This is the Pre-Published Version

Blocking-Aware Processor Voltage Scheduling for Real-Time Tasks *

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

As mobile computing is getting popular, there is a growing need for techniques that minimize energy consumption on battery-powered mobile devices. Processor voltage scheduling can effectively reduce processor energy consumption by lowering the processor speed. In this paper, we study voltage scheduling for real-time periodic tasks with non-preemptible sections. Three schemes are proposed: The static speed algorithm derives the minimum static feasible speed based on the Stack Resource Policy (SRP). Due to blocking, this static speed is usually higher than the speed required for scheduling fully preemptible tasks (called the utilization speed). Two dynamic speed algorithms are then introduced to further reduce energy consumption. The novel dual speed algorithm operates the processor at the utilization speed whenever possible and switches to the higher static speed only when blocking occurs. The dual speed dynamic reclaiming (DSDR) algorithm reserves time budget for each job, reclaims the unused time budget from completed jobs and redistributes it to subsequent jobs so they can run at a lower speed whenever possible. Feasibility conditions for real-time task sets have been derived and proved mathematically. Simulation results show that the proposed voltage scheduling algorithms dramatically reduce processor energy consumption over non-power-aware scheduling algorithms. Furthermore, the two dynamic speed algorithms consistently outperform the static speed scheme in a wide range of system and workload conditions.

keywords: Dynamic power management, non-preemptible sections, power-aware scheduling, real-time systems

*The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6178/03E).

1 Introduction

Wireless personal computing and communication devices are becoming increasingly popular. These devices include laptop computers, PDAs, and various wireless embedded systems. Modern mobile systems are often equipped with powerful processors and are capable of running sophisticated programs such as multimedia applications. Unfortunately, faster hardware usually results in faster energy drain. With battery technology not keeping up with the speed of the processors, the limited battery capacity has become a major concern. Techniques that utilize energy efficiently so as to prolong battery life have received much attention recently [5, 9, 14, 18, 20].

Much work exists on evaluating the power consumption of mobile computing systems [7, 12, 19]. It was found that the display (including backlight), processor, hard disk and wireless LAN card account for most of the power consumption. In particular, the processor may consume up to 25% of the power for laptop computers [12]. Since the processor may not be fully utilized all the time, processor voltage scaling (VS) allows the processor to operate at a lower speed when the system load is low, which has proved to be an effective way to reduce energy consumption [17]. Processors supporting dynamic speed adjustment are commercially available (e.g., the Crusoe processor of Transmeta and the Intel StrongARM processor). Basically, voltage scaling changes processor speed by varying the processor supply voltage. The relationship between processor power (P), supply voltage (V_s) and clock frequency (f) can be described by the following formula [6]:

$$P = C \cdot f \cdot V_s^2, \tag{1}$$

where C is the switched capacitance. Furthermore, the operating frequency f (and therefore the processor speed) is approximately proportional to V_s [9]. Therefore the processor power is proportional to the cubic of the supply voltage. For the same amount of work, a lower processor speed takes longer time to complete the job but consumes less energy.

Despite the increased processing delay, voltage scaling is of particular interest to real-time applications because longer processing time can be tolerated as long as the timing constraints are not violated. In practice, voltage scaling is usually implemented in the process scheduler, so we shall refer to processor scheduling with voltage scaling simply as "voltage scheduling" for short in the rest of this paper.

Most previous studies on voltage scheduling have assumed the tasks are fully preemptible [2,17,21,22]. However, in practice tasks may contain sections that are not preemptible. Take Linux for example, the standard Linux kernel is non-preemptive. At most one execution flow is allowed in the kernel space at any time. Therefore, although a user process is preemptible while executing in the user mode, when it invokes a system call (for example, `read()` or `write()`) and enters the kernel space, the process becomes non-preemptible and remains so until it returns to the user mode. In a recent study, it was found that the non-preemptible sections due to system calls involving disk I/O can be as long as 28ms in a standard Linux kernel [1]. Non-preemptible sections may also be enforced in systems where resources are shared among multiple processes. For example, the Non-Preemptible Critical Section Protocol controls resource

access by disabling preemption when a process is holding a shared resource [15]. As a process can be delayed by a low priority process executing in the non-preemptible section, assuming fully preemptible tasks in this case may cause deadline misses or result in incorrect computation.

In this work, voltage scheduling for periodic real-time tasks with non-preemptible sections is studied for dynamic priority scheduling under the earliest-deadline-first algorithm[1] (EDF) as well as for fixed priority scheduling under the rate-monotonic algorithm (RM). We shall show how to calculate a static speed by which all admitted tasks can be feasibly scheduled with minimal energy. A dynamic dual speed algorithm is also proposed to run the processor at an even lower speed in some intervals. Moreover, as the actual execution time is usually shorter than the declared worst-case execution time (WCET), a reservation-based scheme which dynamically reclaims processor time for redistribution is proposed to further reduce energy consumption. We shall show that the proposed algorithms can effectively reduce processor energy consumption without violating the time constraints. The two dynamic speed algorithms, in particular, can save a significant amount of energy compared with the static speed algorithm. Note that the proposed voltage scheduling algorithms determine the processor speed only, and they must be used together with an existing task scheduling algorithm which will decide the task execution sequence. We shall assume throughout this paper that EDF and RM are used for dynamic priority scheduling and fixed priority scheduling, respectively. For each proposed voltage scheduling scheme, we shall first present the underlying rationale and the general algorithm which are applicable to both types of scheduling. The specific issues related to EDF and RM are then discussed.

The rest of the paper is organized as follows: Section 2 reviews related research on voltage scheduling. The system models are introduced in Section 3. Section 4 derives the feasibility conditions and formulas for the static speed algorithm. Section 5 presents the two dynamic speed algorithms. Performance evaluation results are presented in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

Since the seminal paper by Weiser [21], much work has been done on voltage scheduling. Research in this area can be classified into two categories: interval-based scheduling [13, 16, 21] and profile-based scheduling [2, 9, 17, 22].

Interval-based scheduling estimates the processor utilization based on observations in the past interval(s). For example, the PAST algorithm records the processor utilization in the previous interval. If the utilization exceeded the upper threshold, the supply voltage is incremented; if the utilization was below the lower threshold, the supply voltage is decremented [21]. The weighted-AVG algorithm [16] makes use of the processor utilization in the previous interval as well as the average utilization in all past intervals. Thus prediction is based on both short-term and long-term system behaviors. Simulations using real-life traces were carried out to evaluate the performance of these algorithms. More recently, Lorch and Smith [13] proposed to use distributions to estimate the processing requirements of future

[1] An earlier work on EDF was presented in IEEE Real-Time System Symposiums, 2002 [23].

tasks from the recent behaviors of similar tasks. They suggested using the gamma model for its low complexity and good predicating capability.

Interval-based scheduling assumes the workload is more or less stable (or follows some distributions). The effectiveness in energy reduction greatly depends on the accuracy in predicting future system behaviors, which turns out to be very difficult to achieve [8]. Furthermore, interval-based scheduling algorithms are not suitable for real-time applications. As time constraints are not taken into consideration, these algorithms may improperly lower the processor speed and cause deadline misses.

Profile-based scheduling provides service guarantees for real-time applications and has received more attention recently. It assumes the characteristics of the tasks (e.g., period, deadline and execution time) are known. Yao et al. [22] proposed an optimal offline algorithm for scheduling aperiodic real-time tasks. An online approximation algorithm was also presented in the same paper. Hong et al. [9] used a similar planning-based approach to handle non-preemptible tasks.

The traditional real-time periodic task model is considered in some recent work. For dynamic priority scheduling, optimal static speeds can be found to schedule the tasks with minimum energy [2]. Dynamic speed algorithms utilizing processor slack time have also been proposed. The dynamic reclaiming algorithm (DRA) [2] reserves time for jobs to complete their WCETs at the optimal static speed. The time leftover by the jobs that do not reach their WCETs are reclaimed so subsequent jobs can utilize it and execute at lower speeds. Some greedy algorithms take into consideration the processing requirements in the future [17]. These algorithms delay the future jobs as much as possible so the current jobs can execute at the lowest speed possible.

