Dynamic Power management: A Continuous-Time Stochastic Approach

CENG 99-02

Qinru Qiu, Qing Wu and Massoud Pedram

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213-740-4458)
March 1999

Dynamic Power management: A Continuous-Time Stochastic Approach

Qinru Qiu, Qing Wu and Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California, Los Angeles, California, USA
{qinru, qwu, pedram}@usc.edu

Abstract—This paper introduces a continuous-time, controllable Markov process model of a power-managed system. The system model is composed of the corresponding stochastic models of the service queue and the service provider. The system environment is modeled by a stochastic service request process. The problem of dynamic power management in such a system is formulated as a policy optimization problem and solved using an efficient "policy iteration" algorithm. Compared to previous work on dynamic power management, our formulation allows better modeling of the various system components, the power-managed system as a whole, and its environment. In addition it captures dependencies between the service queue and service provider status. Finally, the resulting power management policy is asynchronous, hence it is more power-efficient and more useful in practice. Experimental results demonstrate the effectiveness of our policy optimization algorithm compared to a number of heuristic (time-out and N-policy) algorithms.

I. INTRODUCTION

With the rapid progress in the semiconductor technology, the chip density and operation frequency have greatly increased, making power consumption in battery-operated portable devices a major concern. The goal of low-power design for battery-powered devices is to extend the battery lifetime while meeting the performance requirement. Reducing power dissipation is a design goal even for non-portable devices since excessive power dissipation results in increased cost of packaging and cooling as well as potential reliability problems. Many computer aided design methodologies and techniques for low power have been proposed [1].

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. Digital components often consume 20-40% of the total power (mainly in memory and CPU). Hence reducing power consumption in the digital VLSI circuits by even 10X does not reduce the power by more than 18-36%. 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 which 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] which 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 (in addition to the "on" and "off" mode) for system components, which is a key requirement for approaches based on Markov decision processes to out-perform heuristic approaches.

An event-driven computing device can be thought of as a finite state system that can be in two states: *Active* and *Idle*. When the device is idle, it is desirable to shut it down by lowering its power supply or by turning off its clock; in this way its power dissipation can be drastically reduced. If we call T_A and T_I the average time spent by the device in the active and in the idle states, respectively, we have that the maximum power improvement achievable through shut down is given by: $1 + T_I / T_A$. Improvement figures closer to the upper bound are however rarely obtained by static shut-down strategies. In fact, normally a device is put in its power-down mode only T time units after it has entered the idle state. This is because it is assumed that there is a high chance for the system to be idle for a much longer time if it has been in the Idle state for at least T time units. Obviously, this simple policy is not efficient.

To overcome the limitations of the static shut-down policy, Srivastava et al. [16] proposed a *predictive* power management strategy, whose main feature exploits the past history of the active and idle intervals to predict the length of an idle interval as soon as the system enters the idle state. In practice, two approaches are suggested: One is based on obtaining a regression equation that predicts the value of T_I based on a quadratic function of the previous values of both T_A and T_I. The other is based on the simple observation that if the T_A immediately preceding a T_I is shorter than the minimum value of T_A ever experienced, it is highly probable that the next T_I will be longer than the minimum time for which it is convenient to shut down the system. In [17], Hwang and Wu have introduced a more complex predictive shut-down strategy that performs better than the methods of Srivastava et al. The use of a technique for correcting possible idle period mis-predictions, along with a pre-wake-up mechanism, account for the higher efficiency and the decreased delay penalty provided by the new approach.

The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a new kind of constrained optimization problem which is of great relevance for low-power electronic systems. This problem is often referred to as the *policy optimization* (PO) problem. In [11], Paleologo et al. proposed a stochastic model for a rigorous mathematical formulation of the problem and give a procedure for its exact solution. The solution is computed in polynomial time by solving a linear optimization problem. Their approach is based on a stochastic model of power-managed devices and workloads and leverages stochastic optimization techniques based on the theory

of *discrete-time Markov decision chains*. That is, the time is divided into small slices of length L. It is assumed that the system can only change its state at the beginning of a time slice. During period (nL, (n+1)L), transition probability of the system depends only on the state of the system at time nL and the command issued by the power manager. The system model consists of four components: a *power manager* (PM), a *service provider* (SP), a *service requestor* (SR) and a *service request queue* (SQ). Each of the components is modeled as a discrete-time Markov chain. The entire power-managed system is modeled as a joint process of the Markov chains. Once the model and its parameters have been determined, an optimal power management policy is obtained to achieve best power-delay trade-off. This approach offers significant improvement over previous power management techniques in terms of its theoretical foundation and a robust system model. This approach however has some shortcomings. Firstly, the power-managed system is modeled in the discrete-time domain, which limits its in real applications. Secondly, the model does not distinguish between the busy state and the idle state of the SP (they are lumped into the "power-up" state), therefore the state transition probability of the system model cannot be calculated accurately. Thirdly, the assumption that the transitions of the SQ and the SP are independent is inaccurate and thus affects the overall accuracy of this model. Finally, 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 dissipation).

In this work, we overcome the shortcomings of [11] by introducing a new system model based on *continuous-time Markov decision processes*. This new model has the following characteristics:

1. The new model is based on the continuous-time Markov decision processes, which is closer to the scenarios encountered in practice.

2. The resulting power management policy is asynchronous which is more suitable for implementation as part of the operating system.

3. The new model introduces a *transfer state* in the model of the SQ; in this way it can distinguish between the busy and idle states of the SP.

4. The new model considers the correlation between the state of the SQ and the state of the SP.

5. A policy iteration algorithm is used to solve the policy optimization problem. The new algorithm tends to be more efficient than the linear programming method.

We also explore the class of N-policies and show that under certain conditions, this class of algorithms, which are very easy to implement, produces optimal solutions.

This paper is organized as follows, Section II provides the background for continuous-time Markov processes and continuous-time Markov decision processes. Section III describes our system model for the dynamic power management. Section IV describes the policy iteration algorithm. Section V gives the experimental results. Section VI is the concluding remarks.

II. BACKGROUND

Definition 2.1 A *stochastic process* is a family of random variables $\{X(t), t \geq 0\}$ where t is the time parameter. 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 $t_0 < t_1 < \ldots < t_n < t$, its conditional distribution has the property:

$$P[X(t) \leq x \mid X(t_n) = x_n, \ldots, X(t_0) = x_0] = P[X(t) \leq x \mid X(t_n) = x_n]$$

where $t_0, t_1, \ldots, t_n, t \in T$ and $x_0, x_1, \ldots, x_n \in S$. T and S are called the *parameter space* and *state space* of the Markov process, respectively. When T is a continuous space and 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 $\sigma_{i,j}$. Matrix $\mathbf{G}$ (also known as the *transition rate matrix*) is called a *differential matrix* if its entries satisfy property (2.4).

