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 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 and efficiently using a "policy iteration" approach. Our method is compared with existing heuristic approaches for different workload statistics. Experimental results show that power management method based on Markov decision process outperforms heuristic approaches in terms of power dissipation savings for a given level of system performance.*

I. INTRODUCTION

With the rapid progress in the semiconductor technology, the 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 of 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. Many low power design methodologies and techniques that target digital VLSI circuits have been proposed [1]-[5].

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. System designers have started to respond to the requirement of power-constrained system designs by a combination of technological advances and architectural improvements. Dynamic power management – which refers to selective shut-off or slow-down of system components that are idle or underutilized – has proven to be a particularly effective technique. 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 stated. Best known among them is the *Advanced Configuration and Power Interface* (ACPI) [6] that specifies an abstract and flexible interface between power-manageable hardware components (VLSI chips, disk drivers, display drivers, etc.) and the *power manager* (the system component that controls the turn-on and turn-off of the system components). It is important to mention that, ACPI defines multiple power modes for system components, which is a key requirement for approaches based on Markov decision processes to outperform heuristic approaches.

*This work was supported in part by SRC under contract No. 98-DJ-606 and NSF under contract No. MIP-9628999.

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 and well-known heuristic policy is the "time-out" policy, which is widely used in today's portable computers. In the "time-out" policy, one component will be shut down after it has been idle for a certain amount of time. The predictive system shutdown approach in [7][8] tries to achieve better power-delay trade-off by predicting the "on" and "off" time of the component. This prediction approach uses a regression equation based on the component's previous "on" and "off" time to estimation the next "turn-on" time, such that the component can be turned on immediately before the request comes. Therefore, the system performance can be improved. However, this method is only applicable to few cases in which the requests are highly correlated.

Because heuristic policies do not have a robust system model and solid theoretical background, their major shortcomings are obvious. Firstly, they can never achieve the best power-delay trade-off for the system. Secondly, they cannot deal with complex components that have more than two (on and off) operating modes such as defined in ACPI. In addition, they cannot deal with complex system with multiple and interactive components.

A power management approach based on Markov decision process has been proposed in [9]. The system is modeled as a **discrete-time** Markov decision process by combining the stochastic models of its components. Once the model and its parameters are determined, an optimal power management policy can be obtained to achieve the best power-delay trade-off for the system. This approach offers significant improvements over previous power management techniques in terms of theoretical framework for modeling and optimizing the system. There are however some shortcomings. Firstly, because the system is modeled in the discrete-time domain, some assumptions about the system components may not hold for real applications. Secondly, the state transition probability of the system model cannot be obtained accurately. Moreover, the power management program needs to send control signals to the components in every time-slice, which results in heavy signal traffic and heavy load on the system resources (therefore more power).

The work of [10] overcomes the shortcomings of [9] by introducing a new system model (as well as component models) based on the **continuous-time** Markov decision process. In [10], a power-managed system is modeled in the continuous-time domain, which is closer to the situation encountered in practice; the component models are simpler and can accurately model many realistic applications.

In this paper, we improve the work of [10] in the following ways:

1. We present a new model of the service provider that explicitly distinguishes between the two cases where the server is busy (*on* and servicing some request) and idle (*on* but not servicing any request).
2. We introduce a new model for the service requester to capture complex workload characteristics.
3. We introduce a new model for the service queue that 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.
4. We present a new system model that is composed of the new component models.

This paper is organized as follows, Sections II and III describes the models for the components and the system. Sections 0 and V present the experimental results and conclusions.

II. COMPONENT MODELING

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 of that the system is in state i at time t

G: *generator matrix* of a continuous-time Markov process

λ: service request generation rate for *Service Requestor* (SR)

μ: service rate of the *Service Provider* (SP)

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

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

π: power management policy

The introduction to continuous-time Markov decision process is omitted to save space. Please refer [10] for detailed background.

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

We assume that the system is embedded in an environment where there is only a single source of requests, which is defined as the service requestor (SR). Requests generated by the SR can be divided into two categories: *low-priority requests* and *high-priority requests*, which are generated independent of each other. Requests generated by the SR are serviced by the system. The system itself consists of three components: a server that processes requests (the SP), a queue which stores the requests that cannot be immediately serviced upon arrival (SQ), and a power manager (PM) that issues commands. The SR is an input source, which is outside and independent of the system.

