# Policy Optimization for Dynamic Power Management

Luca Benini, *Member, IEEE*, Alessandro Bogliolo, *Member, IEEE*, Giuseppe A. Paleologo, *Student Member, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*— Dynamic power management schemes (also called policies) reduce the power consumption of complex electronic systems by trading off performance for power in a controlled fashion, taking system workload into account. In a power-managed system it is possible to set components into different states, each characterized by performance and power consumption levels. The main function of a power management policy is to decide when to perform component state transitions and which transition should be performed, depending on system history, workload, and performance constraints.**

**In the past, power management policies have been formulated heuristically. The main contribution of this paper is to introduce a finite-state, abstract system model for power-managed systems based on Markov decision processes. Under this model, the problem of finding policies that optimally tradeoff performance for power can be cast as a stochastic optimization problem and solved exactly and efficiently. The applicability and generality of the approach are assessed by formulating Markov model and optimizing power management policies for several systems.**

***Index Terms*—Energy conservation, energy management, optimization methods.**

## I. Introduction

BATTERY-OPERATED portable appliances impose tight constraints on the power dissipation of their components. Such constraints are becoming tighter as complexity and performance requirements are pushed forward by user demand. Reducing power dissipation is a design objective also for stationary equipment, because excessive power dissipation implies increased cost and noise for complex cooling systems. Numerous computer-aided design techniques for low power have been proposed [1]–[3] targeting digital very large scale integration (VLSI) circuits, i.e., chip-level designs.

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. To further complicate the picture, in most electronic products, digital components are responsible for only a fraction of the total power consumed. Analog, electromechanical, and optical components are often responsible for the largest contributions to the power budget. For example, the power breakdown for a well-known laptop computer [4] 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 noncritical 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. Many others 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. Designers have reacted to the new challenges posed by power-constrained design by mixing technological innovation and power-conscious architectural design and optimization.

One of the most successful techniques employed by designers at the system level is *dynamic power management* [8], [9]. 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.

To shorten the design cycle of complex power-managed systems, several hardware and software vendors [10], [11] are pursuing a long-term strategy to 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 simplifies the task of controlling the operating conditions of the system resources, but it does not provide insight on how and when to power manage them. We call *power management policy* (*policy* for brevity) a procedure that takes decisions upon the state of operation of system components and on the state of the system itself.

The most aggressive policy (that we call *eager* policy) turns off every system component as soon as it becomes idle.

Manuscript received May 29, 1998; revised November 18, 1998. This work was supported in part by the National Science Foundation (NSF) under Contract MIP-9421129 and by the ARPA under Contract DABT-63-95-C-0049. This paper was recommended by Associate Editor R. Camposano.

L. Benini and A. Bogliolo are with the Università di Bologna, DEIS, Bologna, Italy 30165 (e-mail: lbenini@deis.unibo.it).

G. A. Paleologo is with Stanford University, Department of Engineering-Economic Systems and Operations Research, Stanford, CA 94305-4023 USA.

G. De Micheli is with the Computer Systems Laboratory at Gates Computer Science, Stanford University, Stanford, CA 94305-4070 USA.

Publisher Item Identifier S 0278-0070(99)03971-8.

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 degrades performance and may not decrease power dissipation.

For instance, consider a device that dissipates 2 W in fully operational state and no power when set into inactive state. The transition from operational to inactive state is almost instantaneous (hence, it does not consume sizable power). However, the opposite transition takes 2 s. During the transition, the power consumption is 4 W. This device is a highly simplified model of a hard-disk drive (a more detailed model will be introduced later in this paper). Clearly, the eager policy does not produce any power savings if the device remains idle for less than 4 s. Moreover, even if the idle time is longer than 4 s, transitioning the device to inactive state degrades performance. If the eager policy is chosen, the user will experience a 2-s delay *every time* a request for the device is issued after an idle interval.

The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a 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 [12], [14], [15] but no strong optimality result has been proven.

In this paper we propose a stochastic model based on Markov decision processes [22] for the formulation of policy optimization and we describe a procedure for its *exact* solution. The solution of PO is computed in polynomial time by solving a linear optimization problem. We first describe the details and the fundamental properties of the stochastic model, then we show how to formulate and solve policy optimization. The *global optimality* of the solutions obtained is also proved. The procedure can be employed to explore the power versus performance tradeoff curve.

The class of the optimal policies is then studied in detail. We assess the sensitivity of policies to several system parameters. Our results provide insights for system architects designing power managed systems. Our model and optimization procedures can be used to help designers in difficult high-level decisions on how to choose or design components that can be power managed effectively.

Our analysis and our optimality result critically depends on our 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 II, we review related work in the field of dynamic power management. In Section III, we describe our stochastic model, starting from a qualitative description, then moving to a more rigorous mathematical formulation. The policy optimization problem is formulated in Section IV and a procedure for its solution is described. We implemented a tool for automatic power optimization. In Section V, we describe the tool implementation. Section VI is dedicated to the application of policy optimization to realistic case studies and to the analysis of the sensitivity of optimal policies to system parameters. Section VII presents a discussion on modeling issues, where we clarify the basic assumptions and the domain of applicability of our model. Finally, in Section VIII, we summarize our findings and outline future directions of research.

## II. Related Work

The fundamental premise for the applicability of power management schemes is that systems, or system components, experience nonuniform workloads during normal operation time. Nonuniform workloads are common in communication networks and in almost any thinkable interactive system. In the recent past, several researchers have realized the importance of power management for large classes of applications. Chip-level power management features have been implemented in mainstream commercial microprocessors [5]–[7]. Microprocessor power management has two main flavors. First, the entire chip can be shut down in several sleep states through external signals or software control. Second, chip units can be shut down by stopping their local clock distribution. This is done automatically by dedicated on-chip control logic, without user control. Techniques for the automatic synthesis of chip-level power management logic are surveyed in [8].

At a higher level of abstraction, energy-conscious communication protocols based on power management have been studied [16]–[20]. The main purpose of these protocols is to regulate the access of several communication devices to a shared medium trying to obtain maximum power efficiency for a given throughput requirement. Power efficiency is a stringent constraint for mobile communication devices. Pagers are probably the first example of mobile device for personal communication. In [20], communication protocols for pagers are surveyed. These protocols have been designed for maximum power efficiency. Protocol power efficiency is achieved by increasing the fraction of time in which a single pager is idle and can operate in a low-power sleep state without the risk of loosing messages.

With the widespread diffusion of advanced communication devices (cellular phones, portable wireless terminals, etc.) the bandwidth requirements for communication protocols have become much more stringent. More complex and higher-performance protocols are needed for controlling such advanced devices. In [16], a *star* communication network is studied, where several power-constrained devices communicate with each other through a base station that regulates traffic. The contribution of [16] is the formulation of a slot reservation strategy for the communicating devices and a scheduling algorithm for the base station that reduces power consumption while meeting service quality specifications.

The approaches presented in [18] and [19] are primarily focused on how to maximize the efficiency of a single power-constrained communication device operating in a noisy environment. Traditionally, communication devices have been designed to respond to increased noise levels by increasing transmission power and by repeating transmission. This strategy is highly energy-inefficient and can be counterproductive even throughput-wise if decreased transmission quality is caused by interference from other transmitters operating with the same protocol. Both [18] and [19] assume that the worst menace to service quality is mutual interference and propose retransmission protocols that tend to reduce mutual interferences by reducing the average transmission power and by increasing silence time when error rate is high.

Power management schemes have also been studied in [12], [14], and [15]. The system, or a component, is modeled as a *reactive* system that receives requests from the external environment and performs some computational task in response to a request. The arrival rate of incoming requests is not uniform over time, nor it is so high to impose full utilization. Hence, power can be saved by transitioning the system to a sleep state when it is not in use. The power-down strategy impacts performance both in terms of latency and throughput, because of transition delays. The approaches presented in [12], [14], and [15] explore several shutdown policies that minimize power at the cost of a marginal performance reduction.

Disk driver subsystems are studied in [12] and [13]. This work presents an extensive study of the performance of various disk spin-down policies. The problem of deciding when to spin down a hard disk to reduce its power dissipation is presented as a variation of the general problem of predicting idleness for a system or a system component. This problem has been extensively studied in the past by computer architects and operating system designers (the paper by Golding *et al.* [13] contains numerous references on the topic), because idleness prediction can be exploited to optimize performance (for instance by exploiting long idle period to perform work that will probably be useful in the future). When low power dissipation is the target, idleness prediction is employed to decide when it is convenient to spin down a disk to save power (if a long idle period is predicted), and to decide when to turn it on (if the predictor estimates that the end of the idle period is approaching).

The studies presented in [14] and [15] target interactive devices. A common assumption in these works is that future workloads can be predicted by examining the past history. The prediction results can then be used to decide when and how transitioning the system to a sleep state. In [14], the distribution of idle and busy periods for an interactive terminal is represented as a time series, and approximated with a least-squares regression model. The regression model is used for predicting the duration of future idle periods. A simplified power management policy is also introduced, that predicts the duration of an idle period based on the duration of the last activity period. The authors of [14] claim that the simple policy performs almost as well as the complex regression model, and it is much easier to implement. In [15], an improvement over the prediction algorithm of [14] is presented, where idleness prediction is based on a weighted sum of the duration of past idle periods, with geometrically decaying weights. The policy is augmented by a technique that reduces the likelihood of multiple mispredictions.

A common feature of all previous works in the area of power management is that policies are formulated heuristically, then tested with simulations or measurements to assess their effectiveness. Another interesting commonality is that the highly abstract models used to represent the target systems necessarily imply some uncertainty. Uncertainty is caused by abstraction (for instance system response time is uncertain because detailed functionality is abstracted away), and by nondeterminism (for instance, request arrival times are uncertain because they are not controlled by the system).

Probabilistic techniques and models are employed by all previous approaches to deal with uncertainty. Similarly to previous approaches, we will formulate a probabilistic system model, but differently from previously published results, we will rigorously formulate the policy optimization problem within the framework provided by our model, and we will show that it can be solved exactly and in polynomial time in the size of the system model. To obtain this result, we leverage well-known stochastic optimization techniques based on the theory of Markov processes. A vast literature is available on this topic, and the interested reader is referred one of the numerous textbooks for detailed information (see, for instance, [21]–[23]).

## III. Stochastic Model

In this section we first informally describe a system model, then we provide definitions and we analyze the properties of the model. We consider a system embedded in an environment modeled as a single source of requests. Requests issued by the event source are serviced by the system. The system itself consists of two components: a resource that processes requests (the *service provider*), and a *power manager*.

The resource has several states of operation. Each state is characterized by a service rate, which is, roughly speaking, proportional to the average number of requests serviced in a time unit. Some states may have zero service rate. Such states are called *sleep states*, while states with nonnull service rate are called *active states*. Both request arrivals and services are stochastic processes, in other words, service times and interarrival times between requests are nondeterministic. As explained in Section II, nondeterminism models incomplete information and/or uncertainty caused by the high level of abstraction of the model.

The system may contain a *queue* which stores requests that cannot be immediately serviced upon arrival because the service provider is either busy servicing other requests or it has zero service rate. We assume that requests are indistinguishable, hence, service priorities are immaterial. Moreover we assume that the traffic-management component has finite capacity. Whenever the number of enqueued requests exceeds the capacity, requests are lost. Request loss does not model actual lack of service in the system. In our abstract model, request loss represents an undesirable condition that is verified when too many requests are waiting to be serviced. Real-life systems generally implement congestion-control mechanisms based on synchronization primitives that prevent overflowing of internal queues. We do not accurately model such mechanisms because we focus on average-case operating conditions. However we

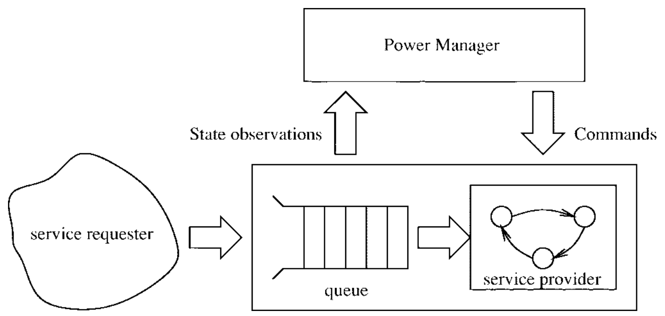


Fig. 1. Components of the system model.

model overflow of normal system capacity because it is undesirable and should be avoided as much as possible.

The power manager is a controller that observes the history of the service provider and of the queue and issues commands. There is a finite number of commands, and their purpose is to cause the transition of the service provider from one state to another. The service provider responds to commands in a nondeterministic fashion. In other words, there is no guarantee that the service provider changes state as soon as a command is issued, but there is a probability that the transition will be performed in the future. Nondeterminism represents the delay of the system in responding to commands and the uncertainty on the actual value of such delay caused by the high abstraction level of the model. The criterion used for choosing what command to issue and when is called *policy*.

The overall system architecture is depicted in Fig. 1. Our goal is to search the space of all possible policies to find the one that minimizes a cost metric. We define two cost metrics: *power* and *performance*. Policy optimization targets the optimization of one cost metric while using the second as a constraint. In Sections III-A and III-B, we formulate a stochastic system model based on Markov chains. Within this model, policy optimization can be rigorously formulated and solved. However, we do not discuss how and when the model is a valid abstraction of a real-life system. This important issue is analyzed in detail in Sections VI and VII.

## *A. System Components*

We assume that the reader is familiar with basic probability theory at the level of [25] and [26]. We use uppercase bold letters (e.g., $\mathbf{M}$) to denote matrices, lowercase bold letters (e.g., $\mathbf{v}$) to denote vectors, calligraphic letters (e.g., $\mathcal{S}$) to denote sets, uppercase italicized letters (e.g., $S$) to denote scalar constants and lowercase italicized letters (e.g., $x$) to denote scalar variables. We will consider a discrete-time (i.e., slotted time) setting, $t_n = Tn$, where $T$ is the time resolution, $n \in \mathbb{N}_+$. We will write $x_n$ in place of $x_{t_n}$. We call *time slice* the time interval between two consecutive values of $t_n$.

