Policy Optimization for Dynamic Power Management

G. A. Paleologo# L. Benini † A. Bogliolo * G. De Micheli †

Stanford University
Dept. of Engineering-Economic Systems and
Operations Research, Stanford, CA 94305-4023

* Università di Bologna
Dip. Informatica, Elettronica,
Sistemistica
Bologna, ITALY 30165

† Stanford University
Computer Systems Laboratory
Stanford, CA 94305-4070

Abstract

Dynamic power management schemes (also called policies) can be used to control the power consumption levels of electronic systems, by setting their components in different states, each characterized by a performance level and a power consumption. In this paper, we describe power-managed systems using a finite-state, stochastic model. Furthermore, we show that the fundamental problem of finding an optimal policy which maximizes the average performance level of a system, subject to a constraint on the power consumption, can be formulated as a stochastic optimization problem called policy optimization. Policy optimization can be solved exactly in polynomial time (in the number of states of the model). We implemented a policy optimization tool and tested the quality of the optimal policies on a realistic case study.

1 Introduction

Battery-operated portable appliances impose tight constraints on the power dissipation of their components. Designers struggle to meet increasingly tight power budgets as complexity and performance requirements are pushed forward by user demand. Numerous computer-aided design techniques [11] for low power have been proposed to help designers reduce time to market and improve the quality of their products. The vast majority of CAD techniques for low power targets digital VLSI circuits, i.e. chip-level designs. Unfortunately, almost every portable electronic appliance is far more complex than a single chip. Portable devices such as cellular telephones and laptop computers contain tens or even hundreds of components. In most electronic products, digital components are responsible for only a fraction of the power consumed. Analog, electromechanical and optical components are a significant fraction of the total, and are often responsible for the largest contributions to the power budget. For example, the power breakdown for a well-known laptop computer [14] shows that, on average 36% of the total power is consumed by the display, 18% by the hard drive, 18% by the wireless LAN interface, 7% by non-critical components (keyboard, mouse etc.), and only 21% by digital VLSI circuitry (mainly memory and CPU). Reducing the power in the digital logic portion of this laptop by *10X* would reduce the overall power consumption by less than 19%. Laptop computers are not an isolated case. Almost all electronic appliances are complex and heterogeneous systems containing a wide variety of devices that do not fall within the scope of the available computer-aided power optimization techniques. Nevertheless, designers have reacted promptly to the new challenges posed by low-cost, low-power portable appliances. Battery lifetime (or time between recharges) is steadily increasing and the physical dimensions of portable devices are progressively shrinking. These surprising results are achieved thanks to a well-balanced mix of technological innovation, architectural design and optimization. One of the most successful techniques employed by designers at the system level is *dynamic power management* [2]. This technique reduces power dissipation by selectively turning off (or reducing the performance of) system components when they are idle (or partially unexploited). Building a complex system that supports dynamic power management is a difficult and error-prone process. Long trial-and-error iterations cannot be tolerated when fast time to market is the main factor deciding the success of a product. Unfortunately, system-level computer-aided design environments are still in their infancy, and EDA vendors are lagging far behind the needs of this segment of the electronic industry. To compensate for this lack of support, several system developers and vendors [10, 9] are aggressively pursuing a long-term, wide-scope strategy to greatly simplify the task of designing large and complex power-managed systems. The strategy is based on a standardization initiative known as the *Advanced Configuration and Power Interface* (ACPI). ACPI specifies an abstract and flexible interface between power-manageable hardware components (VLSI chips, disk drivers, display drivers, etc.) and the *power manager* (the system component that controls when and how to turn on and off functional resources). The ACPI interface specification enormously simplifies the task of controlling the operating conditions of the system resources but it does not provide any insight on how and when to power manage them. We call *power management policy* (*policy* for brevity) an algorithm that takes decisions upon the state of operation of system components. The most aggressive policy (that we call *eager* policy) turns off every system component as soon as it becomes idle. Whenever the functionality of a component is required to carry out a system task, the component must be turned on and restored to its fully functional state. The transition between the inactive and the functional state requires time and power. As a result, the eager policy is often unacceptable because it greatly degrades performance and may not decrease power dissipation. The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a new kind

