Dynamic Power Management for Nonstationary Service Requests

Eui-Young Chung, Luca Benini, Alessandro Bogliolo,
Yung-Hsiang Lu, and Giovanni De Micheli, *Fellow*, *IEEE*

Abstract—Dynamic Power Management (DPM) is a design methodology aiming at reducing power consumption of electronic systems by performing selective shutdown of idle system resources. The effectiveness of a power management scheme depends critically on an accurate modeling of service requests and on the computation of the control policy. In this work, we present an online adaptive DPM scheme for systems that can be modeled as finite-state Markov chains. Online adaptation is required to deal with initially unknown or nonstationary workloads, which are very common in real-life systems. Our approach moves from exact policy optimization techniques in a known and stationary stochastic environment and it extends optimum stationary control policies to handle the unknown and nonstationary stochastic environment for practical applications. We introduce two workload learning techniques based on sliding windows and we study their properties. Furthermore, a two-dimensional interpolation technique is introduced to obtain adaptive policies from a precomputed look-up table of optimum stationary policies. The effectiveness of our approach is demonstrated by a complete DPM implementation on a laptop computer with a power-manageable hard disk that compares very favorably with existing DPM schemes.

Index Terms—Adaptation, DPM, low power, nonstationarity, OS, policy optimization, sliding window.

1 Introduction

Design methodologies for energy-efficient system-level design are receiving increasing attention [2], [11], [14], [17] because of the widespread use of portable electronic appliances (e.g., cellular phones, laptop computers, etc.) and of concerns about the environmental impact of electronic systems.

Battery life time in portable systems can be prolonged in two ways—by increasing battery capacity per unit weight and by reducing power consumption with minimal performance loss. Between these two alternatives, the latter has been the major concern of designers because battery capacity (Watt-hours/lb) has only improved by a factor two to four over the last 30 years, while the computational power of digital ICs has increased by more than four orders of magnitude [17].

Energy efficient design requires the development of new computer-aided design techniques to help explore the trade-off between power and conventional design constraints, i.e., performance and area. Numerous computer-aided design techniques [14] have been researched and implemented at all levels of the design hierarchy to reduce power consumption and many of these techniques target VLSI digital components. Modern portable systems are heterogeneous [19]. For example, the power breakdown for a well-known laptop computer [18] shows that the power consumed by VLSI digital components is only 21 percent, while the display, hard disk drive, and wireless LAN card consume 36 percent, 18 percent, and 18 percent, respectively.

Power-conscious design above the chip level, i.e., system-level low-power design, is regarded as one of the most effective ways to tackle power constraints. Among these techniques, *dynamic power management* (*DPM*) [1], [13], [46] and its extensions, such as application-driven/assisted power management [39], [40], and *dynamic voltage scaling* (*DVS*) [41], [42] have been extensivly applied with good results.

DPM is a flexible and general design methodology aiming at controlling performance and power levels of electronic systems by exploiting the idleness of their components. A system is provided with a *power manager* (PM) that monitors the overall system and component states and controls the power state of each component. This control procedure is called *power management policy*. The problem in DPM is that changing power state (e.g., spin up and down a disk drive) imposes a penalty in terms of both power and performance. Many real-life devices have several sleep states in a trade-off between power saving and transition penalty. Some advanced electronic components have multiple active states, characterized by different supply voltages. For these devices, dynamic power management generalizes into a DVS problem [41], [42]. Unfortunately, variable-voltage devices are not widespread yet. Hence, this paper focuses primarily on DPM, whose effects can be experimentally validated in off-the-shelf systems.

Fig. 1 shows 1) the system power consumption level over time without DPM, 2) the case when the ideal DPM is applied, and 3) the case when nonideal DPM is applied.

- *E.-Y. Chung, Y.-H. Lu, and G. De Micheli are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305. E-mail: {eychung, luyung, nanni}@stanford.edu.*
- *L. Benini is with the Department of Electrical Engineering and Computer Science, University of Bologna, 40136, Italy. E-mail: lbenini@deis.unibo.it.*
- *A. Bogliolo is with the Department of Engineering, University of Ferrara, Ferrara, 44100, Italy. E-mail: abogliolo@ing.unife.it.*

Manuscript received 18 Feb. 2000; revised 2 Aug. 2001; accepted 25 Jan. 2002. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111538.

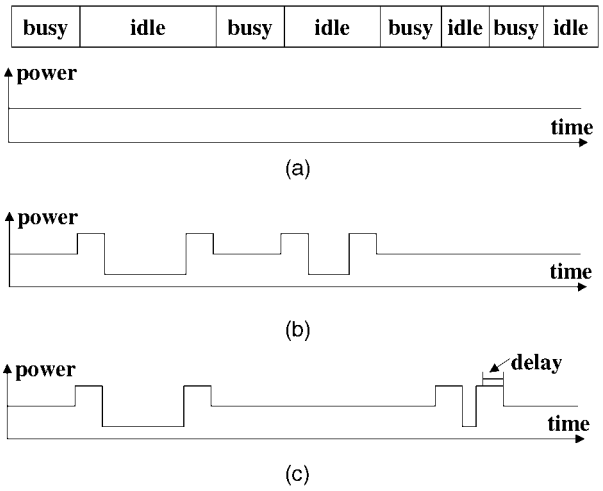


Fig. 1. System power consumption level variation with *DPM*. (a) Without DPM. (b) With ideal DPM. (c) With nonideal DPM.

Nonideal *DPM* wastes the idle interval at the second idle period and pays a performance penalty at the third idle period. These inefficiencies come from the inaccurate prediction of the duration of the idle period or, equivalently, the arrival time of the next request for an idle component. Thus, the ultimate goal in *DPM* is to minimize the performance penalty and wasted idle intervals while, at the same time, minimizing power. This objective can be achieved by an ideal PM with complete knowledge of present, past, and future workloads. In some cases, such knowledge can be provided by applications that provide hints on future requirements of system resources [39], [40]. Unfortunately, the application-driven approach is not viable when applications cannot be modified. In the following, we take an application-independent viewpoint because it can be applied to available systems with no changes to the applications they support.

In general, we can model the unavoidable uncertainty of future requests by describing the workload as a stochastic process. Even if the realization of a stochastic process is not known in advance, its properties can be studied and characterized. This assumption is at the basis of many stochastic optimal control approaches that are routinely applied to real-life systems [3], [4], [5]. *DPM* has been formulated and solved as a discrete-time stochastic optimal control problem by Benini et al. [12]. Also, continuous-time stochastic approaches were proposed in [32], [33], [44], [45].

Unfortunately, in many instances of real-life *DPM* problems, it is hard, if not impossible, to precharacterize the workload of a power-manageable system. Consider, for instance, a disk drive in a personal computer (PC). The workload for the drive strongly depends on the application mix that is run on the PC. This, in turn, depends on the user, on the location, and similar "environmental conditions" which are not known at design or fabrication time. In these cases, the stochastic process describing the workload is *initially unknown*. Even worse, the workload is subject to large variations over time. For instance, hard disk workloads for a workstation drastically change with the time of the day or the day of the week. This characteristic is called *nonstationarity*. It is generally hard to achieve robust and high-quality *DPM* without considering this effect.

Optimal control of nonstationary stochastic systems is a well-developed field [6], [7], [8]. Several adaptive control approaches are based on an estimation procedure that "learns" online the unknown or time-varying parameters of the system, coupled with a flexible control scheme that selects the optimal control actions based on the estimated parameters. This approach is also known as the *principle of estimation and control* (PEC). Our work exploits the same paradigm to adapt power management policies to nonstationary workload.

The contributions of this work can be summarized as follows: We propose parameter-learning schemes for the workload source (e.g., the user, also called service requestor) that capture the nonstationarity of its behavior. We present two techniques based on sliding windows to capture the time-varying parameters of the stochastic model of the workload source. Next, we show how policies for dynamic power management under nonstationary service request models can be determined by interpolating optimal policies computed under the assumption that user requests are stationary. Finally, we implement the proposed algorithm on a laptop computer with a hard disk and show its feasibility in a real system environment.

The paper is organized as follows: In Section 2, we review related work in system-level dynamic power management. In Section 3, we introduce system model and policy optimization problem when the service requestor is a known and stationary stochastic process. In Section 4, a *dynamic power management* scheme when the service requestor is stationary, but unknown, is described. In Section 5, we present sliding window techniques to capture the nonstationarity of service requests and we describe the overall *dynamic power management* scheme based on sliding windows. Section 6 is dedicated to the issues raised by the implementation of adaptive DPM schemes on general-purpose computers. The experimental results obtained from simulation, as well as system measurements, are shown in Section 7. Finally, we summarize our work and propose future directions in Section 8.

2 Related Work

Previous approaches to *dynamic power management* can be classified into three major categories: timeout-based, predictive, and stochastic. The timeout policy [29] is the most widely used in many applications, such as microprocessors, monitors, hard disk drives, etc., because of its simplicity. The value of the timeout can be fixed (static timeout) or it may be changed over time. An effective static timeout policy sets the timeout to the *break-even time*, T_{be} of the power-managed device [26]. Roughly speaking, T_{be} is the minimum idle time for which it is convenient to shut down the device because the power savings in the *sleep* state should compensate for the shutdown and wakeup costs [13]. It can be shown [26] that setting the timeout to T_{be} produces a *2-competitive* policy, which is guaranteed to be within a factor of two from the power savings that could be

achieved by an ideal policy with perfect knowledge of the future (i.e., an *oracle* policy).

Static timeout policies use a fixed timeout value. Several adaptive timeout policies have been introduced to reduce wasted idle time by changing the timeout threshold depending on previous idle period history [15], [16], [27], [28]. The main shortcoming of timeout policies is that they waste power waiting for the timeout to expire. This inefficiency motivates the search for more effective techniques.

Srivastava et al. [10] proposed heuristic policies to shut down a system as soon as an idle period begins. The basic idea in [10] is to predict the length of idle periods and shut down the system when the predicted idle period is long enough to amortize the cost (in latency and power) of shutting down and later reactivating the system. A shortcoming of the predictive shutdown approach proposed by Srivastava et al. is that it is based on offline analysis of usage traces, hence it is not suitable for nonstationary request streams whose statistical properties are not known a priori. This shortcoming is addressed by Hwang and Wu [9]. They proposed online adaptive methods that predict the duration of an idle period with an exponentially weighted average of previous idle periods.

All predictive shutdown techniques share a few limitations. First, they do not deal with resources with multiple sleep states (recently, a predictive approach which handles multiple sleep state components has been proposed in [36].) Second, they cannot accurately trade off performance losses (caused by transition delays between states of operation) with power savings. Third, they do not deal with general system models where multiple incoming requests can be queued while waiting for service.

