International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 1, pages 137–152, January 2013

Injection Based Dynamic Power Management and a Policy for Multiprocessor Systems

Wei Sun
Cloud Systems Research Labs, NEC Corporation
1753, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa, 211-8666 Japan

Received: January 24, 2012
Revised: May 21, 2012
Revised: June 20, 2012
Accepted: July 19, 2012
Communicated by Sayaka Kamei

Abstract

Power consumption has become a critical issue in designing computer systems. Dynamic power management is an approach that aims to reduce power consumption at system level by selectively placing components into low power states. Time-out and prediction based policies are often adopted in practical systems. However, they have to accurately determine the time in low power state and otherwise the saved power consumption is not worth the loss of performance. In this paper, a power management for multiprocessor systems is proposed to optimally reduce the power consumption of multiple processors. The key feature of the proposed power management is that how long to place a processor into low power state is determined in advance but not decided when a processor becomes idle. Thus, many off-time quanta are pre-determined beforehand. The proposed power management schedules the off-time quanta to processors and a processor is placed into low power state if an off-time quantum is assigned to it. It seems that processors execute special tasks which just reduce the power supplied to them. Hence, the off-time quanta are also named sleep tasks, which are virtual and injected into the original task traffic. By doing so, the inaccurate time length of sleep tasks hardly impacts on the performance, because if a processor is blocked by a sleep task there is another one available except that all the others are blocked at the same time. Then a probabilistic policy is also proposed to optimally assign sleep tasks from the waiting queue to the processors for minimum loss of performance. In the proposed policy, high priority is given to real tasks and sleep tasks are serviced only on necessity. The analysis of the probabilistic policy is performed on a queueing model and shows that the policy is asymptotically optimal. The proposed power management and policy are further examined in empirical studies.

Keywords: Power management, Injection, Policy, Multiprocessor systems

1 Introduction

Energy and power have become key design considerations across a spectrum of computing solutions, from supercomputers and data centers to hand-held phones and other mobile computers [2, 3], since the increasing power and energy consumption is indeed a serious threat to not only the devices which we are using but also the places where we are living. Dynamic power management (DPM),

[0]This paper is the extension of [1].

one of the main approaches of power saving, aims at reducing the power consumption at system level by selectively placing components into low-power state [4]. Although low-power state costs minimum energy, the transitions between states require extra time and energy. Thus, sophisticated power management policies at system level are necessary to well make use of DPM and otherwise both performance and efficiency will deteriorate.

DVS (Dynamic Voltage Scaling) is also one of the most widely adopted techniques and has been fully supported by a growing number of processors from main vendors such as Intel and AMD. DVS allows software to adjust clock frequency and supplied voltage in tandem. In this paper, we only focus on DPM, with which, although, DVS can cooperate to control the power consumption [5].

Policies of power management have been studied prosperously in the recent decades, such as time-out policies, predictive policies and stochastic policies. Most policies must decide how long the time in low power state should last when or before a processor is placed into low power state and some techniques are used to make the estimation more accurate. In this paper, we propose a new technique which decides low power time slots beforehand. The low power time slots are called sleep tasks. From a long term viewpoint, sleep tasks look like being injected into real task traffic. Because the time slots are decided beforehand, the service of real tasks may be impacted. Then, a probabilistic policy is introduced to decide where and when a sleep task should be injected.

Saving energy consumption should be performed in long terms, and consequently the techniques do not make sense if they are only effective for one hour or one day workloads (note that it is reasonable to control power consumption in short time periods to avoid high system temperature or inefficient cooling in practice). Over a long term, it is reasonable to assume that the system is stable, i.e. finite queue length, and the arrival rate of real tasks is constant.

Through analytical studies the parameters of the policy are derived and with these parameters the policy is proved to be optimal in terms of the response time of real tasks. Through empirical studies, our policy is evaluated and also compared to another optimal policy proposed in the recent.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the system model and the problem of transferring to low power state. Section 4 introduces the idea of sleep tasks and the probabilistic policy. The analytical studies are performed in Section 5. In Section 6, empirical studies are also performed through simulations. Finally we conclude our work in Section 7.

2 Related Work

The most common power management policy at system level is the timeout policy [6, 7] implemented in many operating systems. The drawback of this policy is that energy is wasted while waiting of timeout events. Predictive policies[8, 9] developed for interactive terminals force the transition to low power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. Obviously incorrect estimates can cause both performance and energy penalties. The policies based on stochastic models can guarantee optimal results. Stochastic models use distributions to describe system behaviors[5, 10, 11]. In other words, the accuracy and the optimality heavily depend on whether system behaviors meet distributions.

In managing a set of servers, Pinheiro et al. estimated the number of servers to keep powered-on and proposed an algorithm based on a PID feedback controller to reconfigure server clusters [12]. The system administrator tunes the system by setting an elapsed time (the time to wait between reconfigurations to let the system settle) and a degradation percent. Chase et al. estimate the impact of cluster resource availability on workload throughput using economic theory [13]. They suggest that a bound on cluster energy savings can be estimated by the variability in the workload. Elnozahy et al. [14] have studied the relationship between policies that use dynamic voltage scaling and those that turn servers completely off. They found that a policy that both resized the cluster and dynamically varied the voltage (and frequency) of the servers achieved the best energy savings. However, a simple policy that turns off servers when not required is found quite competitive with the more complex policies.

Various On-Off based policies have been applied in server farm management, where setup costs are considerable in comparison with switching on/off processors. Gandhi et al. studied these policies in [17, 18]. It has been shown that the power consumption may not be reduced under certain loads because of setup costs [17]. Furthermore, in [18] an asymptotically optimal policy is proposed and named to be DELAYEDOFF. Under this policy, a server waits for some predetermined amount of time before being turned off. The predetermined time length is constant and decided in terms of setup time, active power and idle power.