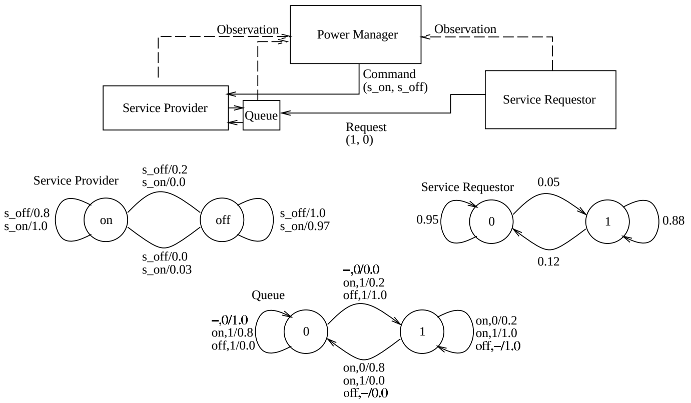

Figure 1: The abstract system model. Components are modeled as Markov chains whose state transition graphs are represented for a simple example situation. The service requestor (SR) has only two states, 0 and 1, representing the number of requests per time period sent to the provider. The queue of the service provider (SQ) has two states, 0 and 1, representing the number of requests to be serviced. The service provider (SP) has two states, on and off, representing its functional state. When on, it serves up to a request per time period taken from the queue. When off it does not serve any request. SR evolves independently, while the transition probabilities of SP depend on the command issued by the power manager (PM) and those of SQ depend on the state of both SP and SR.

of constrained optimization problem which is of great relevance for low-power electronic systems. We call this problem *policy optimization* (PO). Several heuristic power management policies have been investigated in the past [5, 8, 12] but no strong optimality result has been proved. In this paper we propose a stochastic model for the rigorous formulation of policy optimization and we describe a procedure for its *exact* solution. The procedure can be employed to explore the power vs. performance tradeoff curve. The global optimum solution of PO is computed in polynomial time by solving a linear optimization problem. This strong optimality result critically depends on the modeling assumptions. We assess the soundness of our assumptions by constructing the stochastic model for a real-life device (a disk drive) under a realistic workload. We then apply our optimization algorithm and compute optimal policies. The performance and power dissipation of the policies are then validated against simulation. Moreover, the optimal policies are compared with heuristic solutions.

The paper is organized as follows: in Section 2 we outline our stochastic model, starting from a qualitative description, then moving to a more rigorous mathematical formulation. The policy optimization problem is then formulated and a procedure for its solution is described. In Section 3 we carry out modeling and policy optimization for a realistic case study, namely a commercial disk drive with advanced power management support under workload conditions obtained from actual usage traces. In Section 4 we summarize our findings and outline future direction of research.

2 Stochastic model

We introduce a modeling approach that is aimed at providing support for system-level optimization of power-managed systems. The key features of our model are:

• **Generality**. Any power-managed electronic appliance (not only computer systems) can be modeled. Cellular phones, pagers, digital cameras are a few examples of portable appliances for which power is a major concern.

• **High level of abstraction**. The choice of abstraction level is probably the most important decision in system-level power modeling. In order to manage complexity, irrelevant (or marginally relevant) information should be abstracted. In our formulation, resources are described by abstract power and performance models.

• **Non-determinism**. The uncertainty on estimation caused by the abstraction process (and the consequent information loss) is mapped to the inherent uncertainty in the estimation of the expected value of random variables.

In the following we will consider a discrete-time (i.e. slotted time) setting. Thus, time is described by an infinite series of discrete values $t_n = Tn$, where T is the time resolution (or *period*), $n \in \mathbb{N}_+$. We model a system with a single user (or service requestor) whose requests are enqueued in a single queue and serviced by a single service provider. A power manager controls over time the behavior of the service provider (Figure 1). In more detail, the components are described in the following subsections.

A **SERVICE REQUESTOR** (SR). This unit sends requests to the Service Provider. The SR is modeled as a Markov Chain with transition matrix P_{SR}, where the observed variable is the number of requests s_r (with $s_r \in \{0, 1, \dots, S_r - 1\}$) sent to the SR during time interval t_n. We assume that the process and all its relevant parameters are known.

