Stochastic Modeling of a Power-Managed System: Construction and Optimization

Qinru Qiu, Qing Wu, and Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

Abstract -- *The goal of a dynamic power management policy is to reduce the power consumption of an electronic system by putting system components into different states, each representing a certain performance and power consumption level. The policy determines the type and timing of these transitions based on the system history, workload, and performance constraints. In this paper, we propose a new abstract model of a power-managed electronic system. We formulate the problem of system-level power management as a controlled optimization problem based on the theories of continuous-time Markov decision processes and stochastic networks. This problem is solved exactly using linear programming or heuristically using "policy iteration." Our method is compared with existing heuristic methods for different workload statistics. Experimental results show that the power management method based on a Markov decision process outperforms heuristic methods by as much as 44% in terms of power dissipation savings for a given level of system performance.*

I. Introduction

With the rapid progress in semiconductor technology, chip density and operation frequency have increased, making the power consumption in battery-operated portable devices a major concern. High power consumption reduces the battery service life. The goal of low-power design for battery-powered devices is thus to extend the battery service life while meeting performance requirements. Reducing power dissipation is a design goal even for non-portable devices since excessive power dissipation results in increased packaging and cooling costs as well as potential reliability problems. The focus of this paper is, however, on portable electronic systems.

Portable electronic devices tend to be much more complex than a single VLSI chip. They contain many components, ranging from digital and analog to electro-mechanical and electro-chemical. Much of the power dissipation in a portable electronic device comes from non-digital components. Dynamic power management – which refers to a selective shut-off or slow-down of system components that are idle or underutilized – has proven to be a particularly effective technique for reducing power dissipation in such systems. Incorporating a dynamic power management scheme in the design of an already-complex system is a difficult process that may require many design iterations and careful debugging and validation.

To simplify the design and validation of complex power-managed systems, a number of standardization attempts have been initiated. Best known among them is the *Advanced Configuration and Power Interface* (ACPI) [6] that specifies an abstract and flexible interface between the power-managed hardware components (VLSI chips, hard disk drivers, display drivers, modems, etc.) and the *power manager* (the system component that controls the turn-on and turn-off of the system components). The functional areas covered by the ACPI specification are:

- System power management – ACPI defines mechanisms for putting the computer as a whole in and out of system sleeping states. It also provides a general mechanism for any device to wake the computer.
- Device power management – ACPI tables describe motherboard devices, their power states, the power planes the devices are connected to, and controls for putting devices into different power states. This enables the OS to put devices into low-power states based on application usage.
- Processor power management – While the OS is idle but not sleeping it will use commands described by ACPI to put processors in low-power states.

ACPI does not, however, specify the power management policy. It is the objective of the proposed research to provide a framework and supporting tools for constructing optimal power management policies based on modeling the power-managed system as a continuous-time Markov decision process.

The problem of finding a power management scheme (or policy) that minimizes power dissipation under performance constraints is of great interest to system designers. A simple power management system includes four components: Service Provider (SP), Service Requestor (SR), Service Queue (SQ), and Power Manager (PM). Figure 1 shows the information/command flow in a power-managed system. The SR generates service requests for the SP. The SQ buffers the service requests. The SP provides service to the requests in a top-down manner. The PM monitors the states of the SR, SQ, and SP and issues state-transition commands to the SP. A simple and well-known heuristic policy is the "time-out" policy, which is widely used in today's portable computers. In the "time-out" policy, the SP is shut down after it has been idle for a certain amount of time. The predictive system shutdown approach proposed in [7][8] tries to achieve better power-delay trade-off by predicting the "on" and "off" time of each component. This prediction approach uses a regression equation based on the component's previous "on" and "off" times to estimate the next "turn-on" time, such that the SP can be turned on just before the request comes. Therefore, the system performance can be improved. This method is only applicable to the special cases where the requests are highly correlated.

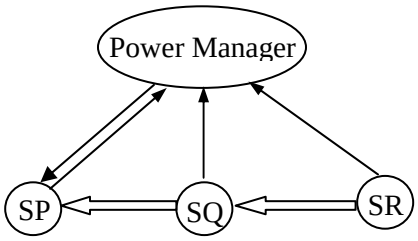


Figure 1 A power-managed system.

In general, heuristic policies cannot achieve the best power-delay trade-off for the system, cannot deal with complex components that have more than two (on and off) operating modes such as defined in ACPI, and cannot deal with a complex system with multiple interactive components.

A power management approach based on a Markov decision process has been proposed in [8]. The system is modeled as a **discrete-time** Markov decision process by combining the stochastic models of each component. Once the model and its parameters are determined, an optimal power management policy for achieving the best power-delay trade-off in the system is generated. This approach offers significant improvements over previous power management techniques in terms of its theoretical framework for modeling and optimizing the system. There are, however, some shortcomings. First, because the system is modeled in the discrete-time domain, some assumptions about the system components may not hold for real applications, such as the assumption that each event comes at the beginning of a time slice, the assumption that the transition of the SQ is independent of the transition of the SP, etc. Second, the state transition probability of the system model cannot be obtained accurately. For example, the discrete-time model cannot distinguish the busy state and the idle state because the transitions between these two states are instantaneous. However, the transition probabilities of the SP when it is in these two states are different. Moreover, the power management program needs to send control signals to the components in every time-slice, which results in heavy signal traffic and a heavy load on the system resources (and therefore more power).

In this work, we overcome these shortcomings by introducing a new system model based on **continuous-time** Markov decision processes. More precisely:

1. The new model is based on continuous-time Markov decision processes, which are more suitable for modeling real systems.
2. The resulting power management policy is asynchronous, which is more appropriate for implementation as part of the operating system.

3. The new model explicitly distinguishes the busy state and the idle state of SP so that the system characterization becomes more accurate.
4. The new model considers the correlation between the state of the SQ and the state of the SP, which is the real-life scenario.
5. The model for the service queue consists of a normal queue and a high priority queue. This is important since some service requests are "urgent" and need immediate response from the server.
6. The service requester model is capable of capturing complex workload characteristics.
7. The overall system model is constructed exactly and efficiently from the component models. We use an analytical base approach to calculate the generator matrix for the joint process of SP-SQ and a tensor sum based calculation to calculate the generator matrix of the joint process of SP-SQ and SR.
8. Both (exact) linear programming and (heuristic) policy iteration algorithms are used to solve the policy optimization problem.

Parts of this work were published in [10] and [11]. This paper is organized as follows: Section II gives a theoretical background of continuous-time controllable Markov processes. Sections III and IV describe the models for the components and the system. Section V describes the solution technique for the optimal policy. Sections VI and VII present the experimental results and conclusions.

II. Background

This section provides a theoretical background on continuous-time Markov decision processes.

We first give the notation that will be used throughout the paper:

$P_{i\Rightarrow j}(t)$: transition probability from state i (directly or indirectly) to state j during time 0 to t

$p_i(t)$: probability that the system is in state i at time t

$X(t)$: value of the stochastic process X at time t

$\boldsymbol{S}$, $\boldsymbol{T}$: state space and parameter space of a stochastic process

$\mathbf{G}$: the *generator matrix* of a continuous-time Markov process

A_i: set of available actions when a system is in state i

$a_i(t)$: the action that the system takes when it is in state i at time t, $a_i(t)\in A_i$

$p_i^{a_i}(t)$: the probability that action $a_i(t)$ is taken when the system is in state i at time t

$\boldsymbol{p}_i^{A_i}(t)$: the vector of $p_i^{a_i}(t)$, for all $a_i\in A_i$

$\boldsymbol{\pi}$: the power management policy

$\sigma_{i,j}$: transition rate from state i to state j

$\sigma_{i,j}^{a_i(t)}$: transition rate from state i to state j at time t when action $a_i(t)$ is taken

$\sigma_{i,j}^{\boldsymbol{p}_i^{A_i}(t)}$: transition rate from state i to state j at time t when actions are taken with probability $\boldsymbol{p}_i^{A_i}(t)$

$r_{i,i}$: reward rate (per unit time) of the system when it is in state i

$r_{i,j}$: transition reward of the system during the time when it makes a transition from state i to state j

r_i: earning rate of the system during the time it is in state i

$r_i^{a_i(t)}$: reward rate of the system when it is in state i and action $a_i(t)$ is taken at time t

$r_i^{\boldsymbol{p}_i^{A_i}(t)}$: reward rate of the system when it is in state i and actions are taken with probability $\boldsymbol{p}_i^{A_i}(t)$ at time t

$v_i(t)$: the total expected reward of the system from time 0 to time t with initial state i

$v_i^{\boldsymbol{\pi}}(t)$: the total expected reward of the system from time 0 to time t with initial state i and policy $\boldsymbol{\pi}$

$v_{i,avg}^{\boldsymbol{\pi}}$: the limiting average reward of the system with the initial state i and policy $\boldsymbol{\pi}$, $v_{i,avg}^{\boldsymbol{\pi}} = \lim_{t\to\infty}\frac{1}{t}v_i^{\boldsymbol{\pi}}(t)$

χ_{s_i,s_j} : the inverse of the average switching time of the SP from state s_i to state s_j

τ_{r_i,r_j} : the inverse of the average switching time of the SR from state r_i to r_j

λ_l (λ_h): SR request generating rate of a low (high) priority request

μ_l (μ_h): SP service rate of a low (high) priority request

A. Stochastic process

Definition 2.1 A *stochastic process* is a family of random variables $\{X(t), t\geq 0\}$, one for each t. t denotes the time parameter. For a specific t, $X(t)$ is a random variable with distribution $F(x, t) = P[X(t) \leq x]$. The values assumed by the process are called the *states*, and the set of possible values is called the *state space*.

Definition 2.2 A stochastic process $X(t)$ is called a *Markov process* if for any set of time instances $t_0<t_1<\ldots<t_n<t$ its conditional distribution has the property:

$$P[X(t)\leq x \mid X(t_n)=x_n, X(t_{n-1})=x_{n-1},\ldots,X(t_0)=x_0] = P[X(t)\leq x \mid X(t_n)=x_n]$$

where $t_0, t_1,\ldots, t_n, t \in \boldsymbol{T}$ and $x_0, x_1,\ldots, x_n \in \boldsymbol{S}$. When $\boldsymbol{T}$ is a continuous space and $\boldsymbol{S}$ is a discrete space, the Markov process is called the *continuous-time Markov process*.

Given a continuous-time Markov process with n states, its *generator matrix* $\mathbf{G}$ is defined as an $n\times n$ matrix as shown in Eqn. (2.1). An entry $\sigma_{i,j}$ in $\mathbf{G}$ is called the *transition rate* from state i to state j. All entries are defined in Eqn. (2.2) and Eqn. (2.3). Eqn. (2.4) gives the relationship between $\sigma_{i,i}$ and $\sigma_{i,j.}$

$$\mathbf{G} = \begin{bmatrix} -\sigma_{0,0} & \sigma_{0,1} & \sigma_{0,2} & \cdots \\ \sigma_{1,0} & -\sigma_{1,1} & \sigma_{1,2} & \cdots \\ \sigma_{2,0} & \sigma_{2,1} & -\sigma_{2,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.1}$$

$$\sigma_{i,i} = \lim_{t\to 0}\frac{1-p_{i\Rightarrow i}(t)}{t} = -p'_{i\Rightarrow i}(0) \qquad i=1,\ 2,\ \ldots,\ n \tag{2.2}$$

$$\sigma_{i,j} = \lim_{t\to 0}\frac{p_{i\Rightarrow j}(t)}{t} = p'_{i\Rightarrow j}(0) \qquad i,\ j = 1, 2, \ldots, n;\ i\neq j \tag{2.3}$$

$$\sum_{j\neq i}\sigma_{i,j} = \sigma_{i,i} \qquad i,\ j = 1, 2, \ldots, n;\ i\neq j \tag{2.4}$$

where $p'_{i\Rightarrow j}(t)$ is the derivative of $p_{i\Rightarrow j}(t)$. Obviously, $p_{i\Rightarrow j}(t)\geq 0$ and $\sum_{j\in \boldsymbol{S}} p_{i\Rightarrow j}(t) = 1$.

The generator matrix in the continuous-time Markov process is the analogue of the transition probability matrix in the discrete-time Markov process. We can calculate the limiting distribution (steady) state probabilities of the continuous-time Markov process from its generator matrix. Theorem 2.1 shows the relation between this matrix and the limiting distribution probabilities [7]. Before stating the theorem, we give some definitions.

Definition 2.3 Let T_{ij} be the first time instance at which that the Markov process visits state j starting from state i. A state i is called *recurrent* if $P(T_{ii} < \infty) = 1$. A state i is called transient if $P(T_{ii} < \infty) < 1$. A recurrent state is said to be *positive recurrent* if $E(T_{ii}) < \infty$, where $E(T_{ii})$ is the expectation of T_{ii}.