The policy proposed in this paper aims at the efficiency over long terms other than short time periods, since saving energy is necessary for the whole life of a system, although in some cases power controlling within a short time is required. Thus, compared to the other policies ours is much simpler and only needs to know the statistics of systems, but the others have to decide the details when to switch on or off a sever or a processor. In our policy, given a system, only the strength of injection needs to be considered beforehand. The runtime behaviors are in the charge of the queues which store the sleep tasks.

3 System Model and Problem

The system considered in this paper consists of m identical processors, each of which works on at least a high power state and a low power state. In order to save power consumption, a processor will be switched into the low power state when it becomes idle. The processors share a global queue, at which tasks arrive.

The state transfer needs extra time and energy. Thus, when a processor is switched into the low power state, the processor has to keep the state for a minimum time period. Otherwise, switching to the low power state will result in both more energy consumption and worse performance. Moreover, the total time in low power states is hoped to be longer since more power consumption can be saved. Consequently, two problems arise: the tasks may be delayed greatly and the system may not be stable, if the state transfer is not controlled well. However, it is not trivial to decide when a processor should be switched to the low power state and how long a processor should be in the state.

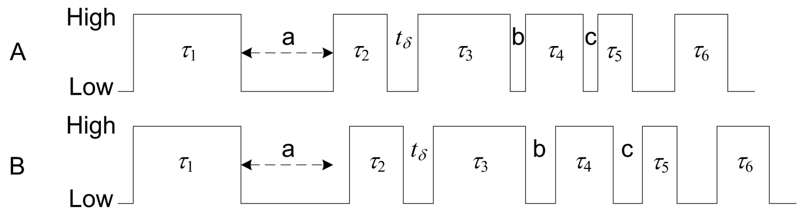

Figure 1: Instances of state transfers

Some examples are provided to facilitate the understanding of the problems in Fig.1, timing charts in which there are 6 tasks $\tau_1, \cdots, \tau_6$ arriving at a processor one by one, and t_δ denotes the shortest time of transferring from the high power state to the low power state and then to the high power state. An ideal case is shown in A. There is no extra costs due to state transfers and the time interval between τ_2 and τ_3 happens to be t_δ seamlessly. Therefore the time intervals of b and c are possible to be shorter than t_δ. The power management may adopt an event-driven policy, in which the events of task arrivals and departures will trigger the state transfers. Time-out is also a kind of event-driven policy and is possible to work well if the time-out can be set accurately, if time can be estimated well. If only an event-driven policy is applied, τ_2, for example in B, may be delayed at least $t_\delta/2$ even if a has been longer than t_δ. As a result, both b and c are equal to t_δ. The situation becomes worse if the estimation is not accurate and in fact no accuracy is guaranteed very well. Moreover, the consequence of the delays is possibly fatal in highly loaded systems, since the highly loaded systems cannot be stable any longer as designed without power management.

The system can be represented by a queueing model. Let λ denote the arrival rate of tasks, which is the the reverse of the mean inter-arrival time or the number of task arrivals per unit time.

Let μ be the service rate of single processors, which is the reverse of mean task running time. The utilization is $\rho = \frac{\lambda}{m\mu}$, which represents the fraction of the time that the system is running actively. Since we consider long term effect, ρ and λ are constant. Note that, both of them are the average values and hence the instant values are not constant, i.e. the instant system load and number of arrivals varying. Moreover, given any two of ρ, λ and μ, another one can be computed. Naturally $1 - \rho$ represents the fraction of the time that the system is idle. Here the idle time of the system refers to the sum of the idle time slots in all processors. If each processor can be appropriately switched to the low power state within the idle time slots, the response times of tasks will not be impacted and the power consumption can be saved with no expense. Unfortunately, the idle time slots cannot be known beforehand and the slots may be too short for the low power state. Some techniques predict the length of the idle time when a processor becomes idle and some of the others estimate the appropriate time length that a processor is in the low power state. As shown in the above example, with an accurate predictor it is still a problem of how to save power consumption in idle time slots without great delays of tasks when ρ is relatively high.

4 Injection Based Management and Probabilistic Policy

The basic idea of this paper is very different from the existing techniques in that we deal with the slots of idle time as sleep tasks. Sleep tasks have the minimum time length longer than the shortest time required by the low power state. When a sleep task starts to run in a processor, the processor will be switched into the low power state at once. The utilization of sleep tasks is ρ'. The sleep tasks are generated and injected into the original task traffic in the rate I. The system should be still stable after sleep tasks are injected. In other words, $\rho + \rho' < 1$. If all idle time slots are in the low power state, $\rho + \rho'$ will be close to 1 infinitely. Thus, $1 - \rho$ is the percentage of saved power consumption to total power consumption without power management. By assuming that the mean time length of sleep tasks equals that of real tasks, the percentage of saved power consumption can be written as $\omega = \frac{I}{I+\lambda}$ which has the upper bound of $1 - \rho$.

Injecting pre-determined sleep tasks is actually not feasible in a single processor, because a new real task has to wait if a sleep task is blocking the processor or the energy is wasted if a sleep task ends early. However, in multiprocessor systems when a processor is blocked by a sleep task newly arriving real tasks are possible to be assigned to the other active processors. Moreover, a sleep task only switches off a processor for a limited time period. Consequently an elegant policy must be able to well harmonize sleep tasks and real tasks. It is not difficult to generate and inject sleep tasks and keep a stable system. The question is how the tasks are serviced to minimize the average response time of real tasks. That is to minimize the impact of sleep tasks on real tasks. To accomplish that goal, a probabilistic policy is shown as follows.

- Give high priority to real tasks and low priority to sleep tasks.
- Inject sleep tasks to an idle processor with probability $1 - p$ if and only if there are $k - 1$ busy processors.
- Inject sleep tasks to an idle processor with probability p if and only if there are k busy processors.

Here, a processor is busy if either a real task or a sleep task is in service. Servicing a sleep task means that a processor is in the low power state. In order to clearly show how the policy works, pseudo-code is provided as follows.

