Chapter 1

DYNAMIC MANAGEMENT OF POWER CONSUMPTION

Tajana Simunic
HP Labs

Abstract Power consumption of electronic devices has become a serious concern in the recent years. Power efficiency is necessary to lengthen the battery lifetime in the portable systems, as well as to reduce the operational costs and the environmental impact of stationary systems. Two new approaches that enable systems to save power by adapting to changes in environment are proposed: *dynamic power management* and *dynamic voltage scaling*. Dynamic power management (DPM) algorithms aim to reduce the power consumption at the system level by selectively placing components into low-power states. A new event-driven power management algorithm that guarantees globally optimal decisions is presented that is based on Time-Indexed Semi-Markov Decision Process model (TISMDP). TISMDP power management policies have been implemented on four devices: two different hard disks, a laptop WLAN card and a SmartBadge portable system [1]. The measurement results show power savings ranging from a factor of 1.7 up to 5.0 with performance basically unaffected. Dynamic voltage scaling (DVS) algorithms reduce energy consumption by changing processor speed and voltage at run-time depending on the needs of the applications running. This work extends the TISMDP power management model with a DVS algorithm, thus enabling even larger power savings. The measurements of MPEG video and MP3 audio algorithms running on the SmartBadge portable device show savings of a factor of three in energy consumption for combined DVS and DPM approaches.

Introduction

Power consumption has become one of the primary concerns in electronic design due to the recent popularity of portable devices and the environmental concerns related to desktops and servers. The battery capacity has improved very slowly (a factor of 2 to 4 over the last 30 years), while the computational demands have drastically increased over the same time frame. Better low-power circuit design techniques have helped to increase battery lifetime [2,3,4]. On the other hand, managing power dissipation at higher levels can

considerably decrease energy requirements and thus increase battery lifetime [5].

System-level *dynamic power management* [6] decreases the energy consumption by selectively placing idle components into lower power states. System resources can be modelled using state-based abstraction where each state trades off performance for power [7]. The transitions between states are controlled by commands issued by a *power manager* (PM) that observes the workload of the system and decides when and how to force power state transitions. The power manager makes state transition decisions according to the *power management policy*. The choice of the policy that minimizes power under performance constraints (or maximizes performance under power constraint) is a constrained optimisation problem. The most common power management policy at the system level is a timeout policy implemented in most operating systems. The drawback of this policy is that it wastes power while waiting for the timeout to expire [8,9]. Predictive policies developed for interactive terminals [10,11] force the transition to a low power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. An incorrect estimate can cause both performance and energy penalties. Both timeout and predictive policies are heuristic in nature, and thus do not guarantee optimal results.

In contrast, approaches based on stochastic models can guarantee optimal results. Stochastic models use distributions to describe the times between arrivals of user requests (*interarrival times*), the length of time it takes for a device to service a user request, and the time it takes for the device to transition between its power states. The system model for stochastic optimisation can be described either with memoryless distributions (exponential or geometric) [12,13,14] or with general distributions [15]. Power management policies can be classified into two categories by the manner in which decisions are made: *discrete time* (or clock based) [12,13] and *event driven* [14,15]. In addition, policies can be *stationary* (the same policy applies at any point in time) or *non-stationary* (the policy changes over time). All stochastic approaches except for the discrete adaptive approach presented in [13] are stationary. The optimality of stochastic approaches depends on the accuracy of the system model and the algorithm used to compute the solution. In both discrete and event-driven approaches optimality of the algorithm can be guaranteed since the underlying theoretical model is based on Markov chains. Approaches based on the discrete time setting require policy evaluation even when in low-power state [12,13], thus wasting energy. On the other hand, event-driven models based on only exponential distributions show little or no power savings when implemented in real systems since the exponential model does not describe well the request interarrival times of users [15].

Dynamic voltage scaling (DVS) algorithms reduce energy consumption by changing processor speed and voltage at run-time depending on the needs of the applications running. Early DVS algorithms set processor speed based on the processor utilization of fixed intervals and did not consider the individual requirements of the tasks running. There has been a number of voltage scaling techniques proposed for real-time systems. The approaches presented in [16,17,18,19] assume that all tasks run at their worst case execution time (WCET). The workload variation slack times are exploited on task-by-task basis in [20], and are fully utilized in [21]. Work presented in [22] introduces a voltage scheduler that determines the operating voltage by analysing application requirements. The scheduling is done at task level, by setting processor frequency to the minimum value needed to complete all tasks. For applications with high frame-to-frame variance, such as MPEG video, schedule smoothing is done by scheduling tasks to complete twice the amount of work in twice the allocated time. In all DVS approaches presented in the past, scheduling was done at the task level, assuming multiple threads. The prediction of task execution times was done either using worst case execution times, or heuristics. Such approaches neglect that DVS can be done within a task or for single-application devices. For, instance, in MPEG decoding, the variance in execution time on frame basis can be very large: a factor of three in the number of cycles [23], or range between 1 and 2000 IDCTs per frame [24] for MPEG video.

This chapter introduces a new model that combines dynamic power management and dynamic voltage scaling. The new approach is based on Time-Indexed Semi-Markov Decision Process model (TISMDP). The power management policy optimisation problem is solved *exactly and in polynomial time with guaranteed optimal results*. Large savings are measured with TISMDP model as it handles general user request interarrival distributions and make decisions in the event-driven manner. The dynamic voltage scaling enables even larger savings than are possible with only DPM. DVS represents the active state as a series of states characterized by varying degrees of performance and energy consumption. The system's processor runs at the minimum frequency and voltage required to sustain the performance level required by an application and thus saves power when the system is active, in addition to the savings obtained by DPM during idle periods. Combined savings of DPM and DVS on the SmartBadge are as high as a factor of 3.

The remainder of the chapter is organized as follows. Section 1 describes the stochastic models of the system components based on the experimental data collected. Time-Indexed Semi-Markov Decision Process model for the dynamic power management policy optimisation problem is discussed in Section 2, followed by the expansion of TISMDP model to support DVS in Section 3. Measured results for power managing two hard disks and a WLAN card

are discussed in Section 4, in addition to the DPM and DVS simulation results for the SmartBadge. Finally, this chapter is summarized in Section 5.

1. System Model

The system model consists of three components: the user, the device and the queue as shown in Figure 1.1. Each system component can be described probabilistically. The user, or the application that accesses each device by sending requests via operating system, is modelled by a request interarrival time distribution. When one or more requests arrive, the user is said to be in the active state, otherwise it is in the idle state. Figure 1.1 shows three different power states for the device: active, idle and sleep. Service time distribution describes the behaviour of the device in the active state. When the device is in either the idle or the sleep state, it does not service any requests. Typically, the transition to the active state is shorter from the idle state, but the sleep state has a lower power consumption. The transition distribution models the time taken by the device to transition between its power states. The queue models a buffer associated with each device. The combination of interarrival time distribution (incoming jobs to the queue) and service time distribution (jobs leaving the queue) appropriately characterizes the behaviour of the queue. Each system component models are described next.