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

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

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

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

where $p_{i \Rightarrow j}(t)$ is the transition probability from state i (directly or indirectly) to state j during time 0 to t, and $p'_{i \Rightarrow j}(t)$ is its derivative.

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 A state i is said to be *recurrent* if and only if, starting from i, eventual return to this state is certain. A recurrent state is said to be *positive recurrent* if and only if the mean time to return to this state is finite. A state i is said to be *transient* if and only if, starting from i, there is a positive probability that the process may not eventually return to this state.

Definition 2.4 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*.

Definition 2.5 If the set of all states of a stochastic process $\mathbf{X}$ form a single communicating class, then $\mathbf{X}$ is *irreducible*.

Theorem 2.1

(1) If the Markov process is irreducible, then the limiting distribution $\lim_{t \to \infty} p_i(t) = p_i$, $i \in S$, exists and is independent of the initial conditions of the process, The limits $\{p_n \mid n \in S\}$ are such that they either vanish identically (i.e., $p_i = 0$ for all $i \in S$) or are all positive and form a probability distribution (i.e., $p_i > 0$ for all $i \in S$, $\Sigma_{i \in S} p_i = 1$).

(2) The limiting distribution $\{p_i, i \in S\}$ of an irreducible positive recurrent Markov process is given by the unique solution of the equation: $p\mathbf{G} = 0$ and $\Sigma_{j\in S} p_j = 1$ where $p = (p_0, p_1, \ldots)$.

Definition 2.6 If we map the states of a Markov process as vertices of a graph and the states transitions as directed edges between the vertices. The Markov process is called a *connected Markov process* if this graph is a connected graph.

For the discussions in the rest of this paper, we will omit the term "continuous-time" for more concise description. Unless otherwise stated, all processes are assumed 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 all 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: $r_i = r_{i,i} + \sum_{j\neq i}\sigma_{i,j} r_{i,j}$.

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:

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

It can 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 $dt\to 0$ and using the definition of earning rate r_i, we have:

$$\frac{d}{dt}v_i(t) = r_i + \sum_{j=1}^{n}\sigma_{i,j}v_j(t) \qquad i = 1, 2, \ldots, n \qquad (2.5)$$

where n is the total number of states of the process. Eqn. (2.5) gives a set of linear, constant coefficient differential equations that relate the total reward in time t from a starting state i to the quantities of r_i and $\sigma_{i,j}$.

Secondly, a *controllable Markov process* is a Markov process whose state transition rates can be controlled by controlling commands (defined as *actions*). When the system is in state i, an action a_i is chosen from a finite set A_i which includes all possible actions for state i. We denote this state action relation as $<i,a_i>$. 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 $<i,a_i(t)>$. The state transition rates $\sigma_{i,j}$ have different values when different actions are taken. We use $\sigma_{i,j}^{a_i(t)}$ to denote the transition rate from state i to state j when action $a_i(t)$ is taken for state i at time t. As a result, the generator matrix of a controllable Markov process can be represented by a parameterized (action is the parameter) matrix.

Definition 2.7 A policy π is the set of state-action pairs for all the states of a controllable Markov process, that is $\pi=\{<i,a_i(t)>|$ $a_i(t)\in A_i, 1\leq i\leq n\}$.

A *Markov decision process* is a controllable Markov process with rewards. In a Markov decision process, since $\sigma_{i,j}^{a_i(t)}$ is action-dependent, the reward rate r_i becomes also action-dependent, which is denoted as $r_i^{a_i(t)}$, $a_i(t)\in A_i$. The expected total reward $v_i(t)$ depends on the chosen action of each state, i.e., it becomes policy-dependent and is denoted as $v_i^\pi(t)$. The generator matrix $\mathbf{G}$ is then also policy-dependent and is denoted as $\mathbf{G}^\pi$.

Let $p_{i\Rightarrow j}^\pi(t)$ denote the probability of being in state j at time t when the initial state is i and the state transition rates are determined by policy π. The total expected reward that the process can earn for a time period of t, can be written as [9]:

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

Given two policies π_1 and π_2, if we can find a time ξ, such that $v_i^{\pi_1}(t) \geq v_i^{\pi_2}(t)$, for all $t>\xi$, $i = 1, 2, \ldots, n$, we denote as $\pi_1 \geq \pi_2$. A policy π is called the *optimal policy*, if $\pi \geq$ {any possible policy for the Markov decision process}.

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

(1) *limiting average reward*:

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

Obviously, maximizing limiting average reward is the same as maximizing the total expected reward.

(2) *discounted reward*:

$$v_{i,dis}^\pi(\alpha) = \lim_{t\to\infty}\int_0^t e^{-\alpha\tau} \sum_{j=1}^{n} p_{i\Rightarrow j}^\pi(\tau) r_j^{a_j(\tau)} d\tau,$$

Both reward models are meaningful for different context. The decision based on the average limiting reward assumes that the system will run forever, therefore, it considers return in near future and far future to have the same significance. While the decision based on the discounted reward considers the future as unpredictable, i.e., the system may be terminated any time in the future. Therefore, it emphasizes the return of the near future. The discount factor α decides how greedy this reward model is. The larger α is, the less it considers the future. When α approaches 0, the discounted reward approaches the total expected reward.

Definition 2.8 A policy π is *stationary* if the decision-making (action) is only a function of the state and independent of time, that is: $\pi=\{<i,a_i>| a_i\in A_i, 1\leq i\leq n\}$.

Theorem 2.2 [9] For any α, there exists a stationary policy which maximizes $v_{i,dis}^\pi(\alpha)$ for all $i = 1, 2, \ldots, n$. Such a policy is called α-optimal.

Definition 2.9 A policy π 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 inside each interval, the policy is stationary.

Theorem 2.3 [9] There exists a stationary policy that is α-optimal for a set of α having 0 as a limit point. This policy maximizes $v_{i,avg}$ over the class of piecewise-stationary policies.

We therefore do not lose generality if our search for the optimum policy is restricted on the set of stationary policies. Actions and policies that we will discuss later are thus time-independent.

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 (delay and power). These two problems are 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 $r_{i,j}$. For the rest of the discussion, our goal is to minimize the cost under either the discounted model or limiting average model.

