Power-Aware Processor Scheduling under Average Delay Constraints *

Fan Zhang and Samuel T. Chanson

Department of Computer Science

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

Email: {zhangfan, chanson}@cs.ust.hk

Abstract

In this paper, voltage scaling strategies for scheduling aperiodic tasks under average delay constraints are studied. Dynamic voltage scaling in single processor systems is formulated as a constrained stochastic optimization problem for which the optimal solution can be obtained using a combination of Lagrange relaxation and the value iteration method. For multiprocessor systems, we present a two-phase approach. In the first phase, the speed settings and static workload distribution of the processors are optimized to minimize the total power dissipation. Dynamic voltage scaling techniques are then applied to each individual processor in the second phase. Both homogeneous and heterogeneous systems have been investigated. Based on queueing theory, the proposed algorithms guarantee conformity to the average delay constraint. Moreover, our simulation experiments have shown they are effective for minimizing power consumption.

1 Introduction

Energy efficiency is an important design goal for two different types of systems — mobile computing systems [3,10] and densely packed server clusters [6,12]. For battery-powered mobile systems, improving energy efficiency minimizes energy consumption and prolongs battery life. For server clusters consisting of a large number of processors, excessive heat dissipation greatly impairs system reliability and poses a heavy burden on the cooling subsystems. Improving energy efficiency for server systems alleviates the heat dissipation problem and reduces the cost of the cooling systems. Voltage scaling, a technique to reduce processor power by lowering the supply voltage, is effective in minimizing the energy consumption (and consequent heat dissipation) of processors [16,3,19]. However, as the processor speed decreases when the supply voltage is reduced, voltage scaling will result in a longer task processing time. Voltage scaling schemes that balance energy saving and service degradation have attracted great interest, especially for tasks with quality of service (QoS) requirements.

Much of the existing work in this area has focused on periodic real-time tasks with fixed periods, hard deadlines and specified worst-case execution times (WCET). Typically, the processor is slowed down as

*The work described in this paper was supported by grants from the National Natural Science Foundation of China and the Research Grants Council of Hong Kong SAR, China (Project No. N_HKUST618/02 and HKUST6178/03E).

much as possible so long as timing constraints are not violated [11, 3, 19]. Another important class of tasks known as aperiodic tasks, however, have dynamic and unpredictable arrival patterns and processing times. In this case, it is difficult to strictly guarantee hard deadlines by scheduling. Fortunately, for many applications, the aperiodic tasks do not have vigorous timing constraints. Instead, soft real-time QoS guarantees (such as average response times) are sufficient. However, results valid for scheduling periodic tasks may not hold for aperiodic tasks. For example, without violating timing constraints, a periodic task set can be partitioned evenly on two processors, each running at half the speed that is required to process the task set on one processor. However, it is well-known from queueing theory that this produces a longer average response time for randomly arriving aperiodic tasks.

We investigate voltage scaling strategies for scheduling aperiodic tasks in both single processor systems and multiprocessor server systems. The systems under consideration include common mobile/embedded systems, stand alone or distributed web servers, database servers, and high performance application servers. Incoming tasks can take the form of web object requests, data queries and general computation requests triggered by users or by external events. Each individual task is assumed not to have a hard deadline. Instead, the goal is to minimize the total energy consumption while keeping the *average* response time below a specified value. For single processor systems, we propose a voltage scaling scheme that dynamically adjusts the processor speed according to the number of tasks (backlog) in the system. The system is modeled as a discrete-time Markov chain, and the average delay of the tasks is mathematically analyzed. The optimal voltage scaling strategy is formulated as a constrained optimization problem and solved using existing optimization techniques [13, 14].

For multiprocessor systems, voltage scaling is divided into two phases. In the first phase, a power-optimized fixed-speed schedule is computed. The output includes a list of processors that should be turned on, workload allocation and the speeds of the active processors. The optimal dynamic voltage scaling technique developed for the single processor case is then applied to each individual processor in the second phase. Both homogeneous systems and heterogeneous systems are investigated. The average task delay is analyzed using queueing theory. In each phase of scheduling, the algorithms are designed to guarantee the average delay requirement. To the best of our knowledge, this is the first work applying queueing analysis in voltage scaling of aperiodic tasks on multiple processors.

The rest of the paper is organized as follows: Section 2 reviews related work in voltage scaling. In Section 3 we study dynamic voltage scaling in single processor systems. The multiprocessor scheduling problem is investigated in Section 4. Performance evaluation results are presented in Section 5. Finally, Section 6 concludes this paper.

2 Related work

When voltage scaling is used, the processor speed decreases proportionally with the supply voltage. The relationship between processor speed s and power P can be described by the formula below [5, 16, 6]:

$$P(s) = As^\alpha + P_{static}, \quad (1)$$

where P_{static} is the static power component independent of the processor speed. As^α accounts for the dynamic power P_{dyn} , where A is a constant coefficient and α is a system dependent parameter usually assumed to be between 2 and 4. For simplicity, we adopt the commonly used value $\alpha = 3$ in the rest of the paper.

Despite the increased processing delay, voltage scaling is of particular interest to real-time applications because longer processing times can be tolerated as long as the timing constraints are not violated. It is shown in [3] that for real-time periodic tasks, the optimal voltage scaling strategy is to run the processor at a fixed normalized speed equal to the utilization bound of the scheduling algorithm. Scheduling tasks with non-preemptible sections usually requires a higher speed compared with scheduling fully preemptible tasks. The dual-speed approach [18, 19] saves energy by allowing some tasks to be processed at a lower speed as if they were scheduled in a fully preemptive environment.