All of the above studies on profile-based scheduling have assumed that tasks are either fully preemptible or completely non-preemptible. None of them is suitable for scheduling tasks containing non-preemptible sections because they may cause deadline misses or incur high scheduling overhead. In this paper, we propose scheduling schemes that maintain real-time guarantees while preserving the flexibility and low complexity as scheduling preemptible tasks.

3 System Model

3.1 Task Model

Real-time periodic tasks are considered in this paper. A periodic task consists of a sequence of jobs generated (or released) at fixed intervals (called the period). We denote the set of tasks by T. Each task $T_i \in T$ is characterized by four parameters:

- A_i: time the task is first released.
- D_i: relative deadline of the task, i.e., the time between a job's arrival time and its deadline.
- P_i: period of the task.
- E_i: worst-case execution time (WCET) of any job in the task.

In this paper, we adopt the common assumption that the relative deadline of a task is equal to its period. We say a job meets the deadline if the job is completed at or before its deadline, and it misses the deadline otherwise. The scheduling algorithms should guarantee no deadline is missed. A task $T_i = (J_{i,1}, J_{i,2}, J_{i,3}, \ldots, J_{i,n})$ consists of n jobs, where job $J_{i,j}$ is characterized by its release time $r_{i,j}$, the execution time $e_{i,j}$ ($\leq E_i$) and the deadline $d_{i,j}$. The execution time is defined as the time required to process the job at the processor's *maximum speed*. Furthermore, jobs are preemptible except when they are running in their non-preemptible sections called *blocking sections*. We say a job is *blocked* only if it cannot execute because a *lower* priority job is executing in its blocking section[2]. A job can have zero, one or more blocking sections, and G_i denotes the length of the longest blocking section of any job in task T_i. The blocking sections are randomly distributed within a job and are non-overlapping.

3.2 Processor Model

The processor is capable of dynamic voltage scaling and its speed is proportional to the supply voltage. The maximum and minimum possible supply voltages are denoted as $V_{\max}$ and $V_{\min}$, respectively, while the corresponding processor speeds are $S_{\max}$ and $S_{\min}$. The processor voltage can be adjusted in discrete steps within the range. Throughout this paper, we assume the processor's maximum speed is 1 and all other speeds are normalized with respect to the maximum speed. We further assume both the voltage transition delay and cost are negligible [17], and the voltage can be adjusted at any time (whether inside or outside a blocking section). We also assume the processor power follows formula (1), which in our case can be simplified to $P = A \cdot V_s^3$, where A is a constant.

4 Static Speed Scheduling

In this section we show how to find a static speed that minimizes energy consumption when processing periodic tasks with blocking sections. In a static speed scheme, the processor speed is changed only when a new task arrives or when an existing task terminates. The earliest-deadline-first (EDF) algorithm and the rate-monotonic (RM) algorithm are considered. Jobs with the same priority are serviced by the first-come-first-serve (FCFS) discipline.

In our task model, a higher priority job cannot start execution if another job is executing in its blocking section. This behavior is similar to the Stack Resource Policy (SRP) proposed by Baker to solve the general resources sharing problem [3]. The core idea of the SRP is that a job cannot start execution unless all the resources it needs are available. Scheduling tasks with blocking sections can be regarded as a special case of scheduling under the SRP, where there is only one non-preemptible shared resource, and every job needs to request the resource to run. If the job does not actually need the resource, which corresponds to a job without a blocking section in our task model, it is still required to request the resource for zero unit of time before it can start execution.

[2]The situation of a higher priority job preventing lower priority jobs from running has nothing to do with the blocking section and is not described as "blocking" in this paper.

The feasibility condition of the SRP with EDF is given below:

Theorem 1 *[3] Suppose n periodic tasks are sorted by their periods. They are schedulable by EDF with the SRP if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{D_i} + \frac{B_k}{D_k} \leq 1,$$

where B_k is the maximum length that a job in T_k can be blocked.

When voltage scaling is allowed, the static processor speed can be reduced according to Theorem 2. Note that we have replaced D_i with P_i since they have the same value in our task model (see Section 3.1).

Theorem 2 *Suppose n periodic tasks are sorted by their periods. They can be feasibly scheduled by EDF with the SRP at processor speed $H(0 < H \leq 1)$ if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{B_k}{P_k} \leq H,$$

where B_k is the maximum length that a job in T_k can be blocked.

Proof: Note that scheduling a task set T at processor speed H is equivalent to scheduling a task set T^* at the maximum processor speed where the execution times and resource holding times of T^* are $1/H$ times the corresponding values in T. Hence, the above inequality can be rewritten as

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i \cdot \frac{1}{H}}{P_i} + \frac{B_k \cdot \frac{1}{H}}{P_k} \leq 1.$$

Note that the maximum blocking time a job in T_i^* may encounter would also be scaled up by $1/H$ times. According to Theorem 1, T^* is schedulable by the SRP at full processor speed (=1), so the original task set T is schedulable at speed H. □

When blocking sections are considered, the maximum length of blocking (B_k) in Theorem 2 can be easily computed (see proof of Corollary 1).

Corollary 1 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by EDF at processor speed $H(0 < H \leq 1)$ if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H, \tag{2}$$

where G_j is the maximum length of any blocking section in T_j and $1 < j \leq n$.

Proof: Note that in EDF scheduling, a job can only be blocked by jobs in tasks with larger periods, so $\max\{G_j | P_k < P_j\}$ is the maximum time that any job in T_k can be blocked. □

The feasibility condition of the SRP with RM is as follows:

```
/* The algorithm finds the minimum processor speed
that can feasibly schedule a set of n tasks. */

Test_Speed( )
    H = 0;
    utilization = 0
    for i = 1 to n  /*(P_1 ≤ P_2 ≤ ... ≤ P_n)*/
        utilization += E_i/P_i;
        speed = utilization + max{G_j|P_i<P_j}/P_i;
        H = max(speed, H);
    end for
    return H;
```

Figure 1: The static speed algorithm for EDF.

Theorem 3 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by RM with the SRP if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{B_k}{P_k} \leq k \cdot (2^{1/k} - 1),$$

where B_k is the maximum length that a job in T_k can be blocked.

Similar to the EDF case, based on the feasibility condition for RM, we can calculate the feasible static speed according to the following corollary:

Corollary 2 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by RM at processor speed $H(0 < H \leq 1)$ if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \cdot k \cdot (2^{1/k} - 1) \quad (1 \leq j \leq n). \tag{3}$$

where G_j is the maximum length of any blocking section in T_j and $1 < j \leq n$.

The static speed H needs to be re-computed only when a task completes or when a new task arrives. Figure 1 shows the static speed algorithm under EDF. The algorithm under RM is similar and is omitted due to space constraint. The objective is to find a minimum value of H such that the inequalities are satisfied. If the return value (H) does not exceed 1, the newly arrived task is admitted and the processor speed is set to H; otherwise the new task is not admitted and the original static speed is maintained. If the processor speed can only be adjusted at discrete levels, then the lowest speed level that is greater than or equal to H is used.

The overhead of the algorithm is very low. By maintaining two task lists sorted respectively by the period and by the maximum length of the blocking sections, H can be computed in $O(n)$ time, where n is the number of tasks. Despite its simplicity, this static speed algorithm can effectively reduce energy consumption when the system load is low. In the next section, we shall introduce two dynamic speed algorithms which achieve even more energy saving at the expense of higher complexity.