These limitations are addressed in [12], where general systems (with multiple states and queuing) and user requests are modeled as discrete-time Markov decision processes. The discrete-time Markov model enables a rigorous formulation of the search for optimal power management policies as a constrained *stochastic optimization* problem whose exact solution can be found in polynomial time. Also, it provides a flexible way to control the trade-off between power consumption and performance penalty depending on the optimization constraints. A few extensions to the discrete-time Markov model have been proposed in the recent past. To reduce the power cost imposed on the power manager, which observes and issues command at regular time-discretization intervals, continuous-time (event-based) formulations have been proposed [32], [33], [44], [54]. The work in [32] was further extended to handle more complex systems (multiple devices) in [43] or to control the power states of the system with the consideration of a side metric—*quality of service* [47].

Unfortunately, a common basic assumption in [12], [32], [33], [44], [45] is that the Markov model is stationary and known in advance. Such an assumption clearly does not hold if the system experiences nonstationary workloads. Also, event-based approaches require a way to correct their wrong decision because they have only one chance to make a decision for a given idle period and the power loss of the wrong decision for a long idle period can be significant. In contrast, a discrete-time approach naturally has the chances to change its wrong decision at each time-discretization interval.

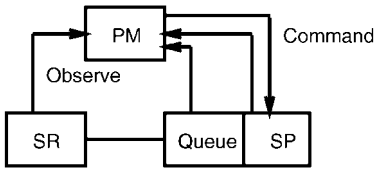


Fig. 2. Overall system model for DPM.

The limitations of previous stochastic optimization approaches to DPM motivate our work. In a nutshell, the techniques presented in this work move from a discrete-time stochastic optimization problem in a stationary and known environment [12], but they consider the nonstationary behavior of user requests for application in a realistic system environment.

We will introduce adaptive DPM by steps. First, we will review the basics of the stochastic optimal control formulation in the case of a known and stationary environment. Then, we will introduce adaptive techniques for the unknown stationary case. In the case of DPM, estimation of the unknown parameters is decoupled from the control of the power-managed system. Hence, we can exploit theoretical results on adaptive stochastic control to prove asymptotic optimality of DPM control policies for initially unknown Markovian workloads. Finally, we will extend our approach to nonstationary, general workloads. In this case, optimality cannot be proven and DPM algorithms are heuristic. The effectiveness of heuristic DPM algorithms will be demonstrated through simulation analysis as well as measurement on real-life examples.

3 DPM in Known Stationary Environment

In this section, we briefly review the system model introduced in [12]. The model for nonstationary requests described in Section 5 can be seen as an extension of the stationary approach of [12], with the consideration of time-varying request probability.

The overall system model for DPM is shown in Fig. 2. An electronic system is modeled as a unit providing a service, called *service provider* (SP), while receiving requests from another entity, called *service requestor* (SR). A *queue* (SQ) buffers incoming unserviced requests. The service provider can be in one of several states (e.g., *active*, *sleep*, *idle*, etc.). Each state is characterized by the ability (or the inability) to provide a service and by a power consumption level. Transitions among states may have a performance penalty (e.g., latency in reactivating a unit) and a power penalty (e.g., power loss in spinning up a hard disk).

The *power manager* (PM) is a control unit that controls the transitions among states. We assume that the power consumption of the PM is negligible with respect to the overall power dissipation.[1] At equally spaced instants in time, the power manager evaluates the overall state of the system (provider, queue, and requestor) and decides to

1. This assumption has been validated in practice for several classes of systems [13], [34] and it will be analyzed in detail later in the paper.

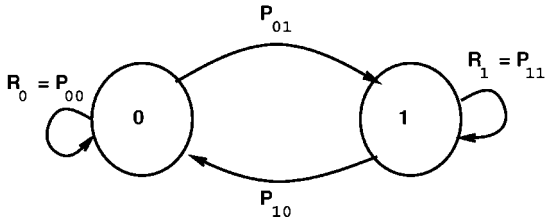


Fig. 3. An example of a Markov chain for stationary SR.

issue a command to stimulate a state transition. For the sake of conciseness, we define the following notation to represent the states of the units and PM commands:

- SP: $s_p = \{0, 1, \cdots, S_p - 1\}$,
- SR: $s_r = \{0, 1, \cdots, S_r - 1\}$,
- SQ: $s_q = \{0, 1, \cdots, S_q - 1\}$,
- A: $a = \{1, 2, \cdots, N_a\}$,

where A is a command set issued by PM to control the power state of SP. We model the system components as discrete-time Markov chains [12]. In particular, we use a controlled Markov chain model for the system provider so that its transition probabilities depend on the command issued by the power manager.

In [12], the SR as well as SP are modeled as stationary processes. A generic requestor can have S_r states. We will discuss the case of $S_r = 2$, as shown in Fig. 3. SR stays in state 0 when no request is issued for the given time slice; otherwise, SR is in state 1. The corresponding transition matrix is denoted by P_{SR}. We call the diagonal elements of P_{SR} *user request probabilities* and we denote them by $R_i, i = 0, 1$. The probabilities $R_i = Prob(s_r(t+1) = i | s_r(t) = i), i = 0, 1$ (and the entire transition matrix P_{SR}) are time-invariant for a stationary workload.

With this assumption, the entire system model does not include any time-variant parameters, thus the complete system can be described by a controlled stationary Markov chain. The *control policy* for the given system model can be optimized under performance or power constraints by solving a linear programming problem. Details are provided in [12].

The optimal policy for a Markov system model is also Markovian, i.e., the decision at any point in time depends only on the current system state instead of the entire past history. Therefore, a policy, the final result from policy optimization, can be thought of as a matrix that associates a probability of issuing each command $(a \in A)$ with each system state. The matrix is called a *decision table* and its dimension is $S \times N_a$, where $S = S_r \times S_p \times S_q$.

A *control policy* is a sequence of decisions. At each time slice, the PM observes the current system state and issues a command based on the probability of each command for the given system state in the decision table. The decision made at each time slice i is denoted as δ_i and the policy π is the sequence of the decisions.

Even though this approach provides a way to obtain an exact solution and control the trade-off between performance and power, the following two assumptions limit its practical application:

- The user request probabilities for the given workload are known through offline analysis.
- The user request probabilities for the given workload are constant over time.

The workload of many practical systems is not stationary in time. Furthermore, statistic workload characteristics may not be available for offline policy optimization. Therefore, we need to extend the approach by relaxing these two assumptions. In Section 4, the approach is first extended to the unknown stationary environment by relaxing the first assumption and in Section 5, it is further extended to the unknown nonstationary environment.

4 DPM in Unknown Stationary Environment

In Section 3, it is assumed that SR can be characterized through offline analysis of stationary workloads. Offline analysis at design time can be impossible in practice, especially for general-purpose systems (such as PCs or workstations), where workload strongly depends on the applications that the end user will run on the general-purpose platform. For these reasons, even the straightforward timeout PM policies implemented on current portable computers are user-customizable. This customization process puts the responsibility for following the trial-and-error process that leads to the choice of an optimal timeout value on the user. While this choice may be acceptable for tuning a single timeout, it is certainly not possible to assume that the user would be able to manage the complex characterization process for a Markov model of system workload. Thus, we need techniques for automatically "learning" a Markov workload model and for computing the corresponding optimal PM policy. Clearly, both estimation and policy optimization overhead should be small for online application in real-life systems.

Our approach will follow classic techniques of adaptive control theory [7], [8], based on the principle of estimation and control. The unknown parameters of the stochastic system (i.e., the parameters that characterize the workload) are estimated with estimators that are guaranteed to converge (with probability one) to the true parameter values. The policy applied at each time step is chosen assuming that the current parameter estimates are the true values. It can be shown that, under some restrictive assumptions (which are verified in our case), this approach (henceforth called *EC* for "estimation and control") leads to a *self-tuning* policy, i.e., a control law that produces the same long-term average cost as the stationary, optimal policy obtained with complete a priori knowledge of parameter values [7].

For the estimation technique, we will adopt Maximum Likelihood Estimation (MLE), which satisfies the requirements for statistical convergence toward true parameter values. MLE is described in Section 4.1. For the computation of the optimal control law for a given ML parameter estimate, we could, in principle, apply the exact optimization techniques introduced in [12]. In practice, this solution may not be applicable because the computational burden of recomputing an optimal policy every time slice could be

$$\mathbf{P_{SR}} = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1m} \\ P_{21} & P_{22} & \cdots & P_{2m} \\ \vdots & \vdots & & \vdots \\ P_{m1} & P_{m2} & \cdots & P_{mm} \end{pmatrix}$$

(a)

state	1	2	$\cdots$	m	n_i
1	n_{11}	n_{12}	$\cdots$	n_{1m}	n_1
2	n_{21}	n_{22}	$\cdots$	n_{2m}	n_2
:	:	:		:	:
:	:	:		:	:
m	n_{m1}	n_{m2}	$\cdots$	n_{mm}	n_m
					n

(b)

Fig. 4. (a) State transition matrix and (b) observed transition table of SR.

sizable, thereby violating the assumption that the power manager takes fast decisions with negligible power.

Therefore, we propose a novel table look-up method with linear interpolation (described in Section 4.2) to compute the control law to be applied at each time step. Using a look-up table is equivalent to enforcing a discretization on the continuous range of optimal control policies. This may, in principle, prevent the achievement of an optimal solution. Fortunately, theoretical results [8] show that a succession of optimal discretized policies tends to the optimal policy as discretization is refined. Furthermore, linear interpolation helps in reducing discretization errors, as demonstrated by our experiments.

4.1 Estimation of Stationary SR

Maximum Likelihood Estimation (MLE) produces estimators that are consistent and, under certain regularity conditions, can be shown to be most efficient in an asymptotic sense (i.e., as the sample size n approaches infinity) [30]. The principle of this method is to select, as an estimate of θ, the value for which the observed sample is most likely to occur.

Suppose there is a service requestor SR, which has the state space $S_r = (1, 2, \cdots, m)$ and the SR is observed until n transitions have taken place. Then, the state transition matrix of SR can be represented as Fig. 4a and the observed transitions can be collected in a tabular form as shown in Fig. 4b, where n_{ij} is the number of transitions observed from state i to state j and $n_i = \sum_{j=1}^{m} n_{ij}$.

Then, MLE of the given SR is $\hat{P}_{ij} = \frac{n_{ij}}{n_i} \;\; i, j = 1, 2, \cdots, m$. We do not show the details of the derivation for MLE, but the complete derivation can be found in [21]. The estimator may be biased for small n, thus, in our approach, every transition from time slice 0 is recorded in the transition table like Fig. 4 for the estimation. As n increases, $\hat{P}_{ij}$ will converge to a certain time-invariant matrix like the transition probability matrix of SR in stationary known environment. In other words, for reasonably large n, $\hat{P}_{ij}(n) \approx P_{ij}(\infty)$.

We can define the convergence time of estimation by introducing the tolerable error, ϵ. Then, the convergence time is the smallest n, such that $|\hat{P}_{ij}(n) - P_{ij}(\infty)| \leq \epsilon$ for $\forall i, j$.

The $n - step$ transition probability can be computed from $1 - step$ transition probability and initial probability vector. Namely, it is a function of $1 - step$ transition probability and n [30].

