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.

MICROSOFT CORP.
EXHIBIT 1013

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 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 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

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

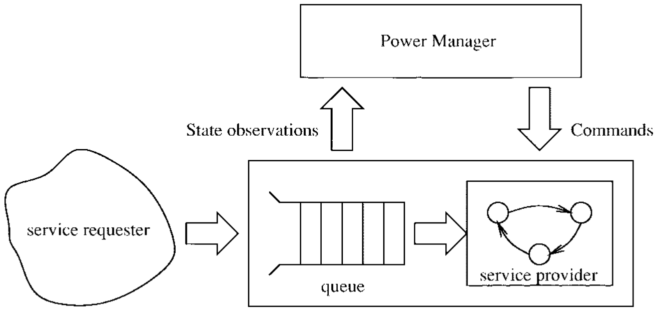

Fig. 1. Components of the system model.

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 $\mathcal{S} = \{s_i, \text{s.t. } i = 1, 2, \cdots, S\}$ whose behavior is such that, at any time t_n, the state probability distribution depends only on the state at time t_{n-1}. $\text{Prob}(x_n = s_j | x_{n-1} = s_i) = p_{s_i, s_j}$ is called *one-step transition probability*. The one-step transition probabilities are conveniently specified in the $\sum_{s\in\mathcal{S}} p_{s_i,s} = 1$. 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*

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

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 $p_{s_i,s_j}(a)$: $\mathcal{A} \rightarrow [0, 1]$, 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* $\mathcal{A} = \{a_i, \text{ s.t. } i = 1, 2, \cdots, A\}$. 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 $\mathcal{S} = \{s_i \text{ s.t. } i = 1, 2, \cdots, S\}$, control set $\mathcal{A}$ and stochastic matrix $\mathbf{P}^{\text{SP}}(a)$; ii) $b(s, a)$ is a function b: $\mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$; and iii) $c(s, a)$ is a function c: $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

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

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

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 $0 \leq b \leq 1$. Zero service rate means that no requests can be serviced and

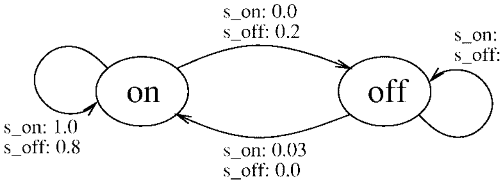

Fig. 2. Markov chain model of the service provider.

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, $\mathcal{S} = \{\mathrm{on}, \mathrm{off}\}$. Assume that two commands are defined $\mathcal{A} = \{s\_\mathrm{on}, s\_\mathrm{off}\}$, 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 $\mathbf{P}^{\mathrm{SP}}(a)$ can be represented by two matrices, one for each command. For example

$$\mathbf{P}^{\mathrm{SP}}(s\_\mathrm{on}) = \begin{array}{ccc} & \mathrm{on} & \mathrm{off} \\ \mathrm{on} & 1 & 0 \\ \mathrm{off} & 0.1 & 0.9 \end{array}$$

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

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

$$b(a, s) = \begin{array}{ccc} & s\_\mathrm{on} & s\_\mathrm{off} \\ \mathrm{on} & 0.8 & 0 \\ \mathrm{off} & 0 & 0 \end{array}$$

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

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

The service requester models the system's environment as a

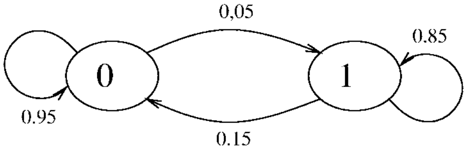

Fig. 3. Markov chain model of the service requester.

$\mathbf{P}^{\mathrm{SR}}$. 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

$$\mathbf{P}^{\mathrm{SR}} = \begin{array}{ccc} & 0 & 1 \\ 0 & 0.95 & 0.05 \\ 1 & 0.15 & 0.85 \end{array}.$$

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 $\mathcal{M}_{\mathrm{SQ}}(a, s, r)$ 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 $\mathbf{P}^{\mathrm{SQ}}(a, s, r)$.

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 $p^{\mathrm{SQ}}_{q_i, q_j}$ 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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase®
Smarter legal research.