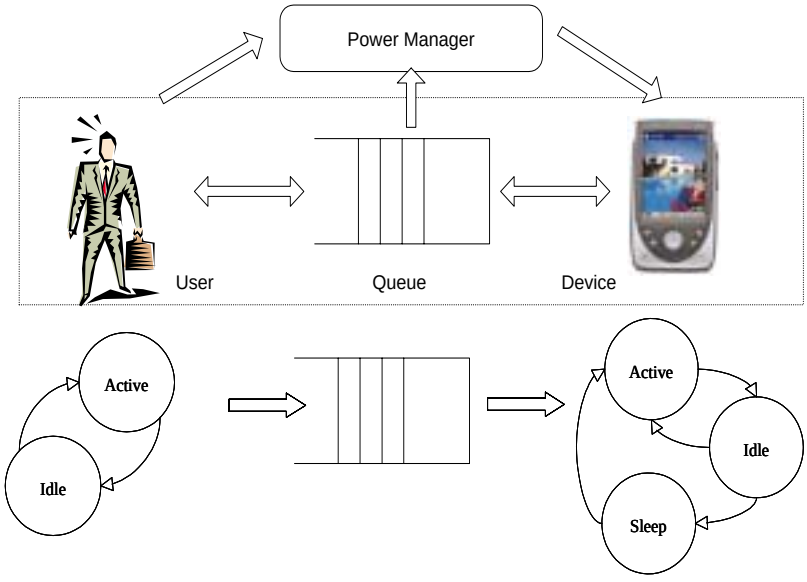

Figure 1.1. System Model

1.1 User Model

The request interarrival times in the active state (the state where at least one user's request is queued up) for all devices are exponentially distributed. Figure 1.2 shows the exponential cumulative distribution fitted to the measurements of 11hr user trace accessing the hard disk of the PC running Windows OS with standard software (e.g Excel, Word, MS VC++). Similar results have been observed for the other devices in the active state [15]. Thus, the user in active state can be modelled with rate λ_U and the mean request interarrival time, $1/\lambda_U$, where the probability of a device receiving a user request within time interval t follows the cumulative probability distribution shown below.

$$F_U(t) = 1 - e^{-\lambda_U t} \tag{1.1}$$

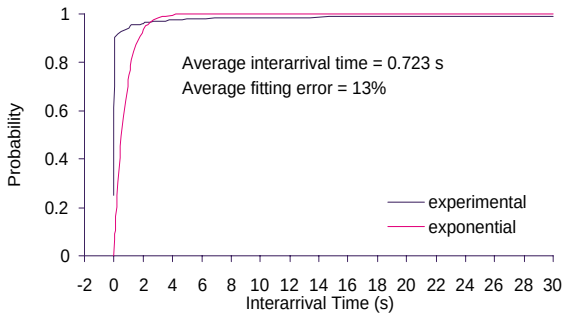

Figure 1.2. User request arrivals in active state for hard disk

The exponential distribution does not model well the first request arrival in the idle state. The tail distribution highlights the probability of longer idle times that are of interest for power management. Figure 1.3 shows the measurements of idle period tail distributions fitted with the Pareto and the exponential distributions for the hard disk and Figure 1.4 shows the same measurements for the WLAN card. The Pareto distribution shows a much better fit for the long idle times as compared to the exponential distribution. The Pareto cumulative distribution is defined in Equation 1.2. The Pareto parameters are a=0.9 and b=0.65 for the hard disk, a=0.7 and b=0.02 for WLAN web requests and a=0.7 and b=0.06 for WLAN telnet requests. SmartBadge arrivals behave the same way as the WLAN arrivals.

$$F_U(t) = 1 - at^{-b} \tag{1.2}$$

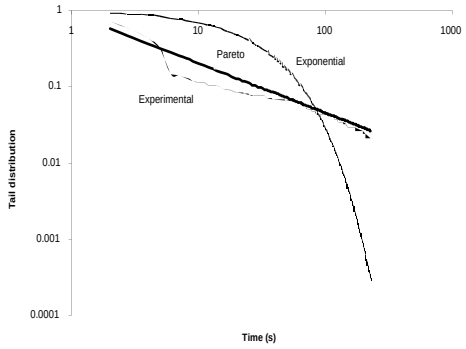

Figure 1.3. Hard disk idle state arrival tail distribution

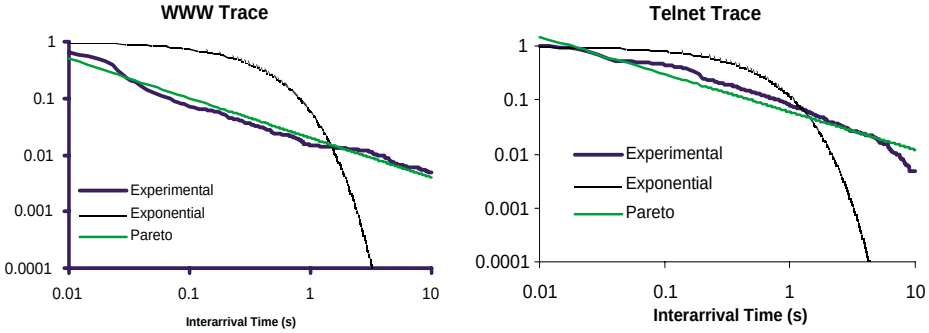

Figure 1.4. WLAN idle state arrival tail distribution

1.2 Portable Devices

A power managed device often has one or more active states in which it services user requests, and one or more low-power states. The power manager can trade off power for performance by placing the device into low-power states or by adjusting the level of service in the active states. Each low power state can be characterized by the power consumption and the performance penalty incurred during the transition to or from that state. Usually higher performance penalty corresponds to lower power states. While the methods presented in this work are general, the optimization of energy consumption under performance constraints (or vice versa) is applied to and measured on the following devices: WLAN card [25] on the laptop, the SmartBadge [1] and laptop and desktop hard disks. More details about the device models are presented below.

1.2.1 SmartBadge. The SmartBadge is used as a *personal digital assistants* (PDAs) that operates as a part of a client-server system. The SmartBadge's components, the power states and the transition times of each component from standby (t_{sby}) and off (t_{off}) state into active state, and the transition times between standby and off states (t_{so}) are shown in Table 1.1.