For example, $P_{00}(n)$ of two-state SR can be represented as follows [30]:

$$\begin{aligned} P_{00}(n) &= \frac{P_{10}(1) + P_{01}(1) * (1 - P_{01}(1) - P_{10}(1))^n}{P_{01}(1) + P_{10}(1)} \\ &= P_{00}(\infty) + \frac{P_{01}(1) * (1 - P_{01}(1) - P_{10}(1))^n}{P_{01}(1) + P_{10}(1)}. \end{aligned} \quad (1)$$

Thus, the convergence time is n, which makes the second term smaller than ϵ. Notice that convergence time depends on the time-step, n, as well as the property of the given stationary process.

It is important to stress the fact that ML estimation of the probability matrix of the SR is completely independent from the PM policy adopted for the SP. In other words, estimation of the unknown parameter does not interfere with control. This *identifiability* condition is sufficient to guarantee that the basic EC adaptive control is self-tuning [7].

4.2 Decision Policy

As mentioned in Section 3, the optimal policy π is the sequence of decisions chosen from the optimized decision table according to the system state in every single time slice. This approach is possible because the transition probability matrix of SR is determined before optimizing the decision table. But, in the unknown stationary environment, it is not possible to build the decision table in advance because the transition probability matrix is unknown. For this reason, it is necessary to provide a decision table for the estimated transition probability matrix dynamically. On the other hand, the EC adaptive policy requires a new policy optimization for every new ML estimate for the SR. This is hard to apply in practice because the computation required to optimize the policy is demanding.

In this section, we describe a table look-up method augmented with a linear interpolation technique that relaxes computational requirements without significantly degrading solution quality. For the sake of simplicity, it is assumed that SR has two states, but this method can handle SRs with more than two states.

4.2.1 Look-Up Table Construction

The transition probability matrix of a stationary SR can be characterized by user request probability, $R_i \in [0, 1]$, $i = 0, 1$. If each dimension, R_i is sampled with a finite number of samples, each sampling point in dimension i is denoted as $R_{ij}, j = 0, i, \cdots, NS_i - 1$, where, NS_i is the number of sampling points for dimension i. Based on these sampling points, a look-up table, called *policy table*, is constructed as shown in Fig. 5.

Each cell of a policy table corresponds to an SR of which the user request probability is (R_{0j}, R_{1k}) ($j = 0, 1, \cdots, NS_0 - 1$ and $k = 0, 1, \cdots, NS_1 - 1$). And, each cell of a *policy table* is also a two-dimensional table, which we call a *decision table* (see Section 3 and [12]). A *decision table* is a matrix with as many rows as the total system states and as many columns as the command issued by the PM to SP. Each

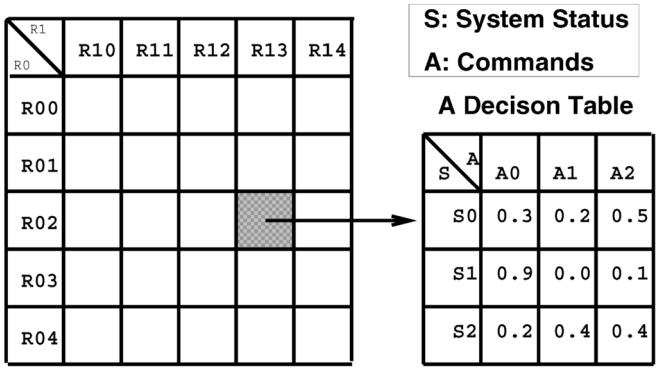


Fig. 5. An example of a 2D policy table ($NS_0 = NS_1 = 5$).

cell of a *policy table* can be indexed as a pair (j, k) and its corresponding request probability pair is (R_{0j}, R_{1k}). For each pair (j, k), a policy optimization is performed to get the corresponding decision table and the obtained decision table is stored to a cell of the policy table with the corresponding index. The overall table is constructed once for all and its size is $NS_0 \times NS_1 \times$ the size of the table used in [12].

4.2.2 Decision Using Interpolation

For a given time slice, $(\hat{R}_0, \hat{R}_1)$ can be obtained using the estimation technique mentioned in Section 3 and two consecutive indices can be chosen for each dimension such that $R_{0j} \leq \hat{R}_0 \leq R_{0(j+1)}$ and $R_{1k} \leq \hat{R}_1 \leq R_{1(k+1)}$. Thus, four decision tables corresponding to the chosen indices can be used to calculate the decision for the given $(\hat{R}_0, \hat{R}_1)$ and current observed system state. From each decision table chosen, a row corresponding to the current system state, denoted by CS is selected as shown in Fig. 6.

From these four rows, the final decision row can be obtained by two-dimensional interpolation technique—applying the one-dimensional interpolation represented in (2) iteratively. In a one-dimensional function $f(x)$, the function value $f(x)$ for any point x which is located in between any two points—x_1 and x_2 can be linearly interpolated as follows:

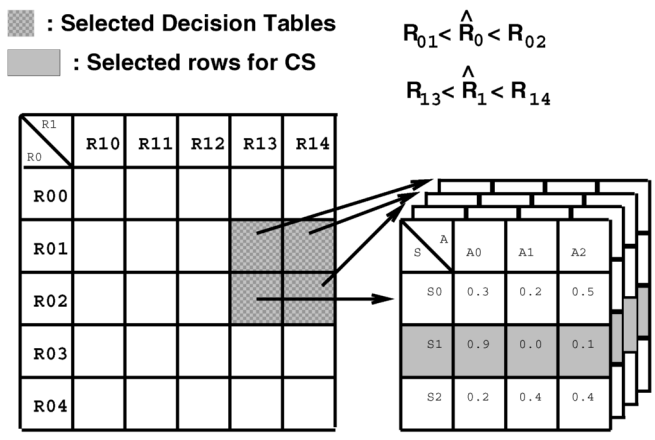


Fig. 6. Decision table and rows selection from policy table.

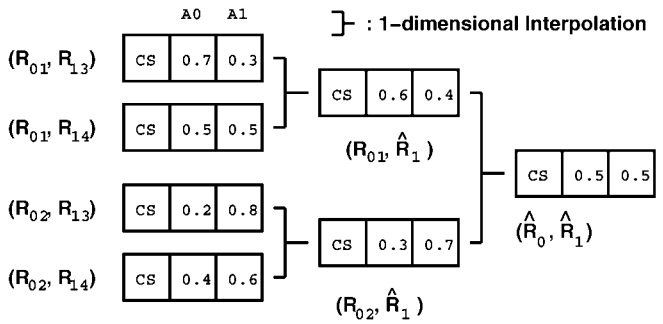


Fig. 7. Visualized two-dimensional interpolation example.

$$f(x) = \frac{(f(x_2) - f(x_1))x + x_2 f(x_1) - x_1 f(x_2)}{x_2 - x_1}. \qquad (2)$$

The iterative procedure is visualized in Fig. 7 and the pseudocode of the interpolation/extrapolation procedure including decision table and row selection is shown in Fig. 8.

Extrapolation is used if $\hat{R}_i > R_{i(NS_i-1)}$ or $\hat{R}_i < R_{i0}$. In all other cases, the interpolated value is computed as three successive one-dimensional linear interpolations on the selected table entries. The proposed technique in this section is described for an SR with two states, but it can be extended to handle SR with more states by increasing the number of dimensions. For n state SR, the required number of dimensions is $n(n-1)$, thus the computational effort is increased proportionally to n^2. In our application, it is enough to model SR with two states with reasonable computation effort.

5 DPM in Nonstationary Environment

5.1 Nonstationary Service Requestor

In many practical applications, the assumption of stationary SR does not hold. Workload is subject to changes over time, as

```
2DInterpolation (CS, cell, R̂0, R̂1, NS0, NS1)
for (i = 0; i < 2; i++) {
  if (R̂i ≤ Ri0) { /* extrapolation */
    id_i^0 = 0;
    id_i^1 = 1;
  } else if (R̂i ≥ R̂i(NSi−1)) { /* extrapolation */
    id_i^0 = NSi - 2;
    id_i^1 = NSi - 1;
  } else { /* interpolation */
    id_i^0 = j s.t. Rij ≤ R̂i ≤ Ri(j+1);
    id_i^1 = j+1;
  }
}
for (i = 0; i < 2; i++) {
  for (j = 0; j < 2; j++) {
    Select a decision d2i+j from cell(id_0^i, id_1^j)
    for State CS;
  }
}
foreach (command) {
  d5 = OneDimInterp(d0, d1);
  d6 = OneDimInterp(d2, d4);
  d7 = OneDimInterp(d5, d6);
}
return(d7);
```

Fig. 8. Two-dimensional interpolation.

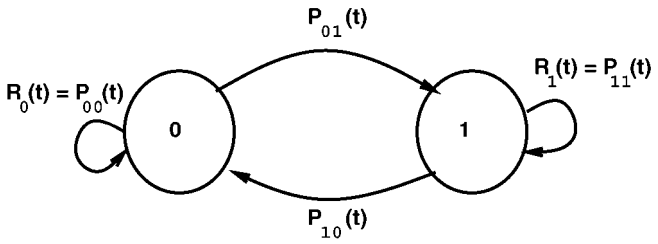


Fig. 9. An example of a Markov chain for nonstationary SR.

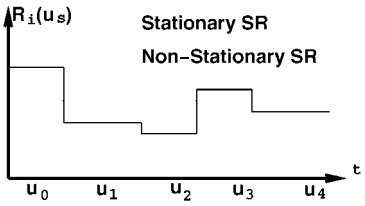


Fig. 10. $R_i(u_s)$ of nonstationary SR vs. stationary SR.

intuitively suggested by observation of typical computer systems. In this section, we describe DPM policies tailored to nonstationary SR models. These adaptive DPM policies are more generally applicable to real system environments than those in Section 3 and 4, they are heuristic because we cannot claim global optimality in a nonstationary environment.

Our first step is to model a nonstationary SR as shown in Fig. 9. A nonstationary workload, denoted by U^l, is modeled by a series of stationary workloads which have different user request probabilities. Each stationary workload is denoted by $u_s, s = 0, 1, \cdots, N_u - 1$, where N_u is the total number of stationary workloads forming the nonstationary workload, U^l. Thus, a nonstationary workload can be represented as $U^{N_u} = (u_0, u_1, \cdots, u_{N_u-1})$. In this model, the R_i becomes a function of the given sequence and can be distinguished from the R_i of stationary SR as shown in Fig. 10.

Notice that the nonstationary SR model is very general: By increasing N_u, we can model any given workload with arbitrary accuracy. In fact, for any given sequence of zeros and ones of length λ, we can set $N_u = \lambda$, and define N_u different two-state Markov chains with deterministic transitions that reproduce exactly the given sequence.

Clearly, the knowledge of such a model at time zero is equivalent to assuming the existence of a perfect oracle that can predict the future with no uncertainty. Realistically, we can only expect to be able to predict the future based on past experience and take into account nonstationarity by limiting the effect that the remote past will have on our current prediction. In other words, we will track changes in transition probability of the nonstationary Markov model by observing the workload on a limited-size time window in the past.