As a simple example, consider the SR with a maximum of one request per period (i.e., two states), as shown in the right hand side of Figure 1.

$$P_{SR} = \begin{array}{ccc} & 0 & 1 \\ 0 & 0.95 & 0.05 \\ 1 & 0.12 & 0.88 \end{array}$$

The example matrix models a "bursty" workload. There is a high probability (0.88) of receiving a request at time t_{n+1} if a request was received at time n, and the mean duration of a stream of requests is equal to $1/0.12 = 8.33$ periods.

A **SERVICE PROVIDER** (SP). The SP is a device which serves incoming requests from a workload source. In each time interval, it can be in only one *state*. Each state $s_p \in \{1, 2, \dots, S_p\}$ is characterized by a performance level and by a power consumption level (to be defined later). In the simplest example, we could have two states ($S_p = 2$): *on* and *off*. At each period, transitions between power states are controlled by a *power manager* (PM) through *commands* $a \in A = \{1, 2, \dots, N_a\}$. For example, we can define two simple commands: switch on (*s_on*) and switch off (*s_off*). When a specific command is issued, the SP will move to a new state in the next period with a fixed probability dependent only on the command a itself, and on the departure and arrival states. In other terms, after being given a transition command by the power manager, the SP can remain in its current state during the next period with a non-zero probability. This aspect of the model takes into account the uncertainty in the transition time between states caused by the abstraction of functional information. Our probabilistic model is equivalent to the assumption that the evolution in time of power states is modeled by a Markov process that depends on the commands issued by the PM. Thus, we define one transition matrix for each command a.

Back to our example with two states and two commands, we could have the following transition matrices:

$$P_{SP}(\textit{s\_on}) = \begin{array}{ccc} & \textit{on} & \textit{off} \\ \textit{on} & 1 & 0 \\ \textit{off} & 0.03 & 0.97 \end{array} \quad P_{SP}(\textit{s\_off}) = \begin{array}{ccc} & \textit{on} & \textit{off} \\ \textit{on} & 0.8 & 0.2 \\ \textit{off} & 0 & 1 \end{array}$$

The Markov chains corresponding to the matrices are pictorially represented in Figure 1. Note that the transition time from *off* to *on* when the *s_on* command has been issued is a geometric random variable with average equal to $1/0.03 = 33$ periods.

Each power state has a specific *power consumption rate c*. It is a function both of the state and the command performed on the state: $c(s_p, a)$. Dependency on the state is obvious: power dissipation depends on the operational state of the SP (for our example $c(\mathit{on}, a) \geq c(\mathit{off}, a)$). Dependence on the command issued is also essential: referring to our example, $c(\mathit{off}, \mathit{s\_on}) \geq c(\mathit{off}, \mathit{s\_off})$, because the activation process of the device requires additional energy consumption. Data on the power dissipation in various operating conditions is usually provided in the data-sheets of the SPs, or it can be directly measured.

A **QUEUE**. When service requests arrive during one period, they are buffered in a queue of length $(S_q - 1)$. In our treatment we will consider a FIFO discipline, although other disciplines can be modeled. The request is processed and serviced within the same period with a probability dependent on the power state of the SP. In this way we model the non-deterministic service time of a request as a geometric random variable, similarly to the exponential service time for the $G/M/1$ class in queueing theory [6]. It follows that also the queue length (denoted by s_q, with $0 \leq s_q \leq S_q$) is a Markov process with transition matrix $P_{SQ}(s_p, s_r)$. We refer again to our example: if $q(\mathit{on}) = 0.8$ and $q(\mathit{off}) = 0$, and the buffer contains at most one request, we have (on the bottom of Figure 1):

$$P_{SQ}(\mathit{on}, 0) = \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 1.0 & 0.0 \\ 0.8 & 0.2 \end{pmatrix} \end{array} \quad P_{SQ}(\mathit{on}, 1) = \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 0.8 & 0.2 \\ 0.8 & 0.2 \end{pmatrix} \end{array}$$

$$P_{SQ}(\mathit{off}, 0) = \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array} \quad P_{SQ}(\mathit{off}, 1) = \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{pmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}.$$

A **POWER MANAGER** (PM). This component communicates with the service provider and attempts to set its state at the beginning of each period, by issuing commands chosen among a finite set A. In our example, the commands can be *s_on*, and *s_off*. It contains all proper specifications and collects all relevant information (by observing SR and PM) needed for implementing a power management policy. The consumption of the power manager is assumed to be much smaller than the consumption of the subsystems it controls and it is not a concern here. The state of the system composed of the SP, the SR and the queue is then a triple $s = (s_r, s_p, s_q)$. Being the composition of three Markov chains, s will be a Markov chain (with $S = S_r \times S_p \times S_q$ states), whose transition matrix $P(a)$ depends on the command a issued to the SP by the PM. Hence, the system is fully described by a set of N_a transition matrices, one for each command.