5 Dynamic Speed Scheduling

The static speed algorithm in Section 4 guarantees the feasibility of the task set. However, the processor could be idle during some intervals in the computed schedule. These idle intervals come from two sources. First, the feasibility test is based on the WCETs and the maximum blocking lengths of the tasks. In reality, the actual execution times and blocking sections of individual jobs are usually shorter. Second and more importantly, given the feasibility condition, the total processor utilization of the admitted tasks is usually lower than the normalized static feasible processor speed H, so the processor is still under-utilized even if all the jobs use their WCETs. Utilizing these idle intervals, some of the running jobs can be processed at a lower speed so energy can be further reduced. However, under a static speed algorithm, further reducing the static speed below the value calculated in Section 4 may incur deadline misses. In this section, we present two new algorithms that dynamically slow down the processor in strategic intervals while still preserving the feasibility of the task set.

5.1 The Dual Speed Algorithm

Fully preemptible tasks can usually be feasibly scheduled at a static speed lower than the one calculated in the last section. For example, a set of tasks can be feasibly scheduled by EDF with a minimum static speed L if

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L.$$

Comparing with (2), it is easy to see $L \leq H$. We refer to L as the *utilization speed*, or simply the "low speed". In contrast, we call the static speed H in Section 4 the "high speed". We propose a dual speed algorithm that allows the processor to operate at speed L and switch to the high speed H only when blocking occurs. The rationale is clear: The high speed is only necessary to guarantee the time constraints of jobs that are blocked. When no job is blocked, the low speed would be adequate. When blocking occurs only infrequently, this strategy would operate the processor at the low speed most of the time and save a significant amount of energy.

The dual speed algorithm is presented in Section 5.1.1. The feasibility conditions of the dual speed algorithm with EDF and RM are given in Sections 5.1.2 and 5.1.3, respectively.

5.1.1 The Algorithm

The algorithm starts by running at the low speed L to process the released jobs according to their priorities. When a job J_i blocks a higher priority job J_k, it will not only delay the blocked job but also transitively delay subsequent higher priority jobs. The processor therefore must operate at the high speed to guarantee the timeliness of these delayed higher priority jobs. However, when a job with priority equal to or lower than the priority of job J_i is selected to run at time t, the processor can switch back to the low speed. This is because all jobs at time t would have the same or lower priority as J_i, so they are not blocked by J_i, directly or indirectly. Moreover, jobs released after time t will not be

```
/* H and L are recomputed by the static speed algorithm as a task joins or
leaves the system. Initially the processor speed is L. */

When job J_i arrives:
    if prt_i > prt_{current_job} /*The new job has a higher priority*/
        if Preempt_Current_Job() is successful
            Execute J_i;
        else /*J_i is blocked*/
            if bprt == -1 /*The first blocking*/
                Set_Speed(H); /*Set the processor speed at H*/
                bprt = prt_{current_job};
            end if
        end if
    end if

When job J_i in the queue is selected to run:
    if J_i == NULL || prt_i ≤ bprt
        bprt = -1;
        Set_Speed(L); /*Set the processor speed at L*/
    end if
    if J_i ≠ NULL
        Execute J_i;
    end if
```

Figure 2: The dual speed algorithm.

transitively delayed by jobs completed before t. Similarly, if all existing jobs are completed and the processor becomes idle, all jobs released thereafter will not be affected by the jobs released before the idle period. As a result, if a new job arrives when the processor is idle, this job can be processed at the low speed L.

The dual speed algorithm is formally presented in Figure 2. The following notations are used in the algorithm:

J_i: current job of task T_i[3].

prt_i: priority of job J_i.

$bprt$: priority of the blocking job in high speed intervals. Equal to -1 otherwise.

Initially the processor operates at speed L and $bprt = -1$. When a job is blocked, the processor speed is switched to H and $bprt$ is set to the priority of the blocking job. The high speed interval is terminated when a job with priority equal to or lower than $bprt$ is executed. Note that the speed change from L to H is necessary only when $bprt \neq -1$, i.e., outside any high speed intervals. Although blocking may also occur inside a high speed interval, the priority of the blocking jobs would be higher than $bprt$ (otherwise the current high speed interval would have already been terminated). In this case, changing $bprt$ to a higher value will not extend the current high speed interval.

[3]Since each task can only have one job present at any time, no ambiguity is introduced.

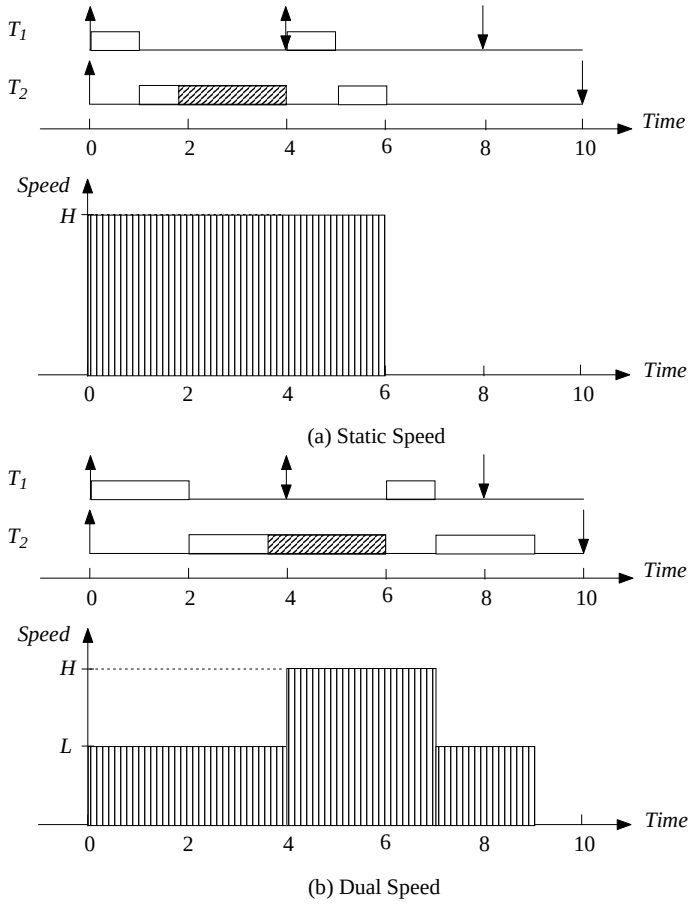

Figure 3: Comparison of static speed and dual speed algorithms under EDF.

In the dual speed scheme, the high speed and the low speed need to be re-calculated only when a task joins or leaves the system. The calculation of H and L takes $O(n)$ time where n is the number of tasks. After that, both procedures in Figure 2 take $O(1)$ time only. As H and L are not frequently changed, the overhead of the algorithm is minimal. The dual speed algorithm can be used with EDF or RM to schedule dynamic priority or fixed priority tasks, respectively. However, different high and low speeds are required for EDF and RM, as will be discussed later.

An example is given in Figure 3 to illustrate how energy is saved under the dual speed algorithm compared with the static speed algorithm. The up and down arrows denote the arrival times and deadlines of the jobs, respectively. The white boxes indicate job execution intervals, while the shaded boxes indicate blocking sections. EDF is used to schedule the tasks in this example. The two jobs in T_1 both need 1 time unit to finish at the high speed H while the job in T_2 needs 4 time units. Suppose the low speed L is half the value of the high speed. Under the static speed algorithm, the processor runs at speed H throughout interval $[0, 6]$ (Figure 3a). Under the dual speed algorithm, the processor runs at speed L before time point 4 at which blocking occurs (Figure 3b), and after time point 7 when the blocked job is completed. Based on formula (1), the dual speed algorithm saves about 38 percent

of energy compared with the case of static speed.

5.1.2 Feasibility under Dynamic Priority Scheduling (EDF)

Although the dual speed algorithm reduces the processor speed in some intervals, the feasibility of the task set is maintained. The following theorem guarantees that under EDF scheduling if a task set is schedulable by the static speed algorithm, it is also schedulable by the dual speed algorithm. Furthermore, the energy consumption under the dual speed algorithm can never be higher than that under the static speed algorithm.