In the nonstationary environment of Fig. 10, the optimal policy is to take a decision based on the decision table optimized for the R_i for each u_s. We call such an ideal policy the **best-adaptive policy** which requires the perfect knowledge of the change of u_s and cannot be implemented in a real situation. Therefore, the objective in this section is to propose techniques that achieve results comparable to *best-adaptive policy*. The look-up table based interpolation technique introduced in Section 4 is still employed for dynamically choosing the most appropriate policy for the estimated SR, but the estimation technique in Section 4 should be replaced due to the nonstationarity of the workloads. We propose two window-based approaches to handle the nonstationarity of the workloads.

For the sake of clarity, we enrich our notation: P_{SR} becomes a function of the sequence and denoted by $P_{SR}(u_s)$. From now on, we will denote the actual values as function of u_s because they are constant over time for a given u_s and the estimated values are represented as a function of time. For example, $R_i(u_s)$ is the actual user request probability of a sequence u_s and $\hat{R}_i(t)$ is the estimated user request probability at time t.

5.2 Single Window Approach

A sliding window stores the recent user-request history to predict future user requests. This approach is a derivation of MLE (Section 4) because it estimates the request ratio depending on recent user history (the information stored in a sliding window) instead of the whole history.

A sliding window, denoted as W, consists of l_w slots and each slot, $W(i), i = 0, 1, \cdots l_w - 1$, stores one previous user request, i.e., $s_r \in 0, 1, \cdots, S_r - 1$. The basic window operation is to shift one slot constantly for every time slice.

An example of a window operation for a two-state user requests is shown in Fig. 11. At each time point, $W(i+1) \leftarrow W(i), i = 0, 1, \cdots, l_w - 1$ and $W(0)$ stores a new user request from SR.

At a given time point t, $\hat{P}_{ij}$ in P_{SR} can be simply estimated by the ratio between the total number of transitions from state i to j and the total number of occurrences of state i observed within the window.

It may be impossible to define $\hat{P}_{ij}$ when the sliding window does not have any information of state i at a certain time point. In this case, we define $\hat{P}_{ij}$ as 0 when $i = j$ and $1/(S_r - 1)$ when $i \neq j$, respectively.

Let us denote the total number of state i observed by the sliding window by A_i, then $A_i = \sum_{k=1}^{l_w-1}(W(k) = i)$ and $\hat{P}_{ij}$ at a given time t can be formally expressed as (3).

$$\hat{P}_{ij}(t) = \begin{cases} \frac{1}{A_i}\sum_{k=1}^{l_w-1}[(\mathrm{W(k)} = \mathrm{i}) \wedge \mathrm{W(k-1)} = \mathrm{j})] & \text{if } A_i \neq 0 \\ 0 & \text{if } A_i = 0 \text{ and } i = j \\ 1/(S_r - 1) & \text{otherwise,} \end{cases} \tag{3}$$

where "=" is the equivalence operation with a Boolean output (i.e., it yields "1" when the two arguments are the same, otherwise returns "0") and where "$\wedge$" is the "*conjunction*" operation.

There exist three possible estimation error sources—resolution error, biased estimation error, and adaption time.

1. **Resolution error** is due to the maximum precision of $\hat{R}_i(t)$, which is limited to $\frac{1}{l_w}$. For example, if $l_w = 10$, $\hat{R}_i(t)$ cannot express two digit effective numbers such as 0.95. The longer l_w is, the smaller the effect of resolution error is.
2. **Biased estimation error** happens when l_w is shorter than the burst lengths of sequences. Suppose an SR

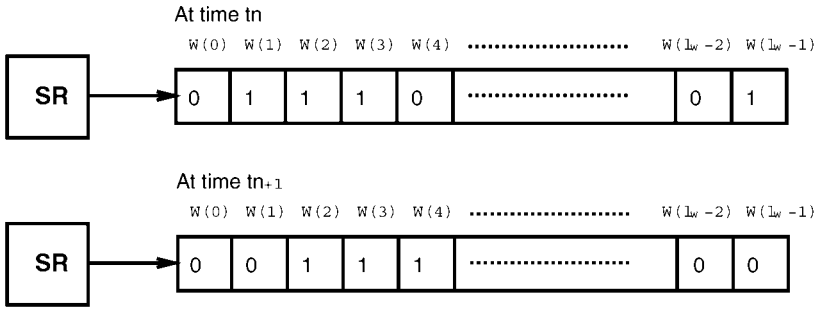


Fig. 11. Single window operation for two-state user requests.

generates $100\ 1s$ after $100\ 0s$ and $l_w = 10$. When the sliding window is in the middle of 0 (1) sequence, the window does not have any information on state 1 (0), which causes the estimator to guess the $\hat{R}_1(t)$ ($\hat{R}_0(t)$) arbitrarily (the second or the third case of (3)). The longer l_w is, the smaller the effect of biased estimation error is.

3. **Adaptation time** is considered when the sliding window is observing u_{s-1} and u_s—the window is experiencing the switching of two stationary processes. The estimation of the new stationary process (u_s) is disturbed by the old stationary process (u_{s-1}). Thus, it is the time required to fill the window, W, fully with the transitions of the new sequence u_s. This error source can be reduced by reducing l_w.

These error sources are graphically represented in Fig. 12.

It is obvious that the resolution error is limited by $1/l_w$ and the adaption time is always l_w independent to u_s. Also, to avoid the biased estimation error, l_w should be larger than the sum of average sequence length of 0 state and 1 state in case of two-state SR. For example, l_w should be larger than 200 in the case of Fig. 12b.

The average burst length of each state i for a given u_s ($I_i(u_s)$) can be expressed as follows:

$$I_i(u_s) = \sum_{k=0}^{\infty} k P_{ii}^k(u_s)(1 - P_{ii}(u_s)) + 1 = \frac{1}{1 - P_{ii}(u_s)}. \quad (4)$$

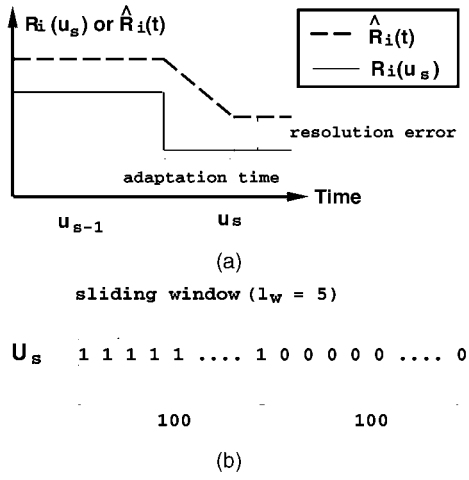


Fig. 12. Estimation error source. (a) Resolution error and adaptation time. (b) Biased estimation error.

This equation represents the average number of self-transitions of each state i whenever state i is first visited. Thus, for two-state SR, the required l_w to avoid biased estimation is simply $I_0(u_s) + I_1(u_s)$ for a given u_s.

Finally, if $l_w \to \infty$, both resolution error and biased estimation error become negligible, but adaption time becomes infinite. Thus, the single window approach becomes the MLE of the unknown stationary environment.

5.3 Multiwindow Approach

In the single window approach, it is not guaranteed that the previous history observed by the window at a given time point always provides complete state information. Due to this limitation, the second and third case of (3) can be frequently used, especially when l_w is small. To avoid this situation, l_w should be increased, but increasing l_w is not desirable because *adaptation time* is also increased. The multiwindow approach is devised to overcome this situation by keeping the previous history of each state separately.

The basic structure for the multiwindow approach is shown in Fig. 13. There are as many windows as S_r of SR and their sizes are the same (l_w). For convenience, each window is denoted by W_i, which is dedicated to the state $s_r = i$. Therefore, W_i stores only the previous transitions from state i. At a time point t, the Previous Request Buffer (PRB) stores $s_r(t-1)$ and controls the window selector to select a window W_i, where $s_r(t-1) = i$. At each time point t, $W_i(j+1) \leftarrow W_i(j)$, $i = s_r(t-1)$, and $j = 0, 1, \cdots, l_w - 1$. Note that only the selected window W_i, $i = s_r(t-1)$, performs the shift operation, while the other windows stay constant. Thus, each window W_i stores l_w previous user requests and plays a role in predicting the transition probabilities from state i to any other states. Each row of $P_{SR}(u_s)$ is mapped to the window corresponding to the state which is the source of the transition and $P_{ij}(u_s)$ can be easily calculated as follows:

$$\hat{P}_{ij}(t) = \frac{\sum_{k=0}^{l_w-1}(W_i(k) = j)}{l_w} \quad for\ all\ i, j. \quad (5)$$

The estimation error sources of the multiwindow approach are resolution error and adaptation time, but there is no biased estimation error because each state has its dedicated window to store past history.

While the resolution error is simply $\frac{1}{l_w}$ (like for the single window approach), the adaptation time is not a constant,

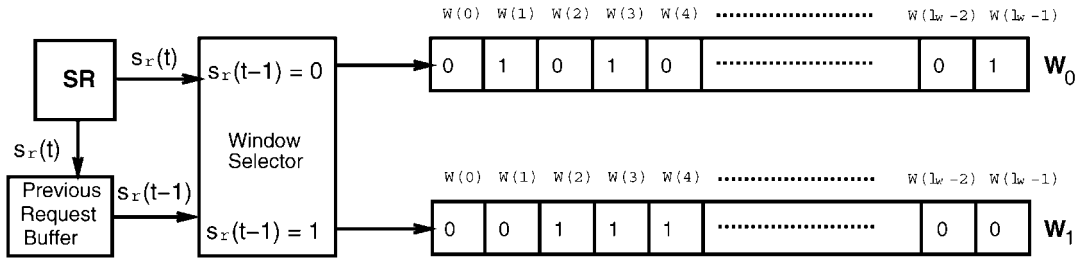


Fig. 13. Multiwindow operation for two-state user requests.

unlike the single window approach. The adaptation time is determined by the window which is fully filled with the new requests (u_s) in the latest.

Example. Suppose a stationary SR which can generate either u_i or u_j depending on the initial state of SR, where $u_i = 00000110000011\cdots$ and $u_j = 11000001100000\cdots$. Also, suppose that l_w of each window (W_0 and W_1) is 10. Then, to completely fill W_0 with u_i, we need two repetitions of sequence 0000011, whereas we need five repetitions of sequence 0000011 for W_1. Thus, the adaptation time is $5 \times 7 = 35$ time slices determined by W_1. On the other hand, for u_j, we only need $5 \times 7 - 5 = 30$ time slices because $0s$ in the last repetition is of no use (W_0 is already filled with u_j). Therefore, the average adaption time for the given SR is 32.5.