In the above description no mention is made of the energy source (i.e., the battery). In this paper our goal is to minimize (or bound) the average power consumption of the SP, and not to maximize the expected battery life. This choice has several advantages: it allows us not to consider the details of the power source (path-dependent discharge characteristics, possible recharge "on the run"), while still retaining the primary feature of minimizing (or constraining) the consumption level. A second desirable feature of the model is that in this it can be used to independently maximize policies in the case of multiple SPs, and consider the average aggregate consumption rate as the only relevant quantity for dimensioning the power source.

Given our stochastic system model, we can now proceed to the formulation of the policy optimization problem. Our objectives are: i) to formally describe the behavior of the PM; ii) to find the particular behavior that optimally reduces power dissipation under performance constraints (or optimizes performance under power constraints). Hence, we need to provide formal definition for power and performance cost metrics as well.

DECISIONS. At the beginning of time period n, the PM observes the "history" H_n of the system (i.e., the sequence of states and commands up to $n-1$) and controls the Service Provider by taking a *decision* δ_n. A *deterministic decision* consists in taking a single action on the basis of the history of the system. Yet, we will consider the much broader set of *randomized decisions*: in this case, the Power Manager assign probabilities to every available command and then chooses the command to issue, according to this probability distribution. In this way, even if the same decision is taken in different periods, the actual commands issued could be different. Analytically, the decision $\delta(H_n)$ can be represented by a set of real numbers, $\{p_a(H_n) | 0 \leq p_a(H_n) \leq 1, \sum_a p_a(H_n) = 1\}$, where each p_a represents the probability of issuing command a given that the history of the system is H_n. A deterministic decision is the special case with $p_{a'} = 1$ for some command a'. The definition of $P(a)$ extends naturally to the case of a randomized decision δ, and we denote the transition matrix P_δ. We stress again the generality of such a class of decisions (randomized and history-dependent). Consider the example of the previous section. Suppose that the PM observes the following history: $s_1 = (0, \mathit{on}, 0)$, $s_2 = (1, \mathit{off}, 0)$ (states in period 0, 1), and $a(0) = \mathit{s\_off}$ (action taken at time 0). Then, a possible decision at time 1, when state s_2 is observed, could consist in assigning probabilities of performing commands *s_on* and *s_off* equal to $p(\mathit{s\_on}) = 0.44$, $p(\mathit{s\_off}) = 0.56$ respectively.

POLICIES. Over an finite time horizon (up to time interval n), the decisions taken by the PM are a finite discrete sequence $\delta_1, \delta_2, \ldots, \delta_n$. We call this sequence a *policy* π. The policy π is the independent variable of our optimization problem. If a policy $\pi = (\delta_1, \delta_2, \ldots, \delta_n)$ is adopted, we define $P_\pi^n := P_{\delta_1} P_{\delta_2} \cdots P_{\delta_n}$; this is simply the transition matrix from period 0 to period n under policy π. Among all policies the *stationary policies* play an important role: in this case δ does not depend on the entire history H_n. For stationary policies, decisions are a function of the current state of the system $s = (s_r, s_p, s_q)$ and the function does not change over time (otherwise, stationarity would be lost). A generic randomized stationary policy can be represented as a $S \times N_a$ matrix M_π. An element $m_{s,a}$ of M_π is the probability of issuing command a given that the state of the system is s.