Aperiodic tasks without strict timing constraints can be scheduled using interval-based schemes. Voltage scaling decisions are based on the observed processor utilization in the last interval [16] or the weighted average utilization in all past intervals [10]. The scheme proposed in [15] adjusts processor speed based on the synthetic utilization bound of aperiodic tasks. Feasibility is guaranteed by keeping the instantaneous synthetic utilization below the bound. Concurrently scheduling mixed periodic tasks and aperiodic tasks is studied recently [4]. Aperiodic tasks are handled by a total bandwidth server (TBS) utilizing the CPU capacity leftover by periodic tasks. The algorithm optimizes processor speeds for running the total bandwidth server and the periodic tasks. However, there is no theoretical analysis given on the average response time of the aperiodic tasks scheduled in this way.

Dynamic power management has also been used in multiprocessor systems and server clusters to minimize energy consumption and heat dissipation [6, 12]. Note that a processor still consumes a substantial amount of static power even when it is idle or running at low speed. If the workload is light, it is advisable to turn off some computing nodes and distribute the workload to the rest of the nodes [12]. The approach of combining voltage scaling with node power-off is studied in [6]. These scheduling schemes usually take a trial-and-error approach. The number of processors and their speeds are increased if the measured utilization exceeds a specified upper threshold, and are decreased if the utilization falls below a lower threshold. No analytical analysis on guaranteeing the average delay constraint is available. Unlike the existing work, we take average delay into consideration as a constraint. Based on queuing analysis, the voltage scaling strategies proposed in our work guarantee the specified average delay constraints are met.

3 Single processor

In this section we study dynamic voltage scaling (DVS) for single processor systems. Under a discrete-time framework, computing the optimal voltage scaling solution is formulated as a constrained stochastic optimization problem. The problem is then solved using a combination of Lagrange relaxation and the value iteration methods.

3.1 System model

First we model the system as a discrete time Markov process. For simplicity of mathematical analysis, we assume the arrival of incoming processing requests is a Poisson process with an average arrival rate λ . When the processor runs at full speed, the execution times of the tasks are exponentially distributed with average service rate μ . We will show later how this model can be extended to other types of task distribution. Time is divided into fixed-length slots. Tasks arriving within a time slot are inserted into the task queue at the end of the time slot. The number of tasks in the system, denoted by x , is used to specify the system state. The state space is finite and is denoted by $\mathbb{R} = \{0, 1, \dots, N\}$, where N is the maximum number of tasks allowed in the system concurrently. Incoming tasks are processed in the first-come-first-serve (FCFS) manner. The processor speed s can be adjusted at the beginning of each time slot only. The value of s is taken from a set of m speed levels $\mathbb{S} = \{S_1, S_2, \dots, S_m\}$, sorted in increasing order and normalized to the full speed $S_m = 1$. The length of a time slot is a system parameter. A short time slot may incur frequent processor speed adjustments and a higher computational overhead at run time. In the extreme case, it is equivalent to changing the speed upon each task arrival and departure. A long time slot, on the other hand, is less sensitive to task dynamics and may miss some energy-saving opportunities. However, it has a lower runtime overhead.

We consider stationary DVS policies that determine the processor speed based on the current system state only. A policy is defined as a function from the system states to the processor speed set, that is, $\pi : \mathbb{R} \rightarrow \mathbb{S}$.

We calculate the transition probabilities of the Markov process as follows. Let p_k be the probability of k arrivals in one time slot, and $\overline{p_k}$ be the probability of k or more arrivals. As task arrival is a Poisson process, $p_k = e^{-\lambda} \cdot \frac{\lambda^k}{k!}$, $\overline{p_k} = 1 - \sum_{i=0}^{k-1} p_i$. When the processor runs at speed s , the probability that k tasks will complete in one time slot is: $p_k(s) = e^{-\mu s} \cdot \frac{(\mu s)^k}{k!}$. The probability of k or more completions is $\overline{p_k(s)} = 1 - \sum_{i=0}^{k-1} p_i(s)$. The probability that the system transits from state q to state r when the processor runs at speed s can then be computed as follows:

$$p_{q,r}(s) = \begin{cases} p_r, & q = 0, r < N; \\ \overline{p_r}, & q = 0, r = N; \\ \sum_{i=0}^{q-1} p_i(s) p_{r-q+i} + \overline{p_q(s)} p_r, & 0 < q \leq r < N; \\ \sum_{i=0}^{q-1} p_i(s) \overline{p_{r-q+i}} + \overline{p_q(s)} \overline{p_r}, & 0 < q \leq r, r = N; \\ \sum_{i=q-r}^{q-1} p_i(s) p_{r-q+i} + \overline{p_q(s)} p_r, & 0 \leq r < q \leq N. \end{cases}$$

We have assumed exponential task distribution to simplify the mathematical formulation. However, in many computing systems the task processing times exhibit a heavy-tailed property [8]. Unlike exponential distribution, heavy-tail distribution is not memoryless. Therefore, in addition to the number of tasks in the system, we need a new variable to track the system state: the amount of time (normalized to full speed) the current task has been processed. Denote this time by z ($z \in \mathbb{Z} = \{0, 1, \dots, Z\}$), where Z is the upper bound of task processing time. A stationary DVS policies in this case is a function $\pi : \mathbb{R} \times \mathbb{Z} \rightarrow \mathbb{S}$. A two dimensional Markov chain is needed to model the computer system. Given a system state (x, z) and processor speed s , we can calculate the probability of one or more task

arrivals/completions. The transition probability matrix can then be built in a similar way as in the previous paragraph.

After modeling the system as a Markov process, we shall formulate the optimal DVS problem as a constrained optimization problem and show how the optimal DVS strategy can be obtained in the next subsection. Note that the optimization process is independent of the formulation of the Markov process. It simply takes as input the transition probabilities calculated from the Markov model.

3.2 Constrained optimization

With each state-action pair in the discrete-time Markov process formulated in Section 3.1, we now define the energy cost $E(x, s)$ and the delay cost $D(x, s)$ ($x \in \mathbb{R}, s \in \mathbb{S}$) in each time slot.