Theorem 4 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by the dual speed EDF algorithm with high speed H and low speed L if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \tag{4}$$

and

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L. \tag{5}$$

Proof: We shall prove the theorem by contradiction.

In EDF scheduling, priorities are assigned to jobs based on their deadlines, and a earlier deadline indicates a higher priority. For clearer presentation, we use deadlines instead of priorities in the following proof.

Suppose the claim is false and t is the earliest time that a job misses its deadline. We find a time point t_1 before t which is the latest time point such that no active job arrived before t_1 has a deadline at or before t. If t_1 does not exist, we let $t_1 = 0$. Constructed in this way, the processor is never idle during $(t_1, t]$ and only two categories of jobs can execute in the interval. The jobs in the first category, denoted by N, are released after t_1 and have deadlines at or before t. The second category, if exists, consists of a single job, denoted by J_m, which has a deadline after t and is executing in its blocking section at time t_1.

We consider two cases: only jobs in N are executed during $(t_1, t]$, and both J_m and jobs in N are executed during the interval.

Let $X = t - t_1$. In the first case, because all tasks are periodic, the processor demand generated by the jobs in N is bounded by $\sum_{i=1}^{n} \lfloor X/P_i \rfloor \cdot E_i$. On the other hand, as the processor is never idle and its speed is greater than or equal to L in the entire interval, the processor demand that can be handled is at least $L \cdot X$. Since a job misses its deadline at time t by assumption, the total processor demand in the interval must exceed what the processor can handle in the same interval. Therefore

$$\sum_{i=1}^{n} \left\lfloor \frac{X}{P_i} \right\rfloor \cdot E_i > L \cdot X.$$

Since $X/P_i \geq \lfloor X/P_i \rfloor$, we have

$$\sum_{i=1}^{n} \frac{X}{P_i} \cdot E_i \geq \sum_{i=1}^{n} \left\lfloor \frac{X}{P_i} \right\rfloor \cdot E_i > L \cdot X \Rightarrow \sum_{i=1}^{n} \frac{E_i}{P_i} > L,$$

which contradicts with (5).

In the second case, the processor demand during $(t_1, t]$ is larger than that in the first case due to the execution of J_m. However, since J_m will no longer be executed once it leaves its blocking section, the total time that J_m can execute during $(t_1, t]$ cannot exceed its longest blocking section, i.e., G_m. Therefore the total processor demand is bounded by $G_m + \sum_{i=1}^{l} \lfloor X/P_i \rfloor \cdot E_i$, where P_l is the longest period that is smaller than or equal to X. As the deadline of J_m is later than t and all jobs in N have deadlines earlier than or equal to t, the processor must have been operating at speed H throughout the whole interval $(t_1, t]$, therefore the amount of work processed is $X \cdot H$. If there is a deadline miss at time t, the processor demand must be greater than the work processed, i.e.,

$$G_m + \sum_{i=1}^{l} \left\lfloor \frac{X}{P_i} \right\rfloor \cdot E_i > X \cdot H.$$

Since $P_l < P_m \Rightarrow G_m \leq \max\{G_j | P_l < P_j\}$, we have

$$\max\{G_j | P_l < P_j\} + \sum_{i=1}^{l} \frac{X}{P_i} \cdot E_i > X \cdot H.$$

Because $P_l \leq X$, we get

$$\sum_{i=1}^{l} \frac{E_i}{P_i} + \frac{\max\{G_j | P_l < P_j\}}{P_l} > H,$$

which contradicts with (4). □

5.1.3 Feasibility under Fixed Priority Scheduling (RM)

Similarly, the schedulability conditions for the dual speed algorithm scheduling fixed priority tasks can be derived as given in Section 5.1.2. When the dual speed algorithm is used with the rate-monotonic (RM) algorithm, a given task set is schedulable if the high and low speeds satisfy the following conditions:

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L \cdot n \cdot (2^{1/n} - 1),$$

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \cdot k \cdot (2^{1/k} - 1) \quad (k < j \leq n).$$

The feasibility of the dual speed algorithm can be proved as follows:

Lemma 1 *Suppose n periodic tasks are sorted by their periods. If*

$$\sum_{i=1}^{k} \frac{E_i}{P_i} \leq L \cdot k \cdot (2^{1/k} - 1), \tag{6}$$

where $1 \leq k \leq n$, *then* $\exists t \in U = \{m \cdot P_i | i = 1, \ldots, k; m = 1, \ldots, \lfloor P_k/P_i \rfloor\}$ *such that*

$$\sum_{i=1}^{k} E_i \cdot \lceil t/P_i \rceil \leq L \cdot t. \tag{7}$$

Proof: The proof is obtained by noticing that expression (6) is a sufficient condition for the k-th task to be schedulable [11] when all tasks are fully preemptible, while expression (7) is a sufficient and necessary condition [10]. □

Lemma 2 *Suppose n periodic tasks with blocking sections are sorted by their periods. If*

$$\sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \cdot k \cdot (2^{1/k} - 1) \quad (k < j \leq n),$$

where $1 \leq k \leq n$, *then* $\exists t \in U = \{m \cdot P_i | i = 1, \ldots, k; m = 1, \ldots, \lfloor P_k/P_i \rfloor\}$ *such that*

$$\sum_{i=1}^{k} E_i \cdot \lceil t/P_i \rceil + \max\{G_j | P_k < P_j\} \leq H \cdot t.$$

Proof: The proof is similar to that given in Lemma 1 by replacing the WCET of task T_k from E_k to $E_k + \max\{G_j | P_k < P_j\}$. □

Theorem 5 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by the dual speed RM algorithm with high speed H and low speed L if* $\forall k, 1 \leq k \leq n$, $\exists t_a, t_b \in U = \{m \cdot P_i | i = 1, \ldots, k; m = 1, \ldots, \lfloor P_k/P_i \rfloor\}$ *such that*

$$\sum_{i=1}^{k} E_i \cdot \lceil t_a/P_i \rceil \leq L \cdot t_a \tag{8}$$

and

$$\sum_{i=1}^{k} E_i \cdot \lceil t_b/P_i \rceil + \max\{G_j | P_k < P_j\} \leq H \cdot t_b. \tag{9}$$

Proof: We shall prove this theorem by contradiction.

Suppose the claim is false and t is the earliest time point that a job, say J_k with priority level prt_k misses its deadline. We find another time t_1 before t which is the latest time point such that there is no job available with a priority higher than or equal to prt_k. If t_1 does not exist, we let $t_1 = 0$. Constructed in this way, the processor is never idle during $(t_1, t]$ and only two categories of jobs can be executed in the interval. The jobs in the first category, denoted by N, are released after t_1 and have priorities higher than or equal to prt_k. The second category, if exists, consists of a single job, denoted by J_m, whose priority is lower than prt_k and it is executing in its blocking section at time t_1.

We consider two cases: i) only jobs in N are executed, and ii) both J_m and jobs in N are executed. Following similar approach for the proof of Theorem 4 and analyzing the processor demand generated and processed during $(t_1, t]$, these two cases can be found to contradict with expressions (8) and (9), respectively. □

Corollary 3 *Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by the dual speed RM algorithm with high speed H and low speed L if*

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L \cdot n \cdot (2^{1/n} - 1), \tag{10}$$

and

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \cdot k \cdot (2^{1/k} - 1) \quad (k < j \leq n). \tag{11}$$

Proof: The proof directly follows from Lemma 1, Lemma 2 and Theorem 5. □

Both Theorem 5 and Corollary 3 provide means to find feasible high and low speeds to use with the dual speed algorithm. Theorem 5 produces a tighter bound (i.e., lower speed), but the complexity of computing the speeds is high when the number of tasks is large. Corollary 3, on the contrary, provides a simpler way to calculate H and L, but the resulting speeds may be higher than those found by Theorem 5. In practice, the choice could be made based on the number of concurrent tasks and the number of jobs each task may contain. If the number of tasks is small while each task contains a large number of jobs (e.g., playing back a movie), preference can be given to Theorem 5; otherwise the speeds can be calculated by the formulas in Corollary 3.

