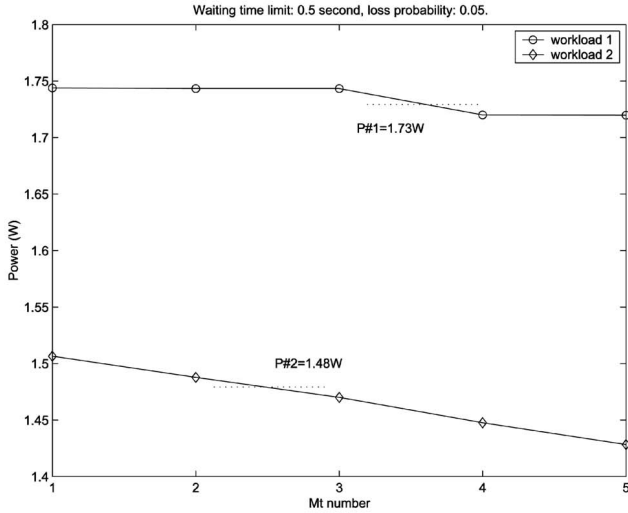
(b)

Fig. 6. Power consumption under stationary and nonstationary usage patterns.

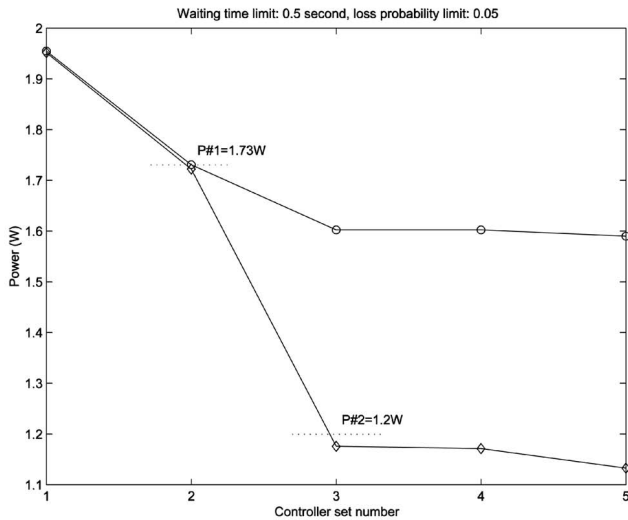
For the other 23 patterns, our approach saves around 9 percent more power.

We now consider the effect of different observable mode sets \mathcal{M}_t s and policy sets \mathcal{C} s on the performance of our adaptive switching rule. Intuitively, the optimal switching rule can achieve more power savings when more observable modes and more policies are used. On the other hand, more observable modes and more policies result in more computations for computing the optimal switching rule because the complexity of Algorithms 1 and 2 grows monotonely with the size of these two sets. Therefore, selection of these two sets has to take both power saving and computation burden into account. One idea is to start with coarse sets and introduce more modes and policies into the sets gradually until a desired power level is achieved. The following simulations demonstrate this idea.

Fig. 7a shows the power consumption of our scheme for five \mathcal{M}_t s:



(a)



(b)

Fig. 7. Power consumption for two usage patterns under different (a) \mathcal{M}_t s and (b) \mathcal{C}_s .

$$\{\lambda_1\}, \{\lambda_1, \lambda_2\}, \{\lambda_1, \lambda_2, \lambda_3\}, \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}, \{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5\}$$

under two randomly generated usage patterns when the policy set is $\mathcal{C} = \{c_1, c_5\}$. Pattern one has an average arrival rate of 0.24 (r/s) and pattern two has one of 0.23 (r/s). In both cases, more power savings are achieved through introducing more modes into \mathcal{M}_t . In the case of pattern one, the mode set $\mathcal{M}_t = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ is needed to achieve an expected power $P_1^\# = 1.73W$. In the case of pattern two, the mode set $\mathcal{M}_t = \{\lambda_1, \lambda_2, \lambda_3\}$ is needed to achieve an expected power $P_2^\# = 1.48W$. The empirical expected power is found by looking at the stationary cases in Table 1.

Fig. 7b shows the power consumption of five \mathcal{C}_s :

$$\{c_5\}, \{c_5, c_4\}, \{c_5, c_4, c_3\}, \{c_5, c_4, c_3, c_2\}, \{c_5, c_4, c_3, c_2, c_1\}$$

under two randomly generated usage patterns when $\mathcal{M}_t = \{\lambda_1, \lambda_3, \lambda_5\}$. Pattern one has an average arrival rate of 0.24 (r/s) and pattern two has one of 0.22 (r/s). In both cases, more power savings are achieved through introducing more policies into policy sets. In the case of pattern

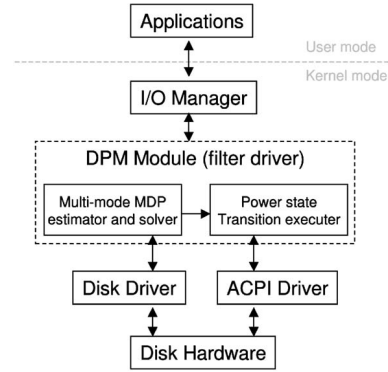


Fig. 8. Implementation of hierarchical DPM.

one, the policy set $\mathcal{C} = \{c_5, c_4\}$ is needed to achieve an expected power $P_1^\# = 1.73W$. In the case of pattern two, the policy set $\mathcal{C} = \{c_5, c_4, c_3\}$ is needed to achieve an expected power $P_2^\# = 1.2W$. The empirical expected power is found by looking at Table 1.