Definition 2.4 A recurrent state i is said to be *periodic* with period d if $d > 1$ is the greatest common divisor of all t_n, which is the nth time instance at which the Markov process returns state i if it starts from state i in time 0. If there is no such d, the state is called *aperiodic*.

Definition 2.5 State j is said to be *accessible* from state i if j can be reached from i within finite time, which is denoted as $i \rightarrow j$. If $i \rightarrow j$ and $j \rightarrow i$, they are said to be *communicate*, which is denoted as $i \leftrightarrow j$. The set of all states of a Markov process that communicate with each other forms a *communicating class*. If the set of all states of a stochastic process $\mathbf{X}$ form a single communicating class, then $\mathbf{X}$ is *irreducible*.

Definition 2.6 A Markov chain is called *ergodic* if it is irreducible, positive recurrent, and aperiodic.

Theorem 2.1 ([12]) If the Markov process is irreducible, then the limiting distribution $\lim_{t\rightarrow\infty} p_i(t) = p_i$, $i \in \mathbf{S}$, exists and is independent of the initial conditions of the process. The limits $\{p_n \mid n \in \mathbf{S}\}$ are such that they either vanish identically (i.e., $p_i = 0$ for all $i \in \mathbf{S}$) or are all positive and form a probability distribution (i.e., $p_i > 0$ for all $i \in \mathbf{S}$, $\Sigma_{i\in \mathbf{S}} p_i = 1$). Furthermore, the limiting distribution $\{p_i, i \in \mathbf{S}\}$ of an irreducible positive recurrent Markov process is given by the unique solution of the equation: $\boldsymbol{p}\mathbf{G} = 0$ and $\Sigma_{j\in \mathbf{S}} p_j = 1$, where $\boldsymbol{p} = (p_0, p_1, p_2, \ldots)$.

B. Continuous-time Markov decision processes

We now give a brief introduction to *continuous-time Markov **decision** processes*. For the discussions in the rest of this paper, we will omit the term "continuous-time" for a briefer description. Unless otherwise stated, all processes are assumed to be continuous-time.

First, we describe a Markov process with *reward*. Assume the system earns a reward at rate $r_{i,i}$ (per unit time) during the time that it occupies state i. When it makes a transition from state i to state j ($i \neq j$), it receives a reward of $r_{i,j}$. Note that $r_{i,i}$ and $r_{i,j}$ have different dimensions. It is not necessary that the system earns according to both reward rates and transition rewards, but these definitions give us generality. We define the "earning rate" of state i as:

$$r_i = r_{i,i} + \sum_{j\neq i} \sigma_{i,j} r_{i,j} \tag{2.5}$$

Let $v_i(t)$ be the expected total reward that the system will earn during a time period of t if it starts in state i. The total expected reward during a time period of $t+dt$, that is $v_i(t+dt)$, can be written as:

$$v_i(t+dt) = (1 - \sum_{j\neq i} \sigma_{i,j} dt)[r_{i,i} dt + v_i(t)] + \sum_{j\neq i} \sigma_{i,j} dt[r_{i,j} + v_j(t)] \tag{2.6}$$

Eqn. (2.6) may be interpreted as follows. During the time interval dt the system may remain in state i or make a transition to some other state j. If it remains in state i for a time dt, it will earn a rate $r_{i,i}dt$ plus the expected reward that it will earn in the remaining t units of time, $v_i(t)$. The probability that it remains in state i for a time dt is $(1 - \sum_{j\neq i} \sigma_{i,j} dt)$. On the other hand, the system may make a transition to some state $j \neq i$ during the time interval dt with probability $\sigma_{i,j}dt$. In this case the system would receive the reward $r_{i,j}$ plus the expected reward to be made if it starts in state j with time t remaining, $v_j(t)$. The product of probability and reward must then be summed over all states $j \neq i$ to obtain the total contribution to the expected values.

With $dt \rightarrow 0$ and using the definition of earning rate r_i, we have:

$$\frac{d}{dt}v_i(t) = r_i + \sum_{j=1}^{n} \sigma_{i,j} v_j(t) \qquad i = 1, 2, \ldots, n \tag{2.7}$$

where n is the total number of states of the process. Eqn. (2.7) gives a set of linear, constant coefficient differential equations that relate the total reward in time t from a starting state i to the values of r_i and $\sigma_{i,j}$.

Second, a *controllable Markov process* is a Markov process whose state transition rates can be controlled by controlling commands (defined as *action*s). When the system is in state i, an action a_i is chosen from a finite set A_i, which includes all possible actions in state i. We denote this state-action relation by $<i,a_i>$. If the chosen action changes as the time changes, we denote the action as a time-dependent variable $a_i(t)$. Hence the state-action pair is written as $<i,a_i(t)>$.

Definition 2.7 A policy $\boldsymbol{\pi}$ is the set of state-action pairs for all the states of a controllable Markov process.

A policy can be either deterministic or randomized. If the policy is deterministic, then when the system is in state i at time t, an action $a_i(t)$ is chosen with probability 1. We denote a deterministic policy as: $\boldsymbol{\pi}=\{ <i,a_i(t)> \mid a_i(t)\in A_i, 1\le i\le n\}$. If the policy is randomized, then when the system is in state i at time t, an action a_i is chosen with probability $p_i^{a_i}(t)$, such that $\sum_{a_i\in A_i} p_i^{a_i}(t) = 1$. We denote a randomized policy as: $\boldsymbol{\pi}=\{ <i,\ \boldsymbol{p}_i^{A_i}(t)> \mid 1\le i\le n\}$, where $\boldsymbol{p}_i^{A_i}(t)$ is a vector of $p_i^{a_i}(t)$ for all $a_i\in A_i$. Notice that the deterministic policy is a special case of randomized policy with one of the $p_i^{a_i}(t)$ equals 1.

In a controllable Markov process, the state transition rates $\sigma_{i,j}$ have different values when different actions are taken. In a deterministic policy, we denote it as $\sigma_{i,j}^{a_i(t)}$. In a randomized policy, we denote it as $\sigma_{i,j}^{p_i^{A_i}(t)}$, and $\sigma_{i,j}^{p_i^{A_i}(t)} = \sum_{a_i\in A_i} \sigma_{i,j}^{a_i} p_i^{a_i}(t)$. As a result, the generator matrix of a controllable Markov process is a parameterized matrix (action is the parameter). A *Markov decision process* is a controllable Markov process with rewards. In a Markov decision process, since $\sigma_{i,j}$ is action-dependent, the reward rate r_i also becomes action-dependent and will thus be denoted as $r_i^{a_i(t)}$ for a deterministic policy and $r_i^{\boldsymbol{p}_i^{A_i}(t)}$ for a randomized policy, $a_i(t)\in A_i$. The expected total reward $v_i(t)$ depends on the chosen action in each state, i.e., it becomes policy-dependent and will be denoted as $v_i^{\boldsymbol{\pi}}(t)$. An example of a Markov decision process is given in Example 2.1.

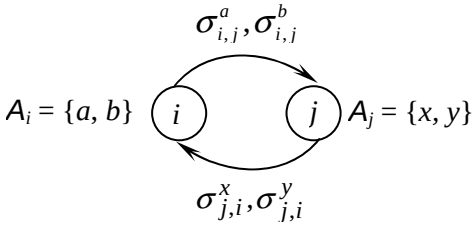


Figure 2 Example of a controllable Markov process.

Example 2.1 Consider a controllable Markov process that consists of only two states i and j (see Figure 2). When the system is in state i, the set of available actions is denoted as A_i; we assume that $A_i=\{a, b\}$. The transition rate from state i to state j equals $\sigma_{i,j}^{a}$ when action a is taken and $\sigma_{i,j}^{b}$ when action b is taken. Similarly, we define $A_j=\{x, y\}$, $\sigma_{j,i}^{x}, \sigma_{j,i}^{y}$. If we always take action a when the system is in state i and always take action x when the system is in state j, we can write the resulting policy as: $\boldsymbol{\pi}=\{<i, a>, <j, x>\}$. The system generator matrix using

policy $\boldsymbol{\pi}$ can be written as $\mathbf{G}=\begin{bmatrix} -\sigma_{i,j}^{a} & \sigma_{i,j}^{a} \\ \sigma_{j,i}^{x} & -\sigma_{j,i}^{x} \end{bmatrix}$. Now consider a randomized policy $\boldsymbol{\pi}'$={<*i*, (0.5, 0.5)>, <*j*, (0.4, 0.6)>}. This means that when the system is in state *i*, action *a* is taken with probability 0.5 and action *b* is taken with probability 0.5; when the system is in state *j*, action *x* is taken with probability 0.4 and action *y* is taken with probability 0.6. Therefore $\sigma_{i,j}^{\boldsymbol{p}_i^{A_i}} = 0.5\sigma_{i,j}^{a} + 0.5\sigma_{i,j}^{b}$ and $\sigma_{j,i}^{\boldsymbol{p}_j^{A_j}} = 0.4\sigma_{j,i}^{x} + 0.6\sigma_{j,i}^{y}$. The system generator matrix is constructed based on these two values.

أ

For the remainder of this paper, we will only use the notation for a deterministic policy.

Let $p_{i\Rightarrow j}^{\boldsymbol{\pi}}(t)$ denote the probability of being in state *j* at time *t* when the initial state at time 0 is *i* and the state transition rates are determined by policy $\boldsymbol{\pi}$. Similarly, let $r_j^{a_j(t)}$ denote the earning rate in state *j* at time *t* and action a_j is taken. The total expected reward that the process can earn for a time period of *t* using policy $\boldsymbol{\pi}$ can be written as ([14]):

$$v_i^{\boldsymbol{\pi}}(t) = \int_0^t \sum_{j=1}^{n} p_{i\Rightarrow j}^{\boldsymbol{\pi}}(\tau) r_j^{a_j(\tau)} d\tau \qquad (2.8)$$

Given two policies $\boldsymbol{\pi}_1$ and $\boldsymbol{\pi}_2$, if we can find a time ξ, such that $v_i^{\boldsymbol{\pi}_1}(t) \geq v_i^{\boldsymbol{\pi}_2}(t)$, for all $t>\xi$, i = 1, 2, …, n, then we say that policy $\boldsymbol{\pi}_1$ is superior to $\boldsymbol{\pi}_2$, denoted as $\boldsymbol{\pi}_1 \geq \boldsymbol{\pi}_2$. A policy $\boldsymbol{\pi}$ is *optimal*, if it is superior to all other policies for the Markov decision process.

Let $v_i^{\boldsymbol{\pi}} = \lim_{t\to\infty} v_i^{\boldsymbol{\pi}}(t)$. The goal of a Markov decision process is to find the optimal policy that maximizes $v_i^{\boldsymbol{\pi}}$ for all *i*. However, in practice, we cannot use $v_i^{\boldsymbol{\pi}}$ directly for calculating the optimal policy since $v_i^{\boldsymbol{\pi}}(t)$ approaches infinity when *t* approaches infinity. A commonly used alternative quantity is the *limiting average reward*:

$$v_{i,avg}^{\boldsymbol{\pi}} = \lim_{t\to\infty} \frac{1}{t} \int_0^t \sum_{j=1}^{n} p_{i\Rightarrow j}^{\boldsymbol{\pi}}(\tau) r_j^{a_j(\tau)} d\tau \qquad (2.9)$$

Obviously, maximizing the limiting average reward for any fixed *t* is the same as maximizing $v_i^{\boldsymbol{\pi}}$.

Definition 2.8 A policy $\boldsymbol{\pi}$ is *stationary* if the action is only a function of the state and independent of time, that is, $\boldsymbol{\pi}$={ $<i,a_i> \mid a_i \in A_i, 1\leq i\leq n$} for a deterministic policy or $\boldsymbol{\pi}$={ $<i, \boldsymbol{p}_i^{A_i}> \mid 1\leq i\leq n$} for a randomized policy.

Definition 2.9 A policy $\boldsymbol{\pi}$ is *piecewise stationary* if for any τ, interval $[0, \tau)$ can be divided into a finite number of intervals $[0, t_1)$, $[t_1, t_2)$, …, $[t_{m-1}, \tau)$ such that the policy is stationary inside each interval.

Theorem 2.2 [14] There exists a stationary policy that maximizes $v_{i,avg}^{\boldsymbol{\pi}}$ over the class of piecewise-stationary policies.

Based on this theorem, we conclude that we do not lose generality if our search for the optimum policy is restricted to the set of stationary policies. Therefore, actions and policies that we will discuss from now on are all time-independent and we will reduce the notation $a_j(t)$ and $p_i^{A_i}(t)$ to a_i and $p_i^{A_i}$.