In the dual speed algorithm, the processor speed is always set to H when blocking occurs, independent of the priority of the blocking job and the length of the blocking section. In fact, a lower speed may be adequate if we take these two factors into consideration. Take RM scheduling for example, if the blocking job belongs to T_b, then speed H' can be used instead of H, where

$$\forall k, 1 \leq k < b, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{G_b}{P_k} \leq H' \cdot k \cdot (2^{1/k} - 1).$$

The algorithm then becomes a multi-speed scheme with n possible speeds depending on the possible length of blocking. Unlike the dual speed scheme, the algorithm must maintain nested high speed intervals because they may have different speed requirements, that is, the inner levels require higher speeds. The processor speed therefore needs to be dynamically changed when the system steps into or out of a higher speed interval.

We give a simple example to illustrate this concept. Consider a task set with three tasks. The characteristics of the tasks $(A_i, D_i, P_i, E_i, G_i)$ are respectively $T_1 = (3, 5, 5, 1, 0)$, $T_2 = (1, 10, 10, 2, 2)$ and $T_3 = (0, 20, 20, 1, 1)$. The low speed $L \approx 0.58$. The high speeds are $H_2 = 0.77$ if the blocking job is in T_2 and $H_3 = 0.64$ if the blocking job is in T_3. The task execution is shown in Figure 4. In the

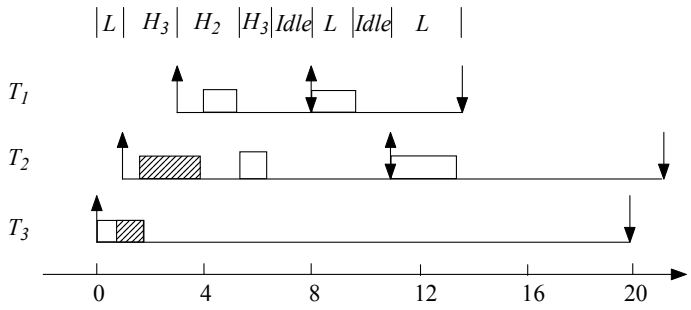

Figure 4: An illustration of nested speed levels.

example, T_3 executes at the low speed until T_2 arrives and is blocked. The processor speed is then increased to H_3. T_1 arrives later and is blocked by T_2, requiring an even higher processor speed H_2. When the first job in T_1 completes, the processor speed is lowered to H_3. During the execution of the second jobs in T_1 and T_2, the processor runs at the low speed L because there is no blocking.

5.2 The Dynamic Reclaiming Algorithm

The speeds in the dual speed algorithms are calculated based on the WCETs of the tasks. The actual processing demand is often lower. When a job completes early, the subsequent jobs can be allowed to run longer without violating their deadlines. Therefore the corresponding processing speed can be further reduced to save energy. We present a reservation-based scheme that allocates time budget to jobs. The time budget specifies the wall clock time that a job can execute, so the time allocated to each job is guaranteed. A dynamic reclaiming mechanism collects the residue time budget from early completions for redistribution. Since this algorithm is an extension of the dual speed algorithm, it is called the Dual Speed Dynamic Reclaiming (DSDR) algorithm. Aydin et al. [2] proposed a similar approach for fully preemptible tasks, but their algorithm is not applicable when blocking sections are present.

5.2.1 The Algorithm

First, we use the term *time budget* (denoted by R) to specify the wall clock time that a job is allowed to execute on the processor. The time budget is computed (to be described shortly) and assigned to a job when it arrives, and is consumed as the job executes. In addition, the assigned time budget has a priority equal to the priority of the job. The *execution time* (denoted by E) of a job, on the other hand, describes the workload imposed by the job. It is defined as the required time to complete the job if the processor runs at its maximum speed. As the processor speed may change in actual execution, the time budget may not be the same as the execution time. In DSDR, a job will always complete its execution before its time budget is depleted. Therefore, given the time budget and the execution time of a job, the processor should operate at speed $S = E/(S_{\max} \cdot R)$, where $S_{\max}$ is the maximum processor speed.

Just like the dual speed algorithm, in DSDR, the system also enters a high speed interval when

blocking occurs. However, the high speed interval in DSDR only affects the computation of the time budget but does not determine the processor speed. Furthermore, the high speed interval will terminate only when all the time budget with priority higher than that of the blocking job has been consumed. We introduce a new term called "base speed" in DSDR. The base speed simulates the processor speed under the dual speed algorithm, and is used to determine the time budget an arriving job will receive. Specifically, a job is allocated the amount of time budget that enables it to complete its WCET at the base speed before depleting its time budget. If a job is released in a high speed interval and has a priority higher than that of the blocking job, it assumes the base speed H and its time budget is $R = E/H$. Otherwise it assumes the base speed L and has budget $R = E/L$. Just as in the dual speed algorithm, H and L are recomputed on task arrivals and departures.

A Free Time Budget list called the FTB-list keeps track of the unconsumed time budget. Each item in the FTB-list contains two values: the amount of time budget and its priority. The list is sorted in decreasing order of the time budget's priority. The free time budget comes from two sources. First, when a job completes, its residue time budget is sent to the list. Second, if the base speed of a job is H, the job will receive less time budget than if the base speed is L. The time budget thus saved is also inserted into the FTB-list with priority *bprt* (the priority of the blocking job that started the high speed interval). The reclaimed time budget in the FTB-list will be used to allow some other jobs to run at a lower speed and therefore saves energy.

Before formally presenting the algorithm, we first introduce some additional notations:

$R_i^r(t)$: time budget of job J_i at time t.

$R_i^F(t)$: total amount of time budget in the FTB-list that job J_i is eligible to at time t.

E_i : worst-case execution time of job J_i under the maximum speed $S_{\max}(= 1)$.

$E_i^r(t)$: worst-case residue execution time of job J_i under $S_{\max}$ at time t.

The core algorithm is presented in Figure 5. H, L, prt_i and *bprt* have the same meanings and initial values as in the dual speed algorithm. When a job is scheduled to run, it is eligible to use its own time budget as well as the time budget in the FTB-list with a higher or equal priority. The actual processor speed is calculated according to the usable time budget $R_i^F(t) + R_i^r(t)$ and the worst-case residue execution time $E_i^r(t)$ of the job that is scheduled to run. However, when blocking occurs, the processor speed is always forced to H during that blocking section in order to minimize the blocking time. To complete the DSDR algorithm, the following rules are used to update the time budget and the worst-case residue execution time of a job as well as the time budget in the FTB-list:

1. As a job executes, its eligible free time budget is consumed at the same speed as the wall clock starting from the front of the FTB-list. Its own time budget will be consumed after the eligible free budget is depleted. $E_i^r(t)$ is decremented by the processor speed per unit time.

When a new job (J_i) arrives:
    $E_i^r(t) = E_i$;
    if $bprt \neq -1$ && $prt_i > bprt$ /*Job released in a high speed interval*/
        $R_i^r(t) = E_i/H$;
        Insert $(E_i/L - E_i/H)$ to the FTB-list with priority $bprt$;
        /*Reclaim unneeded time budget due to running at high speed H*/
    else
        $R_i^r(t) = E_i/L$;
    end if
    if $prt_i > prt_{current_job}$
        if Preempt_Current_Job() is successful
            Select J_i to run;
        else if $bprt == -1$ /*J_i is blocked*/
            Set_Speed(H); /*Force the speed to H in the blocking section*/
            $bprt = prt_{current_job}$;
        end if
    end if

When job J_i is selected to run:
    Set_Speed($\frac{E_i^r(t)}{R_i^F(t)+R_i^r(t)}$); /*Calculate the appropriate processor speed*/
    Execute J_i;

When job J_i is in execution:
    Set $bprt = -1$ if the priority of the budget being consumed is lower than or equal to $bprt$;

When job J_i completes:
    if $R_i^r(t) \neq 0$
        Insert $R_i^r(t)$ to the FTB-list; /*Early completion reclamation*/
    end if

Figure 5: The dual speed dynamic reclaiming (DSDR) algorithm.