III. SYSTEM MODELING

We assume 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 issued by the SR are serviced by the system. The system itself consists of three components: a resource that processes requests (the SP), a queue which stores the requests that cannot be serviced immediately upon arrival (SQ), and a power manager (PM).

Both the request arrival event and request service event are stochastic processes. We assume that they follow 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 (from the SR) and the service time (the time needed by the SP to service a request) follow the exponential distribution with mean $1/\lambda$.

In order to be more general, in our model we assume that the SP has more than one working mode, therefore, it can service the requests with more than one service speed. We also assume that all requests have the same service priority. The service of the requests are based on a FIFO order.

The SP can operate in a number of different power modes. We 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 the SP and the SQ) 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 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. Component modeling

*The **Service Requestor** (SR) has only one request generating mode. The average interval time of requests generated by SR follows the exponential distribution with mean value $1/\lambda$.*

An SR model with one request generating mode is a simplified model of a real SR whose request generating speed varies from time to time depending on the workload. From our observations, however, we find that the average inter-arrival time of a given Poisson process can be estimated within 5% error after observing 50 events. Therefore, if the input stays stable long enough, the power manager can observe and estimate the input rate dynamically, and adaptively change its policy.

*The **Service Provider** (SP) is modeled as a stationary, continuous-time controllable Markov process with state (operation mode) set*

$S=\{s_i \ s.t. \ i=1, 2, ..., S\}$, *action set A and generator matrix $G_{SP}(a)$, $a \in A$. It can be described by a quadruple (χ, $\mu(s)$, pow(s), ene(s_i, s_j)) where: (i) χ is an S×S matrix; (ii) $\mu(s)$ is a function μ: $S \rightarrow \mathbb{R}$; (iii) pow(s) is a function pow: $S \rightarrow \mathbb{R}$; (iv) ene(s_i, s_j) is a function eng: $S \times S \rightarrow \mathbb{R}$.*

We call χ the *switching speed* matrix. 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 $G_{SP}(a)$ can be calculated as:

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

$\delta(s,a)$ equals to 1 if s is the destination state of a, otherwise, $\delta(s,a)$ equals to 0.

A *service rate* $\mu(s)$ represents the service speed of SP in state s. Therefore, $1/\mu(s)$ gives the average time that is needed by SP to complete the service for one request when SP is in state s.

A *power consumption rate* 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), $s_i, s_j \in S$, $s_i \neq s_j$. 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. (3.5) we know that the expected power consumption of SP when it is in state s and action a_s is chosen can be calculated by: $c_s = pow(s) + \sum_{s' \neq s} \sigma_{s,s'}(a_s)ene(s,s')$.

We can divide the state set S into two groups:

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

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

Furthermore, we can divide the matrix $G_{SP}(a)$ into two parts:

$$G_{SP}^{\pi}(a) = \begin{bmatrix} G_{SP}^{AA}(a) & G_{SP}^{AI}(a) \\ G_{SP}^{IA}(a) & G_{SP}^{II}(a) \end{bmatrix}$$

where matrix $G_{SP}^{AA}(a)$ contains the transition rate for transitions between inactive states. Matrix $G_{SP}^{AI}(a)$ contains the transition rates for transitions from any inactive state to any active state. $G_{SP}^{IA}(a)$ and $G_{SP}^{II}(a)$ are defined similarly.

Example 4.1 Consider a SP with three states, $S=\{active, waiting, sleeping\}$. Let the command set be defined as $A=\{wakeup, wait, sleep\}$. Assume that all three commands are valid in any of the three states. The switch speed matrix χ is a 3×3 matrix, with one entry for each state pair. For instance:

$$\chi = \begin{pmatrix} \infty & 1/0.1 & 1/0.2 \\ 1/0.5 & \infty & 1/0.05 \\ 1/1.1 & 1/0.7 & \infty \end{pmatrix}.$$

It means that the average switch time from state *active* to state *waiting* is 0.1sec, the average switch time from state *active* to state *sleeping* is 0.2sec, the average switch time from state *waiting* to state *active* is 0.5sec, etc. The power consumption rate pow(s) can be represented by a vector. For instance: pow(s)=(4w, 2w, 0w). It

means that the power consumption in state active is 4w, the power consumption in state wait is 2w and the power consumption in state sleep is 0w. The switch energy $ene(s_i, s_j)$ can be represented by a two-dimensional tables with one entry for each state pair. For instance:

$$ene(s_i, s_j) = \begin{pmatrix} \infty & 0.1J & 0.2J \\ 0.5J & \infty & 0.05J \\ 1J & 0.8J & \infty \end{pmatrix}.$$

It means that the switch from state *active* to state *waiting* dissipate 0.1J energy, the switch from state *active* to state *sleeping* dissipate 0.2J energy, the switch from state *waiting* to state *active* dissipate 0.5J energy, etc.

Given a policy, the continuous-time Markov decision process model of SP can be fully described by the quadruple $(\chi, \mu(s), pow(s), ene(s_i, s_j))$. For example, if the policy chosen is: $\pi=\{<active, waiting>, <waiting, sleep>, <sleeping, wakeup>\}$, the generator matrix can be written as:

$$G_{SP}^{\pi} = \begin{bmatrix} -1/0.1 & 1/0.1 & 0 \\ 0 & -1/0.05 & 1/0.05 \\ 1/1.1 & 0 & -1/1.1 \end{bmatrix}.$$

Figure 1 gives the illustration of this Markov process. The cost for each state under policy π is:

$$c_{active} = 5w, \ c_{waiting} = 3w, \ c_{sleeping} = 0.909w$$
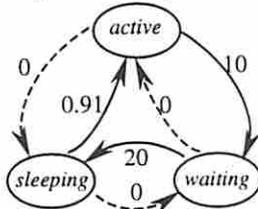

Figure 1 Markov process model of the SP

*The **Service Queue** (SQ) is modeled as a stationary, continuous-time controllable Markov process, with state set $Q=Q_{stable} \lor Q_{transfer}$, where $Q_{stable}=\{q_i \ s.t. \ i=0, 1, 2, ..., Q\}$, $Q_{transfer}=\{q_{i \to i-1} \ s.t. \ i=i, 2, ..., Q\}$, and the generator matrix $G_{SQ}(s, a_{(q,s)})$, where s is the SP state, $a_{(q,s)}$ is the action when SP is in state s and SQ is in state q.*

The model of SQ is constructed based on that of an M/M/1 queue. The queue length is Q. We assume that the request will be lost if the SQ is full at the time the request arrives.