A **stationary Markov chain** $\mathcal{M}$ is a stochastic process over a finite state set $\mathcal{S} = \{s_i$, s.t. $i = 1, 2, \cdots, S\}$ whose behavior is such that, at any time $t_n$, the state probability distribution depends only on the state at time $t_{n-1}$. Prob$(x_n = s_j | x_{n-1} = s_i) = p_{s_i, s_j}$ is called *one-step transition probability*. The one-step transition probabilities are conveniently specified in the form of a *transition probability matrix* $\mathbf{P}$, $0 \le p_{s_i, s_j} \le 1$ and $\sum_{s \in S} p_{s_i, s} = 1$. A Markov chain can also be described by its *state-transition diagram*, a directed graph whose nodes are states, and whose edges are labeled with conditional transition probabilities. State transition times in Markov chains have *geometric distribution*

$$\text{Prob}(t_{s_i, s_j} = nT) = p_{s_i, s_j} p_{s_i, s_i}^{n-1}. \tag{1}$$

A **stationary controllable Markov chain** $\mathcal{M}(a)$ is a Markov chain whose transition probabilities $p_{s_i, s_j}$ are functions of controlling variable $a$. When the independent variable $a$ can take values in a finite set $\mathcal{A}$, the transition probabilities are $p_{s_i, s_j}(a)$: $\mathcal{A} \to [0, 1]$, and the controllable Markov chain can be represented by a set of matrices, one for each value of the independent variable $a \in \mathcal{A}$.

We first define a *command set* $\mathcal{A} = \{a_i$, s.t. $i = 1, 2, \cdots, A\}$. The elements of $\mathcal{A}$ are commands issued by the power manager for controlling the operation of the system.

*Definition 3.1:* A *service provider* (SP) is described by a triple $(\mathcal{M}_{\text{SP}}(a), b(s, a), c(s, a))$ where: i) $\mathcal{M}_{\text{SP}}(a)$ is a stationary, controlled Markov process with state set $\mathcal{S} = \{s_i$ s.t. $i = 1, 2, \cdots, S\}$, control set $\mathcal{A}$ and stochastic matrix $\mathbf{P}^{\text{SP}}(a)$; ii) $b(s, a)$ is a function $b$: $\mathcal{S} \times \mathcal{A} \to [0, 1]$; and iii) $c(s, a)$ is a function $c$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

The SP model is a discrete-time controllable Markov chain and matrix $\mathbf{P}^{\text{SP}}(a)$ is its conditional probability matrix. A *service rate* $b(s, a)$ is associated with each state $s \in \mathcal{S}$ and command $a \in \mathcal{A}$, it represents the probability of completing the service of a request in a time slice, given that SP is in state $s$ and that command $a$ has been issued at the beginning of the time slice. A *power consumption* measure $c(s, a)$ is associated with each state $s \in \mathcal{S}$ and command $a \in \mathcal{A}$. It represents the power consumption of the SP in a time slice, given that command $a$ has been issued and the SP is in state $s$. In each time slice, the service provider can be in only one state. The power manager causes state transitions by issuing commands. However, the response to a command is nondeterministic: the SP may or may not transition to a new state. Clearly, it is possible to model deterministic transitions by specifying a conditional probability value equal to one. In the general case, a command needs to be asserted over several time steps to induce the desired transition. If we assume that the asserted command does not change, the probability that the SP performs the transition increases geometrically with the number of time slices. Thus, the transition time $t_{s_i, s_j}(a)$ has expected value

$$\begin{aligned} \bar{t}_{s_i, s_j}(a) &= T \sum_{k=1}^{\infty} k(1 - p^{\text{SP}}_{s_i, s_j}(a))^{(k-1)} p^{\text{SP}}_{s_i, s_j}(a) \\ &= \frac{T}{p^{\text{SP}}_{s_i, s_j}(a)}. \end{aligned} \tag{2}$$

The value of $\bar{t}_{s_i, s_j}(a)$ is the average time for transitioning from state $s_i$ to state $s_j$, given that the command $a$ is issued at every $t_n$ until the transition is performed.

Each pair $(s, a)$ is characterized by a performance $b(s, a)$ and a power consumption $c(s, a)$. Performance is expressed in terms of service rate, which is the probability of completing a request in a time slice, hence, the value of $b$ is $0 \le b \le 1$. Zero service rate means that no requests can be serviced and the SP is not active. Service rate $b = 1$ means that a request

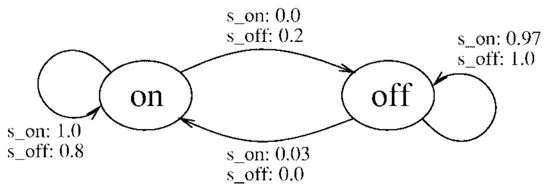


Fig. 2. Markov chain model of the service provider.

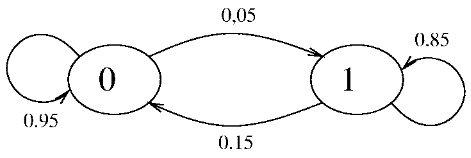


Fig. 3. Markov chain model of the service requester.

is certainly serviced in each time slice. Function $c$ is a general real-valued function that expresses the power consumption in arbitrary units (say Watts). The definitions of $b$ and $c$ are the basis for the computation of the cost metrics employed to evaluate the quality of a policy.

*Example 3.1:* Consider a SP with two states, $\mathcal{S} = \{\text{on, off}\}$. Assume that two commands are defined $\mathcal{A} = \{s\_\text{on}, s\_\text{off}\}$, with the intuitive meaning of "*switch on*" and "*switch off*," respectively. When a command is issued, the SP will move to a new state in the next period with a probability dependent only on the command $a$, and on the departure and arrival states. The stochastic matrix $\mathbf{P}^{\text{SP}}(a)$ can be represented by two matrices, one for each command. For example

$$\mathbf{P}^{\text{SP}}(s\_\text{on}) = \begin{array}{c|cc} & \text{on} & \text{off} \\ \text{on} & 1 & 0 \\ \text{off} & 0.1 & 0.9 \end{array}$$

$$\mathbf{P}^{\text{SP}}(s\_\text{off}) = \begin{array}{c|cc} & \text{on} & \text{off} \\ \text{on} & 0.8 & 0.2 \\ \text{off} & 0 & 1 \end{array}.$$

The Markov chain model of the SP is pictorially represented in Fig. 2. 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 1/0.1 = 10 periods.

Service rate $b(s, a)$ and power consumption $c(s, a)$ can be represented by two-dimensional tables with one entry for each state-command pair. For instance

$$b(a, s) = \begin{array}{c|cc} & s\_\text{on} & s\_\text{off} \\ \text{on} & 0.8 & 0 \\ \text{off} & 0 & 0 \end{array}$$

$$c(a, s) = \begin{array}{c|cc} & s\_\text{on} & s\_\text{off} \\ \text{on} & 3 & 4 \\ \text{off} & 4 & 0 \end{array}.$$

In this example, the SP is active only when it is in the on state and it is not being switched off. Power dissipation is null in the off state, but switching the resource on or off has a sizable power cost: the power consumption of the SP during the switching times (i.e., when the state is on and the command is $s_$off, or when the state is off and the command is $s_$on) is higher than that of the active state.

*Definition 3.2:* A *service requester* (SR) is described by a pair $(\mathcal{M}_{\text{SR}}, z(r))$ where: i) $\mathcal{M}_{\text{SR}}$ is a Markov process with state set $\mathcal{R} = \{r_i \text{ s.t. } i = 0, 1, \cdots, (R-1)\}$ and stochastic matrix $\mathbf{P}^{\text{SR}}$ and ii) $z(r)$ is a function $z$: $\mathcal{R} \rightarrow \mathbb{N}$.

The service requester models the system's environment as a Markov chain with $R$ states and transition probability matrix $\mathbf{P}^{\text{SR}}$. The function $z(r)$ represents the number of requests issued per time slice by the service requester when it is in state $r$. Intuitively, SR states represent traffic conditions, and the value $z(r)$ gives a quantitative measure of the traffic generated in each condition. For instance, if $z(r) = 0$, state $r$ represents an environmental condition where no requests are generated. The Markov process of request generation is completely autonomous and it does not depend on the behavior of the system: it represents the external environment over which the system has no control. Interarrival times have a geometric, memoryless distribution.

*Example 3.2:* Consider a SR with two states, $r_0$ and $r_1$, where function $z(r)$ is defined as follows: $z(r_0) = 0$, $z(r_1) = 1$. Since there is a one-to-one correspondence between values of $z$ and SR states, we will use the values of $z$ as names for the states ($r_0$ will be called 0, and $r_1$ will be called 1). At any time $t_n$ only two possibilities are given: either a single request or no request is received. An example of a stochastic matrix of SR is

$$\mathbf{P}^{\text{SR}} = \begin{array}{c|cc} & 0 & 1 \\ 0 & 0.95 & 0.05 \\ 1 & 0.15 & 0.85 \end{array}.$$

The Markov chain of the SR is shown in Fig. 3. The SR models a "bursty" workload. There is a high probability (0.85) of receiving a request during period $n+1$ if a request was received during period $n$, and the mean duration of a stream of requests is equal to 1/0.15 = 6.67 periods.

Remember that, although we have discussed examples of two-state SR models, the number of states of the model can be larger than two, and function $z(r)$ can take arbitrary integer values.

*Definition 3.3:* A *service queue* is described by a stationary controllable Markov chain $\mathcal{M}_{\text{SQ}}(a, s, r)$ with state set $\mathcal{Q} = \{q_i \text{ s.t. } i = 0, 1, \cdots, (Q-1)\}$, control set $\mathcal{A} \times \mathcal{S} \times \mathcal{R}$ and stochastic matrix $\mathbf{P}^{\text{SQ}}(a, s, r)$.

When service requests arrive during one period, they are buffered in a queue of length $(Q-1)$. The queue is in state $q_i$ when $i$ requests are waiting to be serviced. The queue is bounded: if new requests arrive when its state is $q_{Q-1}$, the state does not change (in this case we say that requests are *lost*). We call $q_{Q-1}$ the *queue full* state, and $q_0$ the *queue empty* state. The conditional probabilities of the SQ $p^{\text{SQ}}_{q_i, q_j}$ are completely determined by the other system components. The SP controls how fast the queue is emptied, while the SR controls how fast the queue is filled. Given the triple $(a, s, r)$ we know $b(a, s)$ (the service rate) and the number of request arrivals $z(r)$. The probability of servicing an enqueued request (or an incoming one) is $b(a, s)$, while the probability that no requests are serviced is $1 - b(a, s)$. States $q_0$ (queue empty) and $q_{Q-1}$ (queue full) are corner cases. If $q = q_0$ and $z(r) = 0$ (i.e., no arrivals), then the state will remain $q_0$ with probability 1. If the queue is full, its state will change with probability $b(a, s)$ if $z(r) = 0$, while it will remain $q_{Q-1}$ with probability 1 if $z(r) > 0$. One last corner case is verified when the number of arrivals $z(r)$ is large enough to exceed the maximum queue length, i.e., if the queue state is $q_i$ and $i + z(r) \geq Q$. In this case the new queue state will be $q_{Q-1}$ (queue full) with probability 1. Whenever an incoming request cannot be enqueued because the queue is full, we say that we have a *request loss*. The transition probabilities outside corner cases can be expressed as follows:

$$p^{\mathrm{SQ}}_{q_i, q_j}(a, s, r) = \begin{cases} 1 - b(a, s), & \text{if } j = i + z(r), \text{ and} \\ & \quad 0 < i + z(r) < Q, \\ b(a, s), & \text{if } j = i + z(r) - 1, \text{ and} \\ & \quad 0 < i + z(r) < Q + 1 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Although the formal description of matrix $\mathbf{P}^{\mathrm{SQ}}(a, s, r)$ may seem complex, its construction is intuitive, and can be best clarified through an example.

*Example 3.3:* Consider the SP and SR models introduced in the previous examples, and assume a maximum queue length of one, thus $\mathcal{Q} = \{q_0, q_1\}$. The SR has two states, the SP has two states and two commands can be issued. Hence $\mathcal{A} \times \mathcal{S} \times \mathcal{R}$ contains eight triples. Matrix $\mathbf{P}^{\mathrm{SQ}}(a, s, r)$ can be described by eight $2 \times 2$ matrices, one for each value of the triple $(a, s, r)$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{on}, \mathrm{on}, 0) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1.0 & 0.0 \\ 0.8 & 0.2 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{on}, \mathrm{on}, 1) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 0.8 & 0.2 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{on}, \mathrm{off}, 0) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{on}, \mathrm{off}, 1) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{off}, \mathrm{on}, 0) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{off}, \mathrm{on}, 1) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{off}, \mathrm{off}, 0) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}$$

$$\mathbf{P}^{\mathrm{SQ}}(s\_\mathrm{off}, \mathrm{off}, 1) = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 0.0 & 1.0 \\ 0.0 & 1.0 \end{pmatrix} \end{array}.$$

Notice that the SP is actively servicing requests with rate 0.8 only when its state is on and the command issued is $s$_on (this situation corresponds to the two top matrices). In all other combinations of command and state the service rate is zero. When there are no requests the queue states does not change, i.e., $\mathbf{P}^{\mathrm{SQ}}$ is the identity matrix. If there are incoming requests, the queue can only be filled. For instance, consider the matrix for state ($s$_on, off, 1): the SP is off, the command is $s$_on and a request arrives. If the queue was empty, it is filled with probability one. If the queue was already full, it remains full and we have a request loss.

*Definition 3.4:* A *power manager* (PM) is a control procedure that issues a command $a \in \mathcal{A}$ to the SP every time period $t_n$. The decision on which command to issue is based on the observation of the system history $H_n$ up to $t_n$, where $H_n \in (\mathcal{S} \times \mathcal{R} \times \mathcal{Q})^n$.

The flow of state information and command between the PM and the system is depicted in Fig. 1. To understand the implications of the PM definition, observe that the set $\mathcal{H}_n = (\mathcal{S} \times \mathcal{R} \times \mathcal{Q})^n$ contains all possible sequences of $n$ triples $(s, r, q)$. A generic element $H_n \in \mathcal{H}_n$ represents the entire system history for $n$ time slices (from time 1 to time $t_n$): $H_n = ((s_1, r_1, q_1), (s_2, r_2, q_2), \cdots, (s_n, r_n, q_n))$. In other words, $\mathcal{H}_n$ contains all possible system trajectories from time $t_1$ to time $t_n$.