Obviously this policy is designed by considering performance a priori, but later we will prove that the idle time per unit time is constant for a given I. In terms of the average response time of real tasks, given proper p and k the policy is optimal. The optimality of the policy and the valuation of p and k are briefly analyzed in the following.

Pseudo-code 1: Generate sleep tasks - GT() and Inject sleep tasks - IT()

```
Input: Two queues $Q_k$ and $Q_{k-1}$ (Equivalent to use only one global queue with marking tasks).
Input: Computed $k$ and $p$
// GT() is called when a sleep task $\tau$ is about to be generated in terms of a certain
   distribution eg. Poisson. IT() is called when the number of busy processors is $k$
   or $k-1$. Thus, a system monitor is assumed.
GT( )
{
generate a new sleep task $\tau$ with given time length;
store $\tau$ into $Q_k$ with the probability $p$, otherwise store into $Q_{k-1}$;
}

IT( )
{
if $k$ busy processors then
    Take a task from $Q_k$ and assign this task to an idle processor;
else
    Take a task from $Q_{k-1}$ and assign this task to an idle processor;
end
}
```

5 Analytical Studies

The system with such a policy is represented by a queueing model, in which a state is the number of all real tasks in the whole system plus the number of sleep tasks only in processors. We only consider stationary system in which the queue is stable and the process is stationary and ergodic. The mean time length of sleep tasks is assumed to be the same as real tasks and then the service rate to all tasks is μ. The assumption of the mean time length is feasible because it is free completely to combine and split sleep tasks. In [15], an analysis technique is provided for multi-server with priority queues. In this paper, we modify and extend the similar technique.

We define the speed of injecting a sleep task to transfer the state i to $i+1$ to be I_i, which is, note that, the limit of the number of sleep tasks which are injected by time t and force the state from i to $i+1$. Assuming that the number of sleep tasks which have transferred the state from i to $i+1$ by time t is $N_i(t)$, then we know I_i should be

$$I_i = \lim_{t\to\infty} \frac{N_i(t)}{t}, i = 0, 1, \cdots, m-1. \tag{1}$$

For the stationary process, we have the balance equations.

$$\begin{aligned} I_0 + \lambda P_0 &= \mu P_1 \\ I_1 + \lambda P_1 &= 2\mu P_2 \\ I_2 + \lambda P_2 &= 3\mu P_3 \\ &\vdots \\ I_{m-1} + \lambda P_{m-1} &= m\mu P_m \\ \lambda P_m &= m\mu P_{m+1} \\ &\vdots \end{aligned}$$

P_i is the fraction of time of the state i and also the limiting probability. They can be solved from the balance equations.

$$P_i = \begin{cases} \sum_{j=0}^{i-1} \frac{I_{i-1-j}\lambda^j}{\frac{i!}{(i-1-j)!}\mu^{j+1}} + \frac{\lambda^i}{i!\mu^i}P_0, & 1 \le i \le m-1 \\ \left(\frac{\lambda}{m\mu}\right)^{i-m} \left(\sum_{j=0}^{m-1} \frac{I_{m-1-j}\lambda^j}{\frac{m!}{(m-1-j)!}\mu^{j+1}} + \frac{\lambda^m}{m!\mu^m}P_0\right), & i \ge m \end{cases}$$

In order to simplify the expression of P_i, we define

$$x_i = \frac{I_i}{\lambda f_i}, i \ge 0, \text{ and } X = P_0, \tag{2}$$

where

$$f_i = \begin{cases} \frac{\lambda^i}{i!\mu^i}, & 0 \le i \le m \\ \frac{\lambda^i}{m!m^{i-m}\mu^i}, & i \ge m+1 \end{cases} \tag{3}$$

Then we have the simplified expression of P_i,

$$P_i = \left(X + \sum_{i=0}^{\min\{i-1,m-1\}} x_i\right) f_i, \forall i \ge 0. \tag{4}$$

The stationary system implies that the overall task arrival rate equals the overall task service rate at infinite time instants. No exception are the arrival rate and the service rate of sleep tasks. Hence, we know

$$\sum_{j=0}^{m-1} I_j = I \tag{5}$$

By (2), we have

$$\sum_{j=0}^{m-1} f_j x_j = \frac{\omega}{1-\omega} \tag{6}$$

Also, because of the stationary process, the limiting probabilities, P_i must add up to 1. Combining (4), we have

$$G_0 X + \sum_{i=0}^{m-1} G_{i+1} x_i = 1, \tag{7}$$

where

$$G_i = \sum_{j=i}^{\infty} f_j. \tag{8}$$

In terms of the policy, it is easy to know that a real task has to wait only if all processors are blocked by either real or sleep tasks, i.e. all processors being busy. The average response time of real tasks depends on the average waiting time of real tasks and the probability that a real task arrives and has to wait equals the probability that all processors are busy, which is

$$\sum_{i \ge m}^{\infty} P_i = \left(X + \sum_{i=0}^{m-1} x_i\right) G_m. \tag{9}$$

Our target is to minimize the probability that a real task has to wait and (9) becomes the objective function and (6) and (7) are the constraints under our definition of the system and the problem. Although X is not independent of x_i, the optimization problem must be reduced to a linear programming problem with a specific X. Such a linear programming problem at hand only have two binding constraints, and therefore at most two out of $x_0, x_1, \cdots, x_{m-1}$ will be nonzero. Thus, given a different X, we can have a different linear programming problem and it is possible to find the optimal solution to the corresponding X. Since an X is a linear combination of $I_0, I_1, \cdots, I_{m-1}$ and a specific vector $\{I_0, I_1, \cdots, I_{m-1}\}$ is a policy to the proposed power management, we intend to find the optimal one within all possible X and $\{I_0, I_1, \cdots, I_{m-1}\}$. That is the optimal one from a population of policies.