Table 1.1. SmartBadge components

Component	Active Pwr (mW)	Idle Pwr (mW)	Standby Pwr (mW)	t_{sby} (ms)	t_{off} (ms)	t_{so} (ms)
Display	1000	1000	100	100	240	110
RF Link	1500	1000	100	40	80	20
SA-1100	400	170	0.1	10	35	10
FLASH	75	5	0.023	0.6	160	150
SRAM	115	17	0.13	5.0	100	90
DRAM	400	10	0.4	4.0	90	75
Total	3.5 W	2.2 W	200 mW	110 ms	250 ms	160 ms

The StrongARM processor on the SmartBadge can be configured at run-time by a simple write to a hardware register to execute at one of 11 different frequencies. The number of frequencies is predefined by the design of the StrongARM processor. The transition between two different frequency settings takes 150 microseconds. For each frequency, there is a minimum voltage the SA-1100 needs in order to run correctly, but with lower energy consumption. Figure 1.5 shows the frequency-voltage tradeoff.

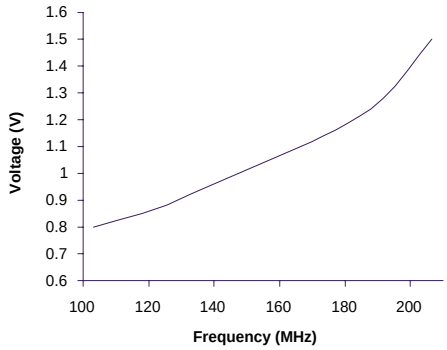

Figure 1.5. Frequency vs. Voltage for SA-1100

1.2.2 Hard Disks. Two different hard disks whose characteristics are shown in Table 1.2 have been used in experiments: the Fujitsu MHF 2043AT hard disk in a laptop and the IBM hard disk in a desktop. Although the power savings in the sleep state are large (P_{active} versus P_{sleep}), the disks should be placed into the sleep state only when the idle period between successive accesses is long enough to justify the performance (T_{active} and T_{sleep}) and power overhead incurred during the transition.

Table 1.2. Disk Parameters

Model	P_{sleep} (W)	P_{active} (W)	T_{sleep} (sec)	T_{active} (sec)
IBM	0.75	3.48	0.51	6.97
Fujitsu	0.13	0.95	0.67	1.61

Hard disk service times in the active state follow an exponential distribution as shown in Figure 1.6. The SmartBadge and the WLAN card measurements have similar patterns. Equation 1.3 defines the cumulative probability of the device servicing a user request within time interval t with the average service rate λ_D.

$$F_D(t) = 1 - e^{-\lambda_D t} \tag{1.3}$$

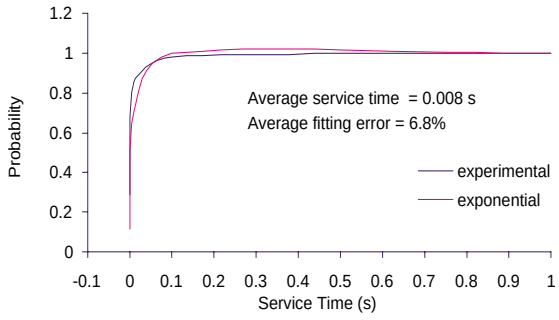

Figure 1.6. Hard disk service time distribution

The transition from sleep to active state requires a spin-up of the hard disk, which is very power intensive. This transition, in addition to the transition from idle to the sleep state, is best described using the uniform distribution shown in Equation below, where t_0 and t_1 can be defined as $t_{ave} - \Delta t$ and $t_{ave} + \Delta t$ respectively. Figure 1.7 shows the measurement results for the transition of

Fujitsu hard disk from sleep to active state. Similar results can be obtained for the other devices.

$$F_D(t) = \begin{cases} 0 & t \leq t_0 \\ \frac{t-t_0}{t_1-t_0} & t_0 < t \leq t_1 \\ 1 & t > t_1 \end{cases} \tag{1.4}$$

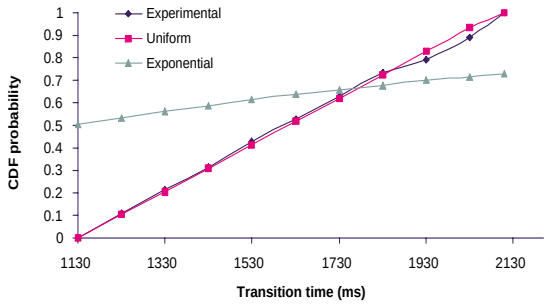

Figure 1.7. Hard disk transition from sleep to active state

1.2.3 WLAN card. The wireless local area network (WLAN) card has multiple power states: two active states, *transmitting*, *receiving*, and two inactive states, *doze* and *off*. Transmission power is 1.65W, receiving 1.4W, the power consumption in the doze state is 0.045W [25] and in the off state it is 0W. Once both receiving and transmission are done, the card automatically enters the doze state. Unfortunately, savings of only 5-10% in power have been measured with this approach, due to the overhead of having to be awake every 100ms to find out if any communication needs to take place. In client-server systems, such as the laptop used in this work, it is clear when communication is finished on the client side. Thus, the power manager can turn the card off once the communication is finished, and turn in back on when the client wishes to resume communication. Once in the off state, the card waits for the first user request arrival before returning back to the doze state. The transitions between the doze and the off states have been measured and are best described using the uniform distribution. The transition from the doze state into the off state takes on average $t_{ave} = 62ms$ with variance of $t_{var} = 31ms$. The transition back takes $t_{ave} = 34ms$ with $t_{var} = 21ms$ variance.

1.3 Queue

Portable devices have a buffer for storing requests that have not been serviced yet. This buffer is modelled as a queue. An experiment on a hard disk

using a typical user trace measured a maximum queue size of 10 jobs. Because the service rate in the SmartBadge and WLAN card is higher, and the request arrival rate is comparable, the same maximum queue size can be used. As there is no priority associated with requests coming into the queue, active and low-power states are differentiated only by the number of jobs pending for service.

1.4 Model Overview

Table 1.3 shows the probability distributions used to describe each system component derived from the experimental results. User request interarrival times with at least one job in the queue are best modelled with the exponential distribution. Pareto distribution best models the arrival of the first user's request when the queue is empty. The service times in the active state follow the exponential distribution. The transitions to and from the low power states are uniformly distributed. The combination of these distributions is used to derive the state of the queue. Although the experimental section of this Chapter utilizes the fact that the non-exponential user and device distributions can be described with well-known functions (Pareto or uniform), the models presented are general in nature and thus can give optimal results with both experimental distributions obtained at run time.