Definition 3.4 fully characterizes a very general class of decision procedures (policies) for the choice of the commands to be issued during system operation. At each time $t_n$, given $H_n$, the power manager issues command $a \in \mathcal{A}$. A policy can be *deterministic* or *randomized*. If the policy is deterministic, the knowledge of system history at a given time $t_n$ uniquely determines the command $a \in \mathcal{A}$ that must be issued, i.e., the policy is a function $\mathcal{H}_n \rightarrow \mathcal{A}$ for all $n$. In contrast, if the policy is randomized, the command than can be issued for a given history is *not* uniquely determined by the history. Rather, $H_n$ uniquely determines a *probability distribution* of the commands. In other words, given history $H_n$ at time $t_n$, the probability of issuing command $a$ is uniquely defined. The actual command to be issued is obtained by randomly selecting a command from $\mathcal{A}$ with the given probability distribution. We will formalize these concepts in Section IV.

Notice that the policy (deterministic or randomized) can be arbitrarily complex, since at each step it takes into account the entire system history. In the next subsection we will introduce several classes of policies and cost metrics to evaluate them. We assume that the implementation of the power manager consumes negligible power (compared to the power consumption of the SP).

*Example 3.4:* In the previous examples we have assumed that $\mathcal{A} = \{s\_\mathrm{on}, s\_\mathrm{off}\}$. Hence, the power manager can, at any time $t_n$, issue either command $s$_on or command $s$_off, depending on the policy adopted. A trivial example of deterministic policy is the *constant policy* where always the same command is issued. A more realistic deterministic policy observes the status of the queue. If there is at least a request waiting, it issues the $s$_on command, otherwise it issues the $s$_off command. We call this policy "*eager*," because it attempts to turn off the SP as soon as it is idle. Finally, an example of randomized policy is the following: at any given time $t_n$, $n > 0$, the last three system states are observed. If the system has been in state (on, 0, 0) for the last three time slices, then the command $s$_off is issued with probability 0.6.

Having defined all system components, we can now provide a formal definition for the system as a single entity. The system can be seen as the composition of the Markov chains of service provider, service requester, and queue. Thus, it is a controlled Markov chain whose state is the concatenation of the states of SP, SR, and SQ, i.e., a triple $x = (s, r, q)$. The state set is $\mathcal{X} = \mathcal{S} \times \mathcal{R} \times \mathcal{Q}$ with cardinality $X = S \cdot R \cdot Q$. The system's stochastic matrix is $\mathbf{P}(a)$, a $X \times X$ matrix of functions of command $a \in \mathcal{A}$, that can be represented by a set of $A$ stochastic matrices, one for each command. The generic element $p_{x_i, x_j}(a)$ of matrix $\mathbf{P}(a)$ has the form

$$\begin{aligned} p_{x_i, x_j}(a) &= \text{Prob}(x_j = (s', r', q') | x_i = (s, r, q), a) \\ &= p^{\text{SP}}_{s, s'} \cdot p^{\text{SR}}_{r, r'} \cdot p^{\text{SQ}}_{q, q'} \\ &= \begin{cases} p^{\text{SP}}_{s, s'}(a) \cdot p^{\text{SR}}_{r, r'} \cdot b(s, a), & \text{if } q' = q + z(r) - 1 \\ p^{\text{SP}}_{s, s'}(a) \cdot p^{\text{SR}}_{r, r'} \cdot (1 - b(s, a)), & \text{if } q' = q + z(r) \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{4}$$

The first case represents transitions occurring when there is at least an enqueued request and the SP services it. The second case represents transitions occurring when the SR does not complete a service request. Finally, some state transitions are simply not possible, hence, they have zero probability. This expression does not account for corner cases (i.e., when the queue is empty and when the queue is full). Formal expressions for $p_{x_i, x_j}(a)$ in corner cases are obtained with similar reasoning as for the state of the queue. We do not analyze corner cases in detail because they do not add much to the understanding of the following material. The construction of the system's Markov chain can be clarified through an example.

*Example 3.5:* Consider the system with two SR states, two SP states (with bursty behavior), two commands and queue length two. The Markov chain of the system has eight states and can be represented by two $8 \times 8$ stochastic matrices, one for each command. We represent the values of the transition probabilities in Fig. 4 on the corresponding edges of the state graph of the Markov chain. Each edge is labeled with two transition probabilities, one for each command. Only a few edges are shown for the sake of illustration.

Consider the transition between state $x_1 =$ (on, 0, 0) (i.e., active SP, no requests, empty queue) and state $x_3 =$ (on, 1, 0) (i.e., one request comes and it is serviced right away). If the command issued by the PM is $s$_off the transition has null conditional probability because the SP cannot service requests when the s_off command has been issued [service rate $b(\text{on}, s\text{\_off}) = 0$]. If the command is s_on, the conditional probability of the transition is

$$p_{(\text{on}, 0, 0), (\text{on}, 1, 0)} = p^{\text{SP}}_{\text{on, on}}(s\text{\_on}) \cdot p^{\text{SR}}_{0, 1} \cdot b(\text{on}, s\text{\_on})$$

where $p^{\text{SR}}_{0, 1} = 0.05$ is the probability of an incoming service request given that no requests were issued in the previous time slice, $b(\text{on}, s\text{\_on}) = 0.8$ is the service rate of the SP when it is on and the $s$_on command has been issued, and $p^{\text{SP}}_{\text{on, on}}(s\text{\_on}) = 1$ is the probability for the SP to remain on if the $s$_on command has been issued.

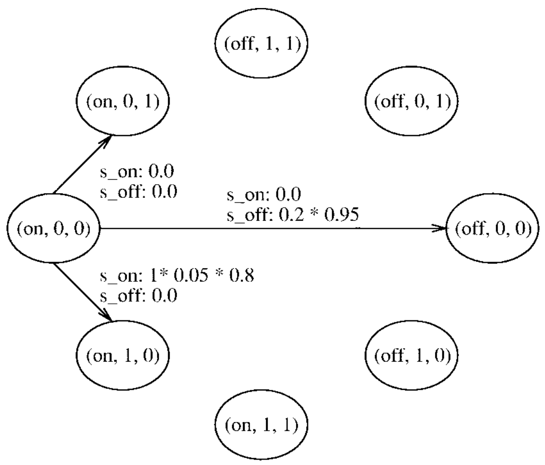


Fig. 4. Fragment of the Markov chain of the entire system. Only three state transitions are shown.

Notice again that the transition time (with fixed command) between any pair of states $x_i$, $x_j$ is a random variable with geometric distribution and average value $T/p_{x_i, x_j}(a)$. Concluding this subsection, we want to stress the key property of the Markov model: at every time $t_n$, the future evolution of the system depends only on the current state $x$ and the command issued. Our approach to the exact and efficient solution of the policy optimization problem relies upon this fundamental property.

## B. Cost Metrics

In Section III-A, we defined a very general class of PM policies (Definition 3.4). The purpose of policy optimization is to search the space of all policies and to find optimal ones. We will show that the optimal policy lies within a specific class whose representation is very compact and whose implementation is straightforward. We now define some important concepts that will be useful in formulating the policy optimization problem and its solution.

*Definition 3.5:* A decision $\delta(H_n) = \{p_a(H_n) \text{ s.t. } a \in \mathcal{A}\}$ at time $t_n$ is a set of functions $p_a$: $\mathcal{H}_n \rightarrow [0, 1]$ with the property $\sum_{p_a \in \delta} p_a(H_n) = 1$.

In simple terms, given a system history $H_n$, a decision $\delta(H_n)$ is a discrete probability distribution that associates a probability value $p_a(H_n)$ with each command $a \in \mathcal{A}$. At the beginning of time slice $n$, the power manager observes the history $H_n$ of the system and controls the SP by issuing a command $a$ with probability $p_a(H_n)$.

In the following, we use the shorthand notation $\delta^{(n)} = \delta(H_n)$. A *deterministic decision* consists of taking a single action with probability 1 on the basis of the history of the system [only one $p_a(H_n)$ is equal to one, all others are zero]. Deterministic decisions are just a limiting case for the more general *randomized decisions* described by Definition 3.5. Notice that even if the same randomized decision is taken in different periods, the actual commands issued could be different.

Remember that the transition matrix of the system $\mathbf{P}(a)$ is a function of the command $a$. Given a (randomized) decision $\delta^{(n)}$, we use the notation $\mathbf{P}_{\delta^{(n)}}$ to denote the transition matrix of the system given decision $\delta^{(n)}$

$$\mathbf{P}_{\delta^{(n)}} = \sum_{p_a \in \delta^{(n)}} p_a \mathbf{P}(a). \tag{5}$$

$\mathbf{P}_{\delta^{(n)}}$ is the weighted sum of the $\mathbf{P}(a)$ weighted with the probabilities $p_a$ assigned to the commands by the decision.

*Example 3.6:* In our example system, we defined two commands: $s\_\mathrm{on}$ and $s\_\mathrm{off}$. Assume that a decision $\delta^{(n)} = \{p_{s\_\mathrm{on}}, p_{s\_\mathrm{off}}\}$ at $t_n$, is given, with $p_{s\_\mathrm{on}} = 0.8$ and $p_{s\_\mathrm{off}} = 0.2$. This means that there is 80% probability of issuing command $s\_\mathrm{on}$ and 20% probability of issuing command $s\_\mathrm{off}$. The transition matrix $\mathbf{P}_{\delta^{(n)}}$ of the system, given decision $\delta^{(n)}$ is

$$\mathbf{P}_{\delta^{(n)}} = 0.8 \cdot \mathbf{P}(s\_\mathrm{on}) + 0.2 \cdot \mathbf{P}(s\_\mathrm{off}).$$

Consider an infinite sequence of time slices $[1, 2, \cdots)$. The decisions taken by the PM are a discrete sequence $[\delta^{(1)}, \delta^{(2)}, \cdots)$. The sequence completely describes the PM *policy* $\pi$ which is the unknown of our optimization problem. If a policy $\pi = [\delta^{(1)}, \delta^{(2)}, \cdots)$ is adopted, we define $\mathbf{P}_\pi^n = \mathbf{P}_{\delta^{(1)}}\mathbf{P}_{\delta^{(2)}}, \cdots, \mathbf{P}_{\delta^{(n)}}$; this is simply the transition matrix from period 0 to period $n$ under policy $\pi$. Among all policies some classes are particularly relevant, as defined next.

*Definition 3.6:* Stationary policies are policies where the same decision $\delta^{(n)} = \delta$ is taken at every time $t_i$, $i = 1, 2, \cdots$, i.e., $\pi = [\delta, \delta, \cdots)$.

For stationary policies, decisions are denoted by $\delta$, which is a function of the system state $x$. Thus, stationarity means that the *functional dependency* of $\delta$ on $x$ does not change over time. When $x$ changes, however, $\delta$ can change. Furthermore, notice that even a constant decision *does not* mean that the *same command* is issued at every period. A decision is a probability distribution that assigns a probability to each command $a \in \mathcal{A}$. Thus, the actual command that is issued is obtained by randomly selecting from $\mathcal{A}$ with the probabilities specified by $\delta$.

*Definition 3.7:* Markov stationary policies are policies where decisions $\delta$ do not depend on the entire history $H_n$, but only on the state of the system $x = (s, r, q)$ at time $t_n$.

Markov stationary policies have a compact representation. *Randomized* Markov stationary policies can be represented as a set of $X$ decisions (one for each state) $\delta_x$, $x \in \mathcal{X}$, which is equivalent to a $S \times A$ matrix $\mathbf{M}_\pi$. An element $m_{x,a}$ of $\mathbf{M}_\pi$ is the probability of issuing command $a$ given that the state of the system is $x$. Notice that for stationary Markov policies the dependency of $\delta$ from the time index $n$ is lost, and it is replaced by a dependency from the system state $x$.

*Deterministic* Markov stationary policies can still be represented by matrices where only one element for each row has value one and all other elements are zero. Moreover, they have an even more compact representation as a $S$-dimensional vector $\mathbf{m}_\pi$, with the $x$th element being the command to issue when in state $x$. We call $\Delta$ the class of deterministic Markovian stationary policies. Notice that, while there are infinite randomized policies, the cardinality of $\Delta$ is finite and equal to $S^A$.

The importance of these two classes of policies stems from two facts: first, they are *easy to store and implement*, second, we will show that for our system model, *optimum policies belong to these classes*.

*Example 3.7:* Consider the example system introduced in the previous section. We defined two commands and the system has eight states. Thus a generic randomized Markov stationary policy can be represented by a $8 \times 2$ matrix, such as the following:

$$\mathbf{M}_\pi = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{array} \begin{array}{c} \begin{array}{cc} s\_\mathrm{on} & s\_\mathrm{off} \end{array} \\ \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \\ 0.5 & 0.5 \\ 1.0 & 0.0 \\ 0.4 & 0.6 \\ 0.8 & 0.2 \\ 0.8 & 0.2 \\ 1.0 & 0.0 \end{pmatrix} \end{array}.$$

Decision $\delta_{x_1}$, the first row of matrix $\mathbf{M}_\pi$ specifies that when the system is in state $x_1$, the PM will issue a $s\_\mathrm{on}$ command with probability 0.4 or a $s\_\mathrm{off}$ command with probability 0.6. A deterministic Markov stationary policy can be represented by an eight-dimensional vector such as the following:

$$\mathbf{m}_\pi = \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{array} \begin{pmatrix} s\_\mathrm{off} \\ s\_\mathrm{on} \\ s\_\mathrm{on} \\ s\_\mathrm{on} \\ s\_\mathrm{off} \\ s\_\mathrm{on} \\ s\_\mathrm{on} \\ s\_\mathrm{on} \end{pmatrix}.$$

The first element of the vector, for instance, is the command that is issued by the PM when the system is in state $x_1$.