The energy consumption in each time slot depends on the processor speed. When the task queue is empty, the processor enters the idle state and the power dissipation is P_{static} . Otherwise the energy cost in one time slot is $E(x, s) = (As^3 + P_{static}) \cdot 1$ according to formula (1). The speed transition time of modern processors is usually under $100\mu s$, and the time the processor is unavailable during the transition is as short as $10\mu s$ [7]. These times are very short compared to the tasks' execution times. Consequently, the time and energy overhead during speed adjustment are considered to be negligible.

It is difficult to analyze the task delay T directly. However, the number of tasks x in the system in each time slot can be measured easily. According to Little's Law [9], the average delay $\bar{T}$ and the average number of tasks $\bar{x}$ follow the simple relationship $\bar{x} = \lambda\bar{T}$. We therefore measure delay cost in terms of the number of tasks in the system, i.e., in each time slot $D(x, s) = x$. Note that $D(x, s)$ in any time slot does not represent the actual delay experienced by any particular task. Instead, the sum of $D(x, s)$ is used to obtain the average value of delay. Furthermore, the processing capability is not specified in Little's Law. The processing unit can be regarded as a black box, independent of the number of actual processors and their speeds. This property enables us to apply Little's Law to voltage scaling and later to the multiprocessor scenarios (see Section 4).

Let x^i and s^i be the state and processor speed at the beginning of time slot i respectively. When the initial state is x , the average energy cost $\bar{E}_\pi(x)$ and the average delay cost $\bar{D}_\pi(x)$ under policy π can be defined as follows:

$$\begin{aligned}\bar{E}_\pi(x) &= \limsup_{n \rightarrow \infty} \frac{1}{n} \mathbf{E} \left[\sum_{i=0}^{n-1} E(x^i, s^i) | \pi, x^0 = x \right], \\ \bar{D}_\pi(x) &= \limsup_{n \rightarrow \infty} \frac{1}{n} \mathbf{E} \left[\sum_{i=0}^{n-1} D(x^i, s^i) | \pi, x^0 = x \right],\end{aligned}$$

where $\mathbf{E}$ represents the expectation value. Assume the average delay constraint is D . A policy π is feasible if the delay constraint can be satisfied ($\bar{D}_\pi(x) \leq D, \forall x \in \mathbb{R}$). Use Δ to denote the set of all feasible policies. The optimization problem is to find a policy $\pi^* \in \Delta$ that minimizes the average energy cost:

$$\bar{E}_{\pi^*}(x) = \inf_{\pi \in \Delta} \bar{E}_\pi(x), \quad \forall x \in \mathbb{R}. \quad (2)$$

Figure 1: Average energy cost of dynamic and fixed speed processing ($\lambda = 0.4$).

Using the Lagrange relaxation technique, we can transform the above constrained stochastic optimization problem into an unconstrained optimization problem. Let $\Lambda \in [0, \infty)$ be a Lagrange multiplier, the relaxed form of the original optimization problem is:

$$J_{\pi}(x, \Lambda) = \limsup_{n \rightarrow \infty} \frac{1}{n} \mathbf{E} \left[\sum_{i=0}^{n-1} E(x^i, s^i) + \Lambda \sum_{i=0}^{n-1} D(x^i, s^i) \mid \pi, x^0 = x \right], \quad \forall x \in \mathbb{R}. \quad (3)$$

For a fixed Λ , the relaxed problem can be solved using the value iteration method. The optimal solution to the original problem is obtained by iteratively adjusting the value of Λ . This approach was first proposed by Sennott [14] to solve the general class of average-cost constrained stochastic optimization problems. A brief description of this approach is given in the appendix. Interested readers are referred to [14] for details.

3.3 Numerical results

The optimal dynamic speed strategy obtained using the approach in Section 3.2 is evaluated against an optimal fixed speed voltage scaling scheme. Most current commercial processors only support a few discrete speed levels for voltage scaling. We considered the Intel PXA27x processor family [2] in our experiments. This processor supports 6 speed levels: 0.2, 0.3, 0.4, 0.6, 0.8 and 1 after normalization. As the static power P_{static} is relatively constant and independent of voltage scaling strategies, we plot the dynamic power P_{dyn} only. The power is normalized so $P_{dyn} = 1$ when the processor runs at full speed. The task arrival rate was fixed at 0.4, and the delay constraint (in terms of the average number of tasks in the system) was varied between 0.75 and 4.

For each value of the delay constraint, the fixed speed algorithm runs at the lowest speed level that satisfies the constraint. In Figure 1, the energy consumption of the optimal dynamic speed scheme decreases smoothly as the delay constraint is relaxed. In contrast, constant speed scheduling produces a saw-tooth shape. There are three horizontal segments in Figure 1, respectively corresponding to speed

levels 1, 0.8 and 0.6 that produce a finite average delay. If the delay constraint falls between the delay values of two speed levels, the higher speed is required to satisfy the constraint. In dynamic speed scheduling, the processor can be made to run at a lower speed when the number of tasks is small, and at a higher speed as the waiting queue builds up. As a result, dynamic speed scheduling achieved about 10 ~ 45% energy saving over fixed speed scheduling. Even when the delay constraint is equal to the delay produced by running the processor at a fixed speed level (see delay = 1 and 2 in Figure 1), dynamic speed scheduling could still adaptively change the processor speed to neighboring speed levels and therefore achieve an additional energy saving of about 10%.