The goal of a Markov decision process is to find a policy that maximizes the expected reward. In our case, we want to find a policy that minimizes our cost function (i.e., power dissipation). These two problems become equivalent if we use the negative of cost as the reward. In the remainder of the paper, we will use the term *cost* instead of *reward* and use $c_{i,i}$ and $c_{i,j}$ instead of $r_{i,i}$ and $r_{i,j}$, $c_{i,avg}$ instead of $v_{i,avg}$. In our work, we actually used a

constraint Markov decision process to model the power-managed system. In the constraint Markov decision process, for each state, there is an object cost *c_obj* and several constraint costs *c_con*. The goal of the constraint Markov decision process is to find a policy that minimizes the expected value of objective cost while meeting the given constraint. That is:

$$\text{Minimize}_{\pi}(c\_obj^{\pi}_{i,avg}), \text{ s.t. } c\_con^{\pi}_{i,avg} \le Constraint$$

The value of $C\_obj^{\pi}_{i,avg}$ and $C\_con^{\pi}_{i,avg}$ can be calculated based on *c_obj* and *c_con* using Eqn. (2.9). We will introduce how we define and calculate the objective cost and the constraint cost in our system model in Section V.

III. Component modeling

In this section, we describe the mathematical models of the components in a power-managed system.

A. Assumptions

The power-managed system consists of four components: a server that processes requests with different power modes (SP), a generator that generates the service request (SR), a queue which stores the requests that cannot be immediately serviced upon arrival (SQ), and a power manager (PM) that issues mode-switching commands to the SP. The SR is independent of the rest of the system. Requests generated by the SR can be divided into two categories: *low-priority requests* and *high-priority requests*, which are generated independently of each other.

Both the request arrival events and the request service completion events are stochastic processes and follow the Poisson distribution. When we state that the request arrival event is a Poisson process, it means that during time (0, *t*] the number of the events follows a Poisson distribution with mean λt. Consequently, the request inter-arrival time follows an exponential distribution with mean $1/\lambda$. Notice a request that arrives when the SQ is full will be rejected. We also assume that the time that is needed for the SP to switch from one state to another follows an exponential distribution. In reality, the switching time for the SP is usually a small fixed value. We know that if the expected value of an exponentially distributed random variable is *a* then its variance is a^2. If *a* is very small, then the variance will be negligible. Therefore, if the switching time of the SP is much shorter than the service time or input generation time then it can be modeled by an exponential distribution without introducing much error.

In the remainder of this paper, we will use upper case bold letters (*e.g.*, **M**) to denote matrices, lowercase bold letters (*e.g.*, **v**) to denote vectors, italicized Arial-Font letters (*e.g.*, $\mathcal{S}$) to denote sets, uppercase italicized letters (*e.g.*, *S*) to denote scalar constants, and lower case italicized letters (*e.g.*, *x*) to denote scalar variables.

B. Model of the Service Provider

*The **Service Provider** (SP) is modeled as a stationary, continuous-time Markov decision process with state (operation mode) set $\mathcal{S}=\{s_i \text{ s.t. } i=1, 2, \ldots, S\}$, action set $\mathcal{A}$, and parameterized generator matrix $\mathbf{G}_{SP}(a)$, $a \in \mathcal{A}$. It can be described by a quadruple (χ, $\mu(s)$, pow(s), ene(s_i, s_j)) where (i) χ is an $S \times S$ matrix; (ii) $\mu_l(s)$ and $\mu_h(s)$ are functions, μ_l, μ_h: $\mathcal{S} \to \mathbf{R}$; (iii) pow(s) is a function, pow: $\mathcal{S} \to \mathbf{R}$; (iv) ene($s_i$, s_j) is a function, ene: $\mathcal{S} \times \mathcal{S} \to \mathbf{R}$.*

We call χ the switching speed matrix of the SP. The (*i*,*j*)th entry of χ is denoted by χ_{s_i,s_j} and represents the switching speed from state s_i to state s_j. The average switching time from state s_i to state s_j is then $1/\chi_{s_i,s_j}$. We set χ_{s_i,s_i} to be ∞ because the switch from state s_i to itself is instantaneous.

The entries of the parameterized generator matrix $\mathbf{G}_{SP}(a)$ can be calculated as:

$$\sigma_{s_i,s_j}(a) = \delta(s_j, a) \cdot \chi_{s_i,s_j}, \; s_i \neq s_j; \tag{3.1}$$

$$\sigma_{s_i,s_i}(a) = -\sum_{s_j \neq s_i} \sigma_{s_i,s_j}(a) \tag{3.2}$$

$$\text{where } \delta(s,a) = \begin{cases} 1 & \text{if } s \text{ is the destination state of action } a \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The *service rates* $\mu_l(s)$ and $\mu_h(s)$ represent the service speed of the SP for low-priority requests and high-priority requests in state s, respectively. Therefore, $1/\mu_l(s)$ ($1/\mu_h(s)$) gives the average time that is needed by the SP to complete the service for a low (high) priority request when it is in state s.

A *power consumption value* $pow(s)$ is associated with each state $s \in S$. It represents the power consumption of the SP during the time it occupies state s. The cost rate $c_{s,s}$ of state s is equal to $pow(s)$.

A *switching energy value* $ene(s_i, s_j)$ is associated with each state pair (s_i, s_j), $s_i, s_j \in S$, $s_i \neq s_j$. It represents the energy needed for the SP to switch from state s_i to state s_j. The cost c_{s_i,s_j} is equal to $ene(s_i, s_j)$.

From Eqn. (2.5), we know that the expected power consumption (cost rate) of the SP when it is in state s and action a_s is chosen can be calculated as:

$$c_s = pow(s) + \sum_{s' \neq s} \sigma_{s,s'}(a_s) ene(s,s') .$$

In reality, states (i.e., working modes) of the SP can be divided into three groups: busy, idle, and power-down. In busy states, the SP is fully powered and working on the first request in the SQ. We assume that each request service is atomic so that the SP cannot switch to any other state when it is working on some request. In other words, the switch from the busy state to another state is not controllable. It only occurs when the SP finishes one service. For each busy state, there exists a corresponding idle state. In the idle states, the SP is fully powered, but it is not working on any request. An idle state is the only state that connects to its corresponding busy state. When the SP finishes a service, it will automatically switch from the busy state to its corresponding idle state. When SP wants to switch from some other state to a busy state, it first switches to the corresponding idle state then goes to the busy state. Notice that the idle states are not physical states of SP. When the SP is in an idle state, it is in the same power mode as when it is in the corresponding busy state. We only use the idle state to make our modeling convenient. In power-down states the SP is partially or completely shut down, i.e., it is not operational. The SP may have multiple power-down states (e.g., standby, soft off, or hard off).

Not all actions in A are valid in all SP states. Constraints on a valid action can be stated as follows:

1. When the SP is in a busy state, its transition is not controllable, $A_{busy}=\Phi$.
2. The action cannot make a transition from a power-down state to a busy state.
3. The action cannot make a transition from an idle state to a busy state other than its corresponding busy state.

Definition 3.1 Power-down state s_1 is more *vigilant* than inactive state s_2 if the SP in state s_1 wakes up (switches to an active state) faster than the same SP does in state s_2.

Different busy states may be used to model a component working under different supply voltages. We associate power and delay (service rate) values to each of these states to model the server performance under different supply voltages. Therefore, our policy optimization approach (cf. Section V) can also find the best policy for *dynamic voltage scaling* as it finds the optimal policy for power management.

Example 3.1 Consider a SP with six states, $S=\{busy_1, busy_2, idle_1, idle_2, wait, sleep\}$. When the SP is in state $busy_1$, it services the requests at a low speed. Assume that the average time needed for each service (for both low-priority requests and high-priority requests) is 5 ms. Therefore, $\mu_l(busy_1)$ and $\mu_h(busy_1)$ are 0.2. Also assume that in state $busy_2$, the SP services the request at a higher speed, e.g., $\mu_l(busy_2)=\mu_h(busy_2)=0.4$. $\mu_l(idle_1)$, $\mu_h(idle_1)$, $\mu_l(idle_2)$, $\mu_h(idle_2)$, $\mu_l(wait)$, $\mu_h(wait)$, $\mu_l(sleep)$, and $\mu_h(sleep)$ are all 0. Let the command set be defined as

A={*go_busy*$_1$, *go_busy*$_2$, *go_idle*$_1$, *go_idle*$_2$, *go_wait*, *go_sleep*}. Notice that not all four commands are valid (or available) in all states. The switching speed matrix χ is given by:

$$\chi = \begin{bmatrix} \infty & 0 & 0.2 & 0 & 0 & 0 \\ 0 & \infty & 0 & 0.4 & 0 & 0 \\ \infty & 0 & \infty & 10 & 1 & 0.5 \\ 0 & \infty & 10 & \infty & 1 & 0.5 \\ 0 & 0 & 0.454 & 0.454 & \infty & 1.5 \\ 0 & 0 & 0.166 & 0.166 & 1.5 & \infty \end{bmatrix}$$

By default, the order of the states in the rows and columns of this matrix are the same as the left-to-right order of the states in S. χ_{s_i,s_i} =∞ means that the SP can transfer from state s_i to s_j immediately. χ_{s_i,s_j} =0 means that the SP can never transfer from state s_i to s_j. In this example, the transfer from any state to itself needs no time. The SP can transfer from the *busy* state to the *idle* state with the transition rate equal to the service rate because it autonomously goes to the *idle* state immediately after it finishes a request. The SP cannot switch between the *busy* state and the *wait* state (or *sleep* state) directly (it must go through the *idle* state). Therefore the corresponding entries in the matrix are 0.

The power consumption is: *pow*(*busy*$_1$)=2.3W, pow(*busy*$_2$)=6.5W, *pow*(*idle*$_1$)=2.3W, *pow*(*idle2*)=6.5W, *pow*(*wait*)=0.8W, and *pow*(*sleep*)=0.1W.

The switching energy *ene*(s_i, s_j) matrix is:

$$ene(s_i, s_j) = \begin{bmatrix} 0 & \infty & 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & 0 & \infty & \infty \\ 0 & \infty & 0 & 0.3mJ & 1mJ & 2mJ \\ \infty & 0 & 0.3mJ & 0 & 1mJ & 2mJ \\ \infty & \infty & 4.4mJ & 4.4mJ & 0 & 0.66mJ \\ \infty & \infty & 30mJ & 30mJ & 9mJ & 0 \end{bmatrix}$$

ene(s_i, s_j)=∞ means that the SP cannot switch between the corresponding states. Note that the energy cost of the autonomous state change (busy to idle) is zero.

A graphical illustration of the SP is shown in Figure 3. The transition rates associated with the edges have not been shown in the figure. They can be extracted from $\mathbf{G}_{SP}(a)$ for specific actions. The self-transitions of each state are not shown in the figure.

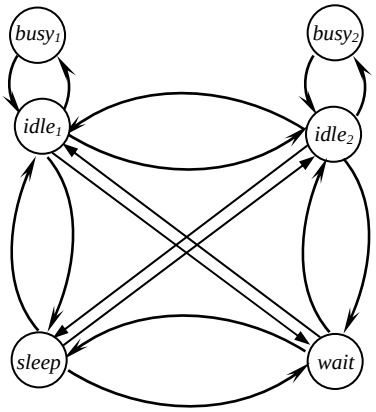


Figure 3 Markov process model of the SP.

C. Model of the Service Requester

*The **Service Requester** (SR) is modeled as a stationary, continuous-time Markov process with state set $R=\{r_i$ s.t. $i=0, 1, 2, \ldots, R\}$ and generator matrix $\mathbf{G}_{SR}$. It can be characterized by a pair $(\boldsymbol{\tau}, \lambda(r))$, where (i) $\boldsymbol{\tau}$ is an $R\times R$ matrix and (ii) $\lambda_l(r)$ and $\lambda_h(r)$ are functions $\lambda: R \rightarrow \mathbf{R}$.*

We call $\boldsymbol{\tau}$ the switching speed matrix of the SR. The (i,j)th entry of $\boldsymbol{\tau}$ is denoted as τ_{r_i,r_j}. We assume that the time needed for the SR to switch from one operation state to another is a random variable with exponential distribution. The average switch time from state r_i to state r_j is given by $1/\tau_{r_i,r_j}$. We set τ_{r_i,r_i} to be ∞ because the switch from state r_i to r_i is instantaneous. The SR model is a continuous-time Markov process with the generator matrix $\mathbf{G}_{SR}$. The value of σ_{r_i,r_j} (the transition rate from state r_i to state r_j) can be calculated as:

$$\sigma_{r_i,r_j} = \tau_{r_i,r_j},\ r_i \neq r_j;\ \sigma_{r_i,r_i} = -\sum_{r_j \neq r_i} \sigma_{r_i,r_j} \tag{3.4}$$

The *request rates* $\lambda_l(r)$ and $\lambda_h(r)$ are associated with state $r \in R$. When the SR is in state r, the generation of the low-priority requests follows the Poisson process with mean value $1/\lambda_l(r)$, whereas the generation of the high-priority requests follows the Poisson process with mean value $1/\lambda_h(r)$.