For two-state SR, it can be generally represented as follows:

$$t_{adapt} = \left\lceil \frac{l_w}{m} \right\rceil (I_0(u_s) + I_1(u_s)) - \frac{1}{2}\left(\left\lceil \frac{l_w}{m} \right\rceil - \frac{l_w}{m} \right)(I_0(u_s) + I_1(u_s)), \quad (6)$$

where $m = \min(I_0(u_s), I_1(u_s))$, thus $\lceil \frac{l_w}{m} \rceil$ represents the number of repetitions required to fill the window for the given sequence of which the length is $I_0(u_s) + I_1(u_s)$. The last term represents the unnecessary part of the sequence in the last repetition. Finally, the last term is divided by 2 to get the average value with the consideration of different initial states.

6 Policy Implementation

The proposed approaches were implemented on both a desktop PC and a laptop PC to control the power state of their hard disk drives. Policy implementation consists of two parts—offline policy table computation and runtime application. The optimization phase is similar to the policy optimization procedure introduced in [12] except that the optimization is repeated many times, for every different set of user request probabilities, $R_i, i = 0, 1$, to construct the policy look-up table. The policy look-up table resides in main memory as a part of PM and PM is implemented on the PC thanks to an open standard called *Advanced Configuration and Power Interface (ACPI)* [22].

ACPI specifies protocols between hardware components and operating systems to enable operating-system directed power management (OSPM); the OS can adopt system-wide power management policies. Fig. 14 shows the ACPI interface; it consists of the OS, which controls the power states, the ACPI interface, and the hardware that responds to ACPI commands and changes power states. An ACPI-compliant device can have up to four power states: `pwerDeviceD0 (D0)`, the working state, and `PowerDeviceD1 (D1)` to `PowerDeviceD3 (D3)`, representing three different sleeping states. But, only $D0$ and $D3$ states are used in our implementation because the hard disk drives provide only single sleep state.

We use Microsoft Windows 2000 in our implementation. In Windows, power management commands are processed as IO commands using *I/O request packets* (IRP). Special IRPs are required to synchronize ACPI commands so that the transient current does not exceed the maximum capability of the power supply. We implement power managers using a *filter driver* (FD) template [24]. A filter driver is a device driver attached upon another device driver; it can intercept commands from the OS and responses from the lower driver. The filter driver observes IO activities generated from OS and applications to update the estimation of stochastic parameters. When the power manager determines to shut down a device, it issues a power IRP to the lower level driver to control the power states.

By implementing power managers in a commercial operating system, we can experiment with different algorithms running realistic workloads. Because our filter drivers are visible only to the OS and its lower-layer driver, application programs can run without any modifications. Based on the software-controlled architecture described above, both single and multiple window approaches were implemented and tested.

7 Experimental Results

We performed two sets of experiments: First, we simulated the proposed algorithms in the context of the system model of Fig. 2; second, we applied them to real-world power-manageable systems. We used simulation to analyze and discuss the inherent properties of the adaptive power management techniques proposed in Section 5, while we carried out real-world experiments to make a fair comparison with other techniques. The experiments were performed on a Sony VAIO PCG-F150 laptop computer, and on a VA Research VArStation desktop computer. The service providers for our experiments were commercial power-manageable HDDs by Fujitsu (VAIO PCG-F150) and IBM (VArStation). Table 1 reports their average power consumption measured in the active and sleep states (P_a and

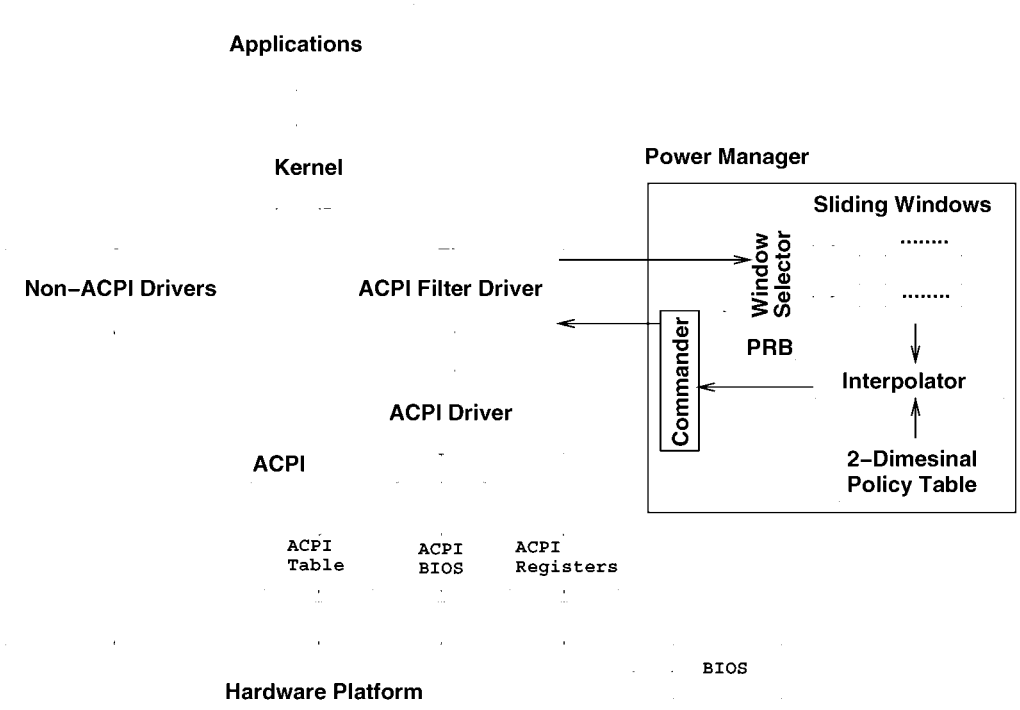


Fig. 14. ACPI interface and PC platform.

P_s) and during shut-down and wake-up transitions (P_{sd} and P_{wu}). Transition times T_{sd} and T_{wu} are also reported in Table 1. The numbers reported in Table 1 are obtained from real measurement using the hardware setup shown in Fig. 15.

The 12V and 5V power lines go through two digital multi-meters, as shown in Fig. 15 and both meters are connected to a data collection computer through the RS232 port. Readers interested in the details of measurement may refer to [37], [38].

7.1 Simulation-Based Analysis

We modeled the Fujitsu's HDD of Table 1 as an SP with four power states, representing *active*, *sleep*, *wakeup*, and *shutdown* operating modes. When active, the SP serves one request per time slice, while it has no throughput when in sleep and transient states. A queue SQ is available to store up to three incoming requests when the SP is not ready to serve them. The SR is a two-state Markov chain that issues a request per time slice when in state 1 and no requests when in state 0. The overall system is a Markov chain with 32 states.

According to the actual behavior of the HDD, wake-up transitions are triggered by incoming requests, while shut-down transitions are triggered by a GO_TO_SLEEP command issued by the PM. In practice, the PM controls the SP by issuing two alternative commands, GO_TO_SLEEP and GO_TO_ACTIVE. When the SP is in active state with no incoming and waiting requests, the GO_TO_SLEEP causes a shut-down transition, while GO_TO_ACTIVE leaves the SP in the active state. On the other hand, when the SP is in sleep state with no incoming and waiting requests, GO_TO_SLEEP leaves the SP in the sleep state, while GO_TO_ACTIVE wakes up the SP. In all other conditions (there are incoming or waiting requests or the SP is in either shutdown or wakeup state), the PM has no control over the SP. Though the complete PM policy can be viewed as a 32×2 matrix, there are only two significant rows, corresponding to a state ($SP = \text{active}$, $SR = 0$, $SQ = 0$) and the other state ($SP = \text{sleep}$, $SR = 0$, $SQ = 0$). For all other states, the power manager does not issue any command, which reduces the computation overhead due to power management.

Moreover, the two entries in the row represent complementary probabilities so that the second one can be obtained as the 1's complement of the first one, representing the conditional probability of issuing a GO_TO_SLEEP command. The value of such a probability is the only degree of freedom available for policy optimization. Thus, the memory space required to store the decision table is only a few bytes. Therefore, total memory required for the policy table is under $1K$ bytes, even when NS_i for each dimension i (number of sampling points for each dimension) is 10. This low memory requirement is especially advantageous when multiple devices must be controlled by our power management policy.

PM policies were designed to minimize power consumption subject to performance constraints expressed in

TABLE 1
Power States of Commercial HDDs from Fujitsu and IBM

HDD	P_s	P_a	T_{sd}	P_{sd}	T_{wu}	P_{wu}
	Watt	Watt	sec	Watt	sec	Watt
IBM	0.75	3.48	0.51	2.12	6.97	7.53
Fujitsu	0.13	0.95	0.67	0.54	1.61	2.72

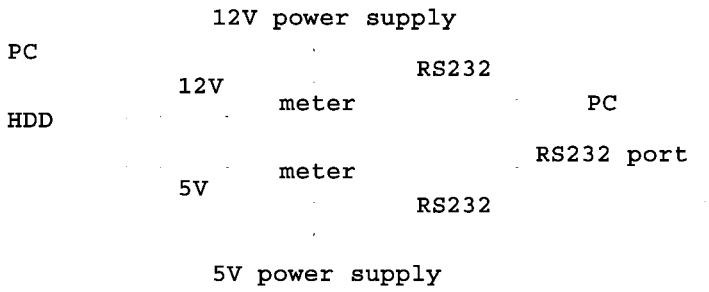


Fig. 15. Hardware setup for HDD power measurement.

terms of upper bounds on two performance metrics: the average waiting time (hereafter called *waiting time* and denoted by W_p) and the average probability of losing an incoming request because of a queue-full condition (hereafter called *request loss* and denoted by L_p).[2]

For our experiments, we used $L_p = 0.05$, representing the probability of losing up to 5 percent of incoming requests, and $W_p = 1$, representing an average delay of one time slice experienced by each service request. The look-up table (LUT) of PM policies was constructed by keeping the constraints unchanged while varying workload parameters R_0 and R_1 with a 0.05 step. For each (R_0, R_1) pair policy, optimization was performed (as described in [12]) and the resulting policy stored in the corresponding entry of the LUT. The entire process took less than 10 minutes on a Sun Sparc2, with a 200MHz clock rate and 520MB of memory.

The estimation and control approach proposed in this paper is characterized by two sources of error: estimation and interpolation. In the following subsections, we report the results of simulations performed to isolate and analyze the effects of estimation and interpolation errors. The overall quality of the estimation and control strategy applied to nonstationary workloads is reported and discussed in Section 7.1.2. All simulations were performed using an in-house cycle-accurate stochastic simulator, with 10^6 time steps by default.

7.1.1 Estimation and Interpolation Error

Since the asymptotic convergence of the maximum likelihood estimators is theoretically demonstrated, we need only to evaluate the dynamic properties of the estimators, i.e., their capability of tracing the time-varying parameters of a nonstationary workload. For this purpose, we used a family of nonstationary two-state SRs with self-loop probabilities defined as sinusoidal functions of time:

$$R_0(t) = 0.5 + A \times sin\left(2\pi\frac{t}{T}\right)$$
$$R_1(t) = 0.5 + A \times cos\left(2\pi\frac{t}{T}\right), \quad (7)$$

where T is the period and A is the amplitude of the variation. Since (7) depend only on T and A, we use the notation $SR(T, A)$ to represent sinusoidal SRs.