In order to study the impact of heavy-tail workload, we have also generated task processing time according to the Bounded Pareto Distribution in our simulation experiments. The probability density function $B(\alpha, k, p)$ is defined as follows [8]:

$$f(x) = \frac{\alpha k^\alpha}{1 - (k/p)^\alpha} x^{-\alpha-1}, \quad (k \leq x \leq p)$$

where k and p are the lower bound and the upper bound of task processing time respectively, and α reflects the variability of processing time. The following values were used to generate the workload: $k = 10$, $p = 20000$, and $\alpha = 2$. Tasks produced under this setting had an average processing time of one time slot when processed at full speed. As can be seen from Figure 1, the difference in average power consumption between heavy-tail and exponentially distributed workload is very small, being less than 5% except for one boundary point. Not shown in the figure, the average delay of the tasks was consistently kept below the specified delay constraint.

4 Multiple processors

Consider a multiprocessor system or server cluster consisting of n processing units (processors) $c_1, c_2, \dots, c_n$. Tasks arriving at the system are assigned to the processors through a central dispatcher. Voltage scheduling under this scenario involves the issues of active processor selection, task allocation and voltage scaling.

When an *active* processor has no task to process, it enters the idle state. The power consumption of an idle processor is P_{static} . Note that when a processor is turned off, it does not consume any power. If the workload is known or is predictable, a set of processors can be selected to stay active for task processing while the rest are turned off to save energy.

Task allocation strategies can usually be classified as dynamic or static based on when the tasks are assigned to the processors. In dynamic allocation, all tasks are queued at the central dispatcher and are allocated to the processors one by one only when a processor becomes idle. Static allocation schemes, on the other hand, assign a task to a processor according to some criteria as soon as it arrives.

When tasks are dynamically allocated, the set of all the processors can be regarded as a single entity. The approach described in the previous section for finding the optimal DVS policy in single processor systems can be easily extended to handle multiprocessor systems in this case. The service rate of the system at anytime is the sum of the service rates of all the non-idle processors. An action

Figure 2: Static task assignment and processing in a multiprocessor system.

is described using a vector $[s_1, s_2, \dots, s_n]$, where s_i denotes the normalized speed of processor c_i . The energy cost is the total energy consumption of all the processors in the system and the delay cost is measured by the number of tasks in the system as before. The constrained optimization problem can be formulated and solved in the same way as described in Section 3. However, as the action space specified by $[s_1, s_2, \dots, s_n]$ grows exponentially with the number of processors, this approach does not scale up well when the number of processors is large.

In the rest of this section we consider static task allocation. A fraction α_i ($0 \leq \alpha_i \leq 1$, and $\sum_{i=1}^n \alpha_i = 1$) of all the incoming tasks is assigned to processor c_i (see Figure 2). In other words, each task arriving at the system has a probability α_i of being allocated to processor c_i .

We propose a two-phase scheme to solve the voltage scaling problem. In the first phase, the active processor set, the static workload allocation for each processor, and the optimal fixed speed at which the processor should run are computed. The average response time on each processor is also calculated. In the second phase, the optimal DVS policy is obtained for each processor individually using the approach described in Section 3. Therefore we shall focus on phase one in the following subsections.

4.1 Homogeneous processors

We first consider a homogeneous multiprocessor system composed of identical processors. Again, to simplify analysis, we assume task arrival is a Poisson process and the processing time is exponentially distributed. Suppose the arrival rate is λ and the service rate is μ when tasks are processed on a single processor at full speed. Suppose m ($m \leq n$) processors are selected to be active while the rest are turned off. Without loss of generality, let the m active processors be $c_1, \dots, c_m$ running at speeds $s_1, s_2, \dots, s_m$, respectively. They are assigned non-zero workload ($\alpha_i \neq 0, \forall i, 1 \leq i \leq m$). By queueing theory [9], the average delay of the tasks on processor c_i can be calculated as:

$$T_i = \frac{1}{s_i\mu - \alpha_i\lambda}, \quad (4)$$

and the average delay of all the tasks in the entire system is:

$$T = \sum_{i=1}^m \alpha_i T_i = \sum_{i=1}^m \frac{\alpha_i}{s_i\mu - \alpha_i\lambda}. \quad (5)$$

As the power functions of the processors are identical, the total power consumption of the m processors can be computed as follows:

$$\begin{aligned} P &= \sum_{i=1}^m P_i(s_i) = \sum_{i=1}^m \frac{\alpha_i \lambda}{s_i \mu} \cdot A s_i^3 + m P_{static} \\ &= \frac{A \lambda}{\mu} \cdot \sum_{i=1}^m \alpha_i \cdot s_i^2 + m P_{static}. \end{aligned} \quad (6)$$

In equation (6), $\alpha_i \lambda / s_i \mu$ is the processor utilization, i.e., the portion of time the processor is processing tasks. The first and the second terms represent the contributions of the dynamic power and the static power, respectively. The objective is to minimize the average total power P . The constraint is that the average response time T must be kept below the given threshold D . Since A , λ , μ and P_{static} are fixed, we use a constant C to replace $\mu P_{static} / A \lambda$. The optimization problem is then formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^m \alpha_i \cdot s_i^2 + m \cdot C \quad (7)$$

$$\text{subject to} \quad \sum_{i=1}^m \frac{\alpha_i}{s_i \mu - \alpha_i \lambda} = D, \quad (8)$$

$$\sum_{i=1}^m \alpha_i = 1, \quad (9)$$

$$0 < \alpha_i, s_i \leq 1, \forall i, 1 \leq i \leq n. \quad (10)$$

For this equality constrained problem, the optimal solutions α_i and s_i must satisfy the following conditions:

$$2s_i \alpha_i - K_1 \frac{\alpha_i \mu}{(s_i \mu - \alpha_i \lambda)^2} = 0, (1 \leq i \leq m), \quad (11)$$

$$s_i^2 + K_1 \frac{s_i \mu}{(s_i \mu - \alpha_i \lambda)^2} + K_2 = 0, (1 \leq i \leq m), \quad (12)$$

where K_1 and K_2 are Lagrange multipliers. Combining the above equations we get $3s_i^2 = -K_2, \forall i = 1, 2, \dots, m$. This indicates all m active processors should run at the same speed. According to (12), the workload on each processor is also equal. Replacing α_i with $1/m$ in (8), we get:

$$s_i = \frac{\lambda}{m \mu} + \frac{1}{D \mu}.$$

The objective function becomes:

$$\sum_{i=1}^m \alpha_i \cdot s_i^2 + m \cdot C = \frac{1}{\mu^2} \left(\frac{\lambda}{m} + \frac{1}{D} \right)^2 + m \cdot C. \quad (13)$$