Example 3.2 Consider an SR with two states, r_1 and r_2. When it is in state r_1, it generates a low priority request every 30 ms on the average and a high priority request every 50 ms on the average. When it is in state r_2, it generates a low priority request every 60 ms and a high priority request every 90 ms. So the request rates in each state are defined as follows: $\lambda_l(r_1)=1/30$, $\lambda_h(r_1)=1/50$, $\lambda_l(r_2)=1/60$, and $\lambda_h(r_2)=1/90$. The switch matrix τ is a 2×2 matrix with one entry for each state pair. For instance:

$$\boldsymbol{\tau} = \begin{bmatrix} \infty & 1/200 \\ 1/400 & \infty \end{bmatrix}.$$

This means that when the SR is in state r_2 the expected time that it will switch to state r_1 is 200 ms, and when the SR is in state r_1 the expected time that it will switch to state r_2 is 400 ms. Therefore, the generator matrix of SR is:

$$\mathrm{G}_{\mathrm{SR}} = \begin{bmatrix} -1/200 & 1/200 \\ 1/400 & -1/400 \end{bmatrix}.$$

Figure 4 gives the illustration of the Markov process model of this SR. The self-transitions of the states are not shown.

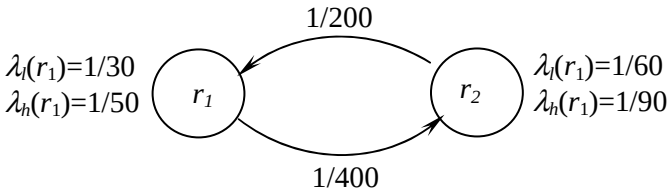


Figure 4 Markov process model of the SR.

D. Model of the Service Queue

*A **Single Service Queue** (SSQ) is modeled as a stationary, continuous-time Markov process, with state set $Q_{SSQ}=\{q_i, i=0, 1, 2, \ldots, Q\}$ and the generator matrix $\boldsymbol{G}_{SSQ}(s, r)$, where Q is the maximum length of the queue, s is the state of SP, and r is the state of SR.*.

The number of waiting requests in the queue decides the state of the SSQ. If there are *i* requests waiting in the queue, then the queue is in the state of q_i. The entries of the parameterized generator matrix $G_{SSQ}(s, r)$ can be calculated as:

$$\sigma_{q_i,q_{i+1}} = \lambda(r),\ \sigma_{q_{i+1},q_i} = \mu(s),\ \sigma_{q_i,q_i} = -\sum_{q_j \neq q_i} \sigma_{q_i,q_j},\ 0 \le i \le Q-1$$

$$\sigma_{q_i,q_j} = 0, \text{ for other state pairs}$$

$\lambda(r)$ and $\mu(s)$ are the request input rate and service rate of the queue. The shortcoming of using SSQ as the stochastic model of the service queue is that we can assign only one delay constraint (i.e., the constraint on the average waiting time of the requests) during the policy optimization. In a power-managed system, the SP, under control of the PM, may thus not service an incoming request immediately in order to achieve better power-delay trade-off. However, there may exist high-priority requests that need immediate service by the SP. In this case, if we use a loose delay constraint (which means the power management policy may not service the request immediately) the consequent long latency is not acceptable for high-priority requests. If we instead use a tight delay constraint to ensure that the high-priority requests are serviced immediately, then there will be undesirable power dissipation related to an unnecessarily tight delay constraint on low-priority requests.

We henceforth model the service queue as a combination of two SSQs: one (denoted as HSQ) for the high-priority requests and the other (denoted as LSQ) for the low-priority requests. The relationships between these two queues are:

1. Two different delay constraints are assigned to the HSQ and the LSQ separately such that the requests in the HSQ have a smaller waiting time than those in the LSQ.
2. The SP will not start serving the requests in the LSQ until it finishes all the requests in the HSQ. Therefore, we define the service rate of the LSQ as a function that relates to both the state of the SP and the state of the HSQ: $\mu'_l(s, hq_i)$, where *s* is the SP state and hq_i is the HSQ state. If i=0 then $\mu'_l(s, hq_0)=\mu_l(s)$, otherwise, $\mu'_l(s, hq_i)$=0.

Although we have introduced two queues in our stochastic model of the service queue, we are actually modeling a single priority queue. The SQ model can be used to model the commonly used priority queue in an operating system where two different priorities are assigned to tasks, and high-priority tasks, when they come, are inserted into the front of the queue. Moreover, obviously, the SQ model can be extended to model a queue of requests that have more than two priority levels.

The formal definition of the SQ model is as follows.

*The **Service Queue** (SQ) is modeled as a stationary, continuous-time Markov process, which is the combination of two SSQs: LSQ and HSQ. The state set of the SQ is given by $Q= Q_{LSQ} \times Q_{HSQ}$, and the generator matrix is given by $\boldsymbol{G}_{SQ}(s, r)= \boldsymbol{G}_{LSQ}(s, r) \oplus \boldsymbol{G}_{HSQ}(s, r, hq)$, where s is the state of SP, r is the state of SR state, and the "$\oplus$" operation is the tensor sum defined in Definition 4.1.*

The number of waiting requests in the HSQ and LSQ decides the state of the SQ. If there are *i* requests waiting in the LSQ and *j* requests waiting in the LSQ, then the queue is in the state of (lq_i, hq_j). The entries of both the parameterized generator matrix $\mathbf{G}_{HSQ}(s, r)$ and $\mathbf{G}_{LSQ}(s, r, hq)$ are calculated in the same way as $\mathbf{G}_{SSQ}(s, r)$ except that they use $\lambda_l(r)$, $\mu'_l(s, hq)$ and $\lambda_h(r)$, $\mu_h(s)$ as the input rate and service rate, respectively.

Example 3.3 Consider an HSQ and an LSQ. Assume that the maximum length of the HSQ is 1, the maximum length of the LSQ is 2. The generator matrices of the HSQ and LSQ are:

$$G_{HSQ}(s,r)=\begin{bmatrix}-\lambda_h(r) & \lambda_h(r)\\ \mu_h(s) & -\mu_h(s)\end{bmatrix},\ G_{LSQ}(s,r,hq)=\begin{bmatrix}-\lambda_l(r) & \lambda_l(r) & 0\\ \mu_l'(s,hq) & -\mu_l'(s,hq)-\lambda_l(r) & \lambda_l(r)\\ 0 & \mu_l'(s,hq) & -\mu_l'(s,hq)\end{bmatrix}.$$

The generator matrix of the SQ is:

$$G_{SQ}(s,r)=\begin{bmatrix}\sigma_{0,0} & \lambda_h(r) & \lambda_l(r) & 0 & 0 & 0\\ \mu_h(s) & \sigma_{1,1} & 0 & \lambda_l(r) & 0 & 0\\ \mu_l(s) & 0 & \sigma_{2,2} & \lambda_h(r) & \lambda_l(r) & 0\\ 0 & 0 & \mu_h(s) & \sigma_{3,3} & 0 & \lambda_l(r)\\ 0 & 0 & \mu_l(s) & 0 & \sigma_{4,4} & \lambda_h(r)\\ 0 & 0 & 0 & 0 & \mu_h(s) & \sigma_{5,5}\end{bmatrix}$$

To save space, we did not write out the value of the diagonal entries.

أ

Request starvation may occur when high priority requests keep coming while there are some low priority requests waiting. However, this problem can be solved by the OS. If a low priority request has been waiting in the queue for too long, the OS will change its priority to high so that it will get serviced. Most of the existing operating system codes have a similar mechanism to prevent starvation. As a result, the power manager need not be concerned with this problem.

IV. System modeling

We first show how to construct the model of the entire system by combining the component models. Then we explain how the power-managed system model is applied to practical applications.

A. Model of the Power-Managed System

*The **Power-Managed System** (SYS) can be modeled as a controllable continuous-time Markov process, which is the composition of the models of the SP, the SR, and the SQ. The state set is given by: $X=S\times Q\times R$–{invalid states where SP is busy and SQ is empty}. A set of all possible actions, which is the same as A in the SP model, is given. A parameterized generator matrix $\boldsymbol{G}_{SYS}(a)$ gives the state transition rates under action a. The SYS state can be represented as (s, r, (lq, hq)), where $s\in S$, $r\in R$, $lq\in Q_{LSQ}$ and $hq\in Q_{HSQ}$.*

Similar to the case of the SP model, not all actions are valid for any system state. The action constraints (which are described in Section III.A) for the SP model still apply to the SYS model. In addition, we add the following constraints related to the SYS model.

(1) When both the LSQ and HSQ are full and the SP is in an inactive state, the SP cannot make a transition to another inactive state that is less vigilant (c.f. Definition 4.1) than the current one. This constraint is reasonable because the SP must go to a fully functional state as soon as possible in this situation.

(2) When both the LSQ and HSQ are full and the SP is in an idle state, the SP cannot make a transition to a power-down state or another idle state whose corresponding busy state has a slower service rate. This constraint is also reasonable because when the SP and SQ are in the above states, it means that the service speed is not enough to match the incoming speed of the requests. Therefore, we need to increase the service rate, not decrease it.

Proposition 4.1 If the SYS satisfies the above two constraints, there is only one ergodic chain in the system, and the states outside the ergodic chain are all transient states.

B. Calculating the generator matrix

We next introduce an efficient method for calculating the generator matrix $\mathbf{G}_{SYS}(a)$ of the system from the generator matrices of the system components: $\mathbf{G}_{SP}(a)$, $\mathbf{G}_{SR}$, and $\mathbf{G}_{SQ}(s, r)$.

First, we show how to calculate the generator matrix of a joint process of two independent continuous-time Markov processes. Proposition 4.2 gives a method to obtain the joint transition rate of two independent continuous-time Markov processes. Proposition 4.3 gives the method for generating the generator matrix of the joint system using matrix operations.

Proposition 4.2 *Given two independent stochastic processes X and Y, let $\sigma_{(x,y),(x'y')}$ denote the transition rate of the joint process from the joint state (x,y) to joint state (x′,y′), where x and x′∈ state space of X and y and y′ ∈ state space of Y. Let $\sigma_{x,x'}$ denote the transition rate of process X from state x to state x′ and $\sigma_{y,y'}$ denote the transition rate of process Y from state y to state y′. Then $\sigma_{(x,y),(x,y')} = \sigma_{y,y'}$, $\sigma_{(x,y),(x',y)} = \sigma_{x,x'}$, $\sigma_{(x,y),(x',y')} = 0$, and $\sigma_{(x,y),(x,y)} = \sigma_{x,x} + \sigma_{y,y}$.*

Let matrices A and B be defined as:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \text{and} \qquad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Definition 4.1 The **tensor product C=A⊗B** is given by $\mathbf{C} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} \end{bmatrix}$. The **tensor sum C=A⊕B** is given by: $\mathbf{C} = \mathbf{A} \otimes \mathbf{I}_{n_2} + \mathbf{I}_{n_1} \otimes \mathbf{B}$, where n_1 is the order of **A**, n_2 is the order of **B**, and $\mathbf{I}_{n_i}$ is the identity matrix of order n_i.

Proposition 4.3 *Given two independent continuous-time Markov processes with generator matrices **A** and **B**, the generator matrix of the joint process is given by **A**⊕**B**.*

We have mentioned in Section III.A that the SR is independent from the rest of the system. Therefore, $\mathbf{G}_{SYS}(a)$ can be calculated as:

$$\mathbf{G}_{SYS}(a) = \mathbf{G}_{SP\text{-}SQ}(a, r) \oplus \mathbf{G}_{SR} \qquad (4.1)$$

where $\mathbf{G}_{SP\text{-}SQ}(a, r)$ is the generator matrix of the joint process of SP and SQ. Notice that $\mathbf{G}_{SYS}(a)$ generator matrix is also a parameterized matrix of action *a*.

The transition of the SP from a busy state to its corresponding idle state is correlated with the transition of the SQ from state (lq_i, hq_i) to state (lq_i, hq_{i-1}) or the transition of the SQ from state (lq_i, 0) to state (lq_{i-1}, 0). This is because whenever the SP makes a transition from a busy state to an idle state (finishes the service for a request), the SQ makes a transition from state (lq, hq) to state (lq', hq') where $lq+hq=lq'+hq'+1$. The other transitions of the SP and SQ are independent. Therefore, we can calculate each entry of $G_{SP\text{-}SQ}$ as this:

Let $\sigma_{x,x'}$ denote the transition rate for the transition from state $x=(s, (lq, hq))$ to state $x'=(s', (lq', hq'))$.

If *s* is a busy state, *s′* is the idle state corresponding to *s*, and $(lq+hq)-(lq'+hq')=1$, then $\sigma_{x,x'}$ equals $\sigma_{(lq,\ hq),(lq',\ hq')}$, which is the transition rate of the SQ from state (lq, hq) to state (lq', hq').

If *s* is a busy state and *s′* is the corresponding idle state, but $(lq+hq) - (lq'+hq') \neq 1$, then $\sigma_{x,x'}$ equals 0.