The state set $Q_{transfer}$ is the set of *transfer states*, which represent the states of the SQ when the service of a request has been finished and the service of the next request has not started. Note that there is a concurrency constraint between the SQ and the SP as follows: Whenever the SQ is in a transfer state, the SP is transitioning from one state to next. Furthermore, the SQ leaves the transfer state exactly when the SP transition is complete.

The state set Q_{stable} is the set of *stable states*, i.e., states of the SQ other than the transfer states. We denote a stable state as q_i, which also implies that there are i requests in the SQ.

Given the state of SP, the transition rates between the states of SQ are fixed. There are four types of possible transitions. They are:

(1) The transition from stable state q_i to stable state q_{i+1}. The SQ will make this transition when it is in state q_i and a request is generated by the SR. The transition rate is: $\sigma_{q_i, q_{i+1}} = \lambda$, $i=0, 1,$ ..., $Q-1$, where λ is the request generation rate of SR.

(2) The transition from stable state q_i to transfer state $q_{i \to i-1}$. The SQ will make this transition when it is in state q_i and the service for the current request is completed by the SP. The transition rate is: $\sigma_{q_i, q_{i \to i-1}} = \mu(s)$, $i=1, 2, ..., Q$

(3) The transition from transfer state $q_{i \to i-1}$ to stable state q_{i-1}. The SQ will make this transition when it is state $q_{i \to i-1}$ and the SP completes switching and starts to provide service for the next request. The transition rate is: $\sigma_{q_{i \to i-1}, q_{i-1}} = \chi_{s,s'}$, $i=0, 1, ..., Q-1$, where s' is the destination state of action $a_{(q,s)}$.

(4) The transition from transfer state $q_{i \to i-1}$ to transfer state $q_{i+1 \to i}$. The SQ will make this transition when it is in state $q_{i \to i-1}$ and a request is generated by the SQ. The transition rate is: $\sigma_{q_{i \to i-1}, q_{i+1 \to i}} = \lambda$, $i=1, 2, ..., Q-1$.

For the sake of brevity, we do not describe the boundary case when the SQ is in state $q_{Q \to Q-1}$ and a request is generated. Transition between states other than these four classes has a rate of 0. The self-transition rate can then be calculated as:

$$\sigma_{q,q} = -\sum_{q' \neq q} \sigma_{q,q'}, \ q, q' \in Q.$$

Based on the above grouping, we can divide the matrix $G_{SQ}(s,a)$ into four parts: $G_{SQ}(s,a) = \begin{bmatrix} G_{SQ}^{SS}(s,a) & G_{SQ}^{ST}(s,a) \\ G_{SQ}^{TS}(s,a) & G_{SQ}^{TT}(s,a) \end{bmatrix}$, where matrix $G_{SQ}^{SS}(s,a)$ contains the transition rate for transitions between stable states. Matrix $G_{SQ}^{ST}(s,a)$ contains the transition rates for transitions from any stable state to any transfer state. $G_{SQ}^{TS}(s,a)$ and $G_{SQ}^{TT}(s,a)$ are defined similarly.

Example 3.3 Consider the SP model given in previous example, and assume a maximum queue length 2. Thus, $Q_{stable}=\{q_0, q_1, q_2\}$; $Q_{transfer}=\{q_{1 \to 0}, q_{2 \to 1}\}$; $Q=Q_{stable} \lor Q_{transfer}$. The policy is based on the joint state of SP, SQ. The generator matrix $G_{SQ}(s, a_{(q,s)})$ is a 5×5 matrix. Assume that whenever the SQ is in transfer state, the PM will issue command *sleep*. Thus $G_{SQ}(s, a_{(q,s)})$ can be written as:

$$G_{SQ}(s,a) = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 \\ 0 & -\lambda-\mu(s) & \lambda & \mu(s) & 0 \\ 0 & 0 & -\mu(s) & 0 & \mu(s) \\ \chi_{A.S.} & 0 & 0 & -\chi_{A.S.}-\lambda & \lambda \\ 0 & \chi_{A.S} & 0 & 0 & -\chi_{A.S.} \end{bmatrix}$$

Figure 2 gives the illustration of the Markov process model of this SQ. This Markov process model of SQ is different when the SP and SR are in different states.
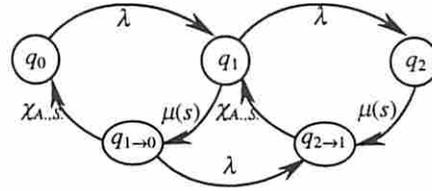

Figure 2 Markov process model of the SQ

B. System modeling

The **Power-Managed System** (SYS) can be modeled as a stationary continuous-time controllable Markov process which is the composition of the Markov processes of the SP and the SQ. The state set is given $X=S \times Q_{stable} \vee S_{active} \times Q_{transfer}$. The action set is the same as that in the SP model. A parameterized generator matrix $G_{SYS}(a)$ gives the state transition rates under action a. A cost function $Cost(a)$ gives the system cost under action a when the SYS is in state x.

There is an action set A_x associated with each state x. When the system is in state x the PM chooses a command from the A_x. The action gives the mode of SP that it should switch to. Not all actions are valid in all states. Constraints on a valid action are as follows:

(1) When the SQ is in stable state, the SP cannot switch from active state to inactive state.

(2) When the SQ is in stable state q_Q (SQ is full), the SP cannot switch from an inactive state to another inactive state with longer wakeup time.

(3) When the SQ is in transfer state $q_{Q \to Q-1}$, the SP cannot switch from an active state to another active state with longer service time.

The first constraint ensures that the work of SP will not be interrupted by the command issued by the PM and all commands issued by the PM will be accepted by the SP with probability 1. The last two constraints ensure that the resulting SYS model is a connected Markov process. Consequently, the limiting distribution of the state probability exists and is independent of the initial state [7]. These two constraints are also reasonable, because when SP and SQ the are in these forbidden states, then the service speed cannot follow the generation speed of the requests. Therefore, we need to increase the service speed.

There is some dependence between the Markov process model of SQ and the Markov process model of SP because the transfer states of SQ are associated with the active states of SP and their transitions are synchronized. When the SQ is in stable state, however, the SQ is independent from the SP.

In this section, we are going to talk about how to construct the generator matrix G_{SYS} based on G_{SQ} and G_{SP}. First, we will show how to calculate the generator matrix of a joint process of two independent continuous-time Markov processes. Proposition 3.1 gives a way to obtain the joint transition rate of two independent continuous-time Markov processes. Proposition 3.2 shows the way to generate the generator matrix of the joint system using matrix operation.