Increasing m minimizes the dynamic power (the first term), but increases the static power (the second term). Therefore, the problem becomes one of determining the optimal number of active processors. If the static power is 0 ($C = 0$), the total power is minimized when all n processors are active. Each processor should run at speed $s_i = \lambda / n \mu + 1 / D \mu$ and the total power dissipation is $A \lambda (\lambda / n + 1 / D)^2 / \mu^3 + n P_{static}$. In the more general case, when the static power is not negligible, increasing the number of

Figure 3: Root of equation (14). ($\lambda = 2, \mu = 1, P_{static} = 0.2, A = 1, D = 2$)

processors may result in higher energy consumption. The optimal value of formula (13) is achieved when its first derivative is equal to 0, i.e.,

$$C - \frac{2}{\mu^2} \cdot \left(\frac{\lambda}{m} + \frac{1}{D} \right) \cdot \frac{\lambda}{m^2} = 0. \quad (14)$$

As m is the only variable in the above cubic equation, a closed form solution exists and can be computed efficiently. Alternatively, the optimal value of m can be obtained by numerically evaluating (13) for all positive integers no larger than n .

Consider the example in Figure 3. The equation has only one positive root which lies between 5 and 6. The values of function (13) when m takes on the values of 5 and 6 are 1.31 and 1.29, respectively. Therefore the optimal number of processors is 6. The workload allocated to each processor and the delay constraint are $\lambda/6$ and D , respectively. Based on this result, the approach presented in Section 3 can be used to determine an optimal DVS policy for each processor.

4.2 Heterogeneous processors

Unlike multiprocessor systems, server clusters are often composed of computing nodes with different configurations and computation capabilities. As different processors may have different power profiles, finding the optimal voltage scaling policy is more complicated.

We introduce n binary variables $b_i \in \{0, 1\}$, where $b_i = 1$ if and only if processor c_i is active. Let $s_i^{\max}$, $P_{i,static}$ and A_i denote the maximum speed, static power and power coefficient of processor c_i ,

Initialization:

```

1    $\alpha = 1,$  /* Normalized total workload */
2    $\alpha_i = \frac{1}{\lambda} \cdot (s_i^{\max} \mu - \frac{1}{D}), \quad \forall i, 1 \leq i \leq n,$ 
3   Sort processors in increasing order of  $s_i^{\max}$ ,
4   for ( $i = 1, i \leq n, i++$ )
5        $\alpha_i = \min(\alpha_i, \alpha/(n - i + 1)),$ 
6        $\alpha = \alpha - \alpha_i,$ 
7        $t_i = D,$  /* Average delay on  $c_i$  */
8        $s_i = \frac{1}{\mu} \cdot (\alpha_i \lambda + \frac{1}{t_i}),$ 
9   end for
```

Figure 4: Pseudo-code of initial workload distribution.

respectively. The power optimization problem can be formulated as follows:

$$\begin{aligned}
 & \text{minimize} && \sum_{i=1}^n \frac{\alpha_i \lambda}{s_i \mu} \cdot b_i A_i s_i^3 + \sum_{i=1}^n b_i P_{i,static} \\
 & \text{subject to} && \sum_{i=1}^n \frac{\alpha_i b_i}{s_i \mu - \alpha_i \lambda} = D, \\
 & && \sum_{i=1}^n b_i \alpha_i = 1, \\
 & && 0 < \alpha_i \leq 1, \quad 0 < s_i \leq s_i^{\max}, \quad i = 1 \dots n.
 \end{aligned}$$

Applying standard linearization techniques [17], the above non-linear optimization problem can be transformed into a mixed integer linear programming (MILP) problem. Existing software packages such as LP-Solve are capable of solving the resulting MILP problem. However, solving a large-scale MILP problem is prohibitively time-consuming. Therefore, we propose an efficient iterative heuristic below.

The heuristic consists of two steps. Initially the workload is evenly distributed on all the processors, and the processor speeds are set to fixed values such that the average delay of tasks on each processor is equal to the delay constraint D . Then we iteratively adjust the active processor set, workload allocation and speed settings. Note that the algorithm is used to build the schedule before any task is actually allocated or processed. The pseudo-codes of the algorithms are given in Figures 4 and 5. We use t_i to denote the average delay constraint of the tasks on processor c_i .

In the initialization stage (Figure 4), the maximum workload each processor can support under delay constraint D is calculated (line 2). The processors are then sorted in increasing order of $s_i^{\max}$. In each step in the *for* loop, the residue workload α is allocated evenly among all the processors that have not been assigned any workload. The workload of each processor should not exceed the maximum workload the processor can support (line 5). Based on the allocated workload and delay constraint $t_i = D$, the processor speed s_i is calculated (line 8).