If *s* is a busy state and $(lq+hq)-(lq'+hq')=1$ but *s′* is not the corresponding idle state of *s*, then $\sigma_{x,x'}$ equals 0.

For all the other $\sigma_{x,x'}$, we can use the value of the corresponding entry of matrix $\mathbf{G}_{SP}(a) \oplus \mathbf{G}_{SQ}(s, r)$. Notice that, after the operation, parameter *s* in $\mathbf{G}_{SQ}(s, r)$ has been removed by substituting the real state of the SP. Therefore, the generator matrix of $\mathbf{G}_{SP\text{-}SQ}$ is parameterized matrix which depends on variables *a* and *r*.

The values of the diagonal entries of $G_{SP\text{-}SQ}$ need to be recalculated using Eqn. (2.4).

Example 4.1 Consider the SP and SQ models given in Examples 3.1 and 3.3. The generator matrix of their joint process can be calculated like this:

$\sigma_{(busy_1,(0,1)),(idle_1,(0,0))} = \sigma_{(0,1),(0,0)}(busy_1) = \mu_h(busy_1)$,

$\sigma_{(busy_1,(1,0)),(idle_1,(0,0))} = \sigma_{(busy_1,(2,0)),(idle_1,(1,0))} = \mu_l'(busy_1, 0) = \mu_l(busy_1)$,

$\sigma_{(busy_1,(1,1)),(idle_1,(0,0))} = \sigma_{(busy_1,(2,1)),(idle_1,(1,1))} = \mu_l'(busy_1, 1) = 0$,

$\sigma_{(busy_1,(0,0)),(idle_1,(0,1))} = \sigma_{(busy_1,(0,0)),(idle_1,(1,0))} = \sigma_{(busy_1,(0,0)),(idle_1,(1,1))} = \cdots = \sigma_{(busy_1,(2,1)),(1,0))} = 0$,

$\sigma_{(busy_1,q),(s',q')} = 0$, where $s' \neq idle_1$, $q, q' \in Q$.

The value of $\sigma_{(busy_2,q),(s',q')}$ can be calculated in the same way as $\sigma_{(busy_1,q),(s',q')}$. The values of the other $\sigma_{x,x'}$ entries equal the values of the corresponding entries of matrix $\mathbf{G}_{SP}(a) \oplus \mathbf{G}_{SQ}(s, r)$.

C. Application issues

Using the SYS model, a power-managed system in real application can work in the following way. When the SP of the system changes states, it sends an interrupt signal SWITCH_DONE to the PM. The PM then reads the states of all the components in the power-managed system (and hence obtains the joint system state) and issues a command according to the chosen policy. The SP receives the command and immediately starts to switch to a state that is dictated by the command. Notice that the command may ask the SP to switch to its current state; therefore the SP state does not change. We assume that after the SP finishes a service, it stays in the idle state for a period of time long enough for it to accept the command from the PM and to switch to another state.

V. Solution techniques

A. Problem formulation

The power management problem is to find the optimal policy (set of state-action pairs) for the PM such that the average system power dissipation is minimized subject to the performance constraints. The performance of a system is usually measured by the average delay of each request. We have the following theorem:

Theorem 5.1 In a power-managed system, if the loss rate of the request is small enough, then $D = Q \cdot \lambda$, where D is the average request delay, Q is the average number of waiting requests in the queue, and λ is the average request inter-arrival time. Furthermore, during any time period of length T, $E_T(d) = E_T(q) \cdot T / X$, where $E_T(d)$ and $E_T(q)$ denote the average request delay and the average number of waiting requests in the queue during time T, and X is the number of incoming requests of this system during time period T.

We define the objective cost of the constraint Markov decision problem $c\_obj_x^{a_x} = c\_pow(x, a_x)$, where $c\_pow(x, a_x)$ denotes the power consumption of the system when it is in state x and action a_x is used. In this work, we only consider the power consumption of the SP. We will explain how to calculate the objective cost in different solution approaches later in this section. We define two constraint cost $c_1\_con_x^{a_x} = c\_lsq(x)$, $c_2\_con_x^{a_x} = c\_hsq(x)$, $c\_lsq(x)$, and $c\_hsq(x)$ denotes the number of waiting requests in the low priority queue and high priority queue, respectively, when the system is in state x. Notice that, although the constraint cost for each state is policy independent, since the state probability for each state is policy dependent, the total expected constraint cost is policy dependent.

The problem can be formally described as:

Find a policy $\boldsymbol{\pi}$ to minimize: $\lim_{t\to\infty}\frac{1}{t}\int_0^t \sum_{x'\in X} p^{\boldsymbol{\pi}}_{x\Rightarrow x'}(\tau)\cdot c\_pow(x', a^{\boldsymbol{\pi}})d\tau$,

$$\text{s.t. } \lim_{t\to\infty}\frac{1}{t}\int_0^t \sum_{x'\in X} p^{\boldsymbol{\pi}}_{x\Rightarrow x'}(\tau)\cdot c\_lsq(x')d\tau \le D_L$$

$$\text{and } \lim_{t\to\infty}\frac{1}{t}\int_0^t \sum_{x'\in X} p^{\boldsymbol{\pi}}_{x\Rightarrow x'}(\tau)\cdot c\_hsq(x')d\tau \le D_H \ , \forall x\in X$$

where $p^{\boldsymbol{\pi}}_{x\Rightarrow x'}(\tau)$ is the state transition (direct or indirect) probability from state *x* to *x'* in a time period of τ under policy $\boldsymbol{\pi}$. D_H and D_L are performance constraints for the high priority queue and low priority queue.

B. The linear programming approach

If the delay constraint for the power management system is an *active constraint* [19], generally, the optimal power management policy will be a randomized policy [15]. The randomized optimal policy can be obtained by solving a linear programming problem. In this section, we first introduce how to solve the unconstraint continuous-time Markov decision process using a linear programming approach and then show how to apply this linear programming approach to our constraint power-management problem.

First we introduce the definition of some variables that are used to measure the Markov process. Let t_{ij} denote the time that the system needs to switch from state *i* to state *j*. $Q_{ij}(t)$ denotes the probability of the event that the next observed state is state *j* and that state *j* is observed no later than time *T*+*t*, given that state *i* is observed at time *T*. We know that $Q_{ij}(t)$ is the probability that t_{ij} is the smallest among all t_{il} and t_{ij} is less than *t*, where *l* is the possible next state of state *i*. We use $t_{ij}^{a_i}$ and $Q_{ij}^{a_i}(t)$ to denote the values of t_{ij} and $Q_{ij}(t)$ when action a_i is taken.

Proposition 5.1 If t_{ij} follows exponential distribution $f(t_{ij}) = \sigma_{ij}\cdot e^{-\sigma_{ij}t}$, $j\in J$, (J is the set of all possible next states of state *i*), then $Q_{ij}(t) = \sigma_{ij}(1-e^{-\sum_{l\in J}\sigma_{il}t})/\sum_{l\in J}\sigma_{il}$.

Let $\tau_i^{a_i}$ denote the expectation of the time that the system will be in state *i* if action a_i is chosen in this state; then $\tau_i^{a_i} = \sum_{j\in J}\int_0^\infty t dQ_{ij}^{a_i}(t)$, and J is the set of all possible next states of *i*. It is easy to show that, in a Markov process, $\tau_i^{a_i} = 1/\sum_{j\in J}\sigma_{ij}^{a_i}$. Let $p_{ij}^{a_i}$ denote the probability that the next system state is *j* if the system is currently in state *i* and action *k* is taken; thus $p_{ij}^{a_i} = Q_{ij}^{a_i}(\infty)$. In a Markov process we have $p_{ij}^{a_i} = \sigma_{ij}^{a_i}/\sum_{l\in J}\sigma_{il}^{a_i}$. Let $\gamma_i^{a_i}$ denote the expected cost of the system during the time it stays in state *i* and action a_i is taken; thus

$$\gamma_i^{a_i} = c_{ii}\tau_i^{a_i} + \sum_{j\in J} c_{ij}p_{ij}^{a_i} \qquad (5.1)$$

Let $x_i^{a_i}$ denote the frequency that the next state of the system will be *i* and action a_i will be taken if we take a random observation of the system. Then $x_i^{a_i}\tau_i^{a_i}$ is the probability that the system is in state *i* and action a_i is taken in a random observation, which is also called *state-action probability*. We know from the definition that

$p_i^{a_i} = x_i^{a_i}\tau_i^{a_i} / \sum_{a_i \in A_i} x_i^{a_i}\tau_i^{a_i}$, $p_i^{\boldsymbol{\pi}} = \sum_{a_i \in A_i} x_i^{a_i}\tau_i^{a_i}$, and $\sum_{i \in S}\sum_{a_i \in A_i} x_i^{a_i}\tau_i^{a_i} = 1$. We also know that if a Markov process is stationary, the input rate to each state needs to be equal to the output rate from that state [15]. That means $\sum_{a_i} x_i^{a_i} - \sum_j \sum_{a_i} x_j^{a_i} p_{ji}^{a_i} = 0$.

For a given policy, if we know that the resulting Markov process is irreducible, Theorem 5.2 shows that we can use the variables *x* and $\gamma$to calculate the limiting average cost.

Lemma 5.1 Given a Markov process, $\sum_{i \in S}\sum_{a_i \in A_i} x_i^{a_i}\gamma_i^{a_i} = \sum_{i \in S} p_i^{\boldsymbol{\pi}} c_i^{\boldsymbol{p}_i^{A_i}}$.

Lemma 5.2 If the Markov process is irreducible, there exists a $C_{avg}^{\boldsymbol{\pi}}, C_{i,avg}^{\boldsymbol{\pi}} = C_{avg}^{\boldsymbol{\pi}} \quad \forall i \in S$. Furthermore, $\sum_{i \in S} p_i^{\boldsymbol{\pi}} c_i^{\boldsymbol{p}_i^{A_i}} = C_{avg}^{\boldsymbol{\pi}}$.

Theorem 5.2 If a Markov process is irreducible, then $\sum_{i \in S}\sum_{a_i \in A_i} x_i^{a_i}\gamma_i^{a_i} = C_{i,avg}^{\boldsymbol{\pi}} \quad \forall i \in S$.

With the above-mentioned variables and their characteristics, we can write the linear programming as [15]:

LP1: $\text{Minimize}_{\{x_i^{a_i}\}} \left(\sum_i \sum_{a_i} x_i^{a_i}\gamma_i^{a_i}\right)$, $a_i \in A_i$ (5.2)

subject to $\sum_{a_i} x_i^{a_i} - \sum_j \sum_{a_j} x_j^{a_j} p_{ji}^{a_j} = 0 \quad i \in S$ (5.3)

$\sum_i \sum_{a_i} x_i^{a_i}\tau_i^{a_i} = 1$ (5.4)

$x_i^{a_i} \geq 0$ all i, a_i (5.5)

Since the action in each state is unknown in the Markov decision process, in general cases, we cannot guarantee that for every possible policy the resulting Markov process is irreducible. For those resulting Markov processes, expression (5.2) may not be equal to their average cost because Theorem 5.2 is no longer applicable. However we have the following theorem:

Lemma 5.3 [15] Let $\{x_i^{a_i}\}$ be a basic feasible solution to LP1. Then set $E = \{i, \sum_{a_i} x_i^{a_i} > 0\}$. $F = \{(i, a_i) : x_i^{a_i} > 0\}$ identifies a unique ergodic chain, and expression (5.2) is the cost rate for this chain.

Theorem 5.3 [15] LP1 is feasible and bounded. The positive variables in its optimal solution identify the states of an ergodic chain whose cost rate is minimized over all chains.

In our power management system, we set constraints on the action sets so that the Markov process model of the resulting power management system contains only one ergodic chain. Therefore, expression (5.2) gives the cost rate for the entire system, and the solution of LP1 identifies an optimal policy.

In our case, we add the delay constraint to the linear program, and formulate it as:

LP2: $\text{Minimize}_{\{x_i^{a_i}\}} \left(\sum_i \sum_{a_i} x_i^{a_i} c\_pow_i^{a_i}\right)$ (5.6)

subject to $\sum_{a_i} x_i^{a_i} - \sum_j \sum_k x_j^{a_j} p_{ji}^{a_j} = 0 \quad i \in X$ (5.7)

$$\sum_i \sum_{a_i} x_i^{a_i} \tau_i^{a_i} = 1 \tag{5.8}$$

$$x_i^{a_i} \geq 0 \quad \text{all } i, a_i \tag{5.9}$$

$$\sum_i \sum_{a_i} x_i^{a_i} c\_hsq_i^{a_i} < D_H \tag{5.10}$$

$$\sum_i \sum_{a_i} x_i^{a_i} c\_lsq_i^{a_i} < D_L \tag{5.11}$$

In LP2, $c\_pow_i^{a_i}$, $c\_hsq_i^{a_i}$, and $c\_lsq_i^{a_i}$ denote the power consumption of the system, the high priority request waiting cost, and the low priority request waiting cost during the time it stays in state i and action a_i is taken. They can be calculated based on (5.1): $c\_pow_i^{a_i} = pow_i \cdot \tau_i^{a_i} + \sum_j ene_{ij} \cdot p_{ij}^{a_i}$, $c\_hsq_i^{a_i} = hq_i \cdot \tau_i^{a_i}$, and $c\_lsq_i^{a_i} = lq_i \cdot \tau_i^{a_i}$. D_H and D_L are performance constraints for the high priority queue and the low priority queue. If constraints (5.10) and (5.11) are implied in constraints (5.7), (5.8), and (5.9), (i.e., they are inactive constraints), the resulting optimal policy must be a deterministic policy. Otherwise, the resulting optimal policy will be a randomized policy.