Table 1.3. System Model Overview

System Component	Component State	Distribution
User	Queue not empty	Exponential
	Queue empty	Pareto
Device	Active	Exponential
	Transition	Uniform

2. Dynamic Power Management

Dynamic power management (DPM) techniques selectively place system components into low-power states when they are idle. A power managed system can be modelled as a *power state machine*, where each state is characterized by the power consumption and the performance. In addition, state transitions have power and delay cost. Usually, lower power consumption also implies lower performance and longer transition delay. When a component is placed into a low-power state, such as a sleep state, it is unavailable for the time period spent there, in addition to the transition time between the states. The break-even time, T_{be}, is the minimum time a device should spend in the low-power state to compensate for the transition cost. The break-even time can

be calculated directly from the power state machine of the device as shown in Equation 1.5, where P_{as}, T_{as} and P_{sa}, T_{sa} are the power consumption and the time to transition between the active and the sleep states, and P_a, P_s are the power consumptions in the active and the sleep states. With devices where the transition cost into inactive state is minimal, the power management policy is trivial (once in the idle state, shut off). In all other situations it is critical to determine the most appropriate policy that the PM should implement.

$$T_{be} = \frac{P_{as}T_{as} + P_{sa}T_{sa} - P_a(T_{as} + T_{sa})}{P_a - P_s} \tag{1.5}$$

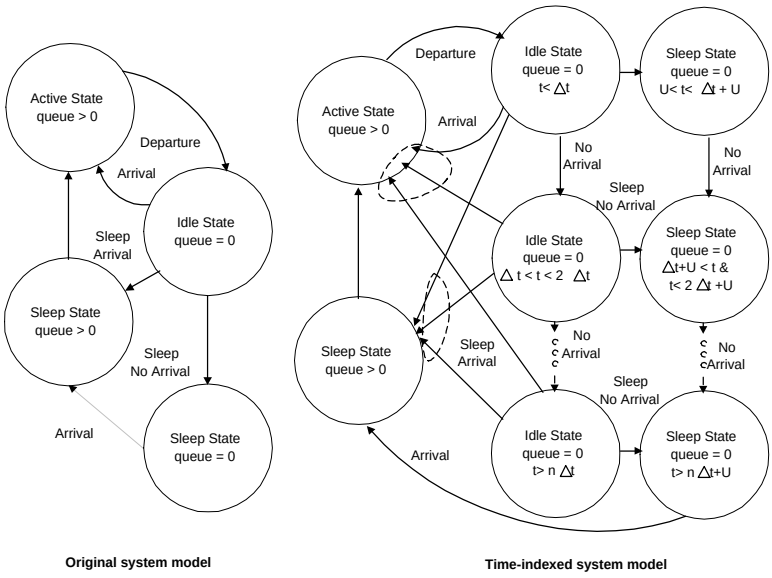

Figure 1.8. Time-indexed SMDP states

This section presents the power management optimisation problem formulation based on *Time-Indexed Semi-Markov Decision Processes* (TISMDP). TISMDP model is a generalization of *Semi-Markov decision processes* (SMDP) model. SMDP allows for at most one non-exponentially distributed transition at a time and enables the power manager to respond on event occurrences instead of being tied to a system clock as in traditional *Discrete-Time Markov Decision Processes*. *Continuous-time Markov decision processes* (CTMDP) [14] can be viewed as a special case of SMDP model in which all transitions are exponentially distributed. TISMDP model introduced in this work, in contrast to SMDP and CTMDP, uses general distributions to describe system transitions, is still event driven and guarantees optimal results. On transitions where none of the processes are exponential (e.g. transition from idle to sleep state, where

user's requests are governed with Pareto distribution and the transition time follows the uniform distribution), time-indexed Markov chain formulation is used to keep the history information [27].

Time-indexing is done by dividing the time line into a set of intervals of equal length Δt. The original state space is expanded by replacing one idle and one queue empty low-power state in SMDP model with a series of time-indexed idle and low-power empty states in TISMDP model as shown in Figure 1.8. The expansion of idle and low-power states into time-indexed states is done only to aid in deriving in the optimal policy. Once the policy is obtained, the actual implementation is completely event-driven in contrast to the policies based on discrete-time Markov decision processes.

The *interevent time set* is defined as $\mathcal{T} = \{t_i, \{s.t.\} i = 0, 1, 2, \ldots, i_{max}\}$ where each t_i is the time between the two successive event arrivals and i_{max} is the index of the maximum time horizon. We denote by $s_i \in \mathcal{S}_i$ the system state at decision epoch i. At each event occurrence, the power manager issues a *command* or an *action* that decides the next state to which the system should transition. An action that is issued at decision epoch i is denoted by $a_i \in \mathcal{A}$. In general, commands given are functions of the state history and the policy. Commands can be either deterministic or randomised. In the former case, a decision implies issuing a command, in the later case it gives the probability of issuing a command. The actions taken by the PM form a sequence which completely describes the optimal power management policy.

The goal of TISMDP optimisation is to minimize the performance penalty under an energy consumption constraint (or vice versa). The linear program minimizing energy consumption under performance constraint is shown in Equation 1.6. Additional constraints can be added easily. Because the problem can be formulated as a linear program, globally optimal solution can be obtained that is *stationary* (the functional dependency of PM actions on the states does not change with time) and *randomised* under the presence of constraints. The problem can be solved in polynomial time as a function of number of states and actions.

$$
\begin{aligned}
\textbf{LPD: } \min \quad & \sum_{s \in S} \sum_{a \in A} cost_{energy}(s,a) f(s,a) \qquad (1.6)\\
\text{s.t.} \quad & \sum_{a \in A} f(s,a) - \sum_{s' \in S} \sum_{a \in A} m(s'|s,a) f(s',a) = 0 \\
& \sum_{s \in S} \sum_{a \in A} y(s,a) f(s,a) = 1 \\
& \sum_{s \in S} \sum_{a \in A} cost_{perf}(s,a) f(s,a) < Constraint
\end{aligned}
$$

The linear program minimizes the cost in energy under a set of constraints. The first constraint is really a set of constraints for each system state. This constraint formulation is called a *balance equation*, because it specifies that the number of ways a system can transition into any given state has to equal the number of ways it transitions out of that state. The second constraint defines the expected time spent in each state when action a is given. The last constraint is the performance constraint. The next set of equations defines all variables in the optimisation problem shown above for both the states that do not need time indexing and for those that need it.