Proposition 3.1 *Given two independent stochastic process X_1 and X_2, let $\sigma_{(x,y),(x'y')}$ denote the transition rate of the joint process from the joint state (x,y) to state (x',y'), where x and $x' \in$ state space of X_1, y and $y' \in$ state space of X_2. Let $\sigma_{x,x'}$ denote the transition rate of process X_1 from state x to state x' and $\sigma_{y,y'}$ denote the transition rate of process X_2 from state y to state y'. Then $\sigma_{(x,y),(x,y')} = \sigma_{y,y'}$, $\sigma_{(x,y),(x',y)} = \sigma_{x,x'}$, $\sigma_{(x,y),(x',y')} = 0$.*

Proof:

Let $p_{x \Rightarrow x'}(t)$ denote the transition probability of process X_1 from state x to x' during t, $p_{y \Rightarrow y'}(t)$ denote the transition probability of process X_2 from state y to y' during t. Let $p_{(x,y) \Rightarrow (x',y')}(t)$ denote the joint transition probability of process $X_1 X_2$ from state (x,y) to (x',y'). From the definition of transition rate, we know that:

$$\lim_{t \to 0} p_{x \Rightarrow x'}(t) = \lim_{t \to 0}(\sigma_{x,x'} t + o(t))$$

$$\lim_{t \to 0} p_{y \Rightarrow y'}(t) = \lim_{t \to 0}(\sigma_{y,y'} t + o(t))$$

$$\lim_{t \to 0} p_{x \Rightarrow x}(t) = \lim_{t \to 0}(1 - \sum_{x' \neq x} \sigma_{x,x'} t + o(t))$$

$$\lim_{t \to 0} p_{y \Rightarrow y'}(t) = \lim_{t \to 0}(1 - \sum_{y' \neq y} \sigma_{y,y'} t + o(t))$$

Since X_1, X_2 are independent, we the following equations:

$$\lim_{t \to 0} p_{(x,y) \Rightarrow (x,y')}(t) = \lim_{t \to 0}(p_{x \Rightarrow x}(t) p_{y \Rightarrow y'}(t)) =$$

$$\lim_{t \to 0}(\sigma_{y,y'} t + \sum_{x' \neq x} \sigma_{y,y'} \sigma_{x,x'} t^2 + o(t)) = \lim_{t \to 0}(\sigma_{y,y'} t + o(t)) \quad \text{(P1.1)}$$

$$\lim_{t \to 0} p_{(x,y) \Rightarrow (x',y)}(t) = \lim_{t \to 0}(p_{x \Rightarrow x'}(t) p_{y \Rightarrow y}(t)) =$$

$$\lim_{t \to 0}(\sigma_{x,x'} t + \sum_{y' \neq y} \sigma_{y,y'} \sigma_{x,x'} t^2 + o(t)) = \lim_{t \to 0}(\sigma_{x,x'} t + o(t)) \quad \text{(P1.2)}$$

$$\lim_{t \to 0} p_{(x,y) \Rightarrow (x',y')}(t) = \lim_{t \to 0}(p_{x \Rightarrow x'}(t) p_{y \Rightarrow y'}(t)) =$$

$$\lim_{t \to 0}(\sigma_{x,x'} \sigma_{y,y'} t^2 + o(t)) = \lim_{t \to 0}(o(t)) \quad \text{(P1.3)}$$

From equation (P1.1), (P1.2) and (P1.3) we know that:

$\sigma_{(x,y),(x,y')} = \sigma_{y,y'}$, $\sigma_{(x,y),(x',y)} = \sigma_{x,x'}$, $\sigma_{(x,y),(x',y')} = 0$

Definition 3.1 Consider two matrices A and B:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$ The **tensor product**

$C = A \otimes B$ is given by $C = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix}$. The **tensor sum**

$C = A \oplus B$ is given by: $C = A \otimes I_{n_2} + I_{n_1} \otimes B$, where n_1 is the order of A, n_2 is the order of B, I_{n_i} is the identity matrix of order n_i.

Proposition 3.2 *Given two independent continuous-time Markov processes with generator matrices A and B, the generator matrix of the joint process is the tensor sum of A and B.*

Proof:

Assume the generator matrix A can be written as:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n_1} \\ \vdots & \ddots & \vdots \\ a_{n_1 1} & \cdots & a_{n_1 n_1} \end{bmatrix}$$

Assume the generator matrix B can be written as:

$$B = \begin{bmatrix} b_{11} & \cdots & b_{1n_1} \\ \vdots & \ddots & \vdots \\ b_{n_1 1} & \cdots & b_{n_1 n_1} \end{bmatrix}$$

Let's denote the new matrix as C, $C = A \oplus B$. From the definition of the Tensor sum we know that

$$c_{i \cdot n_2 + j, m \cdot n_2 + n} = \begin{cases} a_{ii} + b_{jj} & i = m, j = n \\ b_{jn} & i = m, j \neq n \\ a_{im} & i \neq m, j = n \\ 0 & i \neq m, j \neq n \end{cases} \quad \text{(P2.1)}$$

Let's denote the generator matrix of the joint process as **G**. In the joint process, the states are ordered in the lexicography order of A and B, i.e., they are ordered as: $(a_1, b_1), (a_1, b_2), \ldots, (a_1, b_{n_2}), (a_2, b_1), (a_2, b_2), \ldots, (a_2, b_{n_2}), \ldots$.

Therefore, $\sigma_{(a_i,b_j),(a_m,b_n)} = G_{i \cdot n_1 + j, m \cdot n_1 + n}$. From Theorem 4.1 we know that:

$$G_{i \cdot n_1 + j, m \cdot n_1 + n} = \begin{cases} \sigma_{(a_i,b_j),(a_i,b_j)} = \sigma_{a_i,a_i} + \sigma_{b_j,b_j} & i = m, j = n \\ \sigma_{(a_i,b_j),(a_i,b_n)} = \sigma_{b_j,b_n} & i = m, j \neq n \\ \sigma_{(a_i,b_j),(a_m,b_j)} = \sigma_{a_i,a_m} & i \neq m, j = n \\ \sigma_{(a_i,b_j),(a_m,b_n)} = 0 & i \neq m, j \neq n \end{cases}$$ (P2.2)

Because $a_{i,m} = \sigma_{a_i,a_m}$ and $b_{j,n} = \sigma_{b_j,b_n}$, (P2.1) is equal to (P2.2). Therefore, $\mathbf{A} \oplus \mathbf{B}$ is the generator matrix of the joint process of $\mathbf{A}$ and $\mathbf{B}$.