COST METRICS. It is now possible to define the metrics of relevance in the PO problem. In their most general form, they are function both of the state s and of δ_s, i.e., the decision we take when we are in state s. An early example of cost metric is the *expected power consumption level* $c(s_p, \delta_s)$ per unit time, introduced earlier in this section, which represents the expected consumption per period when the SP is in state s_p and decision δ_s is taken. A second parameter of interest is the *performance penalty* per unit time $d(s_q)$ which depends on the queue length (number of jobs in the queue). A natural way to define the performance penalty is the queue length: $d(s) = s_q$. Finally, we consider the *request loss* per period $b(s_r, s_q)$. It takes value 1 when a request arrives ($s_r = 1$) and is forced to balk because the queue buffer is full ($s_q = S_q$); otherwise it is 0. For notational convenience, define the consumption, performance penalty and request loss vectors

$$c_\delta := \begin{pmatrix} c(1,\delta_1) \\ \vdots \\ c(S,\delta_S) \end{pmatrix} \quad d_\delta := \begin{pmatrix} d(1) \\ \vdots \\ d(S) \end{pmatrix} \quad b_\delta := \begin{pmatrix} b(1) \\ \vdots \\ b(S) \end{pmatrix}.$$

For example the s^{th} element of each vector is the expected value of the consumption rate (or performance penalty, or request loss) when the system is in state s and decision δ is taken. Analogous considerations apply to the other vectors. Using the above vectors, we can express the expected value of a cost metric in period n, when decision δ_n is taken, given that the system is in state s in period 0: it is given by the s^{th} element of the vector $P_\pi^n d_{\delta_n}$.

When performing policy optimization, we want to maximize the average performance level over a long time period while keeping the average consumption and request loss below some levels specified by the user. This can be formally expressed as follows:

$$\begin{aligned} \text{PO}: \min_\pi \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} P_\pi^n d_{\delta_n} \\ \text{s.t.} \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} P_\pi^n c_{\delta_n} \leq C_M \\ \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} P_\pi^n b_{\delta_n} \leq B_M \end{aligned}$$

the objective function is an S-dimensional vector. Its s^{th} element is the average (over N time periods) of the expected performance level per period, given that the system starts in state s in period 0, and C_M and B_M are the maximum expected consumption per period and the maximum expected request loss per period respectively (as specified by the user). Strictly speaking, we should solve S optimization problems [4], one for each initial state. Moreover, the optimization is carried the set of *all* possible policies. Hence, solving PO appears to be a formidable task. Fortunately, the this is not the case, thanks to the following result of general validity.

Theorem *[4, 7] When the constraints in PO are active and the Markov process is unichain (i.e., for each policy there is a single communicating class), there exists a unique optimal policy for all the optimization problems of PO, which is stationary, Markovian (and possibly randomized). Furthermore, if constraints are inactive, the policy is deterministic.*

The above theorem has three implications. First, and most importantly, under our modeling assumptions the optimal policies are stationary. This result rules out policies based on statistics taken in periods before the period during which the decision is taken. Second, a stationary policy can be described and implemented in a compact fashion, because no time dependence must be taken into account, and decisions at any time n can be taken only by observing the current state of the system. Third, the optimal policy can be efficiently computed. Several algorithms are available to solve PO when no constraint is active. For constrained problems, it is possible to solve PO by solving a linear programming problem, which is solvable in polynomial time in the number of states. Hordijk and Kallenberg [7], and references therein, describe a solution procedure. The solution procedure is not outlined here because of space limitation. It is however important to mention that the matrix M_π describing the optimum policy that solves PO can be extracted from the solution of the optimization problem in a straightforward fashion. If M_π is available, the decision procedure implemented by the PM is the following: i) observe the state s_n of the system (SP, SR and queue) at time n, and ii) select the command to be issued with probabilities given by the $s(n)^{th}$ row of matrix M_π. For instance, the optimum policy for the system in Figure 1 is represented by a 8 × 2 matrix M_π. Assume that the row of M_π corresponding to state $s = (on, 0, 0)$ is $m_{s,\cdot} = (0.12, 0.88)$. If the system is in state s, the PM will issue the *s_on* command with probability 0.12 or the *s_off* command with probability 0.88. Concluding the section, we want to stress the fundamental assumptions that allows us to compute optimal policies in polynomial time. First, both SR and SP must be modeled by Markov chains with known parameters; also the cost metrics must be known. Second, the time horizon (i.e., the number of time intervals over which the policy is valid) must be long enough to be a good approximation of an infinite horizon. These assumptions will be checked in the next section for a realistic case study.