The iterative approach in Figure 5 improves the initial schedule. There are three ways to reduce the total power consumption:

1) Processor power-off (line 3). If processor c_i is turned off and its workload is allocated to the rest of the m active processors, the workload of each active processor c_j is increased by α_i/m . With the additional workload, the delay constraint t_j is changed to $(\alpha_j t_j + \alpha_i t_i/m)/(\alpha_j + \alpha_i/m)$. The processor

Main loop:

```

1  while true
2      Carry out one of the actions that maximizes power saving ( $\forall i, \forall j$  in
        the active processor set,  $i \neq j$ ):
3          1) Turn off  $c_i$ , distribute  $\alpha_i$  evenly to the other active processors,
4          2) Adjustment delay  $t_i = t_i + \Delta t / \alpha_i$ ,  $t_j = t_j - \Delta t / \alpha_j$ ,
5          3) Move workload  $\Delta \alpha$  from  $c_i$  to  $c_j$ ,  $t_j = (\alpha_j t_j + \Delta \alpha t_i) / (\alpha_j + \Delta \alpha)$ ,
6          Update  $\alpha_i, t_i$  and  $s_i$  of all affected processors,
7          if No change has been made
8              break;
9  end while

```

Figure 5: Pseudo-code of the iterative heuristic.

speed s_j needs to be recalculated using equation (4) based on the new values of α_j and t_j . We compare the power reduced by turning off one processor and the total power increase in the other processors. The processor resulting in maximum power saving will be turned off.

If turning off a processor will not result in any power saving, we resort to the following two approaches:

2) Delay adjustment (line 4). For each processor c_i , we calculate the change in power if the delay constraint is reduced or increased by $\Delta t / \alpha_i$. The processor that produces the maximum power reduction, say c_i , has the delay constraint t_i increased by $\Delta t / \alpha_i$. On the other hand, the processor causing the minimum power increase, c_j , has t_j reduced by $\Delta t / \alpha_j$. The speeds of both processors are recomputed according to the new delay constraint.

3) Workload adjustment (line 5). In each step, a portion $\Delta \alpha$ of the total workload is adjusted between two processors. Suppose c_i is the source processor and c_j is the destination. The delay constraint of c_j should be changed to $(\alpha_j t_j + \Delta \alpha t_i) / (\alpha_j + \Delta \alpha)$ to account for the new workload. After that, s_i and s_j are recomputed and the power dissipation is re-evaluated. The processor pair that maximizes power saving is selected to carry out this move.

In each iteration of the *while* loop in Figure 5, the above three actions are considered sequentially but only one is carried out. For example, 3) will be considered only if neither 1) nor 2) can reduce power. It is not difficult to verify that these actions do not change the system-wide average delay which is fixed at D during initialization. The algorithm terminates if no change can be made after considering all three possible actions.

5 Performance evaluation

In this section, we evaluate the performance of the proposed algorithms for multiple processor systems. The metric was the average power dissipation, which was the total energy consumption (including the energy dissipation at the idle state) divided by the simulated time span.

Two other algorithms are used for comparison. The baseline algorithm which we shall call Load Balance Full Speed (LBFS) balances the workload among all processors and runs every processor at its full speed. The second scheme, Load Balance Scaling (LBS), differs from LBFS in that static

Figure 6: Average power under different delay constraint ($\lambda = 2$).

voltage scaling is used after workload allocation. Under LBS, each processor runs at a reduced speed that maintains an average delay equal to the delay constraint. In fact, LBS has been introduced in Section 4.2 to produce the initial schedule for the iterative algorithm (see Figure 4).

5.1 Homogeneous systems

A multiprocessor system consisting of 8 Intel Pentium 4 processors at 2GHz was used in our experiments. The power dissipation of each processor running at full speed is 54.3W [1]. As static power usually accounts for 15% to 20% of the total power [6], we estimated $P_{static} = 12W$. The power coefficient A is therefore 42.3 from equation (1). We assumed the processor speed was adjustable at discrete steps of 200MHz from 0Hz to 2GHz.

The time unit in the experiments was set at 10ms. Task arrival was a Poisson process, and the arrival rate was varied to produce different workload intensity. The service times of the tasks were exponentially distributed with the service rate set at 1 per time slot on a processor running at full speed.

We simulated task processing for 24 hour (8.64×10^6 time units). For each workload and delay setting, 20 different experiments were conducted and the average result was taken. In the first set of experiments, a light workload with an arrival rate of 2 was used. The delay constraint was varied from 15ms to 100ms. Figure 6 shows the average power dissipation of the system under different algorithms. As LBFS always runs processors at full speed irrespective of the delay constraint, it consumed a constant power of 180W in the experiments. Taking advantage of voltage scaling, LBS consumed 108W when the delay constraint was relaxed to 100ms, a 40% power reduction compared to LBFS. As the processor speed was slowed down, the static power became significant. By turning some processors off, the optimal algorithm described in Section 4.1 further reduced power dissipation to less than 80W, an additional 20% saving compared to LBS.

We then fixed the delay constraint at 100ms and varied the workload from a light 0.5 to a moderate 5

Figure 7: Average power with different workload ($D = 100\text{ms}$).

arrivals per time slot. The results are illustrated in Figure 7. As the workload increased, the total energy for processing increased correspondingly, so did the average power dissipation. The power dissipation of LBS was close to that of LBFS when the workload was light due to the dominance of the static power. By maintaining a minimal set of active processors, the optimal algorithm consumed much less power. For all three schemes, the power dissipation increased with the workload, but in different ways. Under LBFS, the increase in workload incurred a linear increase in processor utilization. LBS, on the other hand, increased processor speed to handle the rising workload, so the power function is convex. Under the optimal algorithm, as more and more processors were turned on, the static power contributed to a large proportion of the power increase. The black dotted line in Figure 7 shows the number of active processors used in the optimal scheme. After the workload exceeded 4, all 8 processors were turned on. In this case, the optimal algorithm is equivalent to LBS.