2. The request loss is a model for the incoming requests when the queue is full. In practice, no request is lost in the implementation because a queue full state will trigger alternative mechanism for buffering. Still, the queue full state is undesirable.

We simulated $SR(T, A)$ for different values of T and we applied single and double-window estimators to trace the variation of R_0 and R_1. Fig. 16 shows the actual behavior of $R_0(t)$ for $T = 1,000$ and $A = 0.45$, together with the estimates provided by a double window of length 20 (DW20), a single window of length 50 (SW50), and a double window of length 100 (DW100). We can compare the estimated waveforms with the original one in terms of: *attenuation*, that is, the ratio between the amplitude of the estimated waveform A_e and that of the original one A, *delay*, evaluated as the time gap between a local maximum of the original waveform and the corresponding maximum of the estimated one, and *noise*, that adds higher-frequency fluctuations to the sinusoidal waveforms. In Fig. 16, it is apparent that DW100 is less noisy than DW20 and SW50, at the cost of sizable estimation delay and attenuation. It is also worth noting that DW20 is closer to the actual waveform and less noisy than SW50.

The above observations are supported by the results of the analysis in the frequency domain, reported in Fig. 17. We repeated the simulation experiment of Fig. 16 for different values of T and we computed attenuation and delay for each estimator. Estimation delays were divided by $T/2\pi$ and expressed in degrees to obtain comparable *phase*

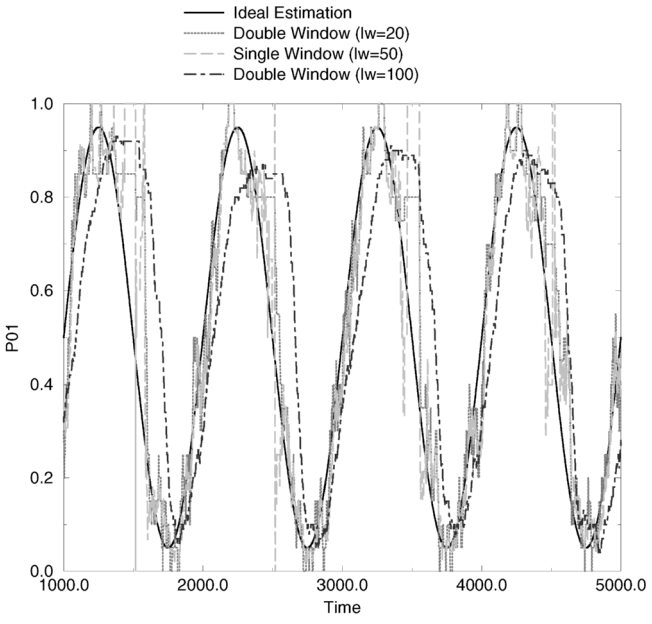


Fig. 16. Ideal and estimated curve at $f = 0.001$.

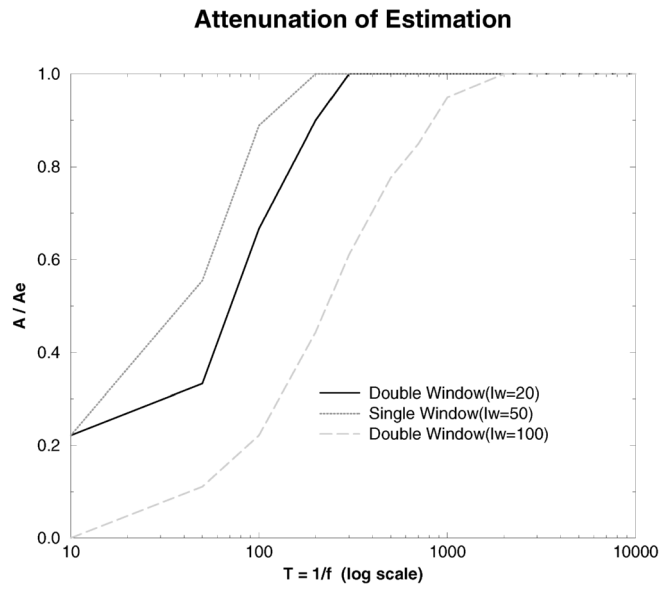


Fig. 17. Attenuation and phase shift of window-based estimates. (a) Attenuation. (b) Phase shift.

shifts. Attenuation and phase shift were then plotted as functions of T, as shown in Fig. 17a and Fig. 17b, respectively.

As expected, all window-based estimators act as low-pass filters: As the period T of workload variations decreases, both the attenuation and the phase shift become critical, while they are negligible for values of T larger than a cut-off value that depends on the estimator. In general, the larger the window the higher the cut-off period. This can be better explained by thinking of a nonstationary workload obtained as the concatenation of two stationary ones, u_0 and u_1. When the workload statistics switch from u_0 to u_1, *adaptation time* proportional to the window length l_w (see Section 5) is required to completely update the contents of the windows in order to estimate the parameters of u_1 independently of u_0. This effect can be appreciated in Fig. 17 by comparing the curves associated with DW20 and DW100, while the comparison between DW20 and SW50 suggests a different trend. This counterintuitive result is due to the inherent capability of double-window estimators of selectively keeping track of significant past events whose distance in time may exceed the window length.

We also conducted experiments to evaluate the accuracy of policy interpolation. For a set of 20 stationary workloads which were artificially generated with random R_0 and R_1, we compared the unknown stationary approach to the known stationary approach. From this experiment, we found that the average penalty of adaptation in unknown stationary approach is below 5 percent in terms of both power consumption and waiting time compared to the known stationary approach.

7.1.2 Overall Quality of Estimation and Control

To evaluate the overall quality of the proposed approach, we simulated an highly nonstationary workload, built as the concatenation of SR traces of variable lengths (ranging from 40,000 to 60,000 time steps) generated by the 20 stationary workloads which were used in the interpolation error estimation. For the sake of conciseness, in the following, we use U^{20} to denote the nonstationary workload trace and u_s ($s = 0, \cdots, 19$) to denote each of the stationary traces that compose it.

We simulated the effect on U^{20} of adaptive control based on single-window (SW) and double-window (DW) estimators with different window sizes. We also implemented and simulated known stationary (KS), unknown stationary (US), and best-adaptive (BA) mentioned Section 5.

We compared the above five policies to the *best oracle* (BO) policy. BO is the most ideal policy in the sense that it perfectly knows the arrival of future requests deterministically. It deterministically decides to shut down the SP at the beginning of idle periods longer than the break-even time t_{be} (i.e., long enough to compensate for the shut-down and wake-up cost) [13]. Also, it wakes up the SP T_{wu} before the next incoming request is issued by SR, thus BO never pays a performance penalty for power saving. This policy cannot be implemented in practice, but its effect can be quantified through offline analysis of any workload. Since it is the "best" possible policy, it is useful for comparisons.

The power consumption and waiting time provided by the power management strategies are reported in Fig. 18a and Fig. 18b as functions of the sliding window size l_w. For a wide range of values of l_w (from 50 to about 5,000) both SW and DW approaches provide almost the same power-performance trade-off of BA policy. Outside this range, estimation errors cause sizeable violations of performance constraints, mainly due to the estimation noise for $l_w < 50$, and to the estimation delay for $l_w > 5000$.

It is also worth mentioning that the given performance constraint represents the maximum performance penalty allowed to achieve minimum power consumption. Thus, if increasing the performance penalty (but still less than the given constraint) does not help to save more power, the waiting time is kept smaller than the given constraint by the control policy. One extreme case can be shown when SR

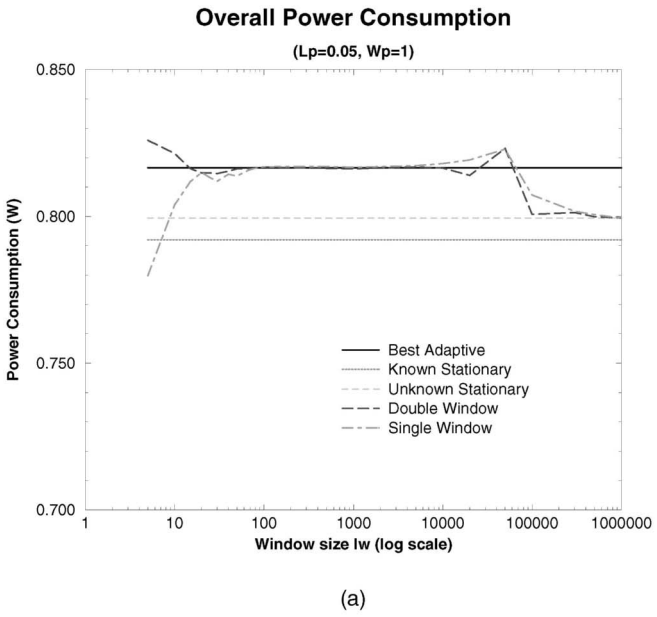


Fig. 18. Power and average waiting time comparison for various window size ($L_p = 0.05$, $W_p = 1$). (a) Power consumption. (b) Average waiting time.

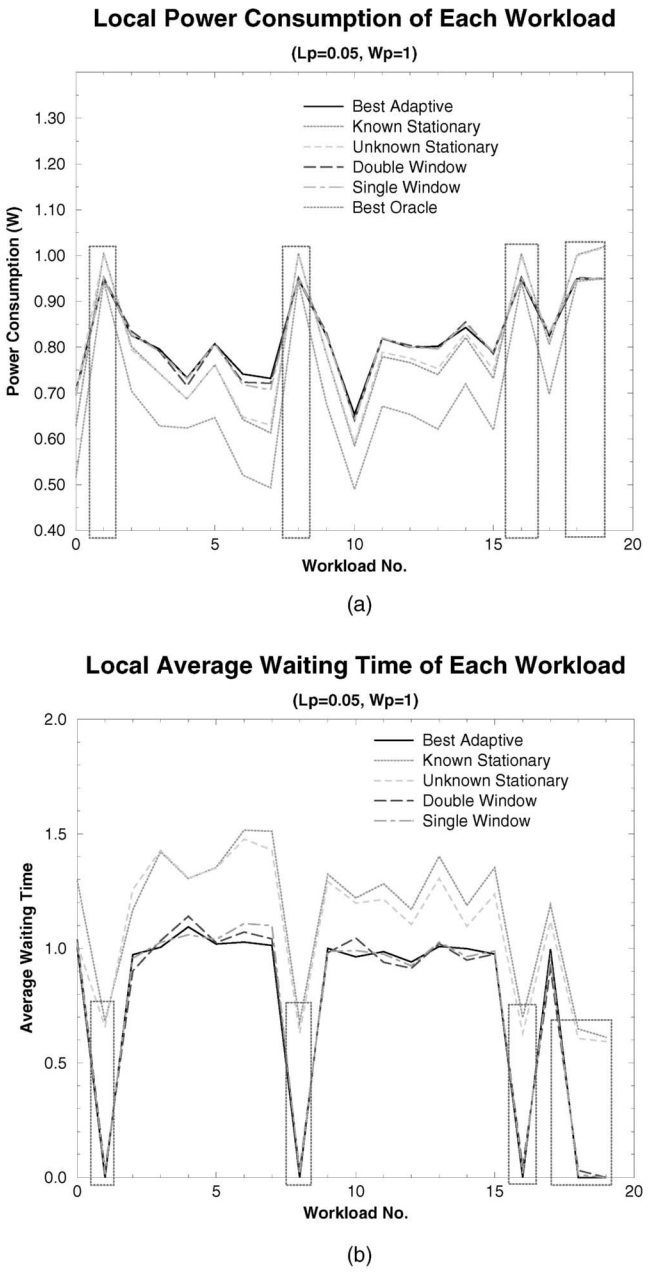


Fig. 19. Local power consumption and average waiting time provided by the PM policies for each subtrace u_s of U^{20} ($L_p = 0.05$, $W_p = 1$). (a) Local power consumption. (b) Local average waiting time.

generates requests without idle periods longer than break-even time T_{be}. In this situation, shutdown does not decrease power consumption, but it increases performance penalty. These points will be further explained by means of Fig. 19.