The state of the SYS can be grouped to two part: $X = X_A \vee X_B$, $X_A = S \times Q_{stable}$, $X_B = S_{active} \times Q_{transfer}$. Based on the grouping, we can divide the matrix $\mathbf{G}_{SYS}(a)$ into four parts: $\mathbf{G}_{SYS}(a) = \begin{bmatrix} \mathbf{G}_{SYS}^{AA}(a) & \mathbf{G}_{SYS}^{AB}(a) \\ \mathbf{G}_{SYS}^{BA}(a) & \mathbf{G}_{SYS}^{BB}(a) \end{bmatrix}$, where matrix $\mathbf{G}_{SYS}^{AA}(a)$ contains the transition rate for transitions between X_A states. Matrix $\mathbf{G}_{SYS}^{AB}(a)$ contains the transition rates for transitions from any X_A states to any X_B states. $\mathbf{G}_{SYS}^{BA}(a)$ and $\mathbf{G}_{SYS}^{BB}(a)$ are defined similarly.

Proposition 3.3 The generator matrix $\mathbf{G}_{SYS}(a)$ can be calculated as:

$$\mathbf{G}_{SYS}(a) = \begin{bmatrix} \mathbf{G}_{SP}(a) \oplus \mathbf{G}_{SQ}^{SS}(a) & \mathbf{M}(a) \\ \mathbf{G}_{SP}^{A}(a) \otimes \mathbf{N} & \mathbf{I}_{S_{active}} \otimes \mathbf{G}_{SQ}^{TT}(a) \end{bmatrix},$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{I}_{S_{active}} \otimes \mathbf{G}_{SQ}^{ST}(a) \\ \mathbf{O}_1 \end{bmatrix}, \quad \mathbf{N} = \begin{bmatrix} \mathbf{I}_Q & \mathbf{O}_2 \end{bmatrix}, \quad \mathbf{G}_{SP}^{A} = \begin{bmatrix} \mathbf{G}_{SP}^{AA} & \mathbf{G}_{SP}^{AI} \end{bmatrix},$$

$\mathbf{O}_1$ is a $S_{inactive} \cdot (Q+1) \times (S_{active} \cdot Q)$ matrix of all zeros, $\mathbf{O}_2$ is a column vector of all zeros. The diagonal entries of $\mathbf{G}_{SYS}(a)$ are calculated as: $\sigma_{i,i}(a) = -\sum_{j \neq i} \sigma_{i,j}(a)$.

Proof:
Because SQ and SP are independent when SQ is in a stable state, from proposition 4.3 we know that:

$$\mathbf{G}_{SYS}^{AA}(a) = \mathbf{G}_{SP}(a) \oplus \mathbf{G}_{SQ}^{SS}(a)$$

According to Proposition 3.1 and the definition of the transfer state, the transition rate from state (s, q_i) to (s, q') is $\sigma_{SQ}^{q,q'}(s,a)$, where $s \in S_{active}$, $q \in Q_{stable}$, $q' \in Q_{transfer}$, the other transition rate of switching from X_A state to X_B state is 0. Therefore, we can write $\mathbf{G}_{SYS}^{AB}(a)$ as:

$$\mathbf{G}_{SYS}^{AB}(a) = \begin{bmatrix} \mathbf{G}_{SQ}^{ST}(a) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{G}_{SQ}^{ST}(a) \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} = \mathbf{M}(a)$$

The transition rate from state (s, q') to (s', q) is $\sigma_{SQ}^{q',q}(s,a)$, when $s', s \in S$, $q \in Q_{stable}$, $q' \in Q_{transfer}$. The other transition rate of switching from X_B state to X_A state is 0. Therefore, we can write $\mathbf{G}_{SYS}^{BA}(a)$ as:

$$\mathbf{G}_{SYS}^{BA}(a) = \begin{bmatrix} \mathbf{N}(\chi_{s_1,s_1}) & \cdots & \mathbf{N}(\chi_{s_1,s_{S_A}}) \\ \vdots & \ddots & \vdots \\ \mathbf{N}(\chi_{s_{S_A},s_1}) & \cdots & \mathbf{N}(\chi_{s_{S_A},s_{S_A}}) \end{bmatrix} \text{ where}$$

$$\mathbf{N}(\chi) = \begin{bmatrix} \chi & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \chi & 0 \end{bmatrix}.$$

Therefore, $\mathbf{G}_{SYS}^{BA}(a) = \mathbf{G}_{SP}^{A}(a) \otimes \mathbf{N}$.

The transition rate from state (s, q) to (s, q') is $\sigma_{SQ}^{q',q}(s,a)$, when $s \in S_{active}$, $q, q' \in Q_{transfer}$. Therefore, we can write $\mathbf{G}_{SYS}^{BB}$ as:

$$\mathbf{G}_{SYS}^{BB}(a) = \begin{bmatrix} \mathbf{G}_{SQ}^{TT} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{G}_{SQ}^{TT} \end{bmatrix} = \mathbf{I}_{S_{active}} \otimes \mathbf{G}_{SQ}^{TT}(a)$$

The diagonal entries of $\mathbf{G}_{SYS}(a)$ are calculated using (2.4).

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. We use the average power consumption $C_{pow}(x,a)$ and the average number of waiting requests C_{sq} to capture the system cost.

Let x be denoted by (s, q), where $s \in S$, $q \in Q$. The power cost can be calculated as: $C_{pow}(x,a) = pow(s) + \sum_{s' \in S, s' \neq s} \sigma_{s,s'}(a) ene(s,s')$.

The delay cost is: $C_{sq} = i$, when SQ is in stable state q_i or transfer state $q_{i+1 \to i}$.

We define a total cost as a weighted summation of the power and delay costs $Cost(x,a) = C_{pow}(x,a) + w \cdot C_{lsq}(x)$ (3.1)

IV. POLICY OPTIMIZATION

The problem of power management is to find the optimal set of state-action pairs for the PM such that the expected power consumption is minimized subject to the performance constraints. This problem can be formally written as:

$$\min_{\pi} \lim_{t \to \infty} \frac{1}{t} \int_0^t \sum_{x \in X} p_{x \Rightarrow x'}^{\pi}(\tau) C_{pow}(x, a^{\pi}) d\tau,$$

s.t. $\lim_{t \to \infty} \frac{1}{t} \int_0^t \sum_{x \in X} p_{x \Rightarrow x'}^{\pi}(\tau) C_{sq}(x) d\tau \leq D_M$, $\forall x, x \in X$,

Where $p_{x \Rightarrow x'}^{\pi}(\tau)$ is the state transition (direct or indirect) probability from state x to x' in a time period of τ under policy π. a^{π} in "$C_{pow}(x, a^{\pi})$" denotes the action in state x.