Assuming x_k and x_{k-1} are the two nonzero variables, let us observe the objective function as X varies. The optimization problem becomes

$$\begin{array}{rl} \textbf{minimize} & Y = \sum_{i \geq m}^{\infty} P_i = (X + x_k + x_{k-1})\, G_m \\ \textbf{s.t.} & f_k x_k + f_{k-1} x_{k-1} = \frac{\omega}{1-\omega} \\ & G_0 X + G_{k+1} x_k + G_k x_{k-1} = 1 \end{array}$$

Solving x_k and x_{k-1} in terms of X from the constraints and reorganizing the objective function, the derivative of the objective function with respect to X is

$$\frac{\partial Y}{\partial X} = G_m \frac{G_0 f_{k-1} - G_{k+1} f_{k-1}}{G_k f_k - f_{k-1} G_{k+1}} \tag{10}$$

Because $G_0 > G_{k+1}, \forall k \geq 0$, the numerator is positive. According to the above definitions, $G_k = G_{k+1} + f_k$ and $f_k = f_{k-1} \frac{\lambda}{k\mu}, 1 \leq k \leq m$.

$$\begin{aligned} f_{k-1} G_{k+1} &= f_k \sum_{j>k}^{\infty} \frac{k\mu}{\lambda} f_j \\ &\leq f_k \sum_{j>k}^{\infty} f_{j-1} \\ &= f_k G_k \end{aligned} \tag{11}$$

The denominator is also positive. As a result, the objective function is a nondecreasing function of X no matter how x_k and x_{k-1} change.

The minimum X is 0. Consequently by ordering X to be 0, x_k and x_{k-1} can be solved.

$$x_k = \frac{\frac{\omega}{1-\omega} G_k - f_{k-1}}{G_k f_k - f_{k-1} G_{k+1}} \tag{12}$$

$$x_{k-1} = \frac{f_k - \frac{\omega}{1-\omega} G_{k+1}}{G_k f_k - f_{k-1} G_{k+1}} \tag{13}$$

Because of (11), the denominators are positive. For the numerator in (12), we have

$$\begin{aligned} \frac{\omega}{1-\omega} G_k - f_{k-1} &= \left(\frac{1}{1-\omega} - 1\right) G_k - f_{k-1} \\ &= \left(\frac{1}{1-\omega}\right) G_k - (f_{k-1} + G_k) \\ &= \left(\frac{1}{1-\omega}\right) G_k - G_{k-1} \\ &> 0 \\ \omega &\geq 1 - \frac{G_k}{G_{k-1}} \end{aligned} \tag{14}$$

The numerator in (13) can also be dealt with similarly and results in

$$\omega \leq 1 - \frac{G_{k+1}}{G_k} \tag{15}$$

It is not difficult to see that $1 - \frac{G_{k+1}}{G_k}$ is non-decreasing in terms of $0 \le k \le m$. Therefore, we need $k = \min\{j > 1 : \omega \le 1 - \frac{G_{j+1}}{G_j}\}$ to guarantee both (14) and (15).

Thus, the minimum probability, a lower bound, that all processors are busy is easy to know in terms of (9), (12), and (13). The more important to this problem is that p in the policy can be known in terms of (2) and x_k and x_{k-1}. With respect to the definition of p, we know

$$p = \frac{I_k}{I_k + I_{k-1}} \tag{16}$$

The optimal is based on $X = P_0 = 0$, which means that the system including the processors and the queue is rarely empty. Although in our analysis the optimality is observed by changing X, in fact X is decided after p and k and P_0 is usually a small positive real number. Assuming a small offset ε to the p derived in the above, we can write it like

$$p + \varepsilon = \frac{I'_k}{I'_k + I'_{k-1}} = \frac{I_k}{I_k + I_{k-1}} + \varepsilon \tag{17}$$

and also

$$I = I'_k + I'_{k-1} = I_k + I_{k-1}. \tag{18}$$

Consequently, we have $x'_{k-1} = \frac{I'_{k-1}}{\lambda f_{k-1}}$ and $x'_k = \frac{I'_k}{\lambda f_k}$ and then

$$x'_{k-1} = x_{k-1} - \frac{\varepsilon I}{f_{k-1}} \tag{19}$$

$$x'_k = x_k + \frac{\varepsilon I}{f_k} \tag{20}$$

Substituting x'_{k-1} and x'_k to (7), we can know that x'_{k-1}, x'_k, and P_0 all are positive, and the optimal solution can be reached as ε is infinitely close to 0. Thus, the policy is asymptotically optimal provided that the system is stable. The optimality is analyzed for minimum average response time of real tasks. It is necessary to discuss how the energy is saved. As mentioned at the end of the last section, the idle time per unit time is constant for a given I. The wasted energy is positively proportional to the idle time. Therefore, let us discuss the wasted energy per unit time, i.e. the wasted power, in the following.

The wasted energy per unit time is denoted as E_w. The supplied power is assumed to be constant and this is true if DVS is not applied. For this reason we can simply order the supplied power to be 1. The amount of energy by the time t is wasted during the idle time and thus we can write E_w to be

$$\begin{aligned} E_w &= \frac{\sum_{i=0}^{m-1}((m-i)P_i t)}{t} \\ &= \sum_{i=0}^{m-1}(m-i)P_i \\ &= aX + \sum_{i=0}^{m-2}\left(x_i \sum_{j=i+1}^{m-1}(m-j)f_j\right) \\ &= \frac{a}{G_0} - \frac{\sum_{i=0}^{m-1} aG_{i+1}x_i}{G_0} + \sum_{i=0}^{m-2} x_i b \\ &= \frac{a}{G_0} + \sum_{i=0}^{m-1}\left(\frac{bG_0 - aG_{i+1}}{G_0 f_i}\frac{I_i}{\lambda}\right) \end{aligned} \tag{21}$$

Here,

$$\begin{aligned} a &= \sum_{j=0}^{m-1}(m-j)f_j & (22) \\ b &= \sum_{j=i+1}^{m-1}(m-j)f_j. & (23) \end{aligned}$$

and $b = 0$, if $i = m - 1$.