State	ΔT	$Power$
active	NA	$2.5W$
idle	$1ms$	$1W$
idleLP	$40ms$	$0.8W$
standby	$2.2sec$	$0.3W$
sleep	$6sec$	$0.1W$

Table 1: State, transition time to active and power dissipation for a hard disk driver

3 Case Study

We tested the validity of our approach by calculating optimal policies for a commercially-available hard disk drive [13] and contrasting them against heuristic policies. The hard disk drive can be operated in 5 different energy states, as shown in Table 1. In four of the five states, the device cannot perform data reads or writes, hence they are all *inactive* states. The inactive states span the tradeoff between power consumption and time required to return to the fully operational *active* state.

More in detail, in the *idle* state the disk is spinning, but some of the electronic components of the drive are turned off. The transition from *idle* to *active* is extremely fast, but only marginal power is saved in the *idle* state. The *low-power idle* state is similar to the *idle* state, but it has decreased power dissipation (and increased transition time to the *active* state). In the *standby* and *sleep* state, the disk is spun down, hence the transition to the *active* state is not only slow, but it causes additional power consumption (the additional current absorbed by the motor to accelerate the disk). It is important to mention that the transition times of Table 1 are explicitly declared as *typical* in the data sheets. In other works, they can be interpreted as expected values of random variables. The simplified transition graph of the service provider that models the disk drive is shown in Figure 2 (a). Several transitions between inactive states (and a few transient states) have been omitted for the sake of readability. The figure shows only the transitions from and to the *active* state, which have a major impact on power and performance. Assume that the disk is in one of the inactive states and a file read or write request is received. The request must be enqueued (in our model, the queue has length two) because the SP is inactive and cannot serve it right away. The PM observes the state of the system

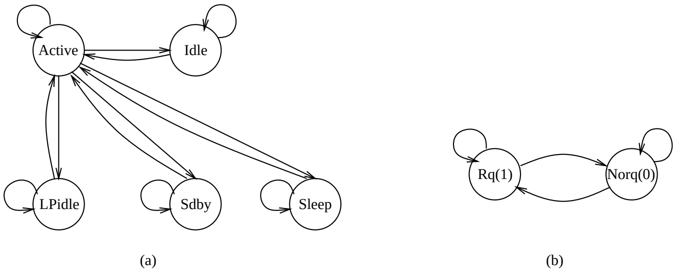

Figure 2: (a) Simplified transition graph for the disk drive. (b) Transition graph for the workload model

and issues a command for transitioning the resource to the active state.

The exit from the inactive states upon assertion of a command from the PM is not instantaneous. Our Markov model has been set up so that the expected exit time from an inactive state when a wakeup command has been issued is equal to the (experimental) average exit time from that inactive state (Table 1). Hence, the self-loops of the idle states have widely varying conditional proabilities. For example the conditional probability of the self-loop from *idle* when a wakeup command has been issued is much smaller than the conditional probability of the self-loop from *sleep* when the same command has been issued. The probabilities on the self-loops of the inactive states model the "inertia" of the system in waking up. Similarly, the self-loop on the active state models the inertia of the system in going to the inactive states. Its conditional probability depends on the command issued by the PM.

The model of the workload source can be automatically extracted from traces of disk accesses (we used the traces provided in [1]). The Markov model of the SR is extracted from usage traces with a modeling procedure that will be only briefly described because of space limitations. Given a time resolution T (proportional to the shortest wakeup time of the SP), the arrival times of the disk access requests are discretized (i.e., the trace is converted into a binary stream that has value 1 in position k if a request is received between time kT and time $(k+1)T$, zero otherwise). For the time resolution and the usage traces of this case, a Markov chain with two states as shown in Figure 2 (b) was sufficient to characterize the system. stream. At any time, the PM can issue a command to the SP. In our case study, the manager can chose among 5 commands: `GO_ACTIVE`, `GO_IDLE`, `GO_IDLELP`, `GO_STBY`, `GO_SLEEP`. The final product of the policy optimization is a matrix with five columns (one for each command) and as many rows as the number of states of the SP, SR and queue system. The complete model of the system (which is slightly more complex than the simplified version described here) has a total of 48 states. The power cost metric is obtained from the data sheets of the disk drive (summarized in Table 1). The performance metric is defined by assigning a cost to each state of the system where the SP is inactive and the queue is full. Roughly speaking, our metric represents a performance *penalty* which is negative and worsens as the average waiting time for a request increases. Finally the request loss metric is defined by assigning a cost to the state of the system where the queue is full and a new request arrives. Notice that request loss in our abstract models does not imply that a read or write request is actually lost (i.e., the system malfunctions). It simply represents an undesirable operating condition where the operating system controlling the drive is required to do extra work to guarantee correct service because the drive's response is very slow. The tool for policy optimization is built around an advanced LP solver based on an interior point algorithm [3]. We implemented an optimization shell that can compute trade-off curves between performance and power. Each point on the curve is a solution of a PO problem with different constraints (i.e., a *Pareto-efficient point*). Additionally, we implemented a simulation tool for validation of the PO results. The simulator can compute the power consumption, performance degradation and request loss of the system by simulating actual usage traces as they are processed by the SP controlled by the policy computed by the policy optimization tool. The cost metrics obtained by simulation should match the values computed by the optimizer. It is important to remember that the policies computed by the optimizer are global optimum ones. In other words, the only sources of error in our procedure come from the modeling process. Simulation is important to assess the quality of the modeling assumptions, and not the quality of the optimization. We tested the two fundamental modeling assumptions, namely the infinite-horizon assumption and the Markov assumption for the SR. Moreover, we compared the optimal policies with heuristic ones. The results of our experiments are shown in Figure 3(a).