As for KS and US, though they provide more power savings than BA, they completely violate performance constraints (again, represented by the average waiting time of BA). Their constraint violations are caused by the wrong hypothesis of stationary Markov SR on which they are based. It is also worth noting that the estimation errors made by the SW and DW approaches when the windows they use are too small or too large are never as critical as those caused by the stationarity assumption.

Fig. 19a and Fig. 19b show the power and performance values achieved by the PM strategies for each stationary subtrace u_s in U^{20}. Index s is reported on the x axis. Boxes are used to point out the cases in which the constraint on W_p was inactive either because it was dominated by that on L_p or because there were no idle periods longer than the break-even time. In both cases, the actual waiting time was well below the given constraint, causing the overall average reported in Fig. 18b to be around 0.5 instead of 1.

Interestingly, the performance of SW and DW (for $l_w = 50$) is always comparable to that of BA, both in terms of power and in terms of waiting time, meaning that both estimation and interpolation errors may be made almost negligible by carefully selecting l_w. On the other hand, the ideal BO strategy often provides a much better (but unreachable) trade-off. As for US and KS, their apparent

advantage in terms of power is paid in terms of performance violations that become evident in Fig. 19. The average waiting time they impose often exceeds by 50 percent the given constraint.

It is also worth noting that, when there are no idle periods longer than the break-even time, all adaptive policies take the correct decision of keeping the resource always on, locally reaching the same quality of BO (see, for instance, $u_s = 8$). In contrast, US and KS policies still issue GO_TO_SLEEP commands that cause both a power waste and a performance penalty.

7.2 Experiments with Policies Implemented on Computers

We implemented single and double-window adaptive control strategies on ACPI-compliant PCs mounting the power-manageable HDDs of Table 1: The MHF2043AT HDD (3.8GB) by Fujitsu was installed on a VAIO PCG-F150 laptop computer from Sony (Pentium II, 32MB memory), while the DTTA-350640 HDD (6.44GB) by IBM was installed on a VArStation desktop computer from VA Research (Pentium II, 256MB memory). The base unit of time slice was set to 1 *second*, which is large enough to tolerate the computation cost of the power manager.

Alternative DPM algorithms [9], [10], [15], [26], [27], [28] and timeout mechanisms [34] were implemented on the same platforms for comparison. All PM schemes were applied under the same workload conditions, represented by an 11-hour trace of disk accesses generated by text editors, debuggers, and graphical tools running on top of Windows-NT. The quality of each control strategy was evaluated based on five metrics:

- P: Average power consumption (unit: W).
- N_{sd}: Number of shutdowns.
- N_{wd}: Number of wrong shutdowns causing a power overhead.
- T_{ss}: Average sleeping time per shut-down (unit: sec).
- T_{bs}: Average idle time before shutdown (unit: sec).

Notice that all of the metrics shown above are related to only HDD, namely, P is the power consumption of HDD alone. The experimental environment described in Section 6 provides the runtime support for online computation of the above metrics.

While average power consumption P provides a direct measure of the objective function of policy optimization, performance metrics are more involved and need some explanations. The number of shutdowns N_{sd} is directly related to the performance penalty that has to be paid (regardless of the PM scheme) to wake up the SP. The number of wrong shutdown decisions N_{wd} is a measure of inefficiency: A wrong decision causes both a performance and a power penalty. On the contrary, the average sleeping time per shut-down, T_{ss}, is a measure of efficiency: The longer the sleeping time, the lower the number of shut-downs required to achieve the same power savings. Finally, T_{bs} can be viewed as a measure of inefficiency since it represents wasted idle time. In summary, good PM strategies should be characterized by low values of P, N_{sd}, N_{wd}, and T_{bs} and by large values of T_{ss}.

TABLE 2
Algorithm Comparison:
(a) Notebook Computer, (b) Desktop Computer

Algorithm	P	N_{sd}	N_{wd}	T_{ss}	T_{bs}
best-oracle	0.33	250	0	118	0
DW	0.43	191	28	127	13.4
[26]	0.44	323	64	79	5.4
[27]	0.47	273	73	88	12.4
[9]	0.50	623	427	37	3.0
$\tau = 30$	0.51	139	7	157	30.0
SW	0.51	226	83	96.05	20.05
[28]	0.52	196	48	109	24.5
US	0.62	173	54	102	35.2
[15]	0.64	881	644	19	2.3
$\tau = 120$	0.67	55	0	255	120.0
always-on	0.95	-	-	-	-

(a)

Algorithm	P	N_{sd}	N_{wd}	T_{ss}	T_{bs}
best-oracle	1.64	164	0	166	0
[26]	1.94	160	15	142	17.6
DW	1.97	168	26	134	18.7
$\tau = 30$	2.05	147	18	142	30.0
[28]	2.09	147	26	138	29.9
[27]	2.19	141	37	135	27.6
[15]	2.22	595	430	41	4.1
SW	2.25	295	188	68.42	14.25
$\tau = 120$	2.52	55	3	238	120.0
US	2.60	105	39	130	48.9
[9]	2.99	595	503	30	7.6
always-on	3.48	-	-	-	-

* **DW:** Double Window; **SW:** Single Window;
US: Unknown Stationary; τ: timeout

(b)

Experimental results are reported in Table 2: Rows are associated with PM algorithms, columns with power/performance metrics. Algorithms are sorted for increasing power consumption. Notice that the reported power value is only consumed by each target HDD.

It is also worth mentioning that DPM aims at reducing average power consumption of the target device, but it can increase the peak power consumption because changing power state (especially when the device is waked up) usually requires more power than active state, as shown in Table 1.

When possible, PM schemes are denoted by a reference to the paper they were presented in. BO, SW, DW, and US stay for best-oracle, single-window, double-window, and unknown-stationary policies, according to the notation introduced in Section 7.1.2. Performance constraints used for policy optimization were $L_p \leq 0.05$ and $W_p \leq 2$, while window sizes of 50 and 20 were chosen for SW and DW, respectively. As for simulation, BO policy was designed based on the offline analysis of the trace. Rows referring to timeout policies are denoted by the corresponding timeout values ($\tau = 30$ and $\tau = 120$). Finally, the *always-on* row reports the power consumption of the HDD without power management.

Notice that we could not implement a best-adaptive (BA) policy as we did for simulation because of the nature of the

TABLE 3
Experimental Results for Window-Size Sensitivity Analysis

l_w	P	N_{sd}	N_{wd}	T_{ss}	T_{bs}
5	1.85	224	61	110.49	6.908
10	1.91	198	45	120.39	10.7
20	1.97	168	26	134	18.7
50	2.23	189	47	104.7	26.9
100	2.35	166	89	100.37	33.68

* performed for desktop computer

workload. In fact, best-adaptive policies can only be conceived for piecewise stationary workloads as those artificially constructed in Section 7.1 for simulation experiments. Real-world workloads are not piecewise stationary.

The method proposed by Karlin et al. [26] achieved the best power-performance trade-off on the desktop computer, but DW provided comparable results both in terms of performance (N_s) and in terms of power (P), the difference being within 5 percent. A different trade-off (with higher consumption at lower performance penalty) was provided by $\tau = 30$, [27] and [28], while all other approaches provided much worse results.

On the laptop computer, DW provided the lowest power consumption, followed by [26] and [28]. We remark, however, that, in this case, results are not comparable: The performance penalty caused by [26] is almost twice that of DW, with lower power savings.

The performance of SW is worth being discussed. Both on desktop and laptop experiments, SW results were much worse than DW, while they provided comparable results on simulation experiments. We believe this is due to the bursty nature of real-world workloads. As discussed in Section 5.2, if the SR does not enter a given state for more than l_w cycles, SW provides no information about state transition probabilities from that state. The arbitrary assumptions made in this case by (3) about workload parameters may cause sizeable estimation errors. On the contrary, this situation does not impair the performance of DW.

We also run a set of experiments on the desktop PC to analyze the sensitivity of DW to window size l_w. Results are shown in Table 3. Power savings increase monotonically as l_w decreases because of a lower adaptation delay. Performance metrics, on the other hand, show that the minimum penalty is achieved when $l_w = 20$. Since a further reduction of l_w does not reduce power significantly, while impairing estimation accuracy, $l_w = 20$ provides the best trade-off between power and performance.

Finally, we measured the power overhead caused by the policy computation and the power saving of the overall system achieved by our approach. For this purpose, we compare DW and the competitive approach proposed in [26] to always-on policy on the laptop computer. There are two reasons to select the competitive approach: 1) Its power saving for HDD is comparable to our approach, 2) its policy computation is very simple because it is a timeout approach by setting the timeout value to the break-even time, whereas our approach requires more complex computation.

We measured the current drawn by the overall system by connecting the multimeter to the AC adaptor. For each policy, we prepared two versions of the power manager. The first version is the same as the power manager used for Table 2, but the second version does not issue any command to HDD. In other words, the second version also performs the entire policy computation, but it does not change the power state of the HDD. The first version is useful to measure the impact of each policy on the power saving of the overall system, while the second version can be used to measure the impact of the policy computation overhead on the overall system. Using the second version, we measured the power overhead of the policies for the entire system; it was very small for both approaches (0.8 percent for ours and 0.6 percent for the competitive approach).

On the other hand, the measurement of the overall power saving by our approach was not obvious due to the power consumption and fluctuation caused by other components such as display and processor. The impact of the power saving for HDD on the entire system can vary significantly, depending on the power control policies of other components. To eliminate such variation, we measured the power reduction by our approach for the OS controllable fraction[3] of the total system power budget. Our approach achieved 20.2 percent of power saving, while the competitive appraoch did 15.7 percent of power saving (both include the power overhead of the policy computation). Also, the peak power of the system was increased by 6 percent for both approaches. To summarize, our approach outperforms the competitive approach in terms of overall power saving, while preserving the same peak power.