2. When the processor is idle, the time budget in the FTB-list is consumed at the same speed as the wall clock unless the FTB-list is empty.

3. When a new *task* arrives at time t, the time budget of every job is set to $E_i^r(t)/H^*$, where H^* is the new high speed value. The FTB-list is then cleared and the free time budget in the list is lost. As new task arrivals are infrequent, the impact of time budget loss caused thereby is minimal.

The above rules do not need to be carried out at every time unit. Instead, Rule 1 is applied only when the current job completes, blocks another job, or is preempted. Rule 2 is used only if a new job arrives when the processor is idle.

5.2.2 Feasibility under Dynamic Priority Scheduling (EDF)

As an extension of the dual speed algorithm, DSDR maintains the same feasibility conditions. Before proving this, we first prove that under DSDR a job is always completed before its time budget is depleted.

Lemma 3 *If a task set is scheduled by DSDR, every job will complete before depleting its time budget.*

Proof: Take any interval in which job J_i is continuously executing. Suppose t is the time J_i starts execution and t' is the end of execution. According to the DSDR algorithm (Figure 5), at time $t+\Delta t$ ($0 \leq \Delta t \leq t' - t \leq R_i^r(t) + R_i^F(t)$), the job's worst-case residue execution time and its time budget are respectively:

$$E_i^r(t + \Delta t) = E_i^r(t) - \frac{E_i^r(t)}{R_i^F(t) + R_i^r(t)} \cdot \Delta t,$$

and

$$R_i^r(t + \Delta t) = \begin{cases} R_i^r(t) & \Delta t \leq R_i^F(t) \\ R_i^r(t) - (\Delta t - R_i^F(t)) & \Delta t > R_i^F(t) \end{cases}. \tag{12}$$

We can further prove that $\forall \Delta t (0 \leq \Delta t \leq R_i^r(t) + R_i^F(t))$,

$$\frac{R_i^r(t + \Delta t)}{E_i^r(t + \Delta t)} \geq \frac{R_i^r(t)}{E_i^r(t)}, \tag{13}$$

which means the ratio between R_i^r and E_i^r will not decrease after any period of execution. Since the initial ratio is at least $1/H$, a job will not exhaust its own budget before it completes. □

The feasibility conditions of DSDR with EDF are given and proved as follows:

Theorem 6 *Suppose n periodic tasks with blocking sections are sorted by their periods. If*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H, \tag{14}$$

where $k < j \leq n$, and

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L, \tag{15}$$

then all jobs can be feasibly scheduled when DSDR/EDF is used with high speed H and low speed L.

Proof: By Lemma 3, all jobs complete before their budget is depleted. Therefore, in order to prove that all jobs meet their deadlines, we only need to prove that budget is always consumed before its deadline. The latter is proved by contradiction as follows.

Suppose the claim is not true. Let t be the earliest instant that the time budget of job J_k is not depleted at its deadline. We choose another time t_1 before t, which is the latest time point such that no pending jobs arrived before t_1 has a deadline at or before t, and the FTB-list does not contain any budget item whose deadline is at or before t. If such a time point does not exist, let $t_1 = 0$. Constructed in this way, the processor never stops consuming time budget throughout the interval $(t_1, t]$. Furthermore, the time budget consumed during the interval is either generated after t_1 and has a deadline before t (denoted as time budget A), or has a deadline after t (denoted as time budget B).

We consider two cases. In the first case, only the time budget in A is consumed. Note that time budget is only generated on job releases. Let $Y = t - t_1$, the total amount of time budget in A is

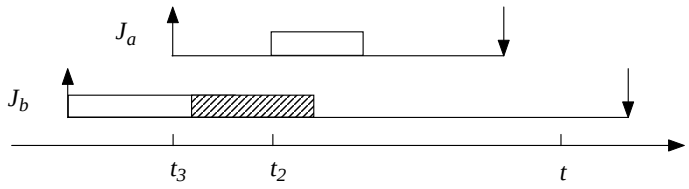

Figure 6: An illustration showing when the time budget in B is consumed.

bounded by $\sum_{i=1}^{n} \lfloor Y/P_i \rfloor \cdot E_i/L$. As there is still time budget left at time t, the amount of time budget in A must be greater than the time budget consumed in the interval, which is Y. Therefore

$$\sum_{i=1}^{n} \left\lfloor \frac{Y}{P_i} \right\rfloor \cdot \frac{E_i}{L} > Y,$$

which contradicts with (15).

In the second case, the time budget in both A and B is consumed in the interval. Unlike the situation in the proof of Theorem 4, the time budget in B may be consumed anywhere in $(t_1, t]$ instead of only at the beginning of the interval. This scenario is illustrated in Figure 6. J_b, with a deadline later than t, starts by consuming the time budget in the FTB-list with deadlines earlier than or equal to t. It depletes the time budget in A after entering its blocking section and starts to consume the time budget in B. In this case, we choose a third time t_2 which is the latest time point before t such that the time budget being consumed has deadline later than t. Since the time budget in B is consumed in the interval $(t_1, t]$, t_2 must exist and $t_1 < t_2 < t$. Before the time budget in the FTB-list is depleted, at least a job with a deadline equal to t or earlier must have arrived and is blocked; otherwise J_b would have been preempted before t_2 or the assumption $t_1 < t_2$ would be violated. Furthermore, there is no time budget in the FTB-list whose deadline is earlier than or equal to t at time t_2. Let J_a be the first job that is blocked by J_b. We use t_3 to denote its arrival time. In the interval $(t_3, t_2]$, the time budget consumed (by J_b) is generated before time t_3. However, in the subsequent interval $(t_2, t]$ only the time budget generated in the period $(t_3, t]$ and whose deadline is earlier than or equal to t is consumed.

Let $Z = t - t_3$. Since the base speed throughout $(t_3, t]$ is H, the amount of time budget generated after t_3 which has a deadline at or before t is bounded by $\sum_{i=1}^{l} \lfloor Z/P_i \rfloor \cdot E_i/H$, where P_l is the maximum period that is smaller than or equal to Z. Moreover, as the processor operates at speed H throughout $(t_3, t_2]$ according to DSDR, the total amount of time budget that can be consumed during $(t_3, t]$ is bounded by $G_b/H + \sum_{i=1}^{l} \lfloor Z/P_i \rfloor \cdot E_i/H$, where G_b is the length of the longest blocking section in J_b. As there is still residue time budget at time t whose deadline is at t and the processor keeps consuming time budget throughout the interval $(t_3, t]$, we have

$$\frac{G_b}{H} + \sum_{i=1}^{l} \left\lfloor \frac{Z}{P_i} \right\rfloor \cdot \frac{E_i}{H} > Z. \tag{16}$$

Because $Z \geq P_l$ and $Z/P_i \geq \lfloor Z/P_i \rfloor$, (16) can be rewritten as

$$\frac{G_b}{P_l} + \sum_{i=1}^{l} \frac{E_i}{P_i} > H,$$

which contradicts with (14). □

As proved in Theorem 6, DSDR maintains the same feasibility condition as the dual speed algorithm under EDF. The same conclusion holds under the rate-monotonic (RM) algorithm for fixed priority scheduling, which will be presented in the following subsection.