The average cost incurred between two successive decision epochs (events) for states that are not time indexed is defined in Equation 1.7 as a sum of the lump sum cost $k(s_i, a_i)$ incurred when action a_i is chosen in state s_i, in addition to the cost in state s_{i+1} incured at rate $c(s_{i+1}, s_i, a_i)$ after choosing action a_i in state s_i. We define $\mathcal{S}_{i+1}$ as the set of all possible states that may follow s_i. Equation 1.8 defines the same cost for the time-indexed states. $\mathcal{S}_{i+1}$ as the set of all possible states that may follow s_i. When action a_i is chosen in system state s_i, the probability that the next event will occur by time t_i is defined by the cumulative probability distribution $F(t_i|s_i, a_i)$. The probability that the system transitions to state s_{i+1} at or before the next decision epoch t_i is defined by $p(s_{i+1}|t_i, s_i, a_i)$.

$$cost(s_i, a_i) = k(s_i, a_i) + \int_0^\infty [E(du|s_i, a_i) \sum_{s_{i+1}} \int_0^u c(s_{i+1}, s_i, a_i) p(s_{i+1}|t_i, s_i, a_i)] dt \tag{1.7}$$

$$cost(s_i, a_i) = k(s_i, a_i) + \sum_{s_{i+1} \in S_{i+1}} c(s_{i+1}, s_i, a_i) y(s_i, a_i) \tag{1.8}$$

The probability of arriving to state s_{i+1} given that the action a_i was taken in state s_i is defined by $m(s_{i+1}|s_i, a_i)$ for states that do not need time indexing as shown in Equation 1.9.

$$m(s_{i+1}|s_i, a_i) = \int_0^\infty p(s_{i+1}|t_i, s_i, a_i) F(dt|s_i, a_i) \tag{1.9}$$

For time indexed state, the probability of transition to the next idle state is defined to be $m(s_{i+1}|s_i, a_i) = 1 - p(s_{i+1}|t_i, s_i, a_i)$ and of transition back into the active state is $m(s_{i+1}|s_i, a_i) = p(s_{i+1}|t_i, s_i, a_i)$. The general cumulative distribution of event occurrences is given by $F(t_i)$, while the probability of getting an arrival is defined using the time indices for the system state where $t_i \leq t \leq t_i + \Delta t$:

$$p(s_{i+1}|t_i, s_i, a_i) = \frac{F(t_i + \Delta t) - F(t_i)}{1 - F(t_i)} \tag{1.10}$$

Expected time spent in each state for the states that are not time indexed is given in Equation 1.11, while the expected time for the time indexed states is given in Equation 1.12:

$$y(s_i, a_i) = \int_0^\infty t \sum_{s_i \in S} p(s_{i+1}|t_i, s_i, a_i) F(dt|s_i, a_i) \tag{1.11}$$

$$y(s_i, a_i) = \int_{t_i}^{t_i+\Delta t} \frac{(1 - F(t))dt}{1 - F(t_i)} \tag{1.12}$$

The $A \cdot S$ unknowns in the LPD, $f(s, a)$, called *state-action frequencies*, are the expected number of times that the system is in state s and command a is issued. The exact and the optimal solution to the TISMDP policy optimization problem belongs to the set of *Markovian randomized stationary policies* [26] that can be compactly represented by associating a value $x(s, a) \leq 1$ with each state and action pair in the TISMDP, as defined in Equation 1.13.

$$x(s_i, a_i) = \frac{f(s_i, a_i)}{\sum_{a_i \in A} f(s_i, a_i)} \tag{1.13}$$

2.1 Policy Implementation

The optimal policy obtained by solving the linear program given in the previous Section can be presented as a table of cumulative probabilities $P(s, a)$ calculated based on the probability distribution described with $x(s, a)$. Once the system enters a decision state (e.g. idle state), a pseudo-random number RND is generated. The device stays in the decision state until either the transition to the low-power state as given by RND and the policy, or until a request arrival forces the transition into the active state. The time interval for which the policy gives the cumulative probability $P(s, a)$ of going to the low-power state greater than RND is the time when the device will start that transition. Thus the policy works like a randomised timeout. Once the device is in the low-power state, it stays there until the first request arrives, at which point it transitions back into the active state.

Example

The SmartBadge has two decision states: idle and standby. From the idle state, it is possible to transition to the standby or to the off state. From standby, only a transition to the off state is possible. The optimal policy (sample is shown in Table 1.4) gives a table of probabilities determining when the transition between the idle, the standby and the off states should occur. A sample policy may specify that if the system has been idle for 50ms, the transition to the standby state occurs with probability of 0.4, the transition to the off state with probability of 0.2 and otherwise the device stays idle. If the SmartBadge was placed into standby state at time 50ms, then the probability to transition into the off state at 100ms would be 0.8, and otherwise the device stays in the standby state. When a user request arrives, the SmartBadge transitions back into the active state.

Table 1.4. Sample Policy

Idle Time (ms)	Idle to Standby Probability	Idle to Off Probability	Standby to Off Probability
0	0	0	0
50	0.4	0.2	0
100	0.1	0.9	0.8

3. Dynamic Voltage Scaling

Dynamic Voltage Scaling (DVS) algorithms adjust the device speed and voltage according to the workload at run-time. Since most systems do not need peak performance at all times, decreasing the device speed and voltage during less busy periods increases energy efficiency. Implementing a DVS algorithm for a processor requires both hardware and software support that is not commonly available yet, even though there have been a few examples of DVS implementation such as in [28].

TISMDP policy presented in Section 2 only decides when to transition the device into one of the low-power states. The addition of DVS algorithm enables power manager to also make decisions on the CPU frequency and voltage setting while in the active state. Thus, instead of having only one active state, now there is a set of active states, each characterized by different performance (CPU frequency) and power consumption (CPU voltage) as shown in Figure 1.9. The transformation from one active into multiple active states is completely compatible with the rest of the original model, as both TISMDP and DVS use exponential distributions to describe the behaviour in the active states.

The newly extended power manager works as follows. At run-time, it observes user request arrivals and service completion times (e.g. frame arrivals and decoding times), the number of jobs in the queue (e.g. number of frames in the buffer) and the time elapsed since last entry into idle state. When in the active state, the power manager checks if the rate of incoming or decoding frames has changed, and then adjusts the CPU frequency and voltage accordingly. Once the decoding is completed, the system enters idle state. At this point the power manager observes the time spent in the idle state, and depending on the TISMDP policy, it decides when to transition into one of the sleep states. When an additional processing request arrives from the user, the power manager transitions the system back into the active state and starts the processing requests. The DVS algorithm consists of two main portions: detection of the change in request arrival or servicing rate, and the policy that adjusts the CPU frequency and voltage. The detection is done using maximum likelihood