The continuous line is the trade-off curve spanned by the optimal policies computed by the optimizer. Its computation took less than 1 minute on a SUN UltraSPARC workstation. The curve is obtained by maximizing performance with varying power constraint and fixed request loss (smaller than 50%). Notice how performance is traded off for power up to a minimum point where power cannot be reduced further because the constraint on request loss is violated. The circles in Figure 3(a) represent the results of simulation of the policies computed by the optimizer with the actual trace and for a finite time (10^6 time slots). The distance of the circles from the curve is a measure of the inaccuracies of the modeling process. It is visually obvious that the inaccuracies are very small, and that the simulated points lie almost perfectly on the tradeoff curve.

The triangles in Figure 3(a) represent heuristic solutions to the PO problem. The downwards triangles represent deterministic policies based on timeouts. Timeout-based policies are widely used for disk power management [5]. They are based on a simple heuristic that shuts down the disk when it has been inactive for a time longer than the timeout T_{to}. The choice of T_{to} is based on simulations and on designer's experience. The upwards triangles are randomized policies where the timeout value and the inactive state are chosen randomly with a given probability distribution. The randomized policies are the heuristic version of the optimal policies computed by our tool.

Although we cannot claim that our heuristic policies are the best that any experienced designer can formulate, some of our policies perform nearly as well as the optimum ones. The important point to notice is that it is much more difficult to control the power and performance produced by the heuristic policies. For example, we could not generate valid policies in the leftmost part of the power vs. performance plane, because the constraint on request loss was always violated. Moreover, it is possible to produce heuristic policies that produce "reasonable" results (such as those around power dissipation of $2.0W$), but there is no way for the designer to estimate if the results can be improved.

In Figure 3(b) we study the dependency of the policies by the workload statistics. The tradeoff curve computed with the Markov model of our usage trace is shown in continuous line. A new tradeoff curve computed with another SR model (i.e., a Markov chain with differ-