8 Conclusions and Future Work

In this paper, we described how to derive adaptive power management policies for nonstationary workloads. Our adaptive approach is based on sliding windows and two-dimensional linear interpolation to find an optimal policy from a optimal policy table which is precomputed. Thus, the online computational requirements are mild (0.8 percent of the overall system power consumption). The proposed approach deals effectively with highly nonstationary workloads. Moreover, our adaptive method offers the possibility of trading off power for performance in a controlled fashion. Simulation results show that our method outperforms nonadaptive policies. In addition, experiments on personal computers show that the proposed approach is superior to other algorithms in terms of both power and performance.

As in the case of most current and previous research, we addressed the problem of power managing a single device (e.g., hard disk) abstracted as a single service provider. We believe that our method can be extended to control multiple devices as long as their number is small. Nevertheless, the problem of performing concurrent power management of multiple devices, under nonstationary workloads, remains

3. This subtracts the quiescent power from the total power. The quiescent power is consumed by the laptop computer when no user program is running and HDD is shut down; it was not controlled by our policy.

a challenging problem for future research that goes beyond the scope of this paper.

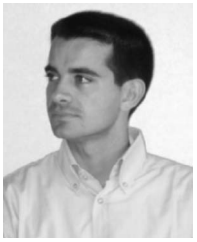
ACKNOWLEDGMENTS

The authors thank the anonymous referees for their careful reviews and many helpful comments. This research was supported in part by the US National Science Foundation under grant CCR-9901190.

REFERENCES

[1] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Kluwer, 1997.
[2] A. Chandrakasan and R. Brodersen, *Low-Power Digital CMOS Design.* Kluwer, 1995.
[3] R. Stengel, *Optimal Control and Estimation.* Dover, 1994.
[4] M. Puterman, *Markov Decision Processes.* Wiley, 1994.
[5] K. Zhou, K. Glover, and J. Doyle, *Robust and Optimal Control.* Prentice Hall, 1995.
[6] K. Astrom and B. Wittenmark, *Adaptive Control.* Addison-Wesley, 1989.
[7] P. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice Hall, 1986.
[8] O. Hernandez-Lerma, *Adaptive Markov Control Processes.* Springer-Verlag, 1989.
[9] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *Proc. Int'l Conf. Computer-Aided Design,* pp. 28-32, 1997.
[10] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Trans. VLSI Systems,* vol. 4, no. 1, pp. 42-55, Mar. 1996.
[11] J. Monteiro and S. Devadas, *Computer-Aided Techniques for Low-Power Sequential Logic Circuits.* Kluwer, 1997.
[12] 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, no. 6, pp. 813-833, June 1999.
[13] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Dynamic Power Management Techniques," *IEEE Trans. VLSI Systems,* Feb. 2000.
[14] *Low-Power Design in Deep Submicron Electronics,* W. Nebel and J. Mermet, eds. Kluwer 1997.
[15] R. Golding, P. Bosh, and J. Wilkes, "Idleness Is Not Sloth," *Proc. Winter USENIX Technical Conf.,* pp. 201-212, 1995.
[16] R. Golding, P. Bosh, and J. Wilkes, "Idleness Is Not Sloth," HP Laboratories Technical Report HPL-96-140, 1996.
[17] *Low-Power Design Methodologies,* J.M. Rabaey and M. Pedram, eds. Kluwer, 1996.
[18] S. Udani and J. Smith, "The Power Broker: Intelligent Power Management for Mobile Computing," Technical Report MS-CIS-96-12, Dept. of Computer Information Science, Univ. of Pennsylvania, May 1996.
[19] J. Lorch and A. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Comm.,* vol. 5, no. 3, June 1998.
[20] P. Whittle, "Some Distribution and Moment Formulae for the Markov Chain," *J. Royal Statistical Soc. B* vol. 17, 1955.
[21] U.N. Bhat, *Elements of Applied Stochastic Process.* John Wiley & Sons, 1984.
[22] http://www.teleport.com/~acpi, ACPI, 1999.
[23] S. Richter and M. Berger ,ACPI4Linux: http://phobos.fachschaften.tu-muenchen.de/acpi/, 1999.
[24] Y.-H. Lu, T. Simunic, and G. De Micheli, "Software Controlled Power Management," *Proc. Int'l Workshop Hardware/Software Codesign,* pp. 157-161, 1999.
[25] 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, no. 6, pp. 813-833, June 1999.
[26] A. Karlin, M. Manasse, L. McGeoch, and S. Owickim, "Competitive Randomized Algorithms for Non-Uniform Problems," *Algorithmica,* vol. 11, no. 6, pp. 542-571, June 1994.
[27] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-Down Policies for Mobile Computers," *Computing Systems,* vol. 8, pp. 381-413, 1995.
[28] Y.-H. Lu and G. De Micheli, "Adaptive Hard Disk Power Management on Personal Computers," *Proc. Great Lakes Symp. VLSI,* pp. 50-53, 1999.
[29] 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.
[30] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice Hall, 1982.
[31] C.L. Liu, *Introduction to Combinatorial Mathematics.* McGraw-Hill, 1968.
[32] Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Process," *Proc. Design Automation Conf.,* pp. 555-561, 1999.
[33] T. Simunic, L. Benini, and G. De Micheli, "Event-Driven Power Management of Portable Systems," *Proc. Int'l Symp. System Synthesis,* pp. 18-23, 1999.
[34] Y.-H. Lu, E.-Y. Chung, L. Benini, and G. De Micheli, "Quantitative Comparison of Power Management Algorithms," *Proc. DATE—Design Automation and Test in Europe Conf. and Exhibition,* pp. 20-26, 2000.
[35] E.-Y. Chung, L. Benini, A. Bogliolo, and G. De Micheli, "Dynamic Power Management for Non-Stationary Service Requests," *Proc. DATE—Design Automation and Test in Europe Conf. and Exhibition,* pp. 77-81, 1999.
[36] E.-Y. Chung, L. Benini, and G. De Micheli, "Dynamic Power Management Using Adaptive Learning Tree," *Proc. Int'l Conf. Computer-Aided Design,* pp. 274-279, 1999.
[37] Y.-H. Lu, T. Simunic, and G. De Micheli, "Software Controlled Power Management," *Proc. Int'l Workshop Hardware/Software Codesign (CODES),* pp. 157-161, 1999.
[38] Y.-H. Lu and G. De Micheli, "Comparing System-Level Power Management Policies," *IEEE Design and Test,* pp. 10-19, Mar. 2001.
[39] R. Kravets and P. Krishinan, "Application-Driven Power Management for Mobile Communication," *Wireless Networks,* vol. 6, pp. 263-277, 2000.
[40] J. Flinn and M. Satyanarayanan, "Energy-Aware Adaptation for Mobile Applications," *Proc. Symp. Operating Systems Principles,* pp. 48-63, 1999.
[41] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 76-81, Aug. 1998.
[42] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Design Automation Conf.,* pp. 134-139, 1999.
[43] Q. Qiu, Q. Wu, and M. Pedram, "Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets," *Proc. Design Automation Conf.,* pp. 352-356, 2000.
[44] Q. Qiu, Q. Wu, and M. Pedram, "Stochastic Modeling of a Power-Managed System: Construction and Optimization," *Proc. Int'l Symp. Low Power Electronic Devices,* pp. 194-199, 1999.
[45] Q. Qiu, Q. Wu, and M. Pedram, "OS-Directed Power Management for Mobile Electronic Systems," *Proc. 39th Power Source Conf.,* pp. 506-509, 2000.
[46] M. Pedram, "Power Management and Optimization in Embedded Systems," *Proc. Asia and South Pacific Design Automation Conf.,* pp. 239-244, 2001.
[47] Q. Wu and M. Pedram, "Dynamic Power Management in a Mobile Multimedia System with Guaranteed Quality-of-Service," *Proc. Design Automation Conf.,* 2001.

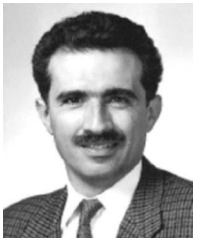
Eui-Young Chung received the BS and MS degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively. He is currently pursuing the PhD degree in electrical engineering at Stanford University and is expected to graduate in June 2002. From 1990 to 1997, he was a research engineer with the CAD Group, Samsung Electronics, Seoul, Korea. His research interests are computer-aided design of VLSI circuits and system-level low power design methodology, including software optimization.



Luca Benini received the PhD degree in electrical engineering from Stanford University in 1997. He is an associate professor in the Department of Electrical Engineering and Computer Science (DEIS) at the Universty of Bologna. He also holds visiting researcher positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, California. Dr. Benini's research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications and in the design of portable systems. On these topics, he has published more than 140 papers in international journals and conferences. He is a member of the organizing commitee of the International Symposium on Low Power Design. He is a member of the technical program committee of several technical conferences, including the Design Automation Conference, International Symposium on Low Power Design, the Symposium on Hardware-Software Codesign.

Alessandro Bogliolo received the Laurea degree in electrical engineering and the PhD degree in electrical engineering and computer science from the University of Bologna, Italy, in 1992 and 1998, respectively. From 1992 to 1999, he was with the Department of Electronics and Computer Science (DEIS), University of Bologna. In 1995 and 1996, he was a visiting scholar at the Computer Systems Laboratory (CSL), Stanford University, Stanford, California. Since then, he has cooperated with the research group of Professor De Micheli at Stanford. In 1999, he joined the Department of Engineering, University of Ferrara, Italy, as an assistant professor. His research interests are in the area of computer-aided design of digital circuits and systems, with emphasis on low-power design, runtime power management, intellectual-property protection, and signal integrity.

Yung-Hsiang Lu received the PhD degree in from the Electrical Engineering Department at Stanford University, Stanford, California, in 2002. He is a postdoctoral researcher in the Electrical Engineering Department at Stanford University. His research focuses on low-power system design.



Giovanni De Micheli is a professor of electrical engineering and, by courtesy, of computer science at Stanford University. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware/software co-design and low-power design. He is the author of: *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994), coauthor and/or coeditor of five other books, and of more than 250 technical articles. He is a member of the technical advisory board of several EDA companies, including Magma Design Automation, Coware, and Aplus Design Technologies. He was a member of the technical advisory board of Ambit Design Systems. Dr. De Micheli is a fellow of the ACM and the IEEE. He received the Golden Jubilee Medal for outstanding contributions to the IEEE CAS Society in 2000. He received the 1987 IEEE Transactions on CAD/ICAS Best Paper Award and two Best Paper Awards at the Design Automation Conference, in 1983 and in 1993. He is president-elect of the IEEE CAS Society in 2002 and he was its vice president (for publications) in 1999-2000. He was editor-in-chief of the *IEEE Transactions on CAD/ICAS* in 1987-2001. He was the program chair and general chair of the Design Automation Conference (DAC) in 1996-1997 and 2000, respectively. He was the program and general chair of the International Conference on Computer Design (ICCD) in 1988 and 1989, respectively. He is a founding member of the ALaRI institute at the Universita' della Svizzera Italiana (USI) in Lugano, Switzerland, where he is currently a scientific counselor.