C. The nonlinear programming approach

Because a randomized policy may be undesirable in some applications, we are interested in finding the optimal deterministic policy instead of the optimal randomized policy. In a deterministic policy, for each state i, there is one and only one $x_i^{a_i}$, $a_i \in A_i$, which is non-zero. Therefore, we can formulate the problem of searching for an optimal deterministic policy into a nonlinear program NLP1, which has a polynomial objective function and a set of linear constraints. In NLP1 λ is an arbitrary large number.

NLP1: $\text{Minimize}_{\{x_i^{a_i}\}} \ (\lambda \sum_i \sum_{a_i \neq l, a_i, l \in A_i} x_i^{a_i} \cdot x_i^l + \sum_i \sum_{a_i} x_i^{a_i} c\_pow_i^{a_i})$

subject to $\sum_{a_i} x_i^{a_i} - \sum_j \sum_{a_i} x_j^{a_j} p_{ji}^{a_j} = 0 \quad i \in X$

$$\sum_i \sum_{a_i} x_i^{a_i} \tau_i^{a_i} = 1$$

$$x_i^{a_i} \geq 0 \quad \text{all } i, a_i$$

$$\sum_i \sum_{a_i} x_i^{a_i} c\_hsq_i^{a_i} < D_H, \qquad \sum_i \sum_{a_i} x_i^{a_i} c\_lsq_i^{a_i} < D_L$$

There are specific algorithms to solve this kind of nonlinear programming problem. A classical algorithm is the *feasible direction algorithm*, which can be found in [16].

D. The branch-bound approach

Another way to find the optimal deterministic policy is to use a branch and bound approach. In a power management system, making different decisions in one system state has a significant effect on the power and performance of the system. For example, given two policies, the first one chooses action "*Go_busy*" when the server is idle and there is 1 waiting request. However, the second one chooses action "*Go_sleeping*" when the system is in the same state. No matter how the actions are chosen in the other state, the best performance of the system using the second policy will be worse than that of the system using the first policy, because there is always at least one request waiting in the queue. This observation is the root cause of the efficiency of the branch and bound algorithm.

The optimal deterministic power-management policy decision tree is a full tree with X levels, where X is the number of states in the controllable Markov process. Each node in level x has A_x child nodes, where A_x is number of available actions in state x. Therefore for each leaf in the decision tree there is a corresponding deterministic power-management policy. We will search for the power-optimal performance-constrained policy based on this decision tree by using a branch and bound algorithm.

For each internal node in the decision tree, there is a partial-decision problem with the same constraints and objective as the original problem. We call it a partial-decision problem because its variables are a subset of those of the original problem. We can find an optimal randomized policy for this partial-decision problem using a linear programming approach. The power consumption of this policy is a lower bound on the power consumption of all policies in this branch. We define the *lower bound operator* as finding the optimal randomized policy for the partial decision problem; similarly, we define the *prune operator* as comparing the power consumption of that randomized policy with that of the best deterministic policy we already have. If the lower bound operator yields a solution that consumes lower power then the best deterministic solution found so far in any branch, then we continue with this branch; otherwise, we prune the branch. Furthermore, if we cannot find a policy that satisfies the performance constraint for the partial-decision problem, then we also prune the branch.

E. The policy iteration approach

The policy iteration algorithm is widely used in searching for an optimal policy without constraint [17]. It starts from a set of randomly selected actions for each state that is called the "initial policy." It then calculates the expected cost of the system under this policy, which is called the "reference cost." Using this reference cost, by some calculation we can find a new policy that has a lower cost than the initial policy. The process is iterated until the resulting policy cannot be improved further. Because of the mathematical nature of the policy optimization problem, this simple greedy algorithm yields the provably optimal solution. Experiments show that the "policy iteration" technique is a fast technique. For a system with 23 states and assuming that three commands are available for each state, the "policy iteration" algorithm can find out the optimal solution within 4 iterations. However, the policy iteration approach cannot be directly applied to the problem of constrained optimization. We make some modification to this approach.

We first consider the power-managed system with only single priority SQ. We define a joint cost as a weighted summation of the power and delay costs:

$$joint\_cost_x^{a_x} = c\_pow_x^{a_x} + w \cdot c\_sq_x \tag{5.12}$$

where $c\_pow_x^{a_x}$ denotes the average power consumption when the system is in state x and action a_x is chosen, c_sq_x is the number of waiting requests in the SQ when the system is in state x. Let x be denoted by (s, q_i), where $s \in S$, $q_i \in Q$. Using Eqn. (2.5) the power cost can be calculated as: $c\_pow_x^{a_x} = pow(s) + \sum_{s' \in S, s' \neq s} \sigma_{s,s'}(a_x) ene(s,s')$. The delay cost is: $c\_sq_x = i$. We start from a randomly selected weight w. Then we find a policy that has optimal *joint_cost* using the policy iteration algorithm. If the delay of this policy is approximately same as the given constraint, then we report this policy; otherwise, we increase or decrease the weight and continue searching using the new weight value.

This modified policy iteration algorithm is a heuristic algorithm. For a certain performance constraint, it may not find the policy that consumes the minimum power. However, we show later that its output policy is one that consumes minimum power among a class of *convex policies* that satisfy the performance constraint. We first give the definition of an *effective policy* and a *convex policy*:

Definition 5.1 An *effective policy* is a policy that consumes the minimum power compared to other policies that have the same or better performance.

Definition 5.2 We sort all the policies in the increasing order of their resulting power consumption and denote the power and delay of each policy p_i by pow_i and $delay_i$. A policy p_i is a *convex policy* if it is an effective policy and $(delay_i - delay_j)/(pow_j - pow_i) < (delay_l - delay_j)/(power_j - pow_l)$, $\forall j, j > i$ and $\forall l, l < i$.

Compared to the policies that consume less power, a convex policy has a better power-delay tradeoff (smaller ratio of increasing delay to the decreasing power). Figure 5 gives an illustration of a convex policy and an effective policy. In that figure, policies *a, b, c, d, e* are all effective policies. However, policy *c* is not a convex policy.

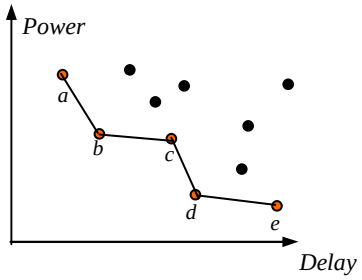


Figure 5 Example to describe policy types.

Theorem 5.4 The output of the modified policy iteration algorithm is a policy that consumes the minimum power among the class of convex policies that satisfy the performance constraint.

In a power management system model with multiple priority queues, the following cost function is used: $cost_x^{a_x} = c\_pow_x^{a_x} + w_1 \cdot c\_lsq_x + w_2 \cdot c\_hsq_x$. By adjusting both of the weight coefficients w_1, w_2, we can find the convex policy.

The policy iteration algorithm can be used as a fast algorithm to perform on-line policy optimization when system parameters change at runtime because it finds the optimal policy based on the previous optimal policy. It is more efficient than the branch and bound algorithm as well as the NLP approach which starts solving from the very beginning.

VI. Experimental results

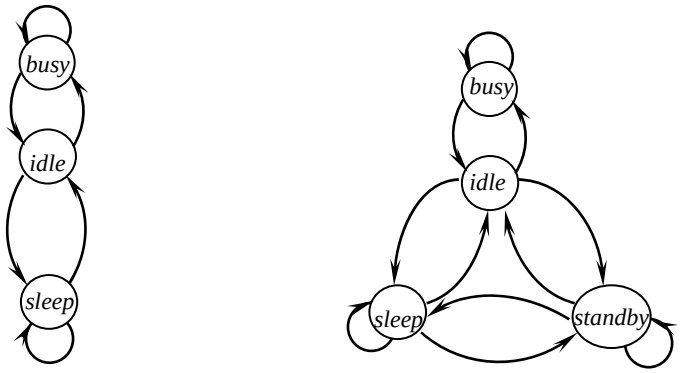
Experiments have been designed to examine the performance of our system model and optimization method.

A. A power-managed system with a single service queue

In this experimental setup, the service requestor (SR) is implemented as an input trace file that stores the time instances when the disk drive read/write requests arrive. The service queue (SQ) and service provider (SP) are implemented in the event-driven simulator. The power manager (PM) is also implemented in the simulator that controls the state transition of the SP based on the policy it reads from the user-specified input file. *Q*, the capacity of the SQ, is set to 20.

We will use two abstract models of the SP for the Fujitsu disk drive [24] in the experiments, which are shown in Figure 6.

When the SP is in the “sleep” and “standby” states, it is inactive, and therefore no request can be serviced. When the SP is in the “idle” state, the SP is active; however, it is not working on any requests, which means that the SQ is empty. The SP makes a transition automatically from “idle” to “busy” whenever a request arrives for service. Similarly, the SP makes a transition automatically from “busy” to “idle” whenever it finishes the service for a request. The transitions between the “sleep,” “standby,” and “idle” states are controlled by the PM, i.e., the DPM policy.



(a) (b)

Figure 6 Three-state and four-state SP models.

The average power consumption and service time (time to finish a read/write operation) for each state of the SP is shown in Table 1. The average energy consumption values for the SP to make transitions among the various states are shown in Table 2. The average transition times for the SP to make transitions among the various states are shown in Table 3. The data is obtained from [24] and [21].

Table 1 Average power consumption values and service times of the SP.

State	"sleep"	"standby"	"idle"	"busy"
Ave. power (W)	0.13	0.35	0.95	2.15
Ave. service time (s)	0	0	0	0.008

Table 2 Average energy consumption values for state transition of the SP.

Ave. energy (J)	"sleep"	"standby"	"idle"	"busy"
"sleep"	0	5.1	7.0	-
"standby"	0.006	0	2	-
"idle"	0.067	0.001	0	0
"busy"	-	-	0	0

Table 3 Average transition times for state transition of the SP.

Ave. transition time (s)	"sleep"	"standby"	"idle"	"busy"
"sleep"	0	0.6	1.6	-
"standby"	0.3	0	1.2	-
"idle"	0.67	0.4	0	0
"busy"	-	-	0	0

We have used five different distributions for the request inter-arrival time. They are:

1. Exponential distribution. This is the key assumption made by the continuous-timed Markov decision process (CTMDP) approach. For the CTMDP policy to be optimal, the request inter-arrival time must follow exponential distribution. In many research works related to hard drive performance, it is found that the request inter-arrival time follows the exponential distribution
2. Combination of the exponential and Pareto distribution. This combination is tried because of the observation made in [21]. The Pareto distribution has a density function: $f(t)=1-at^{-b}$. Compared with exponential distribution, it models an input sequence that shows more bursts.
3. Combination of two exponential distributions. We are using this distribution to approximate distribution 2.
4. Uniform distribution.
5. Normal distribution.

We have used two different distributions for the state transition time of the SP. They are:

1. Exponential distribution. This is another assumption made by the CTMDP approach. For the CTMDP policy to be optimal, the SP transition time must follow exponential distribution.
2. Uniform distribution. This distribution is proposed and used by the authors of [21] in their work.

The mean values we used for these distributions are shown in Table 3. The deviation for the uniform distribution is set to 0.1s, which is again observed in [21].

The policies we have used in this work are as follows:

1) The "time-out" policy

The "time-out" policy has a single parameter T_{out}. This policy can only be used for the three-state system model shown in Figure 6 (a). Under the "time-out" policy, the PM switches the SP from "idle" to "sleep" after the SP has been in the idle state and the SQ has been empty for a time period of T_{out}; the PM switches the SP from "sleep" to "idle" immediately after a request has arrived.

In our experimental setup, we have used different values of T_{out} to study the performance and power characteristics of the "time-out" policy.

2) The N-Policy

The N-policy has a single parameter N, which must be smaller than the capacity of the SQ, Q. Similar to the "time-out" policy; this policy is only applicable to the three-state system model. Under the N-policy, the PM switches the SP from "idle" to "sleep" immediately after the SQ is empty; the PM switches SP from "sleep" to "idle" only after there are more than N requests waiting in the SQ.

In our experimental setup, we have used different values of N to study the performance and power trade-offs of the N-policy.