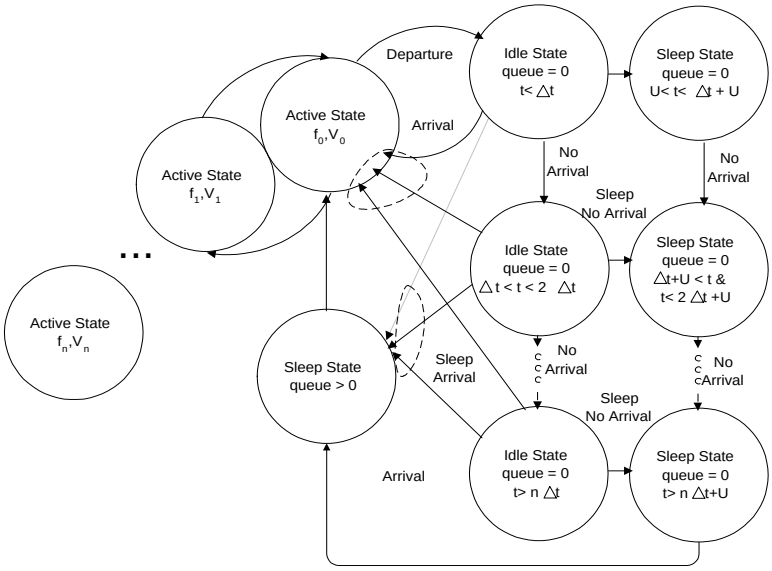

Figure 1.9. Expansion of the active state

ratio that guarantees optimal results. Policy is implemented based on M/M/1 queue results to ensure constant average delay. Detecting the change in rate is a critical part of optimally matching CPU frequency and voltage to the requirements of the user. For example, the rate of MP3 audio frames coming via RF link can change drastically due to the changes in the environment. The servicing rate can change due to variance in computation needed between MPEG frames [23,24], or just by changing the MP3 audio source. The request (frame) interarrival times and servicing (decoding) times follow the exponential distribution as discussed in Section 1. The two distributions are characterized by the user arrival rate, λ_U, and the device servicing rate, λ_D.

The change point detection is performed using maximum likelihood ratio, P_{max}, as shown in Equation 1.14. Maximum likelihood ratio computes the ratio between the probability that a change in rate did occur (numerator in Equation 1.14) and the probability that rate did not change (denominator). The probability that the rate changed is computed by fitting the exponential distribution with an old rate, λ_o, to the first $k-1$ interarrival or servicing times (t_i), and another exponential distribution with a new rate, λ_n, to the rest of the points observed in window of size w (which contains the last w interarrival times of user requests). The probability that the rate did not change is computed by fitting the interarrival or decoding times with the exponential dis-

tribution characterized by the current (or old) rate, λ_o.

$$P_{max} = \frac{\Pi_{i=1}^{k-1} \lambda_o e^{-\lambda_o t_i} \Pi_{i=k}^{w} \lambda_n e^{-\lambda_n t_i}}{\Pi_{i=1}^{w} \lambda_o e^{-\lambda_o t_i}} \tag{1.14}$$

An efficient way to compute the maximum likelihood ratio, P_{max}, is to calculate the natural log of P_{max} as shown below:

$$ln(P_{max}) = (w - k + 1) ln \frac{\lambda_n}{\lambda_o} - (\lambda_n - \lambda_o) \sum_{i=k}^{w} t_i \tag{1.15}$$

The advantage of using $ln(P_{max})$ is that only the sum of interarrival (or decoding) times needs to be updated upon every arrival (or service completion). A set of possible rates, Λ, where $\lambda_o, \lambda_n \in \Lambda$, is predefined, as well as the size of the window w. Variable k is used to locate the point in time when the rate has changed.

The change point detection algorithm consists of two major tasks: off-line characterization and on-line threshold detection. Off-line characterization is done using stochastic simulation of a set of possible rates to obtain the value of $ln(P_{max})$ that is sufficient to detect the change in rate. The results are accumulated in a histogram, and then the value of maximum likelihood ratio that gives very high probability that the rate has changed is chosen for every pair of rates under consideration. In this work 99.5% likelihood is selected.

On-line detection collects the interarrival time sums at run time and calculates the maximum likelihood ratio. If the maximum likelihood ratio computed is greater than the one obtained from the histogram, then there is 99.5% likelihood that the rate change occurred, and thus the CPU frequency and the voltage are adjusted. We found that a window of $w = 100$ is large enough. Larger windows cause longer execution times, while much shorter windows do not contain statistically large enough sample and thus give unstable results. The change point can be checked every $k = 10$ points. Larger values of k interval mean that the changed rate is detected later, while with very small values the detection is quicker, but also causes extra computation.

The adjustment of frequency and voltage is done using M/M/1 queue model [27,29]. The goal is to keep the average processing queue delay constant:

$$Delay = \frac{\lambda_D}{\lambda_U(\lambda_U - \lambda_D)} \tag{1.16}$$

When either interarrival rate, λ_U, or the servicing rate, λ_D, change, the delay is evaluated and the new frequency and voltage are selected that will keep the delay constant. For example, if the arrival rate for MP3 audio frames changes, the equation shown above is used to calculate the required decoding rate in order to keep the frame delay (and thus performance) constant.

If a different frame decoding rate is detected while processor is set to the same frequency, then piece-wise linear approximation based on the application frequency-performance tradeoff curve is used to obtain the new processor frequency setting. In either case, when CPU frequency is set to a new value, the CPU voltage is always adjusted according to Figure 1.5.

4. Results

4.1 Dynamic Power Management

Policy optimisation is performed with a linear program solver [30] in just under 1 minute on a 300MHz Pentium processor. Large savings are measured on three different devices: laptop and desktop hard disks and the WLAN card. All policies implemented are compared with two bounds: *always-on* and *oracle* policies. Always-on policy leaves a device in the idle state, and thus does not save any power. Oracle policy gives the lowest possible power consumption, as it transitions the device into sleep state with the perfect knowledge of the future. It is computed off-line using a previously collected trace. Obviously, the oracle policy is an abstraction that cannot be used in run-time DPM.

Hard disk measurements have the power manager as a part of a filter driver template attached to the vendor-specific device driver [31]. Application programs such as word processors or spreadsheets send requests to the OS. When any event occurs that concerns the hard disk, power manager is notified. When the PM issues a command, the filter driver creates a power transition call and sends it to the device which implements the power transition using Advanced Configuration and Power Interface standard [7]. The change in power state is also detected with the digital multimeter that measures current consumption of the hard disk.

Table 1.5. Hard Disk Measurement Comparison

	Laptop				Desktop			
Algorithm	Pwr (W)	N_{sd}	N_{wd}	$T_{ss}(s)$	Pwr (W)	N_{sd}	N_{wd}	$T_{ss}(s)$
Oracle	0.33	250	0	118	1.64	164	0	166
TISMDP	0.40	326	76	81	1.92	156	25	147
Adaptive	0.43	191	28	127	1.97	168	26	134
Karlin's	0.44	323	64	79	1.94	160	15	142
30s timeout	0.51	147	18	142	2.05	147	18	142
DTMDP	0.62	173	54	102	2.60	105	39	130
120s timeout	0.67	55	3	238	2.52	55	3	238
Always on	0.95	0	0	0	3.48	0	0	0

Comparison of power and performance penalty for all policies measured on the laptop and the desktop is shown in Table 1.5. Performance of the policies is compared using three different measures. N_{sd} is defined as the number of

times the policy issued sleep command. N_{wd} gives the number of times the sleep command was issued and the hard disk was asleep for shorter than the time needed to recover the cost of spinning down and spinning up the disk. Clearly, it is important to minimize N_{wd} while maximizing N_{sd}. The average length of time spent in the sleep state (T_{ss}) should be as large as possible while still keeping the power consumption down. From our experience with the user interaction with the hard disk, our algorithm performs well, thus giving us low-power consumption with still good performance. In comparison, Karlin's policy consumes 10% more power and has worse performance. Karlin's algorithm [8] guarantees to yield a policy that consumes at worst twice the minimum amount of power consumed by the policy computed with perfect knowledge of the user behaviour. In addition, our policy consumes 1.7 times less power than the default Windows timeout policy of 120s and 1.4 times less power than the 30s timeout policy on the laptop. TISMDP policy performs better than the adaptive model [13], and significantly better than the policy based on discrete-time Markov decision processes (DTMDP). The event-driven nature of TISMDP algorithm, as compared to algorithms based on discrete time intervals, saves considerable amount of power while in sleep state as it does not require policy evaluation until an event occurs. Similar results can be observed for the desktops.