Although we consider a relatively simple system in this paper, our approach can be extended to a more complicated application that may consist of multiple SR's, SP's, and SQ's.

Both the request arrival event and the request service event are stochastic processes and follow the Poisson distribution. For example, the request arrival event follows the Poisson process (i.e., during time (0, t] the number of the events has the Poisson distribution with mean λt). Consequently, the request inter-arrival time follow the exponential distribution with mean $1/\lambda$. We assume that the request will be rejected if the SQ is full at the time when it comes.

The SP can operate in a number of different power modes. We also assume that the time needed for the SP to switch from one state to another follows the exponential distribution. The PM is a controller that reads the system state (the joint states of SP, SQ and SR) and issues mode-switching commands to the SP.

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.*, *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.

A. Model of the Service Provider

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

We call **χ**, the switching speed matrix of the SP. The (i,j)th entry of χ is denoted as χ_{s_i,s_j} and represents the switching speed from state s_i to 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:

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

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

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

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

A *power consumption pow*(s) is associated with each state $s\in S$. It represents the power consumption of 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 ene*(s_i, s_j) is associated with each state pair (s_i, s_j), s_i,s_j∈S, $s_i\neq s_j$. It represents the energy needed for 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 (earning rate) of 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, the working modes of the SP can be divided into three groups: busy, idle, and power-down. In busy modes, the SP is fully powered and working on the first request in the SQ. In idle modes, the SP is fully powered, but it is not working on any request. In power-down modes, the SP is partially or completely shut down, i.e., not it is functional. We distinguish idle modes from busy modes, because the SP cannot switch to other state when it is working on some request. In other words if we want to turn the SP off (switch to a power-down mode), it must be switched off from an idle state.

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

For each busy mode, there exists a corresponding idle state. The SP may have multiple power-down modes (e.g. standby, soft off, hard off).

In our mathematical model of the SP, we divide the state set *S* into two subsets:

(1) The set of active states, S_{active}, where $\mu(s_{act})$ is larger than 0 for each $s_{act}\in S_{active}$.

(2) The set of inactive states, $S_{inactive}$, where $\mu(s_{ina})$ is 0 for each $s_{ina}\in S_{inactive}$.

The busy modes belong to the first subset. The idle and power-down modes belong to the second subset.

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

1. The action cannot make a transition between a busy mode to a power-down mode directly. Transitions between them must go through an idle mode.
2. The action cannot cause a transition from a busy mode to its correspondent idle mode. The transition from a busy mode to an idle mode is done autonomously when the SP finishes a service (therefore it is not controllable).
3. The action cannot cause a transition between two busy modes. When the SP is in a busy mode, no transition to any other state is allowed.

Definition 3.1 Inactive 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 in state s_2.

Example 3.1 Consider a SP with four states, S={*busy*, *idle*, *wait*, *sleep*}. When the SP is in state *busy*, it provides the service for the requests. The average time needed for each service (for both low-priority requests and high-priority requests) is 5 second. Therefore, μ_l(*busy*) and μ_h(*busy*) are 0.2. μ_l(*idle*), μ_h(*idle*), μ_l(*wait*), μ_h(*wait*), μ_l(*sleep*) and μ_h(*sleep*) are all 0. Let the command set be defined as A={*go_busy*, *go_idle*, *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:

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

By default, the order of states in rows and columns are the same as the order of 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_i} =0 means that the SP can never transfer from state s_i to s_j. In this example, the SP needs no time to transfer from a state to itself. The SP can transfer from the *busy* state to *idle* state with the transition rate equal to the service rate because it goes to the *idle* state autonomously immediately after it finishes a request. The SP cannot switch between the *busy* state and *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*)=2.3W, *pow*(*idle*)=2.3W, *pow*(*wait*)=0.8W, *pow*(*sleep*)=0.1W.

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

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

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

A graphical illustration of the SP is shown in Figure 1. The transition rates associated with the directed edges have not been shown in the figure. They can be extracted from $\mathbf{G}_{SP}(a)$ for specific actions.