It is now possible to define the metrics of relevance in the policy optimization problem. In their most general form, they are function both of the state $x$ and of $\delta_x$, i.e., the decision we take when we are in state $x$. The first cost metric is the *expected power consumption level* $\overline{c}(x, \delta_x) = \sum_{p_a \in \delta_x} p_a c(s, a)$, where $c(s, a)$ is the power consumed by the SP when it is in state $s$ and command $a$ is issued. The second cost metric of interest is the *performance penalty* per unit time $d(x)$ which relates to the waiting time and the number of jobs in the queue. The simplest way to define function $d(x)$ is to set it equal to the number of requests in the queue: $d(x) = q$. For notational convenience, we define the consumption and performance penalty vectors

$$\overline{\mathbf{c}}_\delta := \begin{pmatrix} \overline{c}(x_1, \delta_{x_1}) \\ \vdots \\ \overline{c}(x_X, \delta_{x_X}) \end{pmatrix} \qquad \mathbf{d}_\delta := \begin{pmatrix} d(x_1) \\ \vdots \\ d(x_X) \end{pmatrix}.$$

## IV. Policy Optimization

As seen in the previous section, the complete system is described by a controlled Markov chain with stochastic matrix $\mathbf{P}(a)$. Given a policy $\pi$ and the $X$-dimensional row vector $\mathbf{p}^{(1)}$, representing the state probability distribution (henceforth called simply probability distribution) of the system at the initial time $t_1$, it is possible to compute the probability distribution of the system at future time $t_n$ according to the formula $\mathbf{p}^{(n)} = \mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}$. Based on this information, we can compute the expected value of the performance penalty and of the consumption at time $n$. They are given by

$$\begin{aligned} E_\pi[\mathbf{d}_{\delta^{(n)}}] &= \mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}\mathbf{d}_{\delta^{(n)}} \\ E_\pi[\overline{\mathbf{c}}_{\delta^{(n)}}] &= \mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}\overline{\mathbf{c}}_{\delta^{(n)}}. \end{aligned} \tag{6}$$

These values are the best estimates of future performance penalties and costs on the basis of our present information about the system. Since we are interested in the long-range behavior of the system, we compute the average over time of the performance metrics, and minimize with respect to such averages. We can formulate two constrained optimization problems: *performance optimization under power constraint* and *power optimization under performance constraint*. In the following, we describe the first problem in detail. The second problem is treated in exactly the same way.

Our objective function is the *average expected* performance penalty, and we constrain the maximum average expected consumption with an upper bound value $C$

$$\begin{aligned} &\min_\pi \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} E_\pi[\mathbf{d}_{\delta^{(n)}}] \\ &\text{s.t.} \lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} E_\pi[\overline{\mathbf{c}}_{\delta^{(n)}}] \le C. \end{aligned} \tag{7}$$

Remember that the unknown of this problem is the policy $\pi$. Yet, we observe that, as it is, the optimization is carried over an *infinite time horizon*. Such formulation has a drawback: it equally weights costs relative to the near future and to the far future. This is usually not the case. In a great variety of applications, optimization over time "discounts" the future, so that immediate rewards (or penalties) are weighted more than rewards in the far future. This is the case for our application, where the time span of interest for the optimization is the finite time window of the desired time between battery recharges. For instance, for typical laptop computers the desired time between recharges ranges between 8 and 12 h, or for aggressive applications, a few days. The behavior of the system for much longer time scales (say 10 yr) is absolutely of no interest for power optimization.

This intuition can be made rigorous according to the following reasoning. We assume that the system is not operating over an infinite horizon, but only over a *finite time horizon* $\hat{N}$ which is *finite* and *random*. We call $\hat{N}$ a *stopping time*. The time period [0, $\hat{N}$] corresponds to the time window of interest (also called a *session*), and randomization takes into account the uncertainty on its exact duration. An important modeling issue is then to define the probability distribution of the stopping time, and its relevant parameters: the distribution of $\hat{N}$ has to be sound, and must allow effective computability of the solution. In the following, we model the system session duration as follows: at the beginning of every period, the session will continue with probability $0 < \beta < 1$ ($\beta$ is called *discount factor*), or will close the session with probability $1-\beta$, independently from the state of the system. After a session has been closed, both consumption and performance penalty are assumed to be zero. We notice that this assumption can result is a slight error in the estimation of the performance metrics, because after the closing of a session some time might be necessary to serve the pending requests and to shut down the system. Yet, this error is small because it occurs only once and a session is generally much longer than the time resolution $T$.

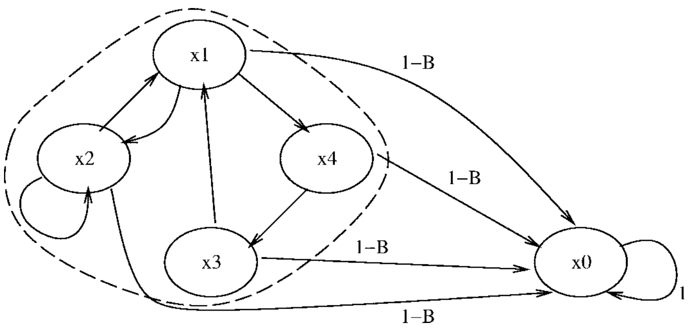


Fig. 5. Markov chain model of a system with trap state $x_0$.

The introduction of the time window is equivalent to assuming that the system's Markov chain has one additional trap state $x_0$, as shown in Fig. 5. All transition probabilities of the original Markov chain are multiplied by $\beta$, and each state $x \in \mathcal{X}$ has a new transition with probability $1-\beta$ toward $x_0$. Once in $x_0$, the system cannot change state. Moreover, both performance penalty and power in state $x_0$ are zero. Based on the above model, $\hat{N}$ is a geometrically distributed random variable with expected value equal $E[\hat{N}] = (1-\beta)^{-1}$.

As the expected stopping time $\hat{N}$ is finite, the *total* expected performance penalty and consumption are finite with probability one. Instead of considering average expected values of $d$ and $c$, it makes now sense to optimize their total expected values over the period [0, $\hat{N}$]. It is easy to see that, at time slice $n$, the probability distribution of the system is given by $\mathbf{p}^{(n)} = \mathbf{p}^{(1)}(\beta\mathbf{P})_\pi^{n-1} = \beta^{n-1}\mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}$. The expected values of $d$ and $c$ are redefined as

$$\begin{aligned} E_\pi[\mathbf{d}_{\delta^{(n)}}] &= \beta^{n-1}\mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}\mathbf{d}_{\delta^{(n)}} \\ E_\pi[\overline{\mathbf{c}}_{\delta^{(n)}}] &= \beta^{n-1}\mathbf{p}^{(1)}\mathbf{P}_\pi^{n-1}\overline{\mathbf{c}}_{\delta^{(n)}}. \end{aligned} \tag{8}$$

The optimization problem we set out to solve is then

$$\begin{aligned} \mathbf{PO1}\text{: } &\min_\pi \sum_{n=1}^{\infty} E_\pi[\mathbf{d}_{\delta^{(n)}}] \\ &\text{s.t.} \sum_{n=1}^{\infty} E_\pi[\overline{\mathbf{c}}_{\delta^{(n)}}] \le C. \end{aligned} \tag{9}$$

Notice that the discounted formulation is *not equivalent* to the formulation of (7). First, in (7), $E_\pi[\mathbf{d}_{\delta^{(n)}}]$ is an expected value, while in (9) it is a discounted expected value. Second, the summation of (9) taken over infinite time slices represents the cumulative discounted expected cost, while the limit in (7) represents an average cost. Convergence of (9) is guaranteed because $\beta < 1$ and $\mathbf{p}^{(1)}\mathbf{P}_{\pi}^{n-1}$ is bounded. In formulation (9), the contribution to the cost function and constraints of terms far in the future is smoothed out by the discount factor. Hence, the discounted formulation expresses the practical problem we want to solve, i.e., policy optimization over a finite time window.

In the PO1 problem, power is a constraint and performance is the objective. The performance-constrained power minimization problem (PO2) can be expressed in the same form

$$\begin{aligned}\mathbf{PO2}\text{: } &\min_{\pi} \sum_{n=1}^{\infty} E_{\pi}[\overline{\mathbf{c}}_{\delta(n)}] \\ &\text{s.t. } \sum_{n=1}^{\infty} E_{\pi}[\mathbf{d}_{\delta(n)}] \leq D. \end{aligned} \tag{10}$$

Since the two problems have the same mathematical formulation, we will focus on the first. All conclusions we derive can be applied to the second.

PO1 and PO2 are stochastic optimization problems, where the *expected value* of a cost function has to be minimized. The solution of PO1 and PO2 is based on classical results of stochastic optimization and it is described in detail in Appendix A. The key result can be summarized as follows: Policy optimization can be formulated as a linear programming (LP) optimization problem, hence, it can be solved *exactly* and in *polynomial time* (in $A \cdot X$).

## *A. The Space of Optimal Policies*

The solution of the LP provides a relation between a prespecified maximum average expected consumption, $C$, and the least performance penalty $D$ that can be attained while fulfilling the constraint on consumption (or vice versa). This relation can compactly be expressed in functional form: $D = g(C)$. If the linear program is infeasible (i.e., the constraint cannot be met), we define $g(C) = \infty$. We call the set $\mathcal{G}^{up} = \{(C, D) \text{ s.t. } D \geq g(C), C \geq 0\}$ the *set of feasible allocations* and the set $\mathcal{G} = \{(C, D) | D = g(C), C \geq 0\}$ the *set of efficient allocations*. Pairs $(C, D) \in \mathcal{G}$ correspond to Pareto points of the power-performance tradeoff curve, i.e., they represent solutions of PO that cannot be improved upon in both directions (power and performance). The following result holds (see [24]):

*Theorem 4.1:* $\mathcal{G}^{up}$ is a convex set.

*Proof:* Let $\mathbf{f}_1$, $\mathbf{f}_2$ be the $S \cdot A$-dimensional state-action vectors corresponding to the maximum consumptions $C_1$ and $C_2$, respectively. The vector $\lambda\mathbf{f}_1 + (1-\lambda)\mathbf{f}_2$ is a feasible solution for LP3, with power consumption constraint $\lambda C_1 + (1-\lambda)C_2$, and minimum performance $\lambda g(C_1) + (1-\lambda)g(C_2)$. The optimal solution will be smaller or equal than this value, so that $g(\lambda C_1 + (1-\lambda)C_2) \leq \lambda g(C_1) + (1-\lambda)g(C_2)$, and $\mathcal{G}^{up}$ is convex.

The theorem has an intuitive interpretation: if we keep reducing the availability of an existing resource (the consumption $C$), the “price” for that resource will keep increasing. By price we mean the increase in the objective function for a unit increase in the available resource. In other terms, price of a resource is a nonincreasing function of available quantity of that resource.

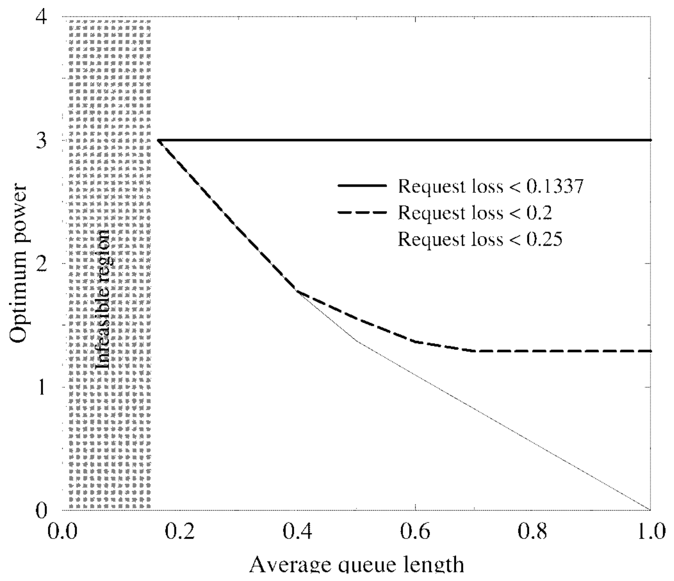


Fig. 6. Pareto curve for the example system.

*Example 4.1:* Pareto curves for the example system are shown in Fig. 6. The curves are obtained by repeatedly solving the LP with different performance constraints. The $x$-axis of the plot reports average queue length (i.e., the performance constraint), while the $y$-axis reports the expected optimum power consumption. Notice the presence of an infeasible region. Even if the resource is never put to sleep, the workload generated by the SR is such that it is impossible to achieve average queue smaller than 0.175.

The three Pareto curves illustrate the results of policy optimization with three different constraint settings. When request loss constraint is not very tight (lowest curve), performance constraint dominates and request loss constraint is never active. On the contrary, when the request loss constraint is very tight it always dominates over performance constraints. The resource is never allowed to turn off, because this would increase request loss, and power consumption is maximum (topmost curve). The middle curve shows an interesting intermediate situation. In the flat region, optimization is dominated by the request loss constraint, that makes the optimal solution insensitive to the performance constraint. Between 0.6 and 0.4 both constraints are active, while under 0.4 performance constraints become dominating and request-loss constraints inactive.

## V. Policy Optimization Tool

We implemented a policy optimization tool for the formulation described in the previous section and Appendix A. The tool is built around PCx, an advanced LP solver based on an interior point algorithm [27]. Interior point algorithms, augmented with presolvers, can efficiently solve very large LP instances with thousands of unknowns. The robustness of interior point-based LP solvers has greatly improved in the last few years, and state-of-the-art implementations such as

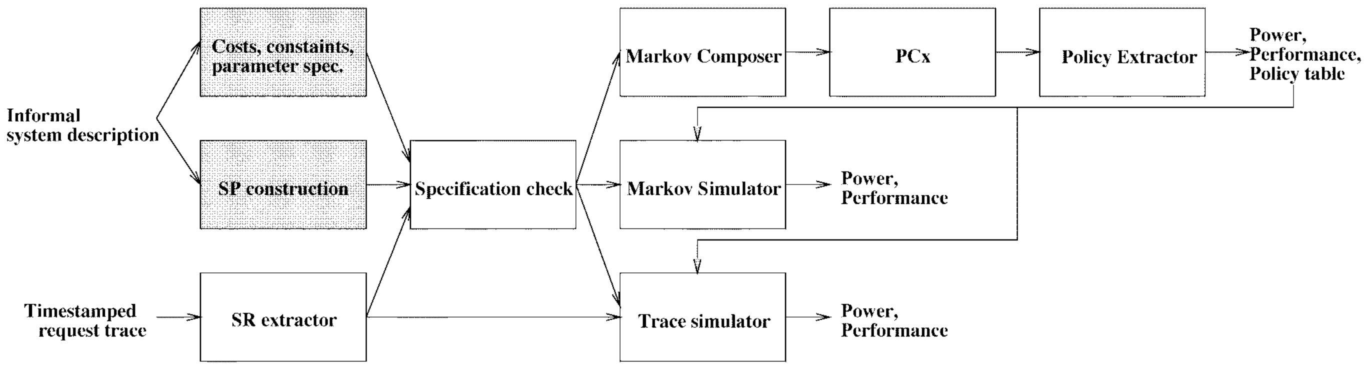


Fig. 7. Block diagram of the policy optimization tool.

PCx are competitive with the best simplex-based traditional LP solvers. PCx is just the computational core of a complex tool whose block diagram is shown in Fig. 7.