Table 1.6. WLAN Measurement Comparison

	WWW				Telnet			
Algorithm	N_{sd}	N_{wd}	$T_p(s)$	$P_{ave}(W)$	N_{sd}	N_{wd}	$T_p(s)$	$P_{ave}(W)$
Oracle	395	0	0	0.467	766	0	0	0.220
TISMDP(a)	363	96	6.90	0.474	798	21	2.75	0.269
TISMDP(b)	267	14	1.43	0.477	782	33	2.91	0.296
Karlin's	623	296	23.8	0.479	780	40	3.81	0.302
TISMDP(c)	219	9	0.80	0.485	778	38	3.80	0.310
CTMDP	3424	2866	253.7	0.539	943	233	20.53	0.361
Default	0	0	0	1.410	0	0	0	1.410

In addition to the hard disks, the measurements have been performed also on Lucent's WLAN 2Mb/s card [25] connected to a Linux laptop. When comparing different policies, a LAN-attached host reads the 2.5 hr WWW and 2hr telnet traces collected by tcpdump [32] utility and delays or drops packets accordingly. Three different versions of TISMDP algorithm (labelled TISMDP a,b,c) with different power and performance penalty are implemented for each application. Since web and telnet arrivals behave differently (see Figure 1.4), the OS observes what application is currently actively communicating and informs the power manager. Performance penalty is determined with three different measures. Delay penalty, T_p, is the time the system had to wait to service a request since the card was in the sleep state when it should not have been.

The total number of shutdowns, N_{sd} and the number of shutdowns where sleep time is too short to make up for the total cost of transition, N_{wd} are measured as well.

The power management policy results presented in Table 1.6, TISMDP a,b,c, show, on average, a factor of three in power savings with a low performance penalty for the WWW application. Karlin's algorithm [8] has low power consumption, but its performance penalty is an order of magnitude larger than for TISMDP policy. A policy that models all stochastic processes with the exponential arrivals only, CTMDP, has a larger performance penalty because its decision is based only on the current system state and not on the previous history. TISMDP policy applied to Telnet trace shows a factor of five in power savings. Telnet allows larger power savings because on average it transmits and receives much less data then the WWW browser, thus giving more chances to shut down the card.

4.2 Dynamic Voltage Scaling

DVS algorithm is implemented as a part of a power manager on the SmartBadge for two different applications: MPEG video decoder and MP3 audio decoder. During the times that the system is idle, the TISMDP power management policy described in this Chapter decides when to transition the SmartBadge into a sleep state. When it is in the active state (the state where audio and video decoding occur), the power manager (PM) observes changes in the frame arrival and decoding rates using change point detection algorithm described previously. Once a change is detected, the PM evaluates the required value of the processor frequency that keeps the frame delay constant. The CPU voltage is set using results shown in Figure 1.5.

Rate change detection algorithm is compared to the ideal detection and to the exponential moving average algorithm. Ideal detection assumes knowledge of the future; thus the system detects the change in rate exactly when the change occurs. The exponential moving average can be defined as follows:

$$\lambda_n = (1 - g)\lambda_o + g\lambda_{cur} \tag{1.17}$$

where λ_n is the new average rate, λ_o is the old average, λ_{cur} is the current measured rate and g is the gain. Figure 1.10 shows the comparison results for detecting a change from 10 fr/sec to 60 fr/sec. Change point detection algorithm presented in this Chapter detects the correct rate within 10 frames and is more stable than either of the two the exponential moving average algorithms (with different gain values).

Three different audio clips totaling 653 seconds of audio, each running at a different set of bit (16,32,64 Kb/s) and sample rates (16 or 32 KHz) have been used to test the DVS algorithm. The ideal detection algorithm, the exponential average approximation used in previous work and the maximum proces-

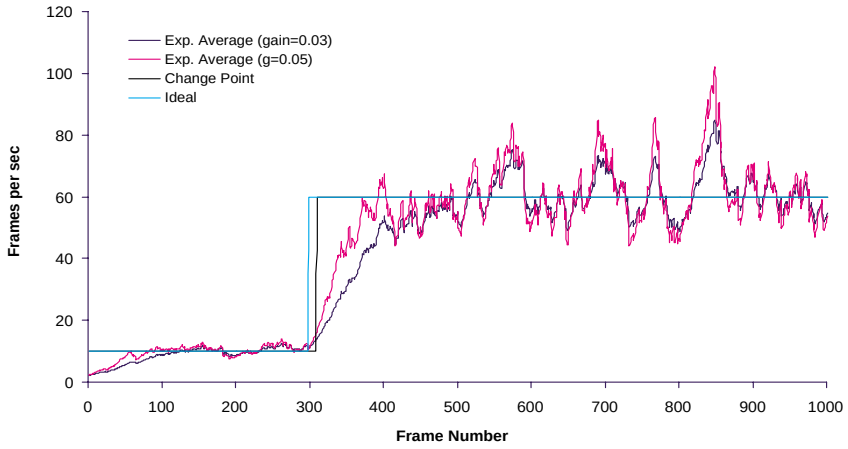

Figure 1.10. Rate Change Detection Algorithms

sor performance are compared to the change point algorithm presented in this Chapter. During decoding, the DVS algorithm detects changes in both arrival and decoding rates for the MP3 audio sequences. The resulting energy (kJ) and average total frame delay (s) are displayed in Table 1.7. The change point algorithm presented in this work performs well, its results are very close to the ideal, with no performance loss as compared to the ideal detection algorithm that allows an average 0.1 s total frame delay (corresponding to 6 extra frames of audio in the buffer).