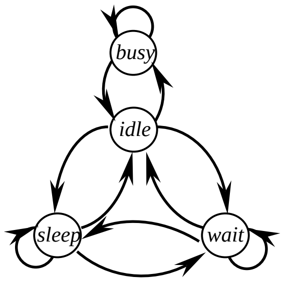

Figure 1 Markov process model of the SP

B. 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, …, R} and generator matrix* $\mathbf{G}_{SR}$. *It can be characterized by a pair (τ, λ(r)), where: (i) τ is an R×R matrix, (ii) $\lambda_l(r)$ and $\lambda_h(r)$ are functions λ: R→**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:

$$\sigma_{r_i,r_j} = \tau_{r_i,r_j},\ r_i \neq r_j;\ \sigma_{r_i,r_i} = -\sum_{r_j \neq r_i} \sigma_{r_i,r_j} \qquad (3.4)$$

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 $\lambda_l(r)$, and the generation of the high-priority requests follows the Poisson process with mean value $\lambda_h(r)$.

C. 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, …, Q} and the generator matrix* $\mathbf{G}_{SSQ}(s, r)$*, where s is the state of SP, r is the state of SR state.*

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. However, in real applications, some service requests may have higher priority than others. Especially in a power-managed system, the PM always buffers the incoming service requests, that is, to achieve the best power-delay trade-off. The SP, under control of the PM, may not service the incoming request immediately even there is no other request in the queue. However, there may exist high-priority requests that need immediate service by the SP. In this case, if we use a loose delay constraint, the power management policy does not serve the request immediately (in order to save power). This long latency may not be acceptable for high-priority requests. We can instead use a tight delay constraint to make sure the high-priority requests are serviced immediately. However, this tight delay constraint is also applied to low-priority requests. Consequently, there will be undesirable power dissipation related to 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 relations between these two queues are:

1. Two different delay constraints are assigned to HSQ and LSQ separately such that the requests in HSQ have smaller waiting time than those in LSQ.
2. The requests in LSQ can be serviced by the SP (under the chosen PM policy) only when there is no request in HSQ.
3. The SP will not start serving the requests in LSQ until it finishes all the requests (under PM policy) in HSQ.

Although we have introduced two queues in our stochastic model of the service queue, we are actually modeling a single priority queue in real applications. 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* $\mathbf{G}_{SQ}(s, r) = \mathbf{G}_{LSQ}(s, r) \oplus \mathbf{G}_{HSQ}(s, r)$*, where s is the state of SP, r is the state of SR state, and the "⊕" operation is the tensor sum defined in* ***Definition 3.2***.

III. SYSTEM MODELING

We first show how to construct the model of the entire system by combining the component models. Next 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 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}. An action set of all possible actions which is the same as A in the SP model. A parameterized generator matrix $\mathbf{G}_{SYS}(a)$ gives the state transition rates under action a. A cost function Cost(x, a) gives the system cost under action a when the SYS is in state x.*

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

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

(2) When both 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 reasonable, because when SP and SQ are in the above states, it means that the service speed cannot catch the incoming speed of the requests. Therefore, we need to increase the service rate.

The SYS state can be represented as (s, r, (lq, hq)), where $s\in S$, $r\in R$, $lq\in Q_{LSQ}$ and $hq\in Q_{HSQ}$. The SYS model is a connected Markov process. Consequently, the limiting distributions of the state probabilities exist and are independent of the initial state.

B. Calculating the generator matrix

We next introduce the method of calculating the generator matrix $\mathbf{G}_{SYS}(a)$ 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 4.1 gives a method to obtain the joint transition rate of two independent continuous-time Markov processes. Proposition 4.2 gives a method of generating the generator matrix of the joint system using matrix operations.

Proposition 4.1 *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, 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 $\sigma_{(x,y),(x,y')} = \sigma_{y,y'}$, $\sigma_{(x,y),(x',y)} = \sigma_{x,x'}$, $\sigma_{(x,y),(x',y')} = 0$.*

Given two matrices A and B as follows:

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

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, $\mathbf{I}_{n_i}$ is the identity matrix of order n_i.

Proposition 4.2 *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 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 Markov processes of the SP and the SQ are however correlated. Because whenever the SP makes a transition from a busy state to an idle state (finishes the service for a request), the SQ must make a transition which decreases the number of requests in SQ by 1.