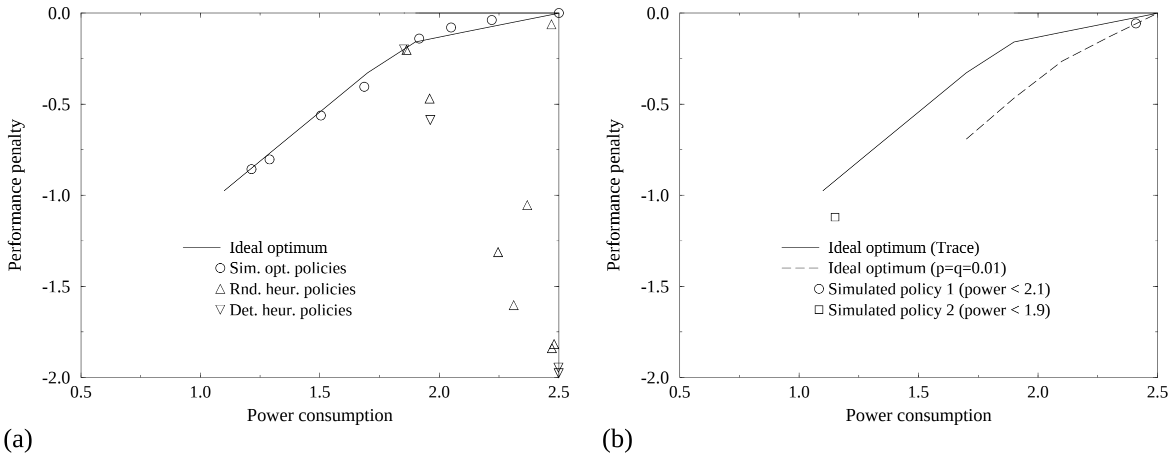

Figure 3: (a) Power consumption versus performance penalty for optimal and heuristic policies. (b) Dependency of optimal policies from SR model.

ent transition probabilities) is shown in dashed line. Clearly, the optimal policies are sensitive to the SR model. To further stress this point, we simulated two optimal policies computed with the second SR model by applying the original trace. The figure shows that both policies perform suboptimally (they are both below the continuous line). Moreover, they provide average power consumptions completely different from those used as optimization constraints. This experiment points out the need of an accurate SR model for obtaining high-quality policies.

4 Conclusion and future work

The identification of optimal power management policies for low-power system is a critical issue that has been addressed using common sense and heuristic solutions. In this work we provided a mathematical framework for the formulation and solution of the *policy optimization* problem. Our approach is based on a stochastic model of power-managed devices and workloads. The constrained policy optimization problem can be solved exactly in our modeling framework. Policy optimization can be cast into a linear programming problem and solved in polynomial time by efficient interior point algorithms. Moreover, tradeoff curves of power versus performance can be computed. The soundness of our modeling assumptions (and consequently the practicality of our power management policies) has been tested on a realistic case study. Our experimental results show that our stochastic model is robust and the optimal policies are flexible and power-efficient. Several extensions to this work are under way. First, we are planning to verify the quality of the results by implementing our power management policies in a portable PC and measuring their power and performance impact. Second, we are extending our model to deal with systems consisting of multiple interacting resources. Finally, we are studying adaptive algorithms that can compute optimal policies in systems where workloads are highly non-stationary and the service provider model changes over time.

Acknowledgements

This work was supported by NSF under contract MIP-9421129 and Toshiba Corp.

References

[1] Auspex File System Traces, available at *http://now.cs.berkeley.edu/Xfs/AuspexTraces/auspex.html* (1993).

[2] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*, Kluwer (1997).

[3] J. Czyzyk, S. Mehrotra, and S. Wright, "PCx User Guide", *Technical Report OTC 96/01*, Optimization Technology Center, May, 1996.

[4] C. Derman, *Finite State Markov Decision Chains*, Academic Press, (1970).

[5] R. Golding, P. Bosh et al, "Idleness is not sloth", in *Proceedings of Winter USENIX Technical Conference*, pp.201-212 (1995).

[6] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, Wiley (1985).

[7] A. Hordijk, and L. C. M. Kallenberg, "Constrained Undiscounted Stochastic Dynamic Programming", *Mathematics of Operations Research*, Vol. 2, pp. 276-289 (1984).

[8] C.-H. Hwang and A. C.-H. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", in *Proceedings of the ICCAD*, pp. 28-32 (1997).

[9] Intel, Microsoft and Toshiba, "Advanced Configuration and Power Interface specification", available at *http://www.intel.com/ial/powermgm/specs.html* (1996).

[10] Microsoft, "OnNow: the evolution of the PC platform", available at *http://www.microsoft.com/hwdev/pcfuture/ONNOW.HTM* (1997).

[11] W. Nebel and J. Mermet (Eds.), *Low power design in deep submicron electronics*, Kluwer (1997).

[12] M. Srivastava, A. Chandrakasan and R. Brodersen, "Predictive system shutdown and other architectural techniques or energy efficient programmable computation", *IEEE Transactions on Very Large Scale Integration Systems* vol. 4, no. 1, pp. 42-55, March 1996.

[13] Technical specifications of hard drive IBM Travelstar VP 2.5-inch, available at *http://www.storage.ibm.com/storage/oem/data/travvp.htm* (1996).

[14] 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, University of Pennsylvania* (1996).