Table 1.7. MP3 audio DVS

MP3 Audio Sequence	Result	Ideal	Change Point	Exp. Ave.	Max
Clip 1	Energy	196	217	225	316
	Fr.Delay	0.1	0.09	0.1	0
Clip 2	Energy	189	199	231	316
	Fr.Delay	0.1	0.09	0.1	0
Clip 3	Energy	190	214	232	316
	Fr.Delay	0.1	0.04	0.1	0

The next set of results are for decoding two different video clips. Results are reported for the ideal detection, the exponential average, the maximum processor performance and change point algorithm presented in this work. The ideal detection algorithm allows for 0.1s average total frame delay equivalent to 2 extra frames of video in the buffer. The arrival rate varies between 9 and

32 frames/second. Energy (kJ) and average total frame delay (s) are shown in Table 1.8. The results are similar to MP3 audio. The exponential average shows poor performance and higher energy consumption due to its instability (see Figure 1.10). The change point algorithm performs well, with significant savings in energy and a very small performance penalty (0.11s frame delay instead of allowed 0.1s).

Table 1.8. MPEG video DVS

MPEG Video Clip	Result	Ideal	Change Point	Exp. Ave.	Max
Football	Energy	214	218	300	426
(875s)	Fr.Delay	0.1	0.11	0.16	0
Terminator2	Energy	280	294	385	570
(1200s)	Fr.Delay	0.1	0.11	0.16	0

Finally, the dynamic voltage scaling is combined with power management algorithm. This experiment uses a sequence of audio and video clips, separated by some idle time. During longer idle times, the power manager has the opportunity to place the SmartBadge in the sleep state. Table 1.9 shows the energy savings with only dynamic voltage scaling, only power management and finally also for the combination of the two approaches. Combined savings are as high as a factor of three.

Table 1.9. DPM and DVS

Algorithm	Energy (kJ)	Factor
None	4260	1.0
DVS	3142	1.4
DPM	2460	1.7
Both	1342	3.1

5. Summary

As most systems do not need peak performance at all times, it is possible to transition system components into low-power states when they are idle (dynamic power management) and to adjust frequency and voltage of operation to the workload (dynamic voltage scaling). This chapter presents an event-driven power management algorithm based on Time-Indexed Semi-Markov Decision Process (SMDP) model that guarantees globally optimal results for systems modeled with general distributions. Large power savings, ranging from a fac-

tor of 1.7 to a factor of 5 have been observed when using TISMDP policy on four different portable devices: the laptop and the desktop hard disks, the WLAN card and the SmartBadge portable device.

TISMDP model has been extended to enable dynamic voltage scaling. The dynamic voltage scaling algorithm consists of two tasks: (i)change point detection to recognize the change in arrival or decoding rates, and (ii) the frequency setting policy that sets the processor frequency and voltage based on the current arrival and decoding rates in order to keep constant performance. The new DVS algorithm gives a factor of 1.4 to 2.2 savings in energy at a small performance penalty for MP3 audio and MPEG video applications running on the SmartBadge. DPM and DVS algorithms implemented together on the SmartBadge have a factor of 3 energy savings.

Acknowledgments

The author wishes to thank Dr. Giovanni De Micheli and Dr. Luca Benini for their input into this work. In addition, this work would not have been possible without the help and support of colleagues at HP Labs and Stanford University.

References

[1] G. Q. Maguire, M. Smith and H. W. Peter Beadle "SmartBadges: a wearable computer and communication system", *6th International Workshop on Hardware/Software Codesign*, 1998.

[2] A. Chandrakasan, R. Brodersen, *Low power digital CMOS design*, Kluwer, 1995.

[3] J. Rabaey, M. Pedram (Editors), *Low power design methodologies*, Kluwer, 1996.

[4] W. Nabel, J. Mermet (Editors), *Lower power design in deep submicron electronics*, Kluwer, 1997.

[5] C. Ellis, "The case for higher-level power management", *7th IEEE Workshop on Hot Topics in Operating Systems*, pp.162–167, 1999.

[6] L. Benini and G. De Micheli, *Dynamic Power Management: design techniques and CAD tools*, Kluwer, 1997.

[7] Intel, Microsoft and Toshiba, "Advanced Configuration and Power Interface specification", 1996.

[8] A. Karlin, M. Manesse, L. McGeoch and S. Owicki, "Competitive Randomized Algorithms for Nonuniform Problems", *Algorithmica*, pp. 542–571, 1994.

[9] D. Ramanathan, R. Gupta, "System Level Online Power Management Algorithms", *Design, Automation and Test in Europe*, pp. 606–611, 2000.

[10] 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, pp. 42–55, March 1996.

[11] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", in *International Conference on Computer Aided Design*, pp. 28–32, 1997.

[12] L. Benini, G. Paleologo, A. Bogliolo and G. De Micheli, "Policy Optimization for Dynamic Power Management", in *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 6, pp. 813–833, June 1999.

[13] E. Chung, L. Benini and G. De Micheli, "Dynamic Power Management for non-stationary service requests", *Design, Automation and Test in Europe*, pp. 77–81, 1999.

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

[15] T. Simunic, L. Benini, P. Glynn, G. De Micheli, ŞEvent-driven Power Management‚Ť *IEEE Transactions on CAD*, pp.840–857, July 2001.

[16] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M. Srivastava, "Power optimization of variable voltage-core based systems," *Proceedings of Design Automation Conference*, pp.176–181, 1998.

[17] I. Hong, M. Potkonjak, M. Srivastava, "On-line Scheduling of Hard Real-time Tasks on Variable Voltage Processor," *Proceedings of International Conference on Computer-Aided Design*, Nov. 1998.

[18] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for dynamically variable voltage processors," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.197–202, 1998.

[19] F. Yao, A. Demers, S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Annual foundations of computer sciend*, pp.374–382, 1995.

[20] Y. Shin, K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proceedings of Design Automation Conference*, pp.134–139, 1999.

[21] S. Lee, T. Sakurai, "Run-time voltage hopping for low-power real-time systems," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.806–809, 2000.

[22] T. Pering, T. Burd, R. Brodersen, "The simulation and evaluation of Dynamic Voltage Scaling Algorithms" *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, 1998.

[23] A. Bavier, A. Montz, L. Peterson, "Predicting MPEG Execution Times," *Proceedings of SIGMETRICS*, pp.131–140, 1998.

[24] A. Chandrakasan, V. Gutnik, T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing," *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, pp.347–352, 1996.

[25] Lucent, *IEEE 802.11 WaveLAN PC Card - User's Guide*, p.A-1.

[26] M. Puterman, *Finite Markov Decision Processes*, John Wiley and Sons, 1994.

[27] H. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, Academic Press, 1998.

[28] L. Geppert, T. Perry, "Transmeta's magic show," *IEEE Spectrum*, vol. 37, pp.26–33, May 2000.

[29] S. Ross, *Stochastic Processes*, Wiley Press, 1996.

[30] S. Skiena, *The Algorithm Design Manual* , 1997.

[31] Y. Lu, T. Šimunić and G. De Micheli, "Software Controlled Power Management", *7th International Workshop on Hardware/Software Codesign*, pp. 157–161, 1999.

[32] V. Jacobson, C. Leres, S. McCanne, *The "tcpdump" Mannual Page*, Lawrence Berkeley Laboratory.