To show how to calculate $\mathbf{G}_{SP\text{-}SQ}(a, r)$ from $\mathbf{G}_{SP}(a)$ and $\mathbf{G}_{SQ}(s, r)$, we need to firstly partition $G_{SP}(a)$ as follows:

$$\mathbf{G}_{SP}(a)=\begin{bmatrix} \mathbf{G}_{SP}^{II}(a) & \mathbf{G}_{SP}^{IA}(a) \\ \mathbf{G}_{SP}^{AI}(a) & \mathbf{G}_{SP}^{AA}(a)\end{bmatrix} \qquad (4.2)$$

Matrix $\mathbf{G}_{SP}^{II}(a)$ contains the transition rates for transitions between inactive states. Matrix $\mathbf{G}_{SP}^{IA}(a)$ contains the transition rates for transitions from any inactive state to any active state. Matrix $\mathbf{G}_{SP}^{IA}(a)$ contains the transition rates for transitions from any active state to any inactive state. Matrix $\mathbf{G}_{SP}^{AA}(a)$ contains the transition rates for transitions between active states.

We can partition $\mathbf{G}_{SP\text{-}SQ}(a, r)$ as:

$$\mathbf{G}_{SP-SQ}(a,r)=\begin{bmatrix} \mathbf{G}_{SP-SQ}^{II}(a,r) & \mathbf{G}_{SP-SQ}^{IA}(a,r) \\ \mathbf{G}_{SP-SQ}^{AI}(a,r) & \mathbf{G}_{SP-SQ}^{AA}(a,r)\end{bmatrix} \qquad (4.3)$$

To calculate $\mathbf{G}_{SP\text{-}SQ}(a, r)$, we first calculate the four sub-matrices in Eqn. (4.3) except the diagonal of $\mathbf{G}_{SP\text{-}SQ}(a, r)$. The entries on the diagonal are calculated using Eqn. (2.4) after the sub-matrices are calculated.

$\mathbf{G}_{SP-SQ}^{II}(a,r)$ defines the transition rates for transitions between any two states (s_1, (lq_1, hq_1)) and (s_2, (lq_2, hq_2)) s.t. s_1, $s_2\in S_{inactive}$ (defined in Section III.A), lq_1, $lq_2\in Q_{LSQ}$ and hq_1, $hq_2\in Q_{HSQ}$. It can be obtained as:

$$\mathbf{G}_{SP-SQ}^{II}(a,r)=\mathbf{G}_{SP}^{II}(a)\oplus\mathbf{G}_{SQ}(s,r) \qquad (4.4)$$

Notice that, after the operation, the parameter s in $\mathbf{G}_{SQ}(s, r)$ has been removed by substituting the real state of the SP. $\mathbf{G}_{SP-SQ}^{II}(a,r)$ is calculated directly by the ⊕ operation because transition between inactive SP states is not correlated with the transition of SQ state.

We let $g_X(x_1, x_2)$ denote the transition rate for the transition from state x_1 to x_2 of a Markov process X. Notice that $g_X(x_1, x_2)$ may be a parameterized quantity as in $\mathbf{G}_{SP}$, $\mathbf{G}_{SQ}$, $\mathbf{G}_{SP\text{-}SQ}$, and $\mathbf{G}_{SYS}$

$\mathbf{G}_{SP-SQ}^{IA}(a,r)$ defines the transition rates for transitions between any two states (s_1, (lq_1, hq_1)) and (s_2, (lq_2, hq_2)) s.t. $s_1\in S_{inactive}$, $s_2\in S_{active}$, lq_1, $lq_2\in Q_{LSQ}$ and hq_1, $hq_2\in Q_{HSQ}$. The rule for calculating the entries of $\mathbf{G}_{SP-SQ}^{IA}(a,r)$ is as follows:

$g_{SP\text{-}SQ}$((s_1, (lq_1, hq_1)), (s_2, (lq_2, hq_2))) is equal to $g_{SP}(s_1, s_2)$ if {(s_1 is an idle state) **AND** (s_2 is the busy state correspondent to s_1) **AND** (lq_1==lq_2) **AND** (hq_1==hq_2)} holds; Otherwise, it is zero.