The tool requires two inputs: a request trace consisting of time-stamped request records (obtained from measurements on a real system), and a system description. The request trace is automatically analyzed by the *SR extractor* that builds a Markov chain model for the service requester. The system description is an informal specification of the information needed to formulate the SP model, various system parameters (time horizon, queue length), cost functions (power and performance), constraints and optimization target. The translation of the informal specification into the stochastic model described in the previous sections is done manually. This step is represented by the shaded blocks in Fig. 7 (shaded blocks represent manual steps, while white boxes are fully automated steps). The system, specified as a set of transition probability matrices, tabular representations of cost functions, constraints and optimization directives, is checked for syntactic correctness and passed to the *Markov composer* that builds a monolithic Markov model by merging the Markov chains of the system components.

The model, cost functions, constraints, and optimization targets are then translated into PCx input format and passed to the LP solver for computing the optimum policy. The output of the LP solver is the set of state-action frequencies, the expected performance and power. State-action frequencies are translated into command probabilities by the *policy extractor* and the policy matrix is obtained. The optimization tool can call the LP solver iteratively, to explore the entire power-performance tradeoff curve. In this case, a set of policies and their corresponding expected performance and power values are computed. This type of design space exploration is efficiently supported by PCx.

The optimal policies computed by the optimizer can be verified by a flexible simulation engine, that takes as input the system specification, the SR model and the optimal policy, then simulates the Markov chain models for computing power and performance. The simulated power and performance values can then be compared against those obtained by the optimizer to check consistency. A second simulation mode is available, where the request trace can be used to directly drive the simulation. This type of simulation is employed to check the quality of the Markov model of the service provider. If the arrival of service requests is poorly modeled by a Markov process, the performance and power values returned by this simulation do not match the expected performance and power computed by the optimizer.

The procedure implemented in *SR extractor* for extracting the Markov model for the SR from a time-stamped request trace is relatively straightforward. Given a time resolution $T$, the arrival times of requests are discretized. The trace is converted into a binary stream that has value one in position $k$ if a request is received between time $nT$ and time $(n+1)T$, zero otherwise. Then, a memory $m$ is chosen for the SR model. The $m$-memory Markov model has $2^m$ states, one for each possible sequence of $m$ consecutive bits. The conditional transition probabilities are computed by counting the occurrences of state transitions, and dividing the count by the total number of times the start state of the transition is visited.

*Example 5.1:* Consider the following time-stamped request trace, represented as an array of request arrival times (in ms): [2, 5, 6, 7, 12]. Initial time is zero. Assume that $T = 1$ ms, thus, the discretized trace becomes [0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1]. Consider a $m = 1$ SR model. The model has two states, marked by values zero and one. The conditional transition probability for the $0 \rightarrow 1$ transition is computed by counting the number of 01-sequences in the discretized trace, and dividing it by the number of zeros in the trace. For our trace, there are three 01-sequences, and eight occurrences of zero. Hence, the conditional probability of the $0 \rightarrow 1$ transition in the Markov model of the SR is 3/8. The other transition probabilities can be computed in a similar fashion.

Notice that given a value of $T$ and $m$, the model extraction procedure always produces a Markov model for the SR. However, there is no guarantee that the model is representative of the statistical properties of the actual trace. The validity of the SR model should always be checked by simulating the behavior of the optimal policies when the system is driven by the actual trace.

## VI. Case Studies

We now focus on how to apply policy optimization. In order to demonstrate the flexibility of the Markov model, we have considered three case studies belonging to different classes of systems. For each example, we will outline its

TABLE I
STATE, TRANSITION TIME TO ACTIVE AND POWER DISSIPATION FOR A HARD DISK DRIVER

| State | $\Delta T$ | $Power$ |
|---|---|---|
| active | NA | 2.5 W |
| idle | 1.0 ms | 1.0 W |
| LPidle | 40 ms | 0.8 W |
| standby | 2.2 s | 0.3 W |
| sleep | 6.0 s | 0.1 s |

distinguishing features and present the results obtained by policy optimization.

In Appendix B, we will also show how the Markov model and policy optimization can be employed as analysis tools for exploring power-efficient system architectures and implementations.

## A. Disk Drive

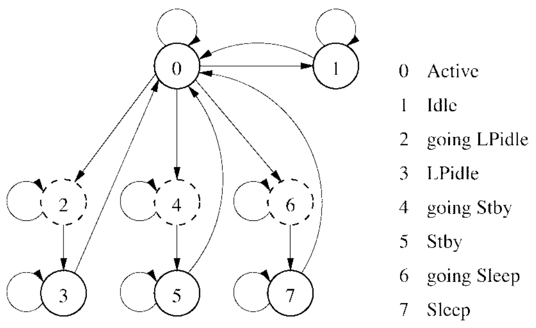
The first case study is a commercially-available hard disk drive.[1] This is an example of a system with a single resource with queueing and a complex state transition structure. The hard disk can operate in five different states, as shown in Table I. In four of the five states, the device cannot perform data reads or writes, hence, they are all *inactive* states.

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. The low-power idle *LPidle* 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 I are explicitly declared as *typical* in the data sheets. In other words, they can be interpreted as expected values of random variables.

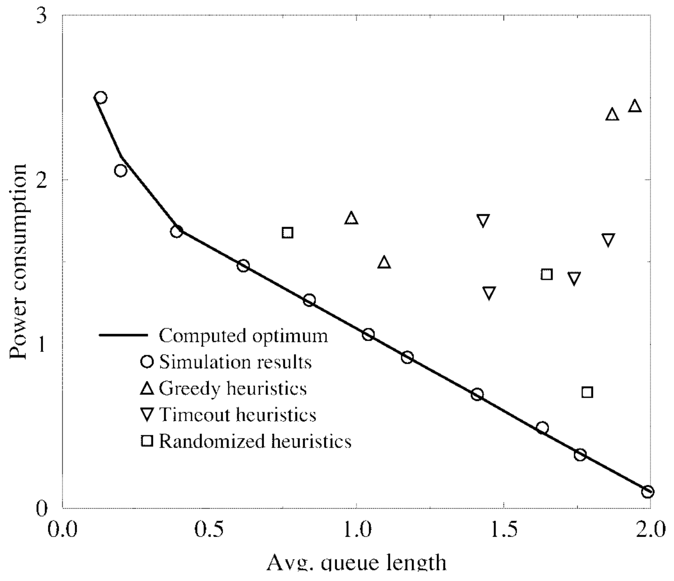
Time resolution $T$ has been chosen based on the expected transition times of the SP. We set time resolution based on the fastest possible transition performed by the disk drive. We are not interested in increasing time resolution beyond $T = 1$ ms, because the system cannot perform faster state transitions. With this choice for time resolution, the transition between *active* and *idle* takes a single time step, while the transition times for all remaining transitions are scaled accordingly.

The transition graph of the SP that models the disk drive is shown in Fig. 8(a). In general, conditional probabilities associated with the edges of the state transition graph of the SP depend on the command issued by the PM. This is the way the PM controls the resource. The active state is denoted by 1, while the four inactive states are denoted by 2, 4, 7, and 10.

States 3, 5, 6, 8, 9, and 11 are transient states. Transitions from transient states have constant conditional probabilities that cannot be controlled by commands. Thus, when in transient states, the behavior of the SP is insensitive to the PM. Transient states are used to model nonunitary-time transitions



(a)



(b)

Fig. 8. (a) Simplified state transition graph of the disk drive. (b) Power consumption versus performance for optimal and heuristic policies.

that cannot be interrupted. When in transient states, the SP has zero service rate but its power consumption is high: 2.5 W.

Transitions between inactive 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. All transition probabilities of the SP Markov model are set up so that the expected transition times (upon assertion of a command from the PM) are equal to the experimental transition times reported on the data sheets (Table I).

If a file read or write request is received when the SP is inactive, the request cannot be serviced right away. In our case study, pending requests are enqueued in a queue of length 2. Requests arriving when the queue is full are lost. Request loss abstractly represents the undesirable condition of too many incoming requests. The workload was modeled by a two-state Markov process as described in Example 3.2. Transition probabilities were extracted from time-stamped traces of disk accesses (we used the traces provided[2]) measured on real machines.

[1] Hard Drive IBM Travelstar VP 2.5-inch, http://www.storage. ibm.com/storage/oem/data/travvp.htm, 1996.

[2] Auspex File System Traces, http://now.cs.berkeley. edu/Xfs/Auspex Traces/auspex.html, 1993.

The complete model of the system has $X = 66$ states, obtained as the product of the 11 SP states, of the two SR states, and of the three SQ states. At any time, the PM can issue a command to the SP. In our case study the manager can choose among five commands: *go_active*, *go_idle*, *go_LPidle*, *go_standby*, *go_sleep*. The final product of the policy optimization is a matrix with five columns (one for each command) and 66 rows (one for each state of the system).

The system was optimized for minimum power under performance constraints. The power cost metric is obtained from the data sheets of the disk drive (summarized in Table I). The user can specify two types of performance constraints. First, a latency constraint can be enforced by specifying a value for maximum expected waiting time $\tau_{\max}$ for an incoming request. Second, a constraint on request loss $L$ can be enforced by specifying the maximum probability $l_{\max}$ for an incoming request to find the queue full.

The results of our experiments are shown in Fig. 8(b). They refer to a time horizon $\hat{N}$ of one million time steps, corresponding to a discount factor $\beta = 10^{-6}$. The continuous line is the power-performance tradeoff curve spanned by the optimal policies computed by the optimizer. Its computation took less than 1 min on a SUN UltraSPARC workstation. Each point on the curve is a solution of a PO problem with different performance constraints. The expected performance and power values returned by the optimizer for the various solutions were checked by simulation.

The circles in Fig. 8(b) represent the results of simulation ($10^7$ time steps) of the policies computed by the optimizer with the actual trace from which the Markov model of the workload was extracted. The distance of the circles from the curve is a measure of the inaccuracies of the modeling process. It is visually obvious that in this case the model is quite accurate, and that the simulated points lie almost perfectly on the theoretical tradeoff curve.

Triangles and boxes in Fig. 8(b) represent heuristic solutions to the PO problem. The upwards triangles represent deterministic greedy policies exploiting different inactive states. The policies shut down the disk (making a transition to a given inactive state) as soon as there are no pending requests on the queue end no new requests coming from the user. A wake-up command is issued whenever a new request arrives. The downwards triangles represent timeout heuristics. Timeout-based policies are widely used for disk power management [12]. They shut down the disk when the user has been inactive for a time longer than the timeout period $T_{to}$. The choice of $T_{to}$ is based on simulations and on designer's experience. Boxes represent 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 provide power-performance points not far from the Pareto curve. We remark, however, that heuristic solutions do not allow the designer to automatically take constraints into account. On the other hand, trial and error approaches may be highly expensive due to the large number of parameters (in our case study the policy is represented by a $66 \times 5$ matrix with 330 entries). Moreover, even if it is possible to produce heuristic policies that produce "reasonable" results, there is no way for the designer to estimate if the results can be improved.

## B. Web Server

In the second case study, we modeled a web server with two processors for a high-traffic web site, which is an example of a system with multiple service providers. Time resolution $T = 5$ s. The time horizon is set to one day, i.e., $\approx 2 \cdot 10^4$ time slices. The two processors are not identical. The second processor has higher performance (1.5 times) and higher power (2 times) than the first one. We model the system as a SP with four states, one for each possible combination of active/sleeping processors. Three of the four states are active, in the sense that the system is able to process workload. The power manager can independently activate or put to sleep the two processors (by issuing one of four different commands), trying to achieve a given average performance level, representing system throughput, with minimum power consumption.

Throughput is maximum (1) when the two processors are both active, the remaining SP states have reduced throughput (i.e., 0.4 when only processor 1 is active, 0.6 when only processor 2 is active and zero when both processors are in sleep state). Performance is constrained by imposing a given minimum expected throughput. Processor power in the active state is 1 W for the first processor, 2 W for the second. Turn-on transition power is an additional 0.5 W over the processor's active power. Shut-down transition power is 0.5 W less than active power. Expected turn-on time is $2T$, while expected shut-down time is $1T$.

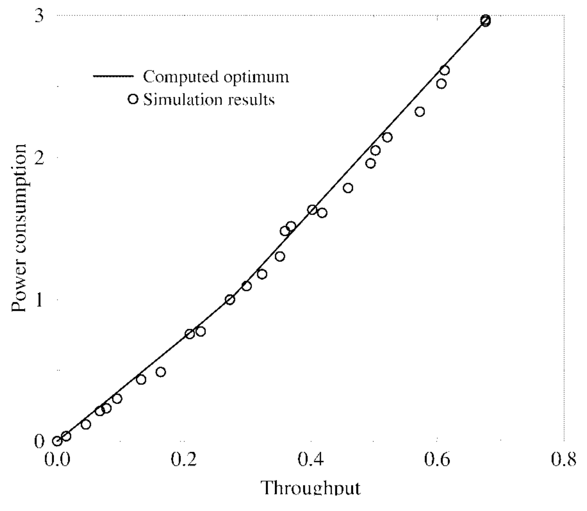
The SR model was extracted from real-life traces obtained by monitoring a busy web server.[3] We constructed a two-state Markov model for the workload. The total number of states for the SP and SR model is $8 = 2 \cdot 4$. Policy optimization results (minimum power under performance constraints) are reported in Fig. 9(a). The continuous-line curve shows the theoretical Pareto curve, while the circles represent the results obtained by simulating the model of the system with the computed optimum policy driven by the actual workload trace.

Analyzing the policies computed by the optimizer, we noticed that the processor with higher performance was never used alone. Intuitively, this is due to the fact that the power consumption of the faster processor is twice as large as the power of the slower one, but its performance is less than two times higher.

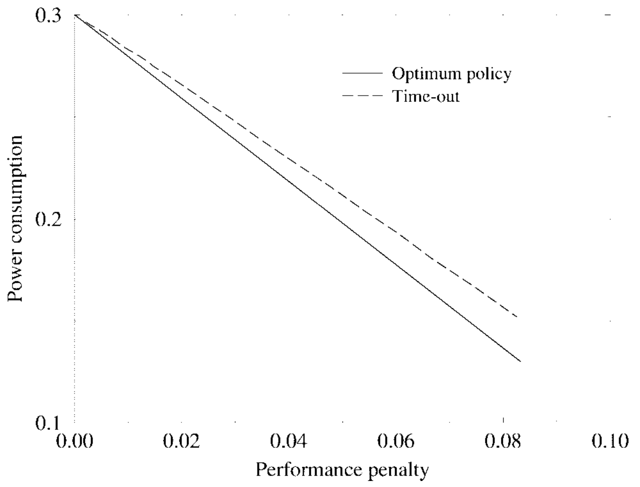
## C. CPU

The third type of system considered is a typical CPU with sleep mode operation. We built our model based on the ARM SA-1100 processor [29]. The actual processor has three power states: *active*, *idle*, and *sleep*. We do not model the idle state because the transitions between *active* and *idle* are very fast and have small cost. Hence we can assume that they are performed greedily, independently of the power

[3] Internet Traffic Archive, http://ita.ee.lbl.gov/.