3) The "always on" and "greedy" policies

The "always on" policy can be regarded as a special case of either the "time-out" policy when T_{out} is infinity, or the N-policy when N is zero. As indicated by its name, the "always on" policy never turns off the SP.

The "greedy" policy can be regarded as a special case of either the "time-out" policy when T_{out} is zero, or the N-policy when N is one.

4) The CTMDP policy

The CTMDP policy is applied to both the three-state model and the four-state model in Figure 6. Given any performance constraint, the power-optimal CTMDP policy can be calculated by solving a linear program.

In our experiments, a series of CTMDP polices were generated for different performance constraints. The CTMDP policies were applied using both the three-state SP model and the four-state SP model. Then the simulator and real traces were used to evaluate this policy in real applications.

5) The 3*CTMDP-Poll Policy*

Because the CTMDP policy is a randomized policy, at times it may not turn off the SP even when there is no request in the SQ. If our stochastic model exactly represents the system behavior, then this policy is optimal. However, in practice, because the stochastic model is not accurate enough, the CTMDP policy may cause unnecessary energy dissipation by not turning off the SP. For example, the real requests pattern on the SP may be quite different from what has been assumed in theory, and the SP idle time may be much longer than we expected based on the assumption of exponential input inter-arrival time. In this case keeping the SP on while it is idle wastes much power. We need to put some mechanism in place that will prevent such a case in real applications.

Based on the original description of the CTMDP policy, we have designed a modified CTMDP policy by adding a *polling state*. The functionality of the polling state is very simple. After adding this state, if the CTMDP policy allows the SQ to stay on when the SQ is empty, the policy will re-evaluate this decision after some random-length period of time. For example, if the SQ is empty and the PM has made a decision (with probability of 0.1) of letting the SQ stay on, then after 2s, if there is no change in the SQ, we enter the polling state, and the PM has to re-evaluate its decision. At this time, the probability for it to still let SQ remain on is again 0.1. So as the time goes on, the total probability of the SQ remaining on reduces in a geometric manner. In this way, we can make sure that the SP will not be idle for too long, resulting in less wasteful energy dissipation. We name this modified policy the **3CTMDP with Polling** (3CTMDP-Poll) policy and implement it in our experiments based on the 3CTMDP policy.

We use the average number (#) of waiting requests in the SQ as the performance metric and the average power consumption (W) as the power metric.

The comparisons of performance-power trade-off curves for the DPM policies for all the combinations of SP state transition distribution (TD) and request inter-arrival time distribution (RD) are shown in the following figures. In the legends of the figures, “3CTMDP” means CTMDP policy using a three-state SP model and “4CTMDP” means CTMDP policy using a four-state SP model. The X-axis gives the performance value, which is represented by the average number of waiting requests in queue. In all the figures, the X-axis is given in logarithmic scale. The Y-axis gives the system power consumption.

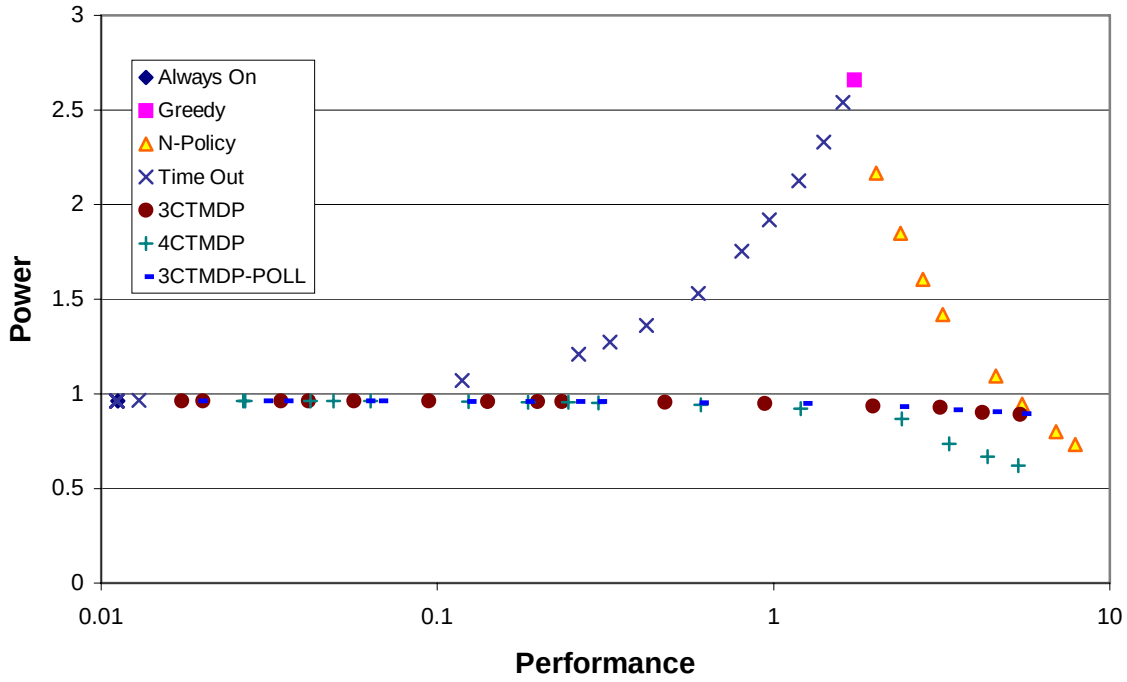


Figure 7 Performance-power trade-off curves for Exp. TD and Exp. RD.

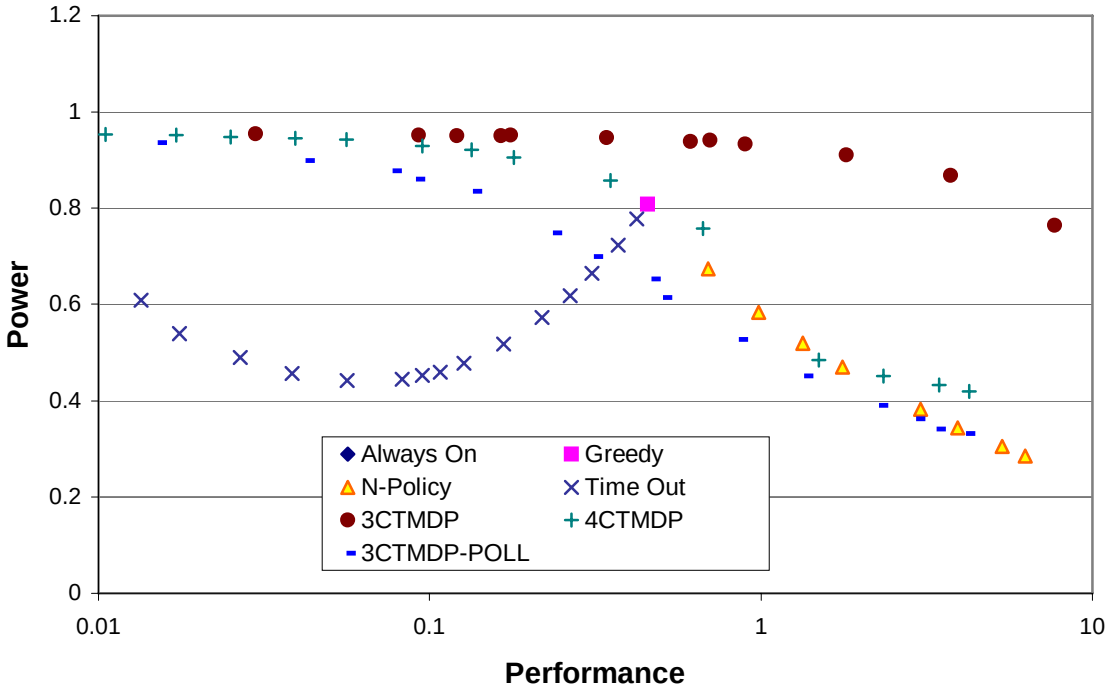


Figure 8 Performance-power trade-off curves for Exp. TD and Exp. & Par. RD.

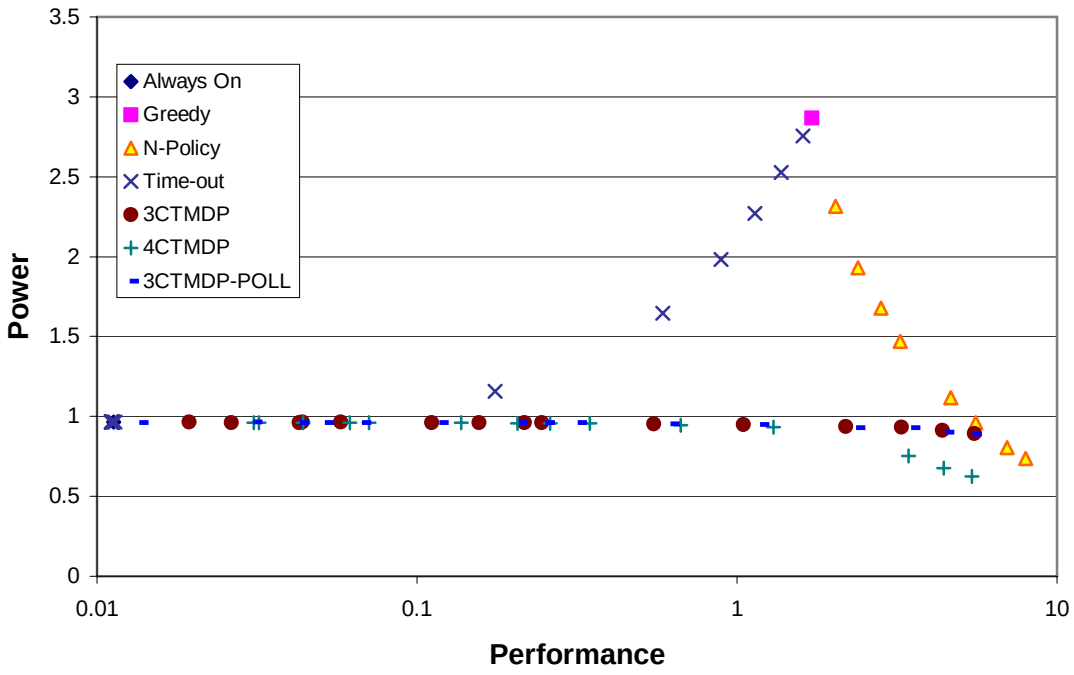


Figure 9 Performance-power trade-off curves for Exp. TD and Uni. RD.

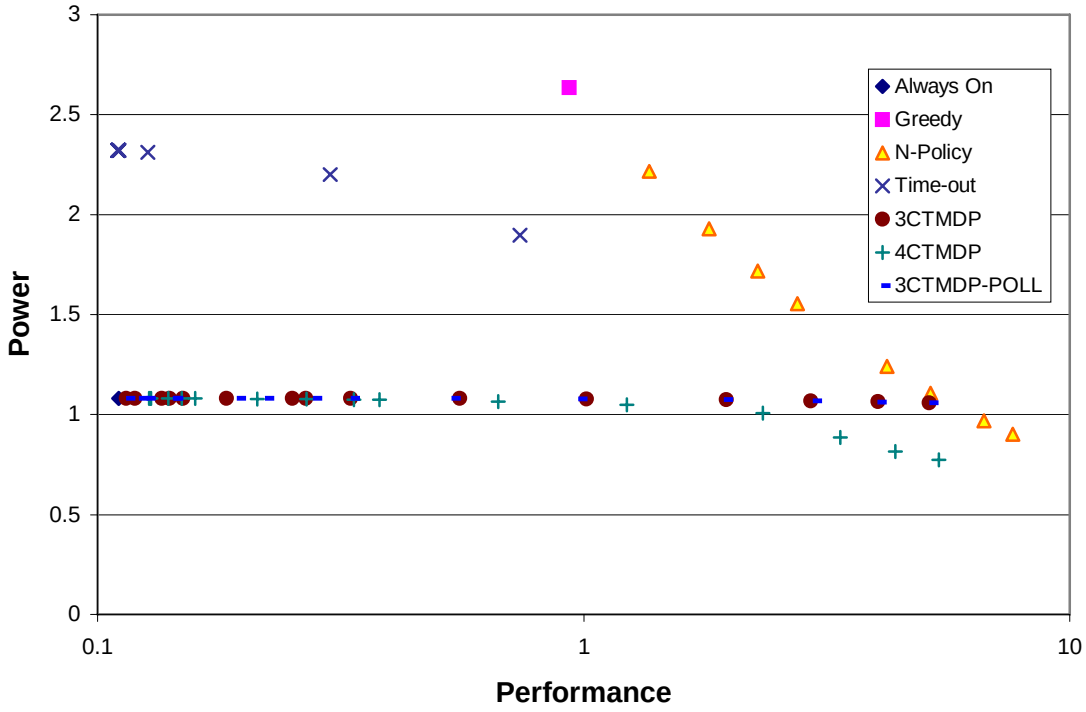


Figure 10 Performance-power trade-off curves for Uni. TD and Nor. RD.