$\mathbf{G}_{SP-SQ}^{AI}(a,r)$ defines the transition rates for transitions between any two states (s_1, (lq_1, hq_1)) and (s_2, (lq_2, hq_2)) s.t. $s_1\in S_{active}$, $s_2\in S_{inactive}$, lq_1, $lq_2\in Q_{LSQ}$ and hq_1, $hq_2\in Q_{HSQ}$. The rule for calculating the entries of $\mathbf{G}_{SP-SQ}^{AI}(a,r)$ is as follows:

$g_{SP\text{-}SQ}$((s_1, (lq_1, hq_1)), (s_2, (lq_2, hq_2))) is equal to $\mu_l(s_1)$ if {(s_1 is a busy state) **AND** (s_2 is the idle state correspondent to s_1) **AND** (lq_1==(lq_2+1)) **AND** (hq_1==hq_2==0)} holds; It is equal to $\mu_h(s_1)$ if {(s_1 is a busy state) **AND** (s_2 is the idle state correspondent to s_1) **AND** (lq_1==lq_2) **AND** (hq_1==(hq_2+1))} holds; Otherwise it is zero.

$\mathbf{G}_{SP-SQ}^{AA}(a,r)$ defines the transition rates for transitions between any two states (s_1, (lq_1, hq_1)) and (s_2, (lq_2, hq_2)) s.t. $s_1\in S_{active}$,

$s_2 \in S_{active}$, lq_1, $lq_2 \in Q_{LSQ}$ and hq_1, $hq_2 \in Q_{HSQ}$. The rule for calculating the entries of $\mathbf{G}_{SP-SQ}^{AA}(a,r)$ is as follows:

$g_{SP\text{-}SQ}((s_1, (lq_1, hq_1)), (s_2, (lq_2, hq_2)))$ is equal to $\lambda_l(r)$ if $\{(s_1==s_2)$ **AND** $(lq_1==(lq_2-1))$ **AND** $(hq_1==hq_2)\}$ holds; It is equal to $\lambda_h(r)$ if $\{(s_1==s_2)$ **AND** $(lq_1==lq_2)$ **AND** $(hq_1==(hq_2-1))\}$ holds; Otherwise it is zero.

C. Calculating the cost function

The cost of the system is related to the state *x* of the SYS and the action *a* taken by the SYS in state *x* . As in [10], we use the average power consumption and the average number of waiting requests to capture the system cost. Therefore, we have three cost functions in our model: the power consumption of the SP $C_{pow}(x)$, the average number of requests in the LSQ of the SQ $C_{lsq}(x)$, and the average number of requests in the HSQ of the SQ $C_{hsq}(x,a)$.

Let *x* be denoted as (*s*, *r*, (*lq*, *hq*)), where $s \in S$, $r \in R_e$, $lq \in Q_{LSQ}$ and $hq \in Q_{HSQ}$.

The power cost can be calculated as:

$$C_{pow}(x,a) = pow(s) + \sum_{s' \in S, s' \neq s} g(s,s')ene(s,s') \qquad (4.5)$$

where *pow*(*s*) and *ene*(*s*,*s*') were defined in Section III.A, and *g*(*s*,*s*') is the transition rate from state *s* to *s*' of the SP. Notice that g(*s*,*s*') is a function of *a*.

In addition, the delay costs are:.

$$C_{lsq}(x)=lq \text{ and } C_{hsq}(x)=hq \qquad (4.6)$$

The average waiting time of the requests is often used as the ***cost of delay***. However, in [10], it is shown that there exists a linear relationship between the average number of requests in the queue and the average waiting time. Therefore, Eqn. (4.6) can be used as the delay cost.

We define the total cost as a weighted summation of the power and delay costs:

$$\text{Cost}(x,a)=w_1 \cdot C_{pow}(x,a)+w_2 \cdot C_{lsq}(x)+w_3 \cdot C_{hsq}(x) \qquad (4.7)$$

where $w_1+w_2+w_3=1$.

The optimal policy for the system model is then solved using the policy iteration algorithm used in [10].

D. 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 state, it sends an interrupt signal SWITCH_DONE to the PM. The PM then reads the states of all components in the power-managed system (hence obtains the joint system state), issues a command according to the chosen policy. The SP receives the command and immediately starts to switch to a state which is given by the command. Notice that the command may ask the SP to switch to its current state, therefore the SP state will not change. We assume that, after the SP finishes a service, it will stay in the idle state for some time that is long enough for it to accept the command from the PM and switch to another state. We also assume that the PM reads the states and issues command in a short time that does not affect the system performance.

IV. EXPERIMENTAL RESULTS

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

A. Experiment for comparing models of the SQ

The original model (i.e., the model which does not consider the request priority) [10] includes:

1. A SP model that is the same as in **Example 3.1**.
2. A SR model with only one state *r*, $\lambda_l(r)$= 1/80 and $\lambda_h(r)$=1/100.
3. A SQ model with a SSQ of length 7.