(a)



(b)

Fig. 9. Power-performance tradeoff curves. (a) Two-processor http server with real-world workload taken from the Internet Traffic Archive: solid line and circles represent optimization and simulation results, respectively. (b) Two-state CPU model with Markovian workload: solid and dashed lines represent the tradeoff achievable by means of optimum stochastic control and time-outs, respectively.

management, and we use a single macro state (that we call *active*) to represent both the active and idle states of the actual processor. As a result, the SP model has only two states, *active* (with power consumption of 0.3 W and full performance) and *sleep* (with null power consumption and null performance). Shut-down and turn-on transitions (from *active* to *sleep* and vice versa) take approximatively 100 ms and have power consumption of 0.3 and 0.9 W, respectively.

The CPU is designed to react to service requests (i.e., interrupt signals) independently of the power manager. In our model, whenever there are incoming requests the SP is insensitive to PM commands, and a turn-on transition is performed unconditionally if a new request arrives when the SP is in sleep state. In practice, only when the SP is active and the SR is idle the PM can control the evolution of the system by issuing a *shut_down* command.

The SR model is a two-state (*active* and *idle*) Markov chain characterized on a CPU workload trace provided by the monitoring package described in [28]. The discretization time step is $T = 100$ ms. Incoming requests are not enqueued. Request arrival when the CPU is sleeping is regarded as an undesirable condition whose occurrence probability has to be constrained. To this purpose, a performance penalty function is defined taking value one when SR = *active* and SP = *sleep*, zero otherwise.

The constrained power optimization problem consists of deciding when to issue the *shut_down* command, in order to minimize power while keeping performance penalty under a given threshold. Optimum policies depend on a single parameter, that is the probability of issuing the *shut_down* command when in state SR = *active* and SP = *sleep*. Notice that this degree of freedom is the same that could be exploited by a traditional shut-down mechanism. Hence, we can use this case study to make a fair comparison between optimum stochastic control and timer-based heuristics.

Comparative results are reported in Fig. 9(b). The solid line is the Pareto curve of optimum stochastic control (obtained by varying the performance penalty constraint). The dashed line is obtained by varying timeout values for a timeout heuristic. We remark that optimum stochastic control performs better than a timeout heuristic even in this case, where the power manager can only control shutdown. The difference in power savings is due to the fact that timeout-based policies waste power while waiting for a timeout to expire.

## VII. Modeling Assumptions: A Critique

The key feature of our policy optimization algorithm is that it allows us to compute a globally optimum policy in polynomial time in the number of states of the system. However, such a strong result is based on several fundamental assumption on the system model. These assumption should be clearly understood in order to assess the domain of applicability of our technique. The basic hypotheses for the validity of our Markov model are as follows:

- The arrival of service requests can be modeled by a $2^m$ state Markov chain.
- The state transition delays in the service provider can be modeled as random variables with a geometric distribution.
- Model parameters and cost functions are available and accurately measured before optimization.

It is possible to envision systems in which one or more of these hypotheses is not verified. For instance, request arrival times can be nonstationary and their distribution can widely change over time.

*Example 7.1:* We applied a highly nonstationary and non-Markovian workload to the case study of Section VI-C. The workload was obtained by merging two real-world traces with completely different statistics, representing to usage of the CPU of a laptop computer with a single user editing a text file and compiling C code [28]. The first trace presents alternating idle and active periods, while the second one has a long activity burst.

A simple two-state Markov model was characterized for the entire trace and used as SR for policy optimization. Optimum

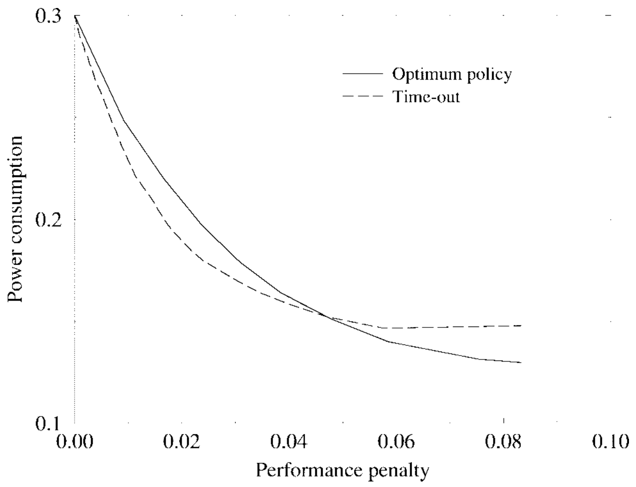


Fig. 10. Power-performance tradeoff for the CPU model of Fig. 9(b) with a nonstationary/non-Markovian workload.

policies were then simulated against the original trace. Simulation results are reported in Fig. 10 together with the results obtained by a timeout-based shut-down mechanism. In some cases, timeout-based shutdown outperforms stochastic control. This is a situation where one of our modeling assumptions is not valid (namely, stationary Markovian workload), hence, Markovian policies may be good but are not provably globally optimum.

Similarly, assuming a geometric distribution of SP transition times may be in many cases an inaccurate model of actual system state transitions. For instance, transition times can be deterministic, or have a bell-shaped distribution around an expected value. Finally, power and performance measures based solely on the SP state may be inaccurate and it may be difficult to obtain data on some system parameters (such as transition probabilities for the SR). For instance, a designer may not be able to collect in advance a time-stamped request trace which is representative of actual workloads, and only approximate information on the request arrival process is available.

It is possible to improve the accuracy of the model for nonstationary workloads, nongeometric transition times and complex cost functions by increasing the number of states in the Markov chain representing the system. Unfortunately, the size of the state space can easily become unmanageable even for powerful advanced LP solvers. Consider for example the case of a deterministic transition delay $D$ from a sleep state to the active state. It is possible to model a deterministic delay by a chain of states in a Markov model. There are $(D-T)/T$ states with the same performance as the sleep state. The last state of the chain has a transition to the active state. All transitions have probability 1. Clearly, if $D$ is much larger than $T$, a huge number of SP states is required to model just one transition. In practice, we need to tolerate model inaccuracies if we want to be able to solve policy optimization with exact LP-based solution. The price to be paid for is in reduced optimality of the power management policies. Good engineering intuition is required to match the approximations made in the model to the desired level of accuracy for constrained policy optimization.

Even if the basic assumptions on the probabilistic model are valid, we can envision systems with a more complex structure than the one described in the previous sections. For instance, we may need to model multiple SR's and/or multiple SP's. Furthermore, the distinction between SP and SR may blur if we assume that a SP can pass requests to other SP's for additional service. A general system model is a network of interacting service providers, multiple service requesters and queues. Even though it is in principle possible to compute a monolithic Markov model for an arbitrary complex network of SP's and SR's, the size of the state space grows exponentially with the number of system components. Again, designer insights are required to formulate decomposition strategies that reduce the state space to a manageable size, without completely compromising the quality of the optimization.

With a correct understanding of the basic model assumption, our policy optimization approach can be effectively employed to reduce power consumption in a number of real-life systems. Promising domains of application are power management for laptop computers, for their components and for portable electronic appliances as well as energy conservation for desktop computers. The two key advantages of our approach over heuristic techniques are the capability of exploring the power-performance tradeoff curve and the high degree of confidence on the quality of the achieved results.

## VIII. Conclusion

The identification of optimal power management policies for low-power systems 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 realistic case studies. Our experimental results show that our stochastic model is robust and the optimal policies are flexible and power-efficient. We believe that our work opens the way to several new research problems. First, it is possible to envision extensions to the basic model to deal with systems consisting of multiple interacting resources as well as with systems where one or more components can be modeled by stochastic processes other than stationary Markov chains. Another interesting direction of investigation is the study of adaptive algorithms that can compute optimal policies in systems where workloads are highly nonstationary and the service provider model changes over time.

## Appendix A
## Solving the Policy Optimization Problem

We first introduce a simplified, unconstrained version of PO1 that can be solved in polynomial time by *linear* *programming*. The solutions of the unconstrained version of PO1 are deterministic stationary Markov policies. Then, we show that the introduction of constraints does not change the complexity of the problem, that can still be solved by linear programming. However, the solutions become randomized stationary Markov policies.

If we remove the power constraint, unconstrained performance optimization (POU) can be formulated as

$$\textbf{POU}\text{: } \min_{\pi} \mathbf{p}^{(1)}\mathbf{v}_{\pi} = \min_{\pi} \sum_{n=1}^{\infty} \beta^{n-1}\mathbf{p}^{(1)}\mathbf{P}_{\pi}^{n-1}\mathbf{d}_{\delta(n)} \quad (11)$$

the unknown in POU is the policy $\pi$, and the cost function $\mathbf{p}^{(1)}\mathbf{v}_{\pi}$ to be minimized is an infinite sum, where each summand is the product of the $X$-dimensional row vector $\mathbf{p}^{(1)}$ (the initial state probability vector) and the column vector $\mathbf{v}_{\pi}^{(n)} = \beta^{n-1}\mathbf{P}_{\pi}^{n-1}\mathbf{d}_{\delta(n)}$. The $x$th component of $\mathbf{v}_{\pi}^{(n)}$ represents the expected performance penalty at time $t_n$, given that the initial state was $x$, and the $x$th component of $\mathbf{v}_{\pi}$ represents the *total* expected performance penalty at time $t_n$, given that the initial state was $x$. If we find a policy that minimizes each component of the column vector, such policy will be the solution of POU, independently from $\mathbf{p}^{(1)}$. The following classical result [21] provides the key insight that leads to the solution of POU

*Theorem A.1:* There exists an optimal policy $\tilde{\pi}$ that minimizes each component of $\mathbf{v}_{\pi}$. Such policy is stationary, deterministic and Markovian. The optimal value $\mathbf{v}_{\tilde{\pi}}$, called $\tilde{\mathbf{v}}$, does not depend on the time index and its components satisfy the optimality equations: $\tilde{v}_x = \min_{a \in \mathcal{A}}\{d(s, a) + \beta \sum_{x' \in \mathcal{X}} p(x'|x, a)\tilde{v}_{x'}\}$. The optimality equations can be written in vector form as

$$\tilde{\mathbf{v}} = \min_{\delta \in \Delta}\{\mathbf{d}_{\delta} + \beta\mathbf{P}_{\delta}\tilde{\mathbf{v}}\}. \quad (12)$$

Proof for this theorem can be found in [21]. Remember that $\Delta$ is the finite set of all possible $X^A$ stationary deterministic policies. We focus here on the interpretation of the optimality equations; for detailed accounts (in order of increasing generality and complexity), see [21]–[23]. The optimality equations state an intuitive fact: by taking an optimal decision $\delta$ at the beginning of time $t_1$, the optimal penalty will be the performance penalty relative to this time slice [i.e., $\mathbf{d}_{\delta}$ in (12)], plus the expected cost from time $t_2$ onward; but, by virtue of the Markovian structure of the system (i.e., conditioning only on the previous slice), this is the discounted ($\beta$) expected value ($\mathbf{P}_{\delta}$) of the optimal penalty ($\tilde{\mathbf{v}}$).

Many algorithms are available to efficiently solve the class of optimization problems POU [21], the most well-known being *policy improvement, successive approximations,* and *linear programming*. We will use the last method, because it allows to generalize our model to the constrained case in a straightforward fashion. We note first that the optimal cost vector $\tilde{\mathbf{v}}$ satisfies the set of vector inequalities

$$\tilde{\mathbf{v}} \le \mathbf{d}_{\delta} + \beta\mathbf{P}_{\delta}\tilde{\mathbf{v}}, \qquad \text{for all } \delta \in \Delta. \quad (13)$$

The set of vector inequalities is equivalent to $X \cdot A$ scalar inequalities. If two vectors $\mathbf{v}$, $\mathbf{w}$ satisfy inequalities (13),

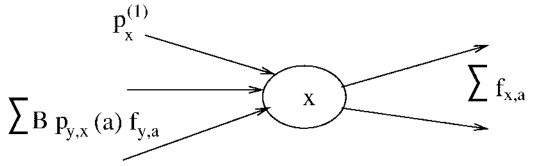


Fig. 11. Balance equation interpretation of the first constraint in LP2.

then the maximum for the two vectors (taken componentwise), $\max\{\mathbf{v}, \mathbf{w}\}$, satisfies the same set of inequalities. A candidate for the optimal objective vector is the $\tilde{\mathbf{v}}$ for which the inequalities become tight. In other terms, it is the $\tilde{\mathbf{v}}$ which satisfies the following linear program:

$$\textbf{LP1}\text{: } \min \mathbf{p}^{(1)}\mathbf{v} \text{ s.t } \mathbf{v} \le \mathbf{d}_{\delta} + \beta\mathbf{P}_{\delta}\mathbf{v}, \qquad \text{for all } \delta \in \Delta. \quad (14)$$

Notice that the solution $\tilde{\mathbf{v}}$ of LP1 is the same independently from the choice of the initial state distribution $\mathbf{p}^{(1)}$. This is because Theorem A.1 guarantees that all components of $\mathbf{v}$ are minimized when $\tilde{\pi}$ is used.

It is possible to show [22] that $LP1$ is equivalent to the following linear program which has a more intuitive interpretation:

$$\begin{aligned} \textbf{LP2}\text{: } \min & \sum_{s \in \mathcal{X}} \sum_{a \in \mathcal{A}} f_{x,a}\, d(x) \\ \text{s.t. } & \sum_{a \in \mathcal{A}} f_{x,a} - \beta \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} p_{y,x}(a) f_{y,a} = p_x^{(1)}, \\ & \text{for all } x \in \mathcal{X} \\ & f_{x,a} \ge 0, \qquad \text{for all } x \in \mathcal{X},\, a \in \mathcal{A}. \end{aligned} \quad (15)$$

The $A \cdot X$ unknowns in LP2, $f_{x,a}$, called *state-action frequencies,* have the following intuitive interpretation: they are the expected number of times that the system is in state $x$ and command $a$ is issued. The objective function is, thus, the total expected penalty under the optimal policy. The first set of constraints expresses the condition that the expected number of times state $x$ is the current state ($\sum_{a \in \mathcal{A}} f_{x,a}$) is equal to the expected initial population of $x$, $p_x^{(1)}$, plus the expected number of times $x$ is reached from another state ($\beta \sum_{y \in \mathcal{X}} \sum_{a \in \mathcal{A}} p_{y,x}(a) f_{y,a}$). This “balance equation” is described pictorially in Fig. 11, that represents the “incoming flow” (transitions toward state $x$) and “outgoing flow” (transitions from state $x$) for state $x$.