Another problem formulation is:

$$\min_{\pi} \lim_{t \to \infty} \frac{1}{t} \int_0^t \sum_{x \in X} p_{x \Rightarrow x'}^{\pi}(\tau) Cost(x, a^{\pi}) d\tau, \quad \forall x, x \in X$$

By adjusting the weights in Eqn. (3.1), we can achieve minimum power under different delay constraints. Figure 3 gives the workflow of our policy optimization algorithm.

It is proved in [10] that for a given policy π, we have:

$$v_i^{\pi}(t) = c_i^{a_i} + \sum_{j=1}^{n} \sigma_{i,j}^{a_i} v_j^{\pi} + t \sum_{j=1}^{N} \sigma_{i,j}^{a_i} g_j^{\pi}, \quad i = 1, 2, ..., n$$

where g_j^{π} and v_j^{π} are related by the equation:

$$\sum_{j=1}^{N} \sigma_{i,j}^{a_i} g_j^{\pi} = 0, \quad i = 1, 2, ..., n$$ (4.1)

and $g_i^\pi = c_i^{a_i} + \sum_{j=1}^{N} \sigma_{i,j}^{a_i} v_j^\pi$, $i = 1, 2, ..., n$ (4.2)

To minimize (4.9), when t is very large, we must find an action a_i that minimizes the term

$$\sum_{j=1}^{N} \sigma_{i,j}^{a_i} g_j^\pi \qquad (4.3)$$

When there are some actions, which give the same minimized value in this term, then minimize the second term

$$\sum_{j=1}^{n} \sigma_{i,j}^{a_i} v_j^\pi \qquad (4.4)$$

An iterative algorithm is proposed in [10] to solve the above optimization problem. The inner iteration of Figure 3 gives the algorithm. It can be proved that a better policy is found in each iteration [10], that is the cost of each successive policy is monotone non-increasing.
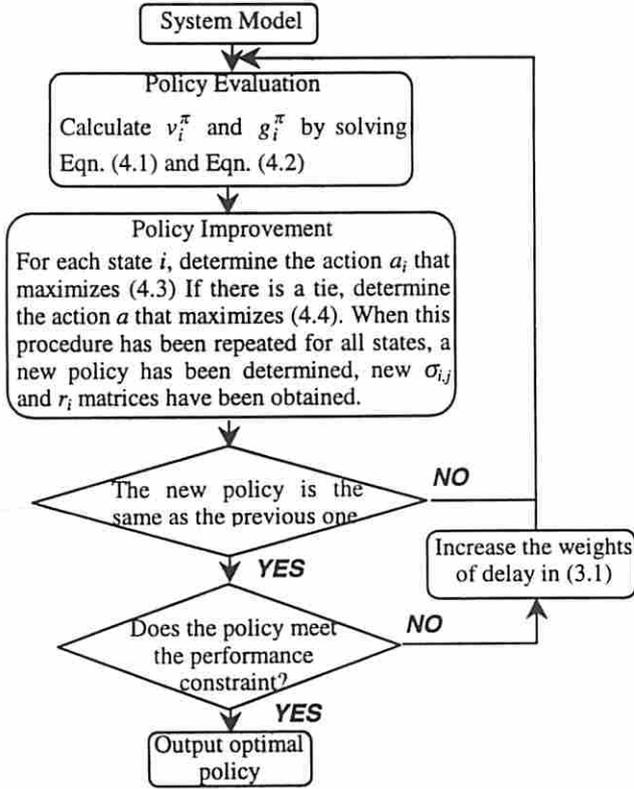

Figure 3 Policy optimization workflow

V.   EXPERIMENTAL RESULTS

First we describe a class of heuristic policies that can give trade off between power and performance. An N-policy is a policy that activates the server when there are N customers waiting for service and deactivates the server when there are no customer in the system [12]. 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 tradeoff.

Our experimental setup is as follows. We have written an event-driven simulator for simulating the real-time operation of a portable system together with the power management policy. The simulator simulates the operations of the server, the queue and the power manager under real-time input requests. The server has three states: *active*, *waiting* and *sleeping*. We set the length of the queue to 5. Tasks are represented by a sequence of events. The interval time between two consecutive requests is generated randomly to follow an exponential distribution with mean value of 6sec. Therefore $\lambda=0.167$ in the stochastic model of the system. The total number of requests is 50,000. The service time of each task is also generated according to an exponential distribution with mean value 1.5sec. Therefore, $\mu(active)=0.67$ in the stochastic model of the system.

When the system state changes, the power manager is triggered and a new command is issued according to the current system state. The switching time of the server is also generated randomly. Eqn. (4.1) (a) gives the experimental value of the average switching time. The time is given in seconds. Note that these refer to the values of $1/\chi_{i,j}$ in the stochastic model. We set the server power dissipation when the server is active, waiting and sleeping to 40w, 15w and 0.1w, respectively. These values are assigned to the corresponding cost rates $c_{i,i}$ in the stochastic model. The energy needed for each transition (given in J) is given in Eqn. (4.1) (b). These values are assigned to the corresponding transition costs $c_{i,j}$ in the model. The performance and the cost metrics are measured by the average number of waiting requests and the average power dissipation of the system during the simulation.

$$(a)\ tr\_time = \begin{bmatrix} - & .1 & .2 \\ .5 & - & .1 \\ 1.1 & .5 & - \end{bmatrix}, (b)\ tr\_energy = \begin{bmatrix} - & .2 & .5 \\ 1 & - & .1 \\ 11 & 25 & - \end{bmatrix} \qquad (5.1)$$