Our new system model includes:

1. A SP model that is the same as the one in **Example 3.1**.
2. A SR model with only one state *r*, $\lambda_l(r)$= 1/80 and $\lambda_h(r)$=1/100.
3. A SQ model with a LSQ of length 5 and a HSQ of length 2.

Our goal is to apply a tight delay constraint *C* on the high-priority requests such that they are serviced within a required amount of time. And as for the low-priority requests, we only want to maintain their throughput (same incoming and outgoing rate). Optimal policies for both models are calculated under following two different scenarios:

1. Since the original model cannot distinguish between high-priority requests and low priority requests, to make sure that the delay of high priority requests meets the constraint *C*, it has to apply the constraint *C* on all requests. Using the new model, we only need to apply *C* on the HSQ and use a looser delay constraint on the LSQ to maintain the throughput of the low-priority requests.
2. Results from scenario 1 shows that, by applying the constraint *C* on all requests, the original model always gets much smaller delay on both high and low priority requests than required. Therefore in this scenario, in the new model, we further tighten the HSQ constraint such that the delay of high-priority requests matches those in the original model.

Different *C* values are used to generate the multiple rows in Table 1 and Table 2. Optimal policies are simulated using an even-driven simulator which has the following setup:

1. The SP is modeled the same as in **Example 3.1**.
2. A total of 20,000 service requests are randomly generated. Both low-priority and high-priority requests are generated independently such that they follow Poisson distributions with parameters $\lambda_l(r)$= 1/80 and $\lambda_h(r)$=1/100, respectively.
3. A queue of length 7 buffers the incoming service requests. An incoming high-priority request will be inserted in front of all other requests except the high-priority requests that came earlier and/or the low-priority request that is being serviced by the SP.

Tables 1 and 2 show the experimental results for both scenarios.

Table 1 Experimental results for scenario 1

Original model			New model			
Ave. # of high-priority requests in the queue	Ave. # of low-priority requests in the queue	Ave. power consump-tion (W)	Ave. # of high-priority requests in the queue	Ave. # of low-priority requests in the queue	Ave. power consump-tion (W)	Power differ-ence
0.26	0.83	1.99	0.62	1.60	0.89	55%
0.42	0.61	1.36	1.02	1.84	0.66	51%
0.63	0.86	1.04	1.26	2.41	0.54	48%

Table 2 Experimental results for scenario 2

Original model			New model			
Ave. # of high-priority requests in the queue	Ave. # of low-priority requests in the queue	Ave. power consump-tion (W)	Ave. # of high-priority requests in the queue	Ave. # of low-priority requests in the queue	Ave. power consumpt ion (W)	Power differ-ence
0.26	0.83	1.99	0.24	0.35	1.85	7%
0.42	0.61	1.36	0.38	1.18	1.14	16%
0.63	0.86	1.04	0.62	1.60	0.89	14%

Notice that the data in both tables are the same for the original model because they have the same delay constraints.

From the results in Tables 1 and 2, we can draw following conclusions:

1. The original model sets the same delay constraints on both low-priority requests and high-priority requests. This results in undesirable increase in power dissipation. In addition, the original model always over-estimates the delay of high-priority requests, i.e., the simulated delay of high-priority requests is always smaller than the pre-set constraint. While in the new model, the simulated delay of both high-priority and low priority requests is always close to the pre-set constraints.
2. Even though the policy based on the old model gets smaller delay (in simulation) for high-priority requests in scenario 1, we can always find an optimal policy while matching the HSQ delay constraint (which is the situation in scenario 2).

The optimal policy based on the new model still saves more power.

In fact, the new model saves more power by taking advantage of being able to setting a different delay constraint on the low-priority requests. If the percentage of the low-priority requests in all requests increases, the optimal policy based on our model saves more power than the optimal policy based on the original model. On the other hand, if the percentage of the low-priority requests decreases, the advantage of the new model will become less significant.

B. Experiments for comparing our method with heuristic policies

The system model used in this part includes:

1. A SP model that is the same as in **Example 3.1**.
2. A SR model with two states r_1 and r_2, $\mathbf{G}_{SR}(r_1,r_2)=\mathbf{G}_{SR}(r_2,r_1)=1/1000$, $\lambda_l(r_1)= 1/10$, $\lambda_h(r_1)=1/40$, $\lambda_l(r_2)= 1/20$, $\lambda_h(r_2)=1/80$.
3. A SQ model with a LSQ of length 5 and a HSQ of length 2.