In the second part of (21), we have

$$\begin{aligned} a &= \sum_{j=0}^{m-1}(m-j)f_j \\ &= m\sum_{j=0}^{m-1} f_j - \sum_{j=0}^{m-1} jf_j \\ &= m\sum_{j=0}^{m-1} f_j - \sum_{j=1}^{m-1} \left(\frac{\lambda}{\mu}f_{j-1}\right) \\ &= m\sum_{j=0}^{m-1} f_j - \frac{\lambda}{\mu}\sum_{j=0}^{m-2} f_j \\ &= \left(m-\frac{\lambda}{\mu}\right)\sum_{j=0}^{m-2} f_j + mf_{m-1} & (24) \end{aligned}$$

and

$$\begin{aligned} b &= \sum_{j=i+1}^{m-1}(m-j)f_j \\ &= m\sum_{j=i+1}^{m-1} f_j - \sum_{j=i+1}^{m-1} jf_j \\ &= m\sum_{j=i+1}^{m-1} f_j - \sum_{j=i+1}^{m-1} \left(\frac{\lambda}{\mu}f_{j-1}\right) \\ &= m\sum_{j=i+1}^{m-1} f_j - \frac{\lambda}{\mu}\sum_{j=i}^{m-2} f_j \\ &= \left(m-\frac{\lambda}{\mu}\right)\sum_{j=i+1}^{m-2} f_j - \frac{\lambda}{\mu}f_i + mf_{m-1} \\ &= a - (m-\frac{\lambda}{\mu})\sum_{j=0}^{i} f_j - \frac{\lambda}{\mu}f_i & (25) \end{aligned}$$

and

$$\begin{aligned}
& G_0 b - G_{k+1} a \\
= \; & G_0 \left(a - (m - \frac{\lambda}{\mu}) \sum_{j=0}^{k} - \frac{\lambda}{\mu} f_k \right) - a G_{k+1} \\
= \; & a \sum_{j=0}^{k} f_j - (m - \frac{\lambda}{\mu}) G_0 \sum_{j=0}^{k} f_j - \frac{\lambda}{\mu} G_0 f_k \\
= \; & a \sum_{j=0}^{k} f_j - m G_0 \sum_{j=0}^{k} f_j + \frac{\lambda}{\mu} G_0 \sum_{j=0}^{k-1} f_j \\
= \; & a \sum_{j=0}^{k-1} f_j - m G_0 \sum_{j=0}^{k-1} f_j + \frac{\lambda}{\mu} G_0 \sum_{j=0}^{k-1} f_j + a f_k - m G_0 f_k \\
= \; & \underline{(a - m G_0 + \frac{\lambda}{\mu} G_0)} \sum_{j=0}^{k-1} f_j + a f_k - m G_0 f_k \qquad (26)
\end{aligned}$$

The underlined part is in fact zero because

$$\begin{aligned}
& a - m G_0 + \frac{\lambda}{\mu} G_0 \\
= \; & m \sum_{i=0}^{m-1} f_i - \frac{\lambda}{\mu} \sum_{0}^{m-2} f_i - m G_0 + \frac{\lambda}{\mu} G_0 \\
= \; & m \left(\sum_{i=0}^{m-1} f_i - G_0 \right) + \frac{\lambda}{\mu} \left(\sum_{i=0}^{m-2} f_i + G_0 \right) \\
= \; & -m G_m + \frac{\lambda}{\mu} \left(\frac{\left(\frac{\lambda}{\mu}\right)^{m-1}}{(m-1)!} + \sum_{i=0}^{\infty} \frac{\left(\frac{\lambda}{\mu}\right)^{m+i}}{m! m^i} \right) \\
= \; & -m G_m + \frac{\left(\frac{\lambda}{\mu}\right)^{m}}{(m-1)!} + \sum_{i=0}^{\infty} \frac{\left(\frac{\lambda}{\mu}\right)^{m+1+i}}{m! m^i} \\
= \; & -m G_m + m \left(\sum_{i=0}^{\infty} \frac{\left(\frac{\lambda}{\mu}\right)^{m+i}}{m! m^i} \right) \\
= \; & -m G_m + m G_m \\
= \; & 0 \qquad (27)
\end{aligned}$$

Therefore, $\forall i$, $0 \leq i \leq m-1$

$$\frac{G_0 b - G_{k+1} a}{f_k G_0} = \frac{a}{G_0} - m \qquad (28)$$

Substituting (27) and (28) into (21), we know that E_w depends on only I and is not relevant to the distribution of I_i. Therefore, we can rewrite E_w.

$$\begin{aligned}
E_w &= \frac{a}{G_0} + \frac{a - m G_0}{G_0 \lambda} I \\
&= m - \frac{\lambda}{\mu} - \frac{I}{\mu} \qquad (29)
\end{aligned}$$

Table 1: Some calculation results ($\mu = 0.9$, $m = 10$, fixed ρ)

	$\rho = 0.99$		$\rho = 0.85$		$\rho = 0.70$		$\rho = 0.55$	
λ	p	k	p	k	p	k	p	k
1	0.898	9	0.405	8	0.863	6	0.344	5
2	0.883	9	0.247	8	0.677	6	0.017	5
3	0.863	9	0.032	8	0.399	6	0.656	4
4	0.835	9	0.815	7	0.951	5	0.870	3
5	0.793	9	0.495	7	0.325	5	-	-
6	0.722	9	0.916	6	0.758	3	-	-
7	0.576	9	0.828	5	-	-	-	-
8	0.110	9	-	-	-	-	-	-

Table 2: Some calculation results ($\mu = 0.9$, $m = 10$, fixed I)

	$I = 3$		$I = 2$		$I = 1$	
λ	p	k	p	k	p	k
1	0.0368	4	0.0724	3	0.0453	1
2	0.0532	5	0.0172	3	0.0778	2
3	0.0021	5	0.0296	4	0.0902	3
4	0.0060	6	0.0364	5	0.0091	3
5	0.0011	7	0.0358	6	0.0216	4

In short, given I, the optimal policy can be computed and then the average slow-down is also known. In practice, it is always possible to find the desired combination of I and the corresponding policy. Note that the results are realistic on stationary systems in which the length of waiting queue is stable and does not increase unlimitedly. Most systems running normally in long time periods meet this assumption. Hence, the policy and the analysis are effective.