5.2 Heterogeneous systems

In this set of experiments, we simulated a 32-processor cluster consisting of 3 types of processors. Their characteristics are listed in Table 1. The values of the maximum power are taken from the Intel Pentium 4 data sheet [1]. Note that the 2GHz processor has a lower maximum power than the 1.4G processor. This is because the 2GHz and 3GHz processors are based on the newer 0.13-micron technology, which improves power efficiency. The processor speed was assumed to be adjustable at discrete steps of 200MHz. The workload was generated in the same way as described in Section 5.1. A processor running at 2GHz had an average service rate of 1 task per time slot. The iterative algorithm proposed in Section 4.2 as well as LBFS and LBS were evaluated. In the iterative algorithm, we set $\Delta\alpha = 0.001$ and $\Delta t = 0.01$. First the workload allocation, fixed speed and delay constraints of each processor were computed. Then the optimal DVS policy obtained using the method described in Section 3 was applied to each individual processor.

We also consider a feedback driven DVS algorithm in our comparison. The algorithm measures the

Processor Type	1	2	3
Max Speed (Hz)	1.4G	2G	3G
Max Power (W)	55.3	54.3	81.9
Static Power (W)	8	12	16
Power Coefficient A	137.9	42.3	19.5
Number of Processors	8	8	16

Table 1: Characteristics of the processors in the cluster.

Figure 8: Average power under different delay constraint ($\lambda = 8$).

average delay every 1000 time units and strives to keep the delay below the delay constraint. If the constraint is violated, the speed of a randomly picked active processor will be increased by one level. If the speed is higher than 80% of the processor’s maximum speed, a non-active processor (if there is one) is turned on to run at its lowest speed. On the other hand, if the measured delay is lower than 95% of the delay constraint, the speed of an active processor is slowed down by one level. Furthermore, if the speed is reduced to lower than 20% of its full speed, the processor is turned off. Similar feedback driven approach have been used in [6] and [12], but their schemes perform voltage scaling according to processor utilization and do not consider any delay constraint.

The results are shown in Figures 8 and 9. The proposed iterative algorithm demonstrated excellent power saving capabilities and consistently outperformed LBFS and LBS under all workload considered. Unlike the case of homogeneous systems, balanced load distribution was not optimal when all processors were active due to the different power characteristics of the processors. For example, as the arrival rate reached 22 in the second set of experiments (Figure 9), all the 32 processors were turned on in the iterative algorithm (see the black dotted line). However, using workload and delay adjustment techniques, the iterative algorithm saved 10% power compared to LBS.

The proposed scheme is also more energy-efficient than the feedback-driven approach (see Figure 9), especially when the system was not saturated or close to saturation. When the system is saturated, our proposed scheme will keep all the processors on. A feedback-driven scheme may save power by dynamically turning off some processors. but frequently turning on/off processors also incurs energy

Figure 9: Average power with different workload ($D = 50\text{ms}$).

overhead, which was not accounted for in our simulation experiments. In addition to the generally better energy-saving capability, our proposed scheme provides average delay guarantee, while existing feedback-driven schemes offer best effort only in this respect.

The computational overhead of the iterative algorithm is low. For example, it took less than 1 second on a 2GHz Pentium 4 processor to compute the workload allocation and speed settings for a 128-processor system in phase one. In phase two, computing the optimal DVS policies took about 10 seconds. Thereafter, voltage scaling simply consists of looking up in a table for the speed level corresponding to the number of tasks currently assigned to the processor. Note that it is necessary to recompute the DVS policies only when the incoming workload distribution changes, which happens infrequently (typically of the order of hours).

6 Conclusion

Improving energy efficiency is not only important for mobile computing systems but also helpful in the server environment for reducing heat dissipation and alleviating the overheating problem. In this work, we have studied voltage scaling strategies for aperiodic tasks in three environments—single processor systems, multiprocessor systems and heterogeneous server clusters. We propose scheduling schemes to minimize the total system power subject to average delay constraints. Queueing models are used to analyze the average delay.

For single processor systems, we propose a dynamic voltage scaling algorithm that adjusts the processor speed according to the number of tasks in the system. The system is modeled as a discrete-time Markov chain. Voltage scaling under delay constraints is formulated as a stochastic constrained optimization problem and solved using a combination of Lagrange relaxation and the value iteration methods. Simulation experiments have shown that the dynamic speed policy obtained using this approach can save up to 40% of power compared to fixed speed voltage scaling.

In the multiprocessor and server cluster environments, some of the processors can be turned off

to save power. For the homogeneous case, we present a method to compute the optimal number of processors that should be kept active. Incoming workload is evenly balanced among all active processors, and the processor speeds are scaled to the same optimal level calculated from queueing analysis. For heterogeneous systems, we propose an algorithm that iteratively refines the set of active processors, workload distribution and speed settings. Both algorithms are effective in power minimization as shown in our experimental results. More importantly, the proposed algorithms analytically guarantee that the average delay is kept within the specified limits.

References

- [1] Intel Pentium 4 Processor Datasheet. <http://www.intel.com/design/Pentium4/documentation.htm>.
- [2] Wireless Intel SpeedStep Power Manager: Optimizing power consumption for the Intel PXA27x processor family, 2004. <http://www.intel.com/design/pca/applicationsprocessors/whitepaper%2F30057701.pdf>.
- [3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, December 2001.
- [4] H. Aydin and Q. Yang. Energy-responsiveness tradeoffs for real-time systems with mixed workload. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.
- [5] A. P. Chandrakasan, S. S. Sheng, and R. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27:473–484, April 1992.
- [6] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proceedings of the Workshop on Power Aware Computing Systems (PACS)*, February 2002.
- [7] Simcha Gochman, Ronny Ronen, Ittai Anati, Ariel Berkovits, Tsvika Kurts, Alon Naveh, Ali Saeed, Zeev Sperber, and Robert C. Valentine. The Intel Pentium M Processor: Microarchitecture and Performance. *Intel Technology Journal*, 7(2):21–36, May 2003.
- [8] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, November 1999.
- [9] Leonard Kleinrock. *Queueing Systems*, volume 1. John Wiley & Sons, 1975.
- [10] T. Pering, T. Burd, and R. W. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pages 76–81, August 1998.