Once the $f_{x,a}$ have been found by solving LP2, the elements $m_{x,a}$ of the optimal stationary policy $\mathbf{M}_{\pi^*}$ are simply given by

$$m_{x,a} = f_{x,a} \Big/ \sum_{a' \in \mathcal{A}} f_{x,a'}. \quad (16)$$

*Example A.1:* Consider the example system introduced in the previous section. The system has eight states $x_1, x_2, \cdots, x_8$ and two actions $s\_\text{on}$ and $s\_\text{off}$. Thus, LP2 has $16 = 8 \times 2$ unknowns, the state-action frequencies $f_{x_1, s\_\text{off}}, f_{x_1, s\_\text{on}}, \cdots, f_{x_8, s\_\text{on}}$. We assume that the time window of interest is [0, $10^5$], hence, the discount factor is $\beta = 1 - 10^{-5} = 0.999\,99$. The performance cost $d(x)$ has only two values: $d(x) = 0$ for the four states where the queue is empty, and $d(x) = 1$ for the four states with queue full. The initial probability distribution is: $p_{x_1} = 1.0$, $p_x = 0.0$ for all $x \neq x_1$ (i.e., the service provider is initially on, no requests are issued and the queue is empty). The transition probabilities $p_{y,x}(a)$ are those described in Example 3.5.

The linear program LP2 has eight equality constraints (one for each state) and 16 inequality constraints (one for each $f_{x,a}$). The cost function is the sum of eight state-action frequencies, those corresponding to states for which $d(x) = 1$ (i.e., states with full queue). The optimal policy minimizes the sum of the $f_{a,x}$ for these states. The command probabilities $m_{x,a}$ are obtained from the state-action frequencies using (16).

The linear programming formulation (LP2) of the unconstrained policy optimization problem has the advantage that it can be easily extended to the constrained case, because it is easy to express constraints on a second cost metric as a linear function of the state-action frequencies. In our case, the total expected power consumption can be written as: $\sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} f_{x,a}c(x, a)$. Therefore, we can solve problem PO1 by simply adding a power constraint in LP2, as follows:

$$\begin{aligned}\textbf{LP3}\text{: } \min & \sum_{s\in\mathcal{X}}\sum_{a\in\mathcal{A}} f_{x,a}\, d(x) \\ \text{s.t.} & \sum_{a\in\mathcal{A}} f_{x,a} - \beta \sum_{y\in\mathcal{X}}\sum_{a\in\mathcal{A}} p_{y,x}(a) f_{y,a} = p_x^{(1)}, \\ & \text{for all } x \in \mathcal{X} \qquad (17) \\ & \sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} f_{x,a}c(x, a) \le C \\ & f_{x,a} \ge 0, \qquad \text{for all } x \in \mathcal{X},\, a \in \mathcal{A}. \qquad (18)\end{aligned}$$

The following result [22] holds for the constrained optimization problem LP3, and shows that the set $\Delta$ of stationary deterministic policies does not contain all optimal solutions:

*Theorem A.2:* If the constraint $\sum_{x\in\mathcal{X}} \sum_{a\in\mathcal{A}} f_{x,a}c(x, a) \le C$ is inactive, the solution of LP3 is a stationary, Markovian deterministic policy. If the constraint (18) is active, and the feasible region is nonempty, the solution is a stationary, Markovian randomized policy.

Proof for this theorem can be found in [22]. The theorem states that the optimal policy is Markovian, stationary and randomized whenever the expected power consumption of the system is exactly $C$. In other words, the most aggressive optimal policies that push power consumption up to its maximum allowed value but achieve maximum performance, are randomized.

Although we have discussed the solution of PO1 in detail, the equivalent problem PO2 can be solved by a linear program as well. The linear program for the solution of LP2 is the following:

$$\begin{aligned}\textbf{LP4}\text{: } \min & \sum_{s\in\mathcal{X}}\sum_{a\in\mathcal{A}} f_{x,a}c(x, a) \\ \text{s.t.} & \sum_{a\in\mathcal{A}} f_{x,a} - \beta \sum_{y\in\mathcal{X}}\sum_{a\in\mathcal{A}} p_{y,x}(a) f_{y,a} = p_x^{(1)}, \\ & \text{for all } x \in \mathcal{X} \\ & \sum_{x\in\mathcal{X}}\sum_{a\in\mathcal{A}} f_{x,a}\, d(x) \le D \\ & f_{x,a} \ge 0, \qquad \text{for all } x \in \mathcal{X},\, a \in \mathcal{A}. \qquad (19)\end{aligned}$$

Observe that linear programs LP3 and LP4 have the same number of unknowns and of constraints. Moreover, the minimum power consumption obtained by solving LP4 for a given performance constraint $D$ is equal to the value $C$ we should assign to the power constraint if we want to obtain a solution of LP3 with minimum performance penalty $D$. Hence, the solutions of LP3 and LP4 are equivalent both from the computational and numeric viewpoint. The choice of what problem to solve is based on how the constraints and optimization targets are specified in practical instances.

One key advantage of the linear programming formulation is that it allows the specification of additional constraints. For instance, it is possible to enforce a constraint on the maximum probability of a *request loss* (defined in Section III). The constraint is specified by adding to LP3 (or LP4) an inequality requiring that the sum of all state-action frequencies corresponding to states where SR issues a request and the queue is full is smaller than a given bound $L$.

*Example A.2:* Consider linear program LP4 for our simple example system, with discount factor $\beta = 0.999\,99$. The initial state is: no request from SR, queue empty and active SP. Thus, $p_x^{(1)}$ has only one element equal to one and all others equal to zero. Performance cost function $d(x)$ is the same as in Example A.1. The power cost function $c(a, x)$ is $c(a, x) = 3$ when SP is "on" and the command is $s_$on; $c(a, x) = 4$ when SP is "on" and the command is $s_$off and when SP is "off" and the command is $s_$on; $c(a, x) = 0$, otherwise. This cost function models a system where forcing SP state transitions is more power-consuming than leaving the SP on, hence, the SR should be shut off only when there is high probability that the SP will not issue requests for a relatively long time in the future.

The average queue length is constrained to be not larger than 0.5. Thus, the performance constraint is set to $D = 0.5 \cdot (1 - \beta)^{-1} = 50\,000$. The probability of losing a request must be smaller than 20%. The request loss $L$ constraint is, thus, $L = 0.2 \cdot (1 - \beta)^{-1} = 20\,000$. LP4 is solved, and the 16 state-action frequencies are computed. Then, the elements of policy matrix $M_\pi$ are computed with (16). The matrix of the resulting optimal policy is

$$\mathbf{M}_\pi = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{array} \begin{array}{c} \begin{array}{cc} s\_\text{on} & s\_\text{off} \end{array} \\ \begin{pmatrix} 0.774 & 0.226 \\ 1.000 & 0.000 \\ 1.000 & 0.000 \\ 1.000 & 0.000 \\ 0.000 & 1.000 \\ 0.395 & 0.605 \\ 0.000 & 1.000 \\ 0.000 & 1.000 \end{pmatrix} \end{array}.$$

The minimum expected power value is $P = 1.554$ W. Notice that the SP power in "on" state is 3 W. Thus, the optimal policy reduces power consumption of almost a factor of two with respect to the trivial policy that never shuts down the SP. Consider the first row of $\mathbf{M}_\pi$. State $x_1$ is the triple $(on, r_0, q_0)$: the SP is on, the SR is not issuing any request, and the queue is empty. The decision prescribes that the $s_$off command can be issued with probability 0.226 and the $s_$on command can be issued with probability 0.774. In other words, the SP is put to sleep with probability 0.226, otherwise it stays active, even if there are no incoming requests. Obviously, the optimum policy is not equivalent to the eager one, that would shut down the SP as soon as it becomes idle.

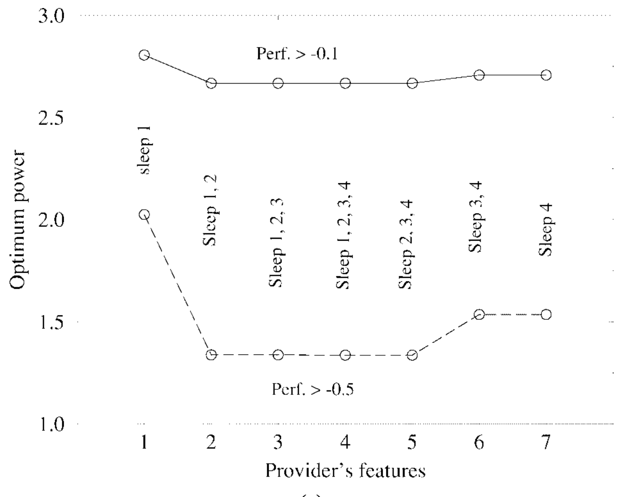
## Appendix B
## Sensitivity to Model Parameters

In Section IV we assumed a Markov system model, and we showed how to compute globally-optimum policies. In this subsection we study how system structure and parameters impact the result of the optimization process. In other words, we exploit our powerful optimization engine to gain deep insights on how to design and tune systems that can be effectively power managed.

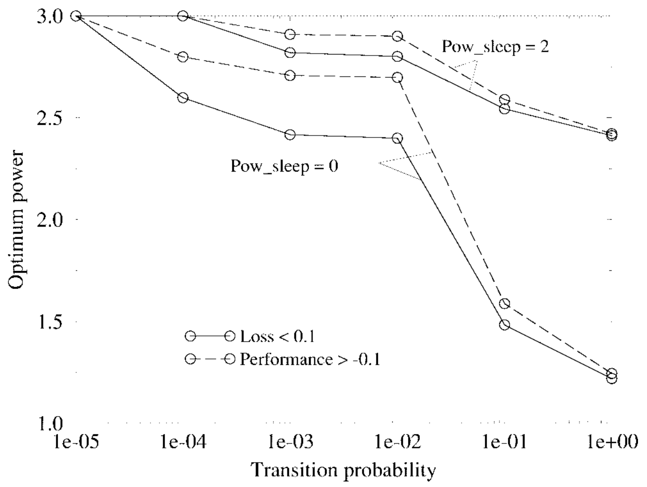
To perform this study, we start from a baseline system implementation, we introduce changes in its structure and parameters and we perform policy optimization on the modified system targeting minimum power consumption with a fixed performance constraint (i.e., we solve LP4). We then compare the expected power consumption obtained by the optimizer for the modified system with the optimum expected power consumption achieved by the baseline system. Our baseline implementation is the following. SP has two states: *active* and *sleep1*. Power consumption is high in active state (3 W) and lower in sleep state (2 W). When the SP is performing a state transition, the power consumption is 4 W. Transitions from active to *sleep1* require only one time slice. The SR model has two states as well. In one state a request is issued, while no request is issued in the other state. The transition probability from one state to another and vice versa is 0.01. The queue has maximum length equal 2.

Our first set of experiments, illustrated in Fig. 12, focuses on the sensitivity to the structure of the service provider. Time horizon is $\hat{N} = 10^4$. In the first experiment, we analyze the impact of multiple sleep states. Sleep state 1 is the baseline, and the power for the baseline system is represented by the leftmost point in the graphs. States 2–4 are deeper sleep states, ordered for decreasing power and increasing transition delay for returning to active. State *sleep2* has power 1 and transition probability 0.1 (i.e., in average 10 clock cycles are required to transition from sleep to active), state *sleep3* has power 0.5 and transition probability 0.01 and state *sleep4* has power 0.0 and transition probability 0.001. Fig. 12(a) shows two sets of data points, representing the minimum power achievable with a given SP structure. The continuous line joins power values obtained with a tight performance constraint, while the dashed line joins power values obtained with a loose performance constraint. Points on the same abscissa have been obtained with the same SP structure. We considered six alternative SP structures with multiple sleep states (the number and type of sleep states for each alternative SP is reported in the figure).

The impact of the number and type of sleep states is quite clear. Having more than one sleep state improves power, but many multiple sleep states are not always useful. For the workload condition specified for the experiment, introducing state *sleep2* brings a sizable power reduction, while introducing additional states does not help much. Notice also that the performance constraint influences the amount of incremental power savings. When the constraint is tight, it is more difficult to use deep sleep states, hence, they are less effective in reducing power. Furthermore, we can improve power even without increasing the number of states. For instance, the system with only the active an the *sleep4* state performs better than the baseline system (that uses sleep state *sleep1*).



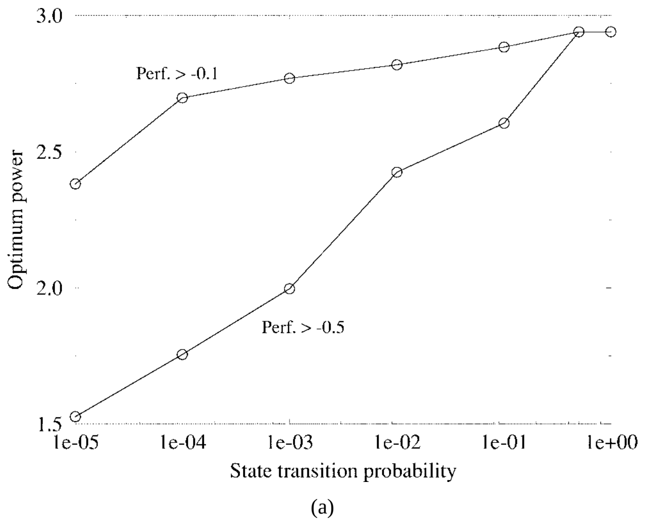
(a)



(b)

Fig. 12. (a) Power consumption versus available sleep states of the SP. (b) Power consumption versus SP state transition cost.

In Fig. 12(b) we study the impact of the transition time between active and sleep and vice versa. We assume that the system has a single sleep state and that the power dissipated during transition is 4 W (which is higher than the 3 W active power consumption). Time horizon is $\hat{N} = 10^5$. Four sets of data-point are shown in the figure. The topmost two sets correspond to SP models where the power dissipated in the sleep states is 2 W, while the other two sets are obtained by assuming zero sleep state power. The data sets joined by continuous lines are obtained by optimizing power with a constraint on request loss, while a performance constraint is enforced for the sets joined by dashed lines. The abscissa reports the value of the transition probability for exiting the sleep state (which is inversely proportional to the average transition time).