6 Empirical Studies

The purpose of empirical studies is to show the system behaviors for better understanding the injection policy and our analysis. A system with 10 identical processors and the service rate of each processor 0.9 are simulated. In terms of our analysis, the asymptotically optimal policies for different system configurations, being represented by p and k, are listed in Table 1 for different arrival rates λ and system utilizations ρ. It is also easy to know the injection rate at $k - 1$ is $1 - p$. Note that for some combinations of arrival rate and utilization the policies do not exist because the arrival rates of real tasks may result in higher utilization than those in the table. In Table 2, for three fixed values of I the policies are calculated at different arrival rates. In Table 3, I and λ are fixed to be 2 and 3 respectively. As we know from Table 2, the optimal policy for $I = 2$ and $\lambda = 3$ is $k = 4$ and $p = 0.0296$. Thus, the other values of k and p shown in Table 3 are not optimal, but all of them guarantee that the queue is stable. The arrivals of 50000 real tasks follow Poisson process and the statistics are collected only from the steady phase.

Corresponding to the above three tables, Fig.2-4 show the experimental results. In each of the figures, there are two Y-axes. The percentage of idle time refers to the ratio of the total idle time to the overall simulation time excluding starting and ending time periods. The percentage follows the left Y-axis, whereas the average response time of tasks follows the right Y-axis. In Fig.2 it compares

Table 3: Some calculation results ($\mu = 0.9$, $m = 10$, fixed I and λ)

k	4	5	6	7	8	9
p	0.0296	0.1021	0.1603	0.2067	0.2429	0.2667

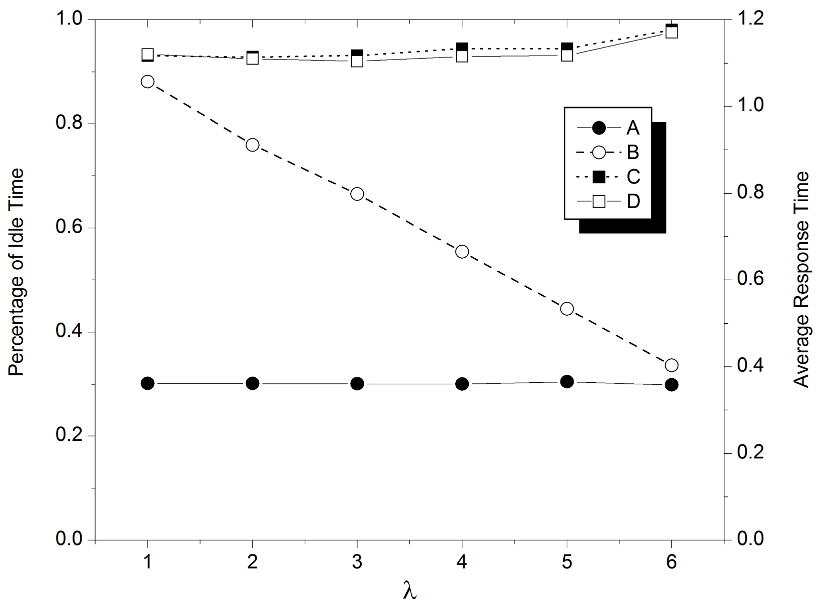

Figure 2: Experimental results for fixed $\rho = 0.7$. A is the percentage of idle time in the simulated system with the injection policy (following the left Y-axis); B is the percentage of idle time in the simulated system without switching off any processor (following the left Y-axis); C is the average response time of real tasks from the same system as A (following the right Y-axis); D is the average response time of real task from the same system as B (following the right Y-axis).

the system with the optimal injection policy at $\rho = 0.7$ and that without switching off any processor at all. As we can observe from Fig.2, the average response time of real tasks is not slowed down by the injection policy obviously. To some extent, the performance is hardly influenced by our policy, whereas the idle time per unit time is reduced greatly. Because power and energy are wasted in idle time, it is reasonable to conclude that power, i.e. energy consumption per unit time, is saved. The total ρ is merely 0.7 and hence the average response time is slightly greater than the average running time (1.11 due to $\mu = 0.9$) for the high priority of real tasks. In spite of that, the increment of average response time is observable as the increasing of λ. For a higher λ, more time is spent on real tasks since more real tasks are going to run in the same time slots. As a result, B is decreasing as λ increases, but A keeps the same because of injected sleep tasks and varying I. At $\rho = 0.7$ in Table 1, the maximum λ guaranteeing a stable system is 6 and A is approaching B. If λ is possible to increase continuously, both A and B will approach such a case in which no sleep task can be injected for the crowded real task traffic. It is believed that B will not be lower than A definitely.

Similar to Fig.2, Fig.3 shows the experimental results for $I = 3$. As analyzed in the last section, the wasted energy per unit time is decided by both I and λ and is proportional to ρ. In Fig.2, ρ is fixed and hence A is almost unchanged as the increasing of λ. However, in Fig.3, we can observe that A is decreasing as λ increases because ρ is also increases for the fixed I. Consistent with the common sense again, the average response time of real tasks is slowed down for higher λ.