From above figures we can see that:

1. In most situations, the stochastic policies out-perform the heuristic policies.
2. The stochastic policies provide a good power-delay trade-off. We can always trade performance to reduce power. However, the heuristic policies such as the time-out policy cannot provide a valid power-delay trade-off. In all experiments, decreasing the time-out threshold increases the average latency for the request. However, decreasing the time-out threshold does not necessarily decrease the average power consumption. With the decrease of the time-out threshold and the increase of the average request latency, the system power consumption first decreases then increases. In these experiments, there is an optimal time-out threshold under which both the average request latency and system power consumption are minimized. However, there is no formal way to find out this optimal threshold.
3. The Three-state CTMDP policy is not efficient with input sequence with Exp. & Pareto inter-arrival time because our stochastic model does not give as accurate a representation of the real system. However, the 3CTMDP-poll policy solves the above problem. The 3CTMDP-poll policy works almost as well as the four-state CTMDP.
4. Four-state CTMDP is robust in different TD and RD distributions. This means that if the device can provide us with more operation modes of different performance and power consumptions, the CTMDP policy can explore these modes to make itself an always-robust DPM policy independent of the distribution of request inter-arrival time or the distribution of SP transition time.

B. A power-managed system with a priority queue

In this setup, we call our algorithm CTMDP policy.

B.1 Comparison of the CTMDP policy with heuristic policy

In this experiment, we will present the comparison of the CTMDP policy and the heuristic policies including greedy policy and time-out policy and show that our policy consumes less power than the heuristic policies.

The system model used in this part includes:

An SP model that is the same as in Example 3.1 except that it has only one busy state, $busy_1$, and one idle state, $idle_1$.

An SR model with two states r_1 and r_2, $G_{SR}(r_1,r_2)=1/200$, $G_{SR}(r_2,r_1)=1/400$, $\lambda_l(r_1)= 1/30$, $\lambda_h(r_1)=1/50$, $\lambda_l(r_2)= 1/60$, $\lambda_h(r_2)=1/90$.

An SQ model with an LSQ of length 3 and an HSQ of length 2.

We compare the CTMDP policy with the “greedy” and “time-out” policies.

Two different traces of requests are used for simulation:

1. Requests are generated to exactly follow the SR model.

2. Requests are generated to follow the SR state transition rate between r_1 and r_2. However, in state r_1, the inter-arrival time of low-priority requests follows the uniform distribution (compared with the exponential distribution in trace 1) with mean value $1/\lambda_l(r_1)=30$, the inter-arrival time of high-priority requests follows the uniform distribution with mean value $1/\lambda_h(r_1)=50$. In state r_2, the inter-arrival time of low-priority requests follows the uniform distribution with mean value $1/\lambda_l(r_2)=60$, the inter-arrival time of high-priority requests follows the uniform distribution with mean value $1/\lambda_h(r_2)=90$.

In both cases, we simulated the heuristic policies to get the average delay of the low priority and high priority requests. We then use these delay values as the delay constraint and searched for a randomized CTMDP policy

using linear programming. Finally, we simulated the CTMDP policies and compared the power and delay value with that of the heuristic policies.

Table 4 Comparison with heuristic policy with input trace 1.

Heuristic Policy				LP-based CTMDP policy	
Name of policy	P (w)	D_l	D_h	P	ΔP (%)
Time-out policy t_{out}=20	1.665	0.399	0.143	0.942	43.43
Time-out policy t_{out}=40	2.080	0.232	0.118	1.156	44.42%
Time-out policy t_{out}=60	2.244	0.183	0.104	1.865	16.89
Greedy policy	1.303	0.151	0.255	1.067	18.11

Table 5 Comparison with heuristic policy with input trace 2.

Heuristic Policy				LP-based CTMDP Policy					
Name of Policy	P(w)	D_l	D_h	P(w)	ΔP (%)	D_l	ΔD_l (%)	D_h	ΔD_h (%)
Time-out policy t_{out}=20	1.515	0.439	0.121	1.121	26.0	0.406	7.51	0.119	1.7
Time-out policy t_{out}=40	2.071	0.243	0.097	1.674	19.2	0.227	6.6	0.094	3.1
Time-out policy t_{out}=60	2.242	0.151	0.085	2.142	4.5	0.149	1.3	0.083	2.4
Greedy policy	1.655	0.227	0.137	1.522	8.0	0.169	25.6	0.107	21.9

Table 4 and Table 5 present the comparison results by simulation. Columns P, D_l, and D_h give the power, low priority request delay, and high priority request delay for each power management policy respectively. Columns ΔP, ΔD_l, and ΔD_h give the CTMDP policy's improvement in power, low priority request delay, and high priority request delay. The values are simulated using a commercial stochastic activity network analysis tool: UltraSAN [20]. The conclusions we have from these experiments are as follows.

If the inter-arrival time of the input requests follows exponential distribution, the CTMDP policy saves more than 30% power on average over heuristic methods while keeping the same delay for both the high priority and low priority requests.

If the inter-arrival time of the input requests follows uniform distribution, the CTMDP policy cannot exactly meet the delay constraint. It saves about 14% power on average, and at the same time it reduces the delay of low priority requests by about 10% and reduces the delay of high priority requests by about 7%.

Further reductions can be achieved using the CTMDP method if we increase the delay constraint. In other words, the CTMDP method can make different power-delay trade-offs by changing the delay constraints. Little can be done by the heuristic method.

The CTMDP method can perform optimal management for a complex system, while the heuristic methods can only perform simple management such as turn-on and turn-off.

The CTMDP method can adjust the optimal policy when workload characteristics change, while the "greedy" and "time-out" methods are not adjustable to workload change.

We did not compare the DPM method with the predictive method, because in our experiments the inter-arrival time of each requests are assumed to be independent. Therefore, the predictive method is not applicable.

B.2 Comparison of the CTMDP policy with the N-policy

In this section we will show that the CTMDP method is powerful in finding power and delay trade-offs. When the server has only two states, *active* and *sleeping*, it can easily be shown that the N-policy gives the minimum power compared to other stationary policies with the same performance constraint. Our experiments show that, however, for a system with more than two server states, the *N*-policy does not give the optimal power-delay trade-off. In the experimental results, we will present the comparison of our policy and the *N*-policy and show that our policy is more efficient in finding power and performance trade-offs than the *N*-policy.

In this experiment, the system model includes:

An SP model that is the same as that in the previous experiment.

An SR model with only one state *r*, $\lambda(r)= 1/20$, and the inter-arrival time of each request follows exponential distribution.

An SQ model with an SSQ of length 5

Table 6 gives the result of the comparison of our policy with the *N*-policy, where *N* varies from 0 to 5. The result shows that there is always a randomized policy that has the exact delay as the *N*-policy and consumes less power than the *N*-policy. The deterministic policy consumes more power than a randomized policy; however, it is still more efficient than the *N*-policy. Furthermore, the *N*-policy in this experiment only gives five different policies. However, a randomized policy can find power optimal policy under any delay constraint, which is achievable. Therefore, our algorithm is more flexible and effective.

Table 6 Comparison of our policy with the N-policy.

N-policy			Randomized-policy				Deterministic-policy			
N	Power	Delay	Power	ΔP(%)	Delay	ΔD(%)	Power	ΔP(%)	Delay	ΔD(%)
5	0.860	2.346	0.754	12.3	2.304	1.8	0.754	12.3	2.304	1.8
4	0.944	1.933	0.802	15.0	1.933	0	0.8107	14.1	1.869	3.3
3	1.045	1.483	0.861	38.7	1.483	0	0.869	16.8	1.425	3.9
2	1.216	1.027	0.952	20.9	1.027	0	0.966	20.6	1.005	2.1
1	1.623	0.608	1.160	28.5	0.607	0.2	1.213	25.3	0.606	0.3
0	2.3	0.332	2.3	0	0.332	0	2.3	0	0.32	0

VII. Conclusions

We have proposed a new system model and method for dynamic power management at the system-level. The problem of system-level power management was formulated as an optimal policy selection problem based on the theories of continuous-time Markov decision processes and stochastic networks. Compared to previous works, we introduced a new and more complete model of the system components as well as a model of the whole system. The proposed mathematical framework captures the characteristics of the real applications more accurately compared to the models proposed by previous researchers. This is mainly because we solve the problem in a continuous-time domain while previous authors solve the problem in a discrete-time domain. We have also proposed CTMDP-Poll techniques, which are better than the DTMDP-based techniques and the simple CTMDP-based techniques in all cases. Furthermore, we have used a priority queue model, which is more general and adaptable to real applications.

In the examples and the experimental results included in this paper, we have assumed that the system contains a single SP. Therefore, all the requests in the SQ are targeted for that SP (which means that they have an identical type). However, please note that this does not mean that all the requests pose the same amount of work load to the SP. The difference in the request work loads is captured by the exponentially distributed service time in the model. More generally, our mathematical framework can handle systems with more than one SP, and therefore, service requests of different types can be handled. In this case, we have to create multiple SQs, one for each type (or class) of service requests. There will also be a compatibility relation describing what type of requests can be serviced by which SPs. This adds complexity to the problem but is quite manageable within our proposed mathematical framework.

A shortcoming of our CTMDP-based technique is that it is very difficult to use the model when attempting to represent complex systems that consist of multiple closely interacting SPs and must cope with complicated synchronization schemes. In this case, we need to use the modeling techniques based on the theory of generalized stochastic Petri Nets (GSPN). Finally, in real applications, the inter-arrival time of service requests may not follow an exponential distribution. Although our experimental results demonstrate that, in practice, our method remains effective for a wide range of distributions, it is still desireable from a theoretical point of view to develop a new mathemnatical framework where arbitrary distributions can be easily handled. Again this problem can be solved by using the "stage method" (a series-parallel connection of exponentially distributed sources) based on the GSPN framework. However, GSPN is not in the scope of this manuscript.

REFERENCES

[1] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, July 1995.

[2] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design." IEEE Symposium on Low Power Electronics, pp.8-11, 1994.

[3] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," 1996 International Symposium on Low Power Electronics and Design", pp.347-352, Aug. 1996.

[4] J. Rabaey and M. Pedram, *Low Power Design Methodologies.* Kluwer Academic Publishers, 1996

[5] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Kluwer Academic Publishers, 1997.

[6] Intel, Microsoft and Toshiba, "Advanced configuration and power interface specification." URL: http://www.intel.com/ial/powermgm/specs.html, 1996.

[7] M. Srivastava, A. Chandrakasan. R. Brodersen, "Predictive system shutdown and other architectural techniques for energy effcient programmable computation," *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1, pp.42-55, 1996.

[8] C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Proceedings of the International Conference on Computer Aided Design*, pp. 28-32, Nov. 1997.

[9] G. A. Paleologo, L. Benini, et.al, "Policy optimization for dynamic power management," *Proceedings of the Design Automation Conference*, pp.182-187, June 1998.

[10] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," *Proceedings of the Design Automation Conference*, pp. 555-561, June 1999.

[11] Q. Qiu, Q. Wu, and M. Pedram, "Stochastic modeling of a power-managed system: construction and optimization," *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999.

[12] U. Narayan Bhat, *Elements of Applied Stochastic Processes*. John Wiley & Sons, Inc., 1984.

[13] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 347-352, Aug. 1996.

[14] B. Miller, "Finite state continuous time Markov decision processes with an infinite planning horizon," *J. Of Mathematical Analysis and Applications*, No. 22, pp. 552-569, 1968.

[15] E. V. Denardo, "On linear programming in a Markov decision problem," Management Sience, Vol. 16, No. 5, pp. 281-288, Jan. 1970.

[16] J. F. Shapiro, *Mathematical Programming: Structures and Algorithms*. John Wiley & Sons, Inc., 1979.

[17] R. A. Howard, *Dynamic Programming and Markov Processes*. New York: Wiley, 1960.

[18] D. P. Heyman and M. J. Sobel, *Stochastic Models in Operations Research*. McGraw-Hill Book Company, 1982.

[19] L. E. Scales, *Introduction to Non-Linear Optimization*. New York: Springer-Verlag New York Inc., 1985.

[20] UIUC, Performability Engineering Research Group, "The UltraSAN Software", http://www.crhc.uiuc.edu/UltraSAN/, 1997.

[21] T. Simunic, L. Benini, and G. De Micheli, "Dynamic power management of portable systems," MOBICOM, 2000.

[22] T. Simunic, L. Benini, and G. De Micheli, "Dynamic power management of laptop hard disk," DATE, 2000.

[23] Y. Lu, E. Chung, T. Simunic, L. Benini, and G. De Micheli, "Quantitative comparison of power management algorithms," DATE, 2000.

[24] "MHE2064AT, MHE2043AT, MHF2043AT, MHF2021AT Disk Drive Product Manual", Fujitsu Limited, 1998.