The choice of the initial mode set and the subsequent order of adding more modes into the set are based on an “order of importance” for the modes. There are several considerations to sort the modes: 1) the variation on power savings over different policies. If, at a stationary mode, the power savings vary dramatically over different policies, it should be incorporated into the mode set since there is a need to switch among policies at this mode. 2) Switching frequency issue: Modes can take different expected return time to themselves and some modes are chosen to obtain desirable average time intervals between policy switchings. In the above experiment, we chose the initial mode set as $\{\lambda_1\}$ because it has the lowest arrival rate and the biggest variation of power savings from the optimal and always-on policy. The order of adding modes also follows the order of the arrival rates.

When choosing the initial policy set and order of adding more policies, we followed an order of policies based on their “safety,” which characterizes a policy’s ability to satisfy performance constraints. This consideration ensures the existence of switching rules feasible to performance constraints. Since c_5 is the policy associated with the workload with the highest arrival rate, it is the safest policy in terms of performance delivery. We therefore chose $\{c_5\}$ as the initial policy set and added subsequent safe policies to the set.

5 CONCLUSION

This paper presents a hierarchical model for adaptive dynamic power management under nonstationary service requests. We derive a multimode model based on modeling the nonstationary service request as a Markov-modulated stochastic process and each mode is associated with a stationary workload. The Power Manager adaptively switches among a set of stationary optimal policies to accommodate the stochastic mode-switching service request.

In Fig. 8, we show the positioning of the hierarchical DPM in OS-directed configuration and power management (OSPM) of Advanced Configuration and Power Interface (ACPI) [7]. The Power Manager resides in a filter driver.

The filter driver intercepts disk read/write IRP (I/O Request packets) to the disk driver, which enables the Power Manager to obtain the state information and estimate the parameters for the multimode MDP. Since each IRP is associated with a certain thread, the mode information can therefore be obtained. The Power Manager then solves the multimode MDP to obtain a power management policy, which executes the power state transition commands using IRP through the ACPI driver.

REFERENCES

- [1] L. Benini, A. Bogliolo, G. Paleologo, and G. De Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 813-833, 1999.
- [2] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. VLSI Systems*, vol. 8, pp. 299-315, 2000.
- [3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, Mass.: Kluwer, 1997.
- [4] E.Y. Chung, L. Benini, A. Bogliolo, Y.H Lu, and G. De Micheli, "Dynamic Power Management for Non-Stationary Service Requests," *IEEE Trans. Computers*, vol. 51, no. 11, pp. 1345-1361, Nov. 2002.
- [5] W. Fischer and K. Meier-Hellstern, "The Markov-Modulated Poisson Process (MMPP) cookbook," *Performance Evaluation*, vol. 18, pp. 149-171, 1992.
- [6] P. Greenawalt, "Modeling Power Management for Hard Disks," *Proc. Int'l Workshop Modeling, Analysis, and Simulation for Computer and Telecomm. Systems*, pp. 62-65, 1994.
- [7] <http://www.acpi.info>, 2002.
- [8] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," *Proc. Design Automation Conf.*, pp. 555-561, June 1999.
- [9] M.L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994.
- [10] Z.Y. Ren and B.H. Krogh, "Mode Matching Control Policies for Multi-Mode Markov Decision Processes," *Proc. 2001 Am. Control Conf.*, 2001.
- [11] Z.Y. Ren and B.H. Krogh, "Switching Control in Multi-Mode Markov Decision Processes," *Proc. 40th IEEE Conf. Decision and Control*, 2001.
- [12] Z.Y. Ren, B.H. Krogh, and R. Marculescu, "Hierarchical Adaptive Dynamic Power Management," *Proc. Design, Automation, and Test in Europe Conf.*, Feb. 2004.
- [13] T. Simunić, L. Benini, P. Glynn, and G. De Micheli, "Event-Driven Power Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 7, July 2001.
- [14] Storage Systems Division of IBM, "Adaptive Power Management for Mobile Hard Disks," http://www.almaden.ibm.com/almaden/mobile_hard_drives.html, Jan. 1999.
- [15] Storage Systems Division of IBM, "IBM Adaptive Battery Life Extender," <http://www.storage.ibm.com/hdd/library/able.htm>, 2002.



Zhiyuan Ren received the PhD degree in electrical and computer engineering from Carnegie Mellon University in 2002. He is an embedded system engineer at General Electric Global Research Center, Niskayuna, New York. He is a member of the IEEE.



Bruce H. Krogh is a professor of electrical and computer engineering at Carnegie Mellon University, Pittsburgh, Pennsylvania. He is a past associate editor of the *IEEE Transactions on Automatic Control and Discrete Event Dynamic Systems: Theory and Applications* and founding editor-in-chief of the *IEEE Transactions on Control Systems Technology*. His current research interests include hybrid dynamic systems and synthesis and verification of embedded control system designs. He is a fellow of the IEEE.



Radu Marculescu received the PhD degree in electrical engineering from the University of Southern California in 1998. He is currently an associate professor in the Department of Electrical and Computer Engineering at Carnegie Mellon University. He is a recipient of the US National Science Foundation's CAREER Award (2001) in the area of design automation of electronic systems. He has received two Best Paper Awards from the Design Automation and Test in Europe (DATE) Conference in 2001 and 2003 and a Best Paper Award from the Asia and South Pacific Design Automation Conference (ASP-DAC) in 2003. He also was awarded the Carnegie Institute of Technology's Ladd Research Award (2002-2003). His current research focuses on developing design methodologies and software tools for embedded systems design, novel approaches for on-chip communication with emphasis on power, performance, and fault tolerance, and ambient intelligence. He is a member of the IEEE and the ACM.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.