Fig.4 demonstrates the fact that the wasted energy per unit time, i.e. the percentage of idle time per unit time, is constant, since A keeps the same for different k, each of which except $k = 4$ represents the injection policy with non-optimal setting. Provided that the system is stable, only average response time should be optimized. The average response time at $k = 4$ is indeed the minimum, however, the others are very competitive. In practical systems, this phenomenon has been observed in [14] that often a simple policy that turns off servers when not required is found quite competitive with more complex policies. It is expensive to find the optimal policy in more complicated systems and hence in practice it is more important and easier to guarantee stable

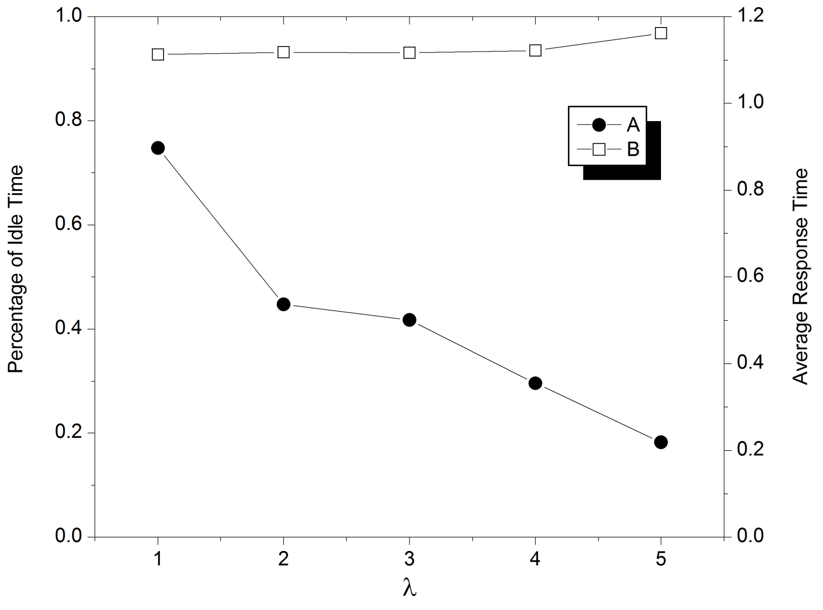

Figure 3: Experimental results for $I = 3$. A is the percentage of idle time per unit time in the simulated system with the injection policy (following the left Y-axis); B is the average response time of real tasks from the same system as A (following the right Y-axis).

systems than the optimality. Nevertheless, in practical systems the injection policy will be robust even if the optimized parameters are not so accurate.

Note that the policy does not imply only some processors can be supplied with low power or switched off, although in the vector there are only two non-zero numbers. Actually, each processor is possible to be in the low power state. In this analysis, the mean time length of sleep tasks has been assumed to equal that of real tasks. This may not be true in all the cases, but sleep tasks can be split or converged freely and easily to meet the assumption because sleep tasks are not real ones.

To close this section, we compare our policy to DELAYEDOFF in [18] since it has been proven to be asymptotically optimal. The predetermined waiting time of DELAYEDOFF is approximately

$$t_{wait} = setup\ time \cdot R_{On/Off} = setup\ time \frac{active\ power}{idle\ power}. \tag{30}$$

According to [16], the ratio of active power to idle power, $R_{On/Off}$, is almost 2. Setup time is usually different in computer systems and factored into architecture, configuration and hardware. Thus, in the simulation we value setup time within a reasonable scope to examine the performance of both our policy and DELAYEDOFF. μ is still 0.9 and λ is set to be 5. Setup time is set to be 0.5 or 1 time unit. We do not consider the case that setup time is longer than average task running time. The parameters of our injection based policy are from the above tables. The results are shown in Table 4, in which ρ is fixed to be 0.99 for our injection based policy, i.e., trying to inject as many sleep tasks as possible. Since the arrival rate of real tasks is 5, the percentage of active time must be the utilization of real tasks. Obviously, the simulation results are consistent with theoretical expectation. It is not surprising that the idle time for our policy is much less than that for DELAYEDOFF, because the number of injected sleep tasks is very close to its limit. The average response time for our policy is very competitive. Compared to DELAYEDOFF, our policy wins at the tradeoff between wasted power and response time as the power consumption is generally constant in a given power state. However, the frequency of On/Off switching for our policy is higher than that for DELAYEDOFF. In some cases, there are power penalties in the transfer from idle states to active states, but the penalties are not generally existing. Moreover, our policy can be

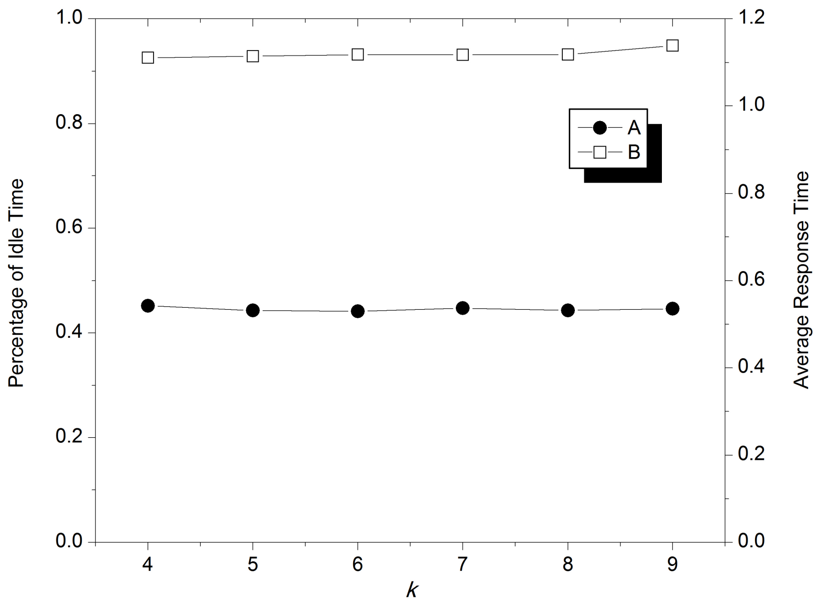

Figure 4: Experimental results for other settings. *A* is the percentage of idle time per unit time in the simulated system with the different *k*(following the left Y-axis); *B* is the average response time of real tasks from the same system as A (following the right Y-axis).