In the first experiment, we changed the value of the performance weight of our algorithm and obtained a set of optimum policies. We also generated a set of N-policies using $N=1, 2, ..., 5$. Figure 4 shows the comparison of the simulated values of perfomance and power of the two sets. Note that the leftmost (rightmost) N-policy solution in Figure 4 corresponds to N=1 (N=5). We find our policy gives better power-delay tradeoff than the N-policy. In the experiments, we also calculated the functional value of the queue length and energy cost (by using the state probability and the state cost) and found that the functional value and the simulated value are almost the same. This shows that our stochastic model of the power-managed system matches the real situation very well.
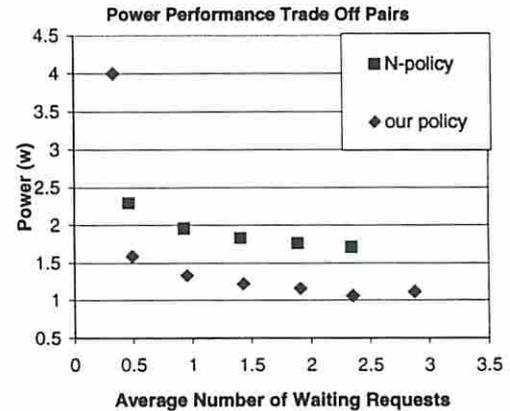

Figure 4 Comparison of our policy and The N-policy

In the second experiment, we assume that the performance constraint of the system is to keep the average output rate (throughput) the same as the input rate. That is, the average time that each task stays in the queue, (i.e. average waiting time), should be equal to or less than the average inter-arrival time of tasks. In the algorithm, the performance constraint is defined in terms of the average number of waiting requests. Therefore, we must convert the average waiting time to the average number of waiting requests. We used the approximataion that the average number of waiting requests equals the input rate times the average waiting time of each request. Table 1 gives the simulated values of the average waiting time and the corresponding queue length. It shows that the approximation is within 5% error of the actual value.

Table 1 Comparison of real average queue length and the approximated average queue length

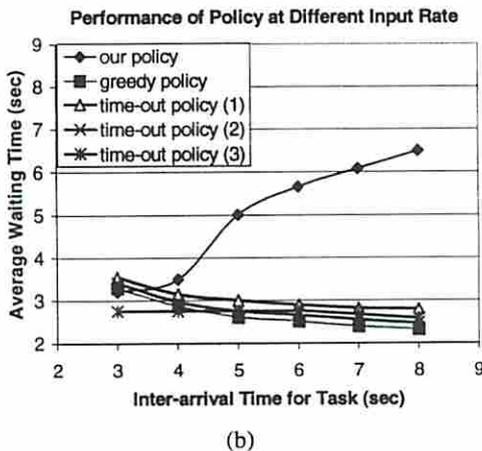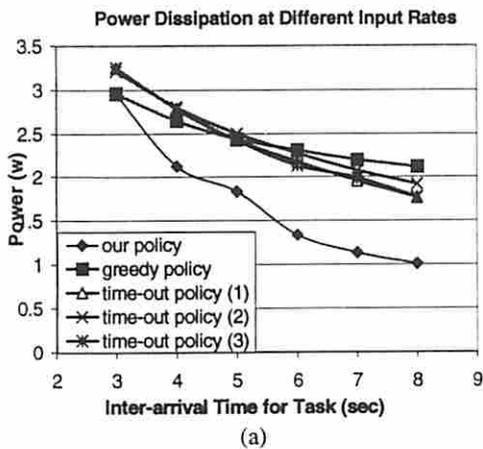| Input Rate (1/sec)	1/8	1/7	1/6	1/5	1/4	1/3
Avg. Waiting Time (sec)	6.493	6.08	5.658	5.008	3.50	3.30
Aprox. # of Waiting Requests	0.811	0.868	0.943	1.001	0.875	1.10
Actual # of Waiting Requests	0.816	0.869	0.94	1.053	0.861	1.05
Error of Apporximation(%)	-0.6	-0.1	0.3	-4.9	1.6	4.7

(a)

(b)

Figure 5 Comparison of our policy and heuristic policies

In the last experiment, we used a set of input tasks where the input rate varied from 1/8 to 1/3. The corresponding average interval time of the tasks varied from 8sec to 3sec. We compared the power-delay curves for our policy with four heuristic algorithms. Among heuristic approaches, one is a greedy algorithm which deactivates (activates) the server as soon as the queue is empty (the queue is not empty). The other three are time-out policies, which deactivate the server n seconds after it becomes idle. In time-out policy (1), n is fixed to 1sec. In policy (2), n is equal to the average inter-arrival time of the input tasks. In policy (3), n is equal to half of the inter-arrival time of input tasks. Figure 5 shows the simulated value of power and the average waiting time. We can see that our algorithm gives best power dissipation while satisfying the performance constraint.

VI. CONCLUSION

We have proposed a new system model and method for dynamic power management in system-level. The problem of system-level power management was formulated as the continuous-time Markov decision process based on the theories of continuous-time Markov decision process, and stochastic network. Compared to previous work, our model can represent the system behavior more intuitively and more accurately by considering the close relationship between the server status and the queue status. By modeling the system as a queue in the domain of continuous-time, the parameters in the model become more realistic such that they can be collected easily and precisely. Experimental results were presented to show that our approach is more flexible and more effective than heuristic approaches to achieve the best power-performance tradeoff.

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] U. Narayan Bhat, "*Elements Of Applied Stochastic Processes*", John Wiley & Sons, Inc. 1984
[8] B. Miller, "Finite State Continuous Time Markov Decision Processes With an Finite Planning Horizon." *SIAM J. Control*, Vol. 5, No. 2, pp. 266-281, 1968.
[9] 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.
[10] R.A.Howard, *Dynamic Programming and Markov Processes*, Wiley, New York, 1960
[11] G. A. Paleologo, L. Benini, et.al, "Policy Optimization for Dynamic Power Management", *Proceedings of Design Automation Conference*, pp.182-187, Jun. 1998.
[12] D. P. Heyman, M. J. Sobel, *Stochastic Models in Operations Research*, McGraw-Hill Book Company, 1982
[13] L. Benini, A. Bogliolo, S. Cavallucci, B. Ricco, "Monitoring System Activity For OS-Directed Dynamic Power Management",

Proceedings of International Symposium of Low Power Electronics and Design Conference, pp. 185-190, Aug. 1998.

[14] L. Benini, R. Hodgson, P. Siegel, "System-level Estimation And Optimization", *Proceedings of International Symposium of Low Power Electronics and Design Conference*, pp. 173-178, Aug. 1998.

[15] G. Bolch, S. Greiner, H. D. Meer and K. S. Trivedi, *Queueing Networks and Markov Chains*, John Wiley & Sons, Inc., 1998

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

[17] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *Proc. of the Intl. Conference on Computer Aided Design*, pages 28-32, November 1997.