5.2.3 Feasibility under Fixed Priority Scheduling (RM)

We assume RM is used with DSDR for fixed priority scheduling. The feasibility conditions are given and proved as follows:

Theorem 7 *Suppose n periodic tasks with blocking sections are sorted by their periods. If* $\forall k, 1 \leq k \leq n$, $\exists t_a, t_b \in U = \{m \cdot P_i | i = 1, \ldots, k; m = 1, \ldots, \lfloor P_k/P_i \rfloor\}$ *such that*

$$\sum_{i=1}^{k} E_i \cdot \lceil t_a/P_i \rceil \leq L \cdot t_a \tag{17}$$

and

$$\sum_{i=1}^{k} E_i \cdot \lceil t_b/P_i \rceil + \max\{G_j | P_k < P_j\} \leq H \cdot t_b, \tag{18}$$

then these tasks can be feasibly scheduled when DSDR/RM is used with high speed H and low speed L.

Proof: We shall prove the theorem by contradiction.

Suppose the claim is not true and t is the earliest time point that a job with priority level prt_k misses its deadline. Let us denote this job by $J_{k,j}$. We choose another time t_1 before t, which is the latest time point such that there is no job or any time budget in the FTB-list with priority higher than or equal to prt_k. If such a time point does not exist, let $t_1 = 0$. Constructed in this way, time budget is continuously consumed throughout the interval $(t_1, t]$. The time budget consumed can be classified into two categories, A and B. The time budget in A is generated after t_1 and has a priority higher than or equal to prt_k, while the time budget in B has a priority lower than prt_k.

We consider two cases: only the time budget in A is consumed, and the time budget in both categories is consumed. Similar to the proof of Theorem 6, these two cases are found to contradict with (17) and (18) respectively after analyzing the amount of budget generated and consumed in the interval. This part of the proof is therefore omitted due to space limitation. □

Based on Theorem 7, the following corollary can be derived:

Corollary 4 *Suppose n periodic tasks with blocking sections are sorted by their periods. If*

$$\sum_{i=1}^{n} \frac{E_i}{P_i} \leq L \cdot n \cdot (2^{1/n} - 1),$$

Task Type	Short Tasks	Medium Tasks	Long Tasks
Period	20∼100ms	100∼1000ms	1000∼5000ms
WCET	1∼20ms	1∼100ms	1∼1000ms

Table 1: Characteristics of the three types of tasks.

and

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i}{P_i} + \frac{\max\{G_j | P_k < P_j\}}{P_k} \leq H \cdot k \cdot (2^{1/k} - 1) \quad (k < j \leq n)$$

where G_j is the length of the longest blocking section in task T_j, then these tasks can be feasibly scheduled by DSDR/RM with high speed H and low speed L.

6 Performance Evaluation

Simulation experiments were carried out to evaluate the effectiveness of the proposed algorithms in saving energy. In section 6.1, we first describe the simulation model and the task sets. The simulation results are then presented and analyzed in Section 6.2.

6.1 Simulation Setup

The simulator is event-driven and simulates the processing of periodic tasks on a single processor that is capable of voltage/speed scheduling. We assume a maximum processor speed of 1 and a minimum processor speed of 0.05. Speed levels between the two bounds are discrete and spaced by 0.05. The processor speed is proportional to the supply voltage and the energy consumption follows formula (1). We also assume the processor does not consume energy when it is idle. The simulation experiments consisted of three phases: task generation, admission control, and job scheduling/execution.

Each task set generated was associated with a *utilization factor* and a *blocking ratio*. The utilization factor indicates the expected total utilization of the tasks generated, that is, $\sum_{i=1}^{n} E_i/P_i$. The blocking ratio specifies the maximum proportion of a blocking section in a job's execution time, so the maximum length of any blocking section is $WCET \times blocking_ratio$. To simulate the mixed workload, three types of tasks were generated. Their characteristics are presented in Table 1. Tasks in each task set had equal probability of being short, medium or long. Within each category, the task periods and WCETs were randomly selected from the corresponding ranges. After the task set was generated, the WCETs of the tasks were scaled such that the total utilization was equal to the specified utilization factor.

Every generated task had to undergo an admission control procedure in the order of its release time. A new task was admitted if a high speed H smaller than the maximum speed could be found satisfying formula (2) for dynamic priority scheduling or formula (9) for fixed priority scheduling; otherwise the task is discarded. The admitted tasks generated jobs periodically according to their periods. Each task was further assigned a best-case execution time (BCET). The actual execution times of the jobs in a task

were uniformly distributed between the task's BCET and WCET. In each experiment a WCET/BCET ratio was specified which was used by all tasks. Each job could have at most one blocking section, and we use the term *blocking probability* to denote the probability that a job has a blocking section. The position of the blocking section was randomly chosen. Released jobs were placed in the job queue and processed according to their priorities (calculated according to EDF or RM). The voltage scheduling algorithms presented in Sections 4 and 5 were used to determine the processor speed for the current executing job.

6.2 Experimental Results

Extensive simulation experiments were conducted to evaluate the performance of the proposed algorithms. In each experiment, 30 tasks were generated. All tasks were released at time 0 and each experiment ran 100,000ms of simulated execution time. Jobs whose deadlines were later than 100,000ms were not processed at all. This guaranteed that the workload processed by the algorithms under evaluation (see below) was exactly the same. To improve accuracy, ten experiments were performed with different task sets for each set of parameter values and the average results were taken.

Including the baseline and the lower bound algorithms, a total of six algorithms were considered in the experiments. The energy consumption of a processor without voltage scheduling (i.e., the processor always runs at its maximum speed) was used as the baseline. The energy consumptions of the three proposed voltage scheduling algorithms[4] were normalized with respect to the baseline for easy comparison. The performance of the proposed algorithms were compared with the Dynamic Reclaiming Algorithm (DRA) [2], which allows a job to execute at a lower speed if its preceding jobs did not use up their WCETs. To guarantee schedulability in case of blocking, the static speed H was taken as the "nominal speed" in DRA. A lower bound in energy consumption was also plotted where the processor operated at the lowest static speed to complete the given workload within the simulated time span (i.e., 100,000ms) disregarding the deadlines of the jobs. Not limited by the timing constraints nor by the discrete speed levels, it consumes less energy than any algorithm that must observe these constraints.

Blocking parameters

In the first set of simulations, we evaluated the effect of blocking sections on energy consumption. The blocking parameters *blocking_ratio* and *blocking_prob* were varied to produce task sets with different blocking characteristics. In these experiments, we let all jobs run their full length of their WCETs, i.e., WCET/BCET = 1. As will be shown later, the energy saving would further increase if the actual execution time is less than the WCET.

We first fixed the *blocking_prob* at 0.5 and varied the *blocking_ratio* between 0 and 0.3. The range of blocking ratio was chosen for two reasons. First, blocking sections are usually short in practical systems. Moreover, as the blocking ratio exceeded 0.3, the number of tasks dropped during the admission control

[4]For the rate-monotonic dual speed algorithm and DSDR, the high and low speeds were calculated according to formulas (9) and (8), respectively.