Table 4: Comparison between DELAYEDOFF and INJECTION

	INJECTION		DELAYEDOFF	
Setup time	0.5	1	0.5	1
Percentage of active time	0.5506	0.5537	0.5533	0.5533
Percentage of idle time	0.0142	0.0104	0.2255	0.2968
Percentage of off/sleep time	0.4356	0.4359	0.2210	0.1497
Average response time	1.3016	1.2970	1.2096	1.2320
On/Off switches per unit time	0.3543	0.3542	0.1300	0.0994

Table 5: Comparison between DELAYEDOFF and INJECTION

	INJECTION under different setup time		DELAYEDOFF under different $R_{On/Off}$	
	0.5	1	3	4
Percentage of active time	0.5580	0.5559	0.5577	0.5563
Percentage of idle time	0.1319	0.1315	0.3237	0.3489
Percentage of off/sleep time	0.3103	0.3126	0.1183	0.0945
Average response time	1.1575	1.1533	1.2030	1.1873
On/Off switches per unit time	0.1532	0.1500	0.0802	0.0689

enhanced by slightly changing the task assignment to schedule consecutive sleep tasks to the same processors and thus the frequency can be reduced. The frequency in DELAYEDOFF will increase as setup time or $R_{On/Off}$ becomes smaller, whereas the frequency in our policy will decrease if less sleep tasks are injected.

In Table 5, the total utilization including all real and sleep tasks is lowered to 0.85 but the arrival rate is still 5. Therefore, the number of injected sleep tasks is smaller than in the above simulation. For DELAYEDOFF, setup time is still 1 but $R_{On/Off}$ is increased to 3 and 4. The results indicate that by tuning the strength of injection our policy can completely ourperform DELAYEDOFF given the same system settings (comparing DELAYEDOFF in Table 4 and INJECTION in Table 5), since (30) has been shown the optimal choice for t_{wait} in [18]. On the other hand, if $R_{On/Off}$ is greater (though this is not true in many production systems), the switching frequency for DELAYEDOFF can be further reduced. That is, INJECTION can be close to DELAYEDOFF by reducing the strength of injection always. It can be concluded that INJECTION offers such a flexibility of tradeoff between energy consumption and On/Off switch frequency.

In order to further understand the energy efficiency in INJECTION and DELAYEDOFF, let us assume that the supplied idle power is p_i. $\forall t$ time period, the wasted energy should be the multiplication of t, p_i and the percentage of idle time. In terms of the data in Table 4 and 5, it is easy to know that the wasted energy of INJECTION is much less than DELAYEDOFF because p_i is constant. Comprehensively comparing our policy to DELAYEDOFF, it is necessary to consider energy efficiency, response time, and switching frequency if power penalties exist. In practice, many systems have small setup time and no power penalties and hence our policy is much better for not only its efficiency but also it stability because the performance of our policy is only decided by the strength of injection.

7 Conclusion

In this paper, a power management for multiprocessor systems is proposed. The significant difference from the others lies in the idea of sleep tasks. Basically, most other techniques have to accurately estimate the time intervals of task arrivals, the times in low power states and the active times of each processor. Therefore, the effect of other techniques depends on the accuracy of the estimation. In this paper, sleep tasks are generated and injected into the real task traffic in advance. It has been proved that the wasted energy per unit time is constant if the system is stable. However, it is needed to guarantee the minimum impact on real tasks. Therefore, a probabilistic policy is applied to optimally inject sleep tasks to minimize the average response time of real tasks. Through our analytical studies, it is shown that given proper parameters, the policy is optimal. Through empirical studies, it is concluded that our policy is better than an existing asymptotically optimal policy.

References

[1] Wei Sun, An optimal power management for stationary multiprocessor systems. In *Proc. of the Second International Conference on Networking and Computing*, Nov. 2011.

[2] D.J. Brown and C. Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3):50-58, 2010.

[3] P. Ranganathan. Recipe for efficiency: principles of power-aware computing. *Communications of the ACM*, 53(4):60-67, 2010.

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

[5] T. Simunic. Dynamic management of power consumption. *Power Aware Computing* edited by R. Graybill and R. Melhem, 2002

[6] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmca*, pages 542-571, 1994.

[7] D. Ramanathan and R. Gupta. System level online power management algorithms. *Design, Automation and Test in Europe*, pages 606-611, 2000.

[8] C-H. Hwang and A. Wu. A predicativee system shutdown method for energy saving of event-driven computation. In *Proc. of International Conference on Computer Aided Design*, pages 28-32, 1997.

[9] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predicative system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems*, 4(1):42-55, 1996.

[10] L. Benini, G. Paleologo, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design*, 18(6):813-833, 1999.

[11] Q. Qiu and M. Pedram. Dynamic power management based on continuous-time markov decision processes. In *Proc. of Design Automation Conference*, pages 555-561, 1999.

[12] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. *Compilers and Operating Systems for Low Power*, L. Benini, M. Kandemir, and J. Ramanujam, eds., Kluwer Academic Publishers, August 2003.

[13] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proc. of the 18th Symposium on Operating Systems Principles*, October 2001.

[14] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of the Second Workshop on Power-Aware Computing Systems*, Februry 2002.

[15] E. A. Peköz. Optimal policies for multi-server non-preemptive priority queues. *Queueing Systems*, vol.42, pages 91-101, 2002.

[16] L.A. Barroso and U.Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, vol.40, Issue 12, pages 33-37, Dec. 2007.

[17] A. Gandhi, M. Harchol-Balter, and I. Adan. Server farms with setup costs. *Performance Evaluation*, vol.67, Issue 11, pages 1123-1138, Nov. 2010.

[18] A. Gandhi, et al. Optimality Analysis of Energy-Performance Trade-off for Server Farm Management. *Performance Evaluation*, vol.67, Issue 11, pages 1155-1171, Nov. 2010.