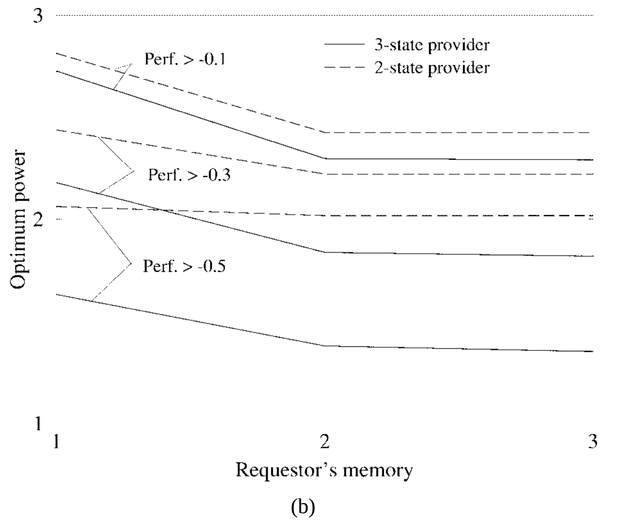
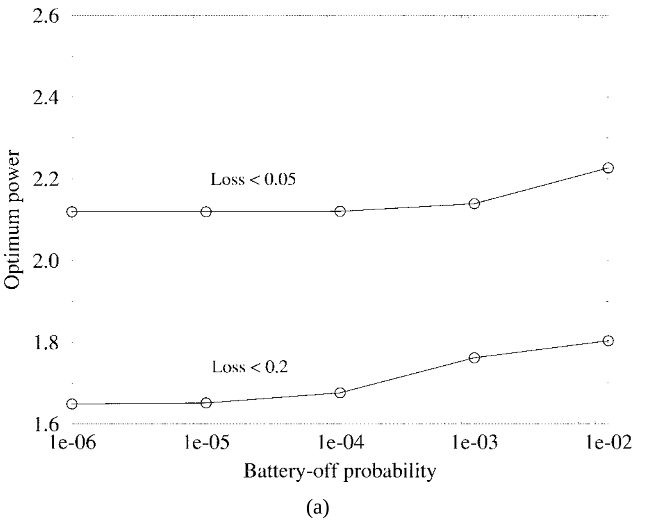


Fig. 13. (a) Power consumption versus SR burstiness. (b) Power consumption versus SR memory.



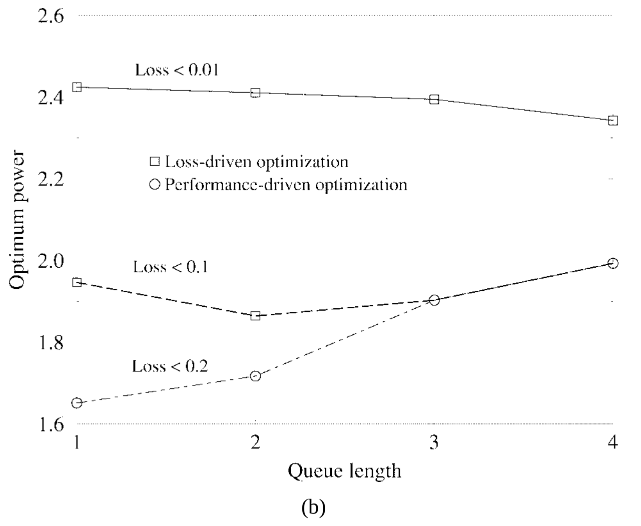


Fig. 14. (a) Power consumption versus time horizon. (b) Power consumption versus queue length.

From the plot we can infer the following observations. Since transitions have high power consumption, the optimal power value is strongly sensitive to transition speed (faster transition times corresponds to points on the right side of the plot). For very slow transition times, performance constraints inhibit the exploitation of sleep states (the two points on the upper left corner). Even when constraints are not active (the two uppermost points of the two top curves), if the average transition times are comparable with the time horizon, sleep states are not used. Notice also that high-power, but fast-transition sleep states may become more convenient than low-power, slow-transition sleep states (the two top-most curves for faster transition times are more power-efficient than the two lower curves for slow transition times).

The experiments summarized by the plots of Fig. 13 assess the impact of SR characteristics. The plot of Fig. 13(a) focuses on SR burstiness. For all data points we used the same SP model (with four sleep states). Time horizon is $\hat{N} = 10^5$. Maximum expected request loss was set to 0.01. Two sets of points are shown for two performance constraints. The abscissa reports the probability of a transition from the state where a request is issued to the state where no requests are issued and vice versa. Points to the left correspond to bursty SR. The interpretation of these results is immediate. The more bursty is the receiver the more effective is power management. It is important to notice that increased burstiness does not imply reduced workload. In fact, the probability of issuing a request is the same (0.5) for all data points in the plot.

In Fig. 13(b) we study the dependency of the optimization on SR memory. A Markov model with memory 1 has two states (this is the baseline SR model). The number of states in the SR is $2^m$, where $m$ is the memory. Intuitively, longer memory means more complex correlations between past and current history. From an optimization point of view, a more complex SR model gives the optimizer more possibilities of exploiting past history to predict request issues and take optimal decisions. Clearly, this increased knowledge about the past is paid for by increased system complexity (the number of states is exponential in $m$). We optimized power with three different performance constraints (time horizon is $\hat{N} = 10^4$). We also considered two different SP: the baseline one, and a SP with two sleep states. Clearly, increased memory improves optimization quality, because the optimizer can better distinguish long request bursts from short ones. Also, the power savings are higher if there are multiple sleep states to chose from, because the optimal policy matches the length of idle periods with the best sleep state.

In the last set of experiments (Fig. 14), we studied optimal power as a function of time horizon and queue length. In Fig. 14(a) we plot optimum power values for the SP with four sleep states, performance better than 0.5, baseline two-state SR and queue length two. Two sets of points are shown, for two request loss constraints. The abscissa reports the probability of a transition to the trap state, i.e., longer time horizons are to the left of the plot. The interpretation of the plot is straightforward. The longer the time horizon the better are the achievable power savings, because the optimizer has a longer time to amortize wrong decisions, hence, more degrees of freedom in selecting aggressive shutdown policies.

The interpretation of Fig. 14(b) is a little more involved. Time horizon is $\hat{N} = 10^5$. The $x$-axis reports the maximum queue length. We have three sets of data points, for different request loss constraints. Performance constraint is the same (performance penalty better than 0.5) for all plots. The SP has four sleep states, and the SR model is the baseline one. For the two sets of data points marked with squares, the constraint on request loss is very tight and dominates the optimization results. For the data points marked with circles, the performance constraint dominates. When optimization is dominated by request loss constraint, larger maximum queue length reduces the probability of a request to find the queue full even if the resource is aggressively shut down. Thus, power dissipation can be reduced more effectively. However, when optimization is dominated by performance constraint, which is related to average waiting time, shorter queue lengths give better results. This is because a high-capacity queue reduces the probability of an arrival with queue full (i.e., a request loss), but implies that enqueued requests wait a longer time for service.

## References

[1] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.

[2] J. M. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Norwell, MA: Kluwer, 1996.

[3] W. Nebel and J. Mermet, Eds., *Low Power Design in Deep Submicron Electronics*. Norwell, MA: Kluwer, 1997.

[4] S. Udani and J. Smith, "The power broker: Intelligent power management for mobile computing," Dept. of Comput. Inform. Sci., Univ. Pennsylvania, Tech. Rep. MS-CIS-96-12, 1996.

[5] S. Gary, P. Ippolito, G. Gerosa, C. Dietz, J. Eno and H. Sanchez, "PowerPC 603, a microprocessor for portable computers," *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 14–23, Win. 1994.

[6] G. Debnath, K. Debnath, and R. Fernando, "The Pentium processor-90/100, microarchitecture and low power circuit design," in *Proc. Int. Conf. VLSI Design*, Jan. 1995, pp. 185–190.

[7] S. Furber, *ARM System Architecture.* Reading, MA: Addison-Wesley, 1997.

[8] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Norwell, MA: Kluwer, 1997.

[9] J. Lorch and A. Smith, "Software strategies for portable computer energy management," *IEEE Personal Commun., Mag.*, vol. 3, no. 5, pp. 60–73, June 1998.

[10] Intel, Microsoft and Toshiba, "Advanced configuration and power interface specification," [Online]. Available www: http://www.intel.com/ial/powermgm/specs.htm, 1996.

[11] Microsoft, "OnNow: The evolution of the PC platform," [Online]. Available www: http://www.microsoft.com/hwdev/onnow.htm, 1997.

[12] R. Golding, P. Bosch, and J. Wilkes "Idleness is not sloth," in *Proc. Winter USENIX Tech. Conf.*, 1995, pp. 201–212.

[13] ——, "Idleness is not sloth" HP Laboratories, Tech. Rep. HPL-96-140, 1996.

[14] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Syst.*, vol. 4, pp. 42–55, Mar. 1996.

[15] C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. Int. Conf. Computer Aided Design*, 1997, pp. 28–32.

[16] K. M. Sivalingam, M. Srivastava, P. Agrawal and J. C. Chen, "Low-power access protocols based on scheduling for wireless and mobile ATM networks," in *Proc. Int. Conf. Universal Personal Communications*, 1997, pp. 429–433.

[17] G. Hadjiyiannis, A. Chandrakasan, and S. Devadas, "A low power, low bandwidth protocol for remote wireless terminals," in *Proc. IEEE Global Telecommunications Conf.*, 1996, pp. 22–28.

[18] M. Zorzi and R. Rao, "Energy-constrained error control for wireless channels," *IEEE Personal Commun.*, vol. 4, no. 6, pp. 27–33, Dec. 1997.

[19] J. Rulnick and N. Bambos, "Mobile power management for wireless communication networks," *Wireless Networks*, vol. 3, no. 1, pp. 3–14, 1997.

[20] B. Mangione-Smith, "Low power communications protocols: Paging and beyond," in *Proc. IEEE Symp. Low-Power Electronics*, 1995, pp. 8–11.

[21] S. Ross, *Introduction to Stochastic Dynamic Programming.* New York: Academic, 1983.

[22] M. Puterman, *Finite Markov Decision Processes.* New York: Wiley, 1994.

[23] M. Y. Kitaev and V. V. Rykov, *Controlled Queuing Systems.* Boca Raton, FL: CRC, 1995.

[24] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization.* Belmont, CA: Athena Scientific, 1997.

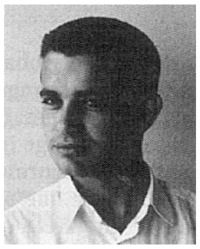
[25] K. Trivedi, *Probability & Statistics with Reliability, Queuing and Computer Science Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[26] S. Ross, *Introduction to Probability Models*, 6th ed. New York: Academic, 1997.

[27] J. Czyzyk, S. Mehrotra, and S. Wright, "PCx user guide," Optimization Technol. Center, Tech. Rep. OTC 96/01, May 1996.

[28] L. Benini, A. Bogliolo, S. Cavallucci, and B. Riccò, "Monitoring system activity for OS-directed dynamic power management," in *Proc. IEEE Int. Symp. Low-Power Electronics and Design*, Aug. 1998, pp. 185–190.

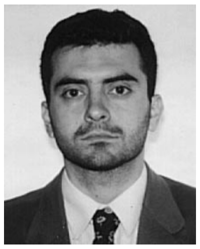
[29] Intel Corporation, *SA-1100 Microprocessor*, Tech. Ref. Manual, Sept. 1998.



**Luca Benini** (S'94–M'97) received the B.S. degree (*summa cum laude*) in electrical engineering from the University of Bologna, Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

Since 1998, he has been an Assistant Professor in the Department of Electronics and Computer Science in the University of Bologna. He also holds visiting researcher positions at Stanford University and Hewlett-Packard Laboratories, Palo Alto, CA. His research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications.

Dr. Benini has been a member of the technical program committee for the Design and Test in Europe Conference, the International Symposium on Low-Power Design, and the International Workshop on Logic Synthesis.

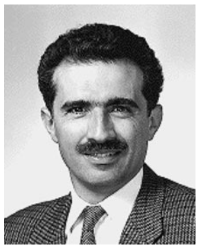


**Alessandro Bogliolo** (S'96–M'97) received the Laura degree *summa cum laude* in electrical engineering in 1992 and the Ph.D. degree in electrical engineering and computer science in 1998 from the University of Bologna, Bologna, Italy.

He is currently a postdoctoral Research Associate with the Department of Electronics, Computer Science and Systems (DEIS) at the University of Bologna. In 1995/1996, he was a Visiting Scholar with the Computer Systems Laboratory (CSL), Stanford University, Stanford, CA. Since then, he cooperates with the research group of Prof. G. De Micheli, at Stanford. His research interests are in the area of computer-aided design of digital integrated circuits and systems, with particular emphasis on high-level power modeling, and optimization.

**Giuseppe A. Paleologo** (S'98) received the B.S. degree in physics from the University of Rome "La Sapienza," and the M.S. degree in engineering-economic systems and operations research (EES and OR) from Stanford University, Stanford, CA. He is currently working toward the Ph.D. degree in EES and OR at Stanford University.

His research interests are in distributed algorithms for congestion control of communication networks and in computer performance modeling and evaluation.



**Giovanni De Micheli** (S'79–M'79–SM'89–F'94) is a Professor of electrical engineering, and by courtesy, of computer science at Stanford University, Stanford, CA. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on synthesis, system-level design, and hardware/software codesign. He is the author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw Hill, 1994); coauthor of *Dynamic Power Management: Circuit Techniques and CAD Tools* (Norwell, MA: Kluwer, 1998); and of three other books.

Dr. De Micheli was granted a Presidential Young Investigator award in 1988. He received the 1987 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS (CAD) Best Paper Award and two Best Paper Awards at the Design Automation Conference in 1983 and in 1993. He is Vice President (for publications) of the IEEE CAS Society. He is the Editor of the IEEE TRANSACTIONS ON CAD/ICAS. He was the Program Chair (for Design Tools) of the Design Automation Conference in 1996 and 1997, and he is currently Vice General Chair of DAC. He was Program and General Chair of International Conference on Computer Design (ICCD) in 1988 and 1989, respectively.