- [11] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.
- [12] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2001.
- [13] L.I. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems*. Wiley, 1999.
- [14] L.I. Sennott. Computing average optimal constrained policies in stochastic dynamic programming. *Probability in the Engineering and Informational Sciences*, 15:103–133, 2001.
- [15] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, December 2003.
- [16] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of Symposium on Operating System Design and Implementation (OSDI)*, pages 13–23, November 1994.
- [17] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 1999.
- [18] F. Zhang and S. T. Chanson. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 235–245, December 2002.
- [19] F. Zhang and S. T. Chanson. Blocking-aware processor voltage scheduling for real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2), May 2004.

A Constrained stochastic optimization

To solve the following constrained optimization problem

$$\bar{E}_{\pi^*}(x) = \inf_{\pi \in \Delta} \bar{E}_{\pi}(x), \quad \forall x \in \mathbb{R}, \quad (15)$$

the method of Lagrange multipliers is employed to transform the problem into an unconstrained problem. Given policy π , the Lagrange relaxation of the original optimization problem can be defined as follows:

$$\begin{aligned} J_{\pi}(x, \Lambda) &= \limsup_{n \rightarrow \infty} \frac{1}{n} \mathbf{E} \left[\sum_{i=0}^{n-1} E(x^i, s^i) + \Lambda \sum_{i=0}^{n-1} D(x^i, s^i) \mid \pi, x^0 = x \right] \\ &\leq \bar{E}_{\pi}(x) + \Lambda \bar{D}_{\pi}(x), \quad \forall x \in \mathbb{R}, \end{aligned}$$

where $\Lambda \in [0, \infty)$ is the Lagrange multiplier. A policy π is called Λ -optimal if $J_{\pi}(x, \Lambda) = \inf_{\pi} J_{\pi}(x, \Lambda), \forall x \in \mathbb{R}$. According to [14], if we can find $\Lambda > 0$ and a Λ -optimal policy π such that $\bar{D}_{\pi}(x) = D, \forall x \in \mathbb{R}$ and $\bar{E}_{\pi}(x)$ is finite, then π solves the constrained optimization problem specified by (15).

Figure 10: An example plot of J_Λ .

Therefore, the problem can be solved by finding a Λ -optimal policy π that satisfies the delay constraint D . The following theorem suggests a possible iterative approach. Figure 10 gives a graphical interpretation of the theorem.

Theorem 1 [14] *If $\mathbb{R}$ is finite and $J(x, \Lambda) \equiv J(\Lambda), \forall \Lambda > 0$ and $\forall x \in \mathbb{R}$, then the following is true:*

1. $J(\Lambda)$ is an increasing, continuous, and concave function consisting of finitely many linear segments each with a non-negative intercept.
2. The unique breakpoints $0 = \Lambda_0 < \Lambda_1 < \dots < \Lambda_M$ define the $M+1$ intervals $[\Lambda_0, \Lambda_1], \dots, [\Lambda_{M-1}, \Lambda_M], [\Lambda_M, \infty)$, on each of which $J(\Lambda)$ is linear.
3. Let $J(\Lambda) = \bar{E}_j + \Lambda \bar{D}_j$, for $\Lambda \in [\Lambda_j, \Lambda_{j+1}]$. Then $\bar{D}_0 > \bar{D}_1 > \dots > \bar{D}_M$ and $\bar{E}_0 < \bar{E}_1 < \dots < \bar{E}_M$.
4. Assume that a stationary policy π is Λ -optimal for some $\Lambda \in [\Lambda_j, \Lambda_{j+1}]$, then $\bar{E}_\pi(x) = \bar{E}_j$ and $\bar{D}_\pi(x) = \bar{D}_j, \forall x \in \mathbb{R}$. Hence π is optimal for $\Lambda \in [\Lambda_j, \Lambda_{j+1}]$. In this case, π is said to be essential in $[\Lambda_j, \Lambda_{j+1}]$.

Based on the above theorem, the objective is to find an interval $[\Lambda_j, \Lambda_{j+1}]$ during which $\bar{D}_j = D$, where $\bar{D}_j$ ($0 \leq j \leq M$) is the slope of the function $J(\Lambda)$. The minimum energy consumption ($\bar{E}_j$) is the y-coordinate of the cross point with the y-axis by extending the linear function segment between $[\Lambda_j, \Lambda_{j+1}]$.

In the following we briefly introduce the procedures to solve the constrained optimization problem. As $\bar{D}_j$ monotonically decreases with Λ , an efficient bisection search method is used to find $\bar{D}_j$ close to the delay constraint D . A value iteration method [13] is used to determine an optimal stationary policy π for each value of Λ in the searching procedure. J_Λ and $\bar{D}_\Lambda$ under π are computed at the same time. If there exists a $\bar{D}_j = D$, then a stationary policy π essential (namely optimal) in $[\Lambda_j, \Lambda_{j+1}]$ is the optimal voltage scaling policy for the entire constrained optimization problem. As the value of $\bar{D}_j$ is not continuous, it is possible only $\bar{D}_j < D < \bar{D}_{j+1}$ can be found. In this case, the optimal policy can

be constructed by randomly switching between two policies π_j and π_{j+1} , which are essential in adjacent intervals $[\Lambda_j, \Lambda_{j+1}]$ and $[\Lambda_{j+1}, \Lambda_{j+2}]$, respectively. In particular, these two policies take different actions at only one state. Policy π_j shall be adopted with probability p and π_{j+1} with probability $1 - p$, where p satisfies equation $p\bar{D}_j + (1 - p)\bar{D}_{j+1} = D$.