Three different traces of requests are used for simulation:

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

Trace 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 a uniform distribution with mean value 10, the inter-arrival time of high-priority requests follows a uniform distribution with mean value 40. In state r_2, the inter-arrival time of low-priority requests follows a uniform distribution with mean value 20, the inter-arrival time of high-priority requests follows a uniform distribution with mean value 80.

Trace 3. Request trace extracted from real operations on a portable computer. In this case, the parameters in the SR model are obtained by curve fitting.

Our method is compared with four heuristic power management methods:

1. Greedy policy: turn on the SP whenever a request comes and turn off the SP whenever the SP is idle and there is no request in the queue.
2. Timeout policy (T=20): turn on the SP whenever a request comes and turn off the SP whenever the SP has been idle for 20 seconds and there is no request in the queue.
3. Timeout policy (T=40): turn on the SP whenever a request comes and turn off the SP whenever the SP has been idle for 40 seconds and there is no request in the queue.
4. Predictive method proposed in [7].

Tables 3, 4 and 5 present the comparison results by simulation. The conclusions we have from these experiments are:

1. The new model saves more than 30% power on average over heuristic methods. The average waiting time of low-priority requests has been increased for less power. Note however that the throughput of low priority requests is maintained.
2. By using our new system model, the latency of the high-priority requests can be kept low, which is required in real applications.
3. With new model, the latency of the high-priority requests is still a little higher compared with heuristic methods, mostly because of the SP switching time from power-down states to functional states (heuristic methods make less switch than DPM method). This situation can be improved in other applications where the SP switches faster than the one in our experimental setup.
4. The new model can handle multiple power modes whereas the heuristic methods can only handle on-off states.
5. The new model can make different power-delay trade-offs by changing the constraints on request waiting time.
6. The new model can adjust the optimal policy when workload characteristics change, while the greedy and timeout methods are not adaptive to the workload.

V. CONCLUSION

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 process, and stochastic network. Comparing with previous work, we introduced new and more complete model of the system components, as well as the model of the whole system. The proposed models are closer to the real applications than other previously proposed models. Experimental results showed that our model has better performance than previous models in real application. We also showed that the dynamic power management method out-perform the heuristic approaches in terms of better and more flexible power-delay trade-off.

Table 3 Experimental results for comparing power management methods using request trace 1

Method	Ave. waiting time of high-priority requests (sec)	Ave. waiting time of low-priority requests (sec)	Ave. Power Dissipation (W)
Greedy	16.4	25.5	2.03
Timeout (T=20)	13.9	19.5	2.29
Timeout (T=40)	12.9	18.3	2.20
Predictive	17.5	25.5	2.17
Our DPM	20.0	33.7	1.80

Table 4 Experimental results for comparing power management methods using request trace 2

Method	Ave. waiting time of high-priority requests (sec)	Ave. waiting time of low-priority requests (sec)	Ave. Power Dissipation (W)
Greedy	9.6	15.5	3.67
Timeout (T=20)	5.6	7.6	2.59
Timeout (T=40)	5.2	6.7	2.30
Predictive	10.3	15.7	3.90
Our DPM	10.2	24.7	1.95

Table 5 Experimental results for comparing power management methods using request trace 3

Method	Ave. waiting time of high-priority requests (sec)	Ave. waiting time of low-priority requests (sec)	Ave. Power Dissipation (W)
Greedy	13.0	11.8	2.39
Timeout (T=20)	9.2	6.4	2.15
Timeout (T=40)	8.7	3.9	2.06
Predictive	15.0	16.5	3.10
Our DPM	15.1	36.5	1.51

REFERENCES

[1] A. Chandrakasan, 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,*” Proc. of the Intl.*

[9] G. A. Paleologo, L. Benini, et.al, “Policy Optimization for Dynamic Power Management”, *Proceedings of the Design Automation Conference*, pp.182-187, Jun. 1998.

[10] Q. Qiu, M. Pedram, “Dynamic Power Management Based on Continuous-Time Markov Decision Processes”to appear, *Proceedings of the Design Automation Conference*, Jun. 1999.