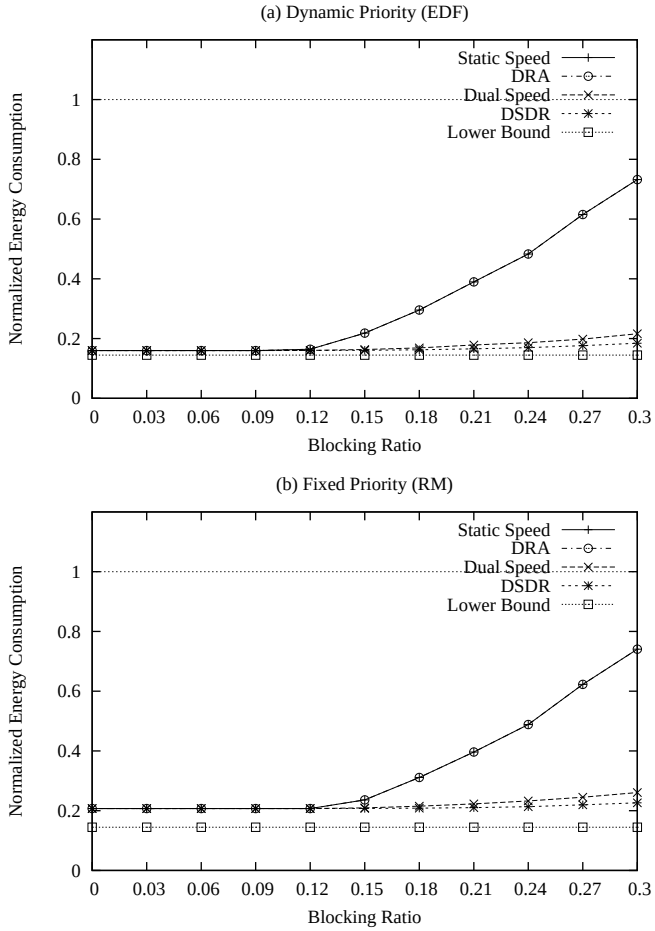

Figure 7: Normalized energy consumption with varying blocking ratio.

increased rapidly, thereby affecting the accuracy of the results. The normalized energy consumption of the proposed algorithms under a utilization factor of 0.4 was given in Figure 7. The proposed algorithms consumed much less energy compared to the baseline. Since all jobs consumed their WCETs in the simulation, the performance of DRA was identical to that of the proposed static speed algorithm. As the maximum length of blocking sections grew with the blocking ratio, the value of high speed H was also increased according to formulas (4) or (9), respectively. Although the energy consumed by all three algorithms proposed grew with H, the energy consumption of the two dynamic speed algorithms grew much more slowly because they operated at the low speed in some intervals. As a result, when the energy consumption of the static speed algorithm increased to 75% of the baseline energy under EDF, the dual speed algorithm and DSDR consumed less than 25% of the baseline energy, only 10% higher than the lower bound. Although there was no slack to reclaim in this set of experiments, DSDR was still able to utilize the time budget reclaimed during the high-speed intervals and performed better than the dual speed algorithm. However, the difference was marginal because blocking happened infrequently and the resulting high speed intervals were short, leaving little room for reclamation. Another observation

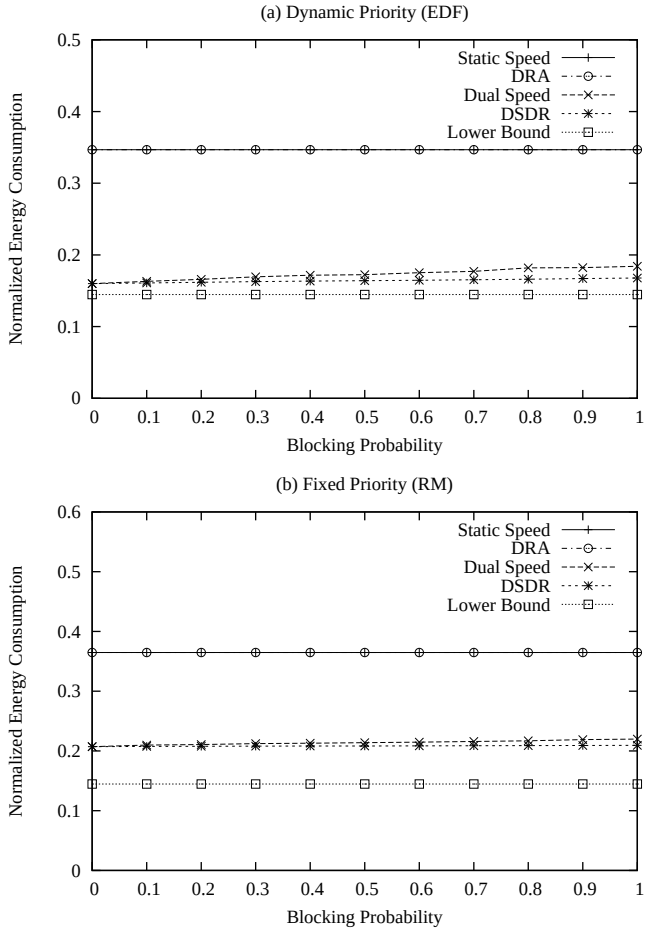

Figure 8: Normalized energy consumption with varying blocking probability.

is that when the blocking ratio was smaller than 0.09, the discrete values of high and low speeds were equal. Therefore the curves of all three algorithms converged (see Figure 7). The energy consumption of the two dynamic speed algorithms under dynamic priority scheduling was very close to the lower bound as shown in Figure 7(a). However, the energy consumption under fixed priority scheduling was slightly higher. This is due to the stricter feasibility conditions (see formulas (8) and (9)). For the same set of periodic tasks, the rate-monotonic algorithm may require a higher speed than that needed by EDF scheduling to guarantee schedulability.

Next, we varied *blocking_prob* with *blocking_ratio* fixed at 0.2. The energy consumptions of the static speed algorithm and DRA were identical and constant. Interestingly, the energy consumed by the dual speed algorithm increased only slightly (about 2%) with increase in the blocking probability (see Figure 8). This is somewhat surprising because both the number and the durations of high speed intervals increase with the blocking probability, leading to potentially higher energy consumption. Closer examination showed that the occurrence of blocking was still infrequent (several hundred in the entire time span of each experiment) despite the increasing blocking probability, and the high-speed intervals

were short. The impact on energy consumption was therefore almost negligible. DSDR was insensitive to changes in *blocking_prob* since the time budget reclaimed in high-speed intervals could be used by subsequent jobs. The overall energy consumption did not change much.

WCET/BCET Ratio

In this set of experiments, we fixed the *blocking_ratio* and the *blocking_prob* at 0.2 and 0.5, respectively. The utilization factor was 0.4. Figure 9 shows the simulation results with the WCET/BCET ratio varied between 1 (no slack) and 10 (90% slack). The static speed and the dual speed algorithms were insensitive to changes in the WCET/BCET ratio. On the other hand, with the ability to reclaim unused time budget from early completions, DSDR was able to process some jobs at a speed even lower than the utilization speed. As a result, the energy consumption decreased when the slack increased. For example, at a WCET/BCET ratio of 5, which is a typical value for video decoding times [4], DSDR obtained an additional 5 percent energy saving under dynamic priority scheduling and 12 percent energy saving under fixed priority scheduling compared to the case when no slack was available. Nonetheless, the gain slowed down as the WCET/BCET ratio increased and more or less stabilized when the ratio reached 5 (80% slack). Due to the job slack time, DRA was also able to save energy compared with the static speed algorithm, but the energy consumption was still higher than those of the dual speed algorithm and DSDR in all cases.

Processor Utilization

We also investigated the effect of different system load on performance. Again, the *blocking_ratio* and the *blocking_prob* were set at 0.2 and 0.5 respectively. The WCET/BCET ratio was kept at 5. Due to blocking, it was not possible to reach full processor utilization. As the utilization factor grew, some tasks failed the admission control and were dropped, causing the actual processor utilization to be lower than the specified value and affected the accuracy of the results. We therefore show the results only for utilization factors between 0.1 to 0.6 when all tasks generated were admitted to the system.

As in the previous experiments, the proposed voltage scaling algorithms consistently outperformed the baseline full speed algorithm, but the normalized energy consumption increased with the utilization factor. This can be explained as follows. The increased utilization factor resulted in linear increase in processing time under the baseline algorithm, so the energy consumption increased linearly. The other five algorithms, however, had to increase the processor speed to meet the deadline under the increased load, resulting in cubic energy increase according to formula (1). As seen in Figure 10, the normalized energy consumptions of the five voltage scheduling schemes increased more or less quadratically with the utilization factor as expected. Nonetheless, since the allowable operating speed cannot exceed the maximum processor speed, the energy consumptions of the proposed algorithms are bounded by that of the baseline algorithm.

In summary, the proposed voltage scheduling algorithms effectively reduced energy consumption compared with the full speed algorithm under all workload and system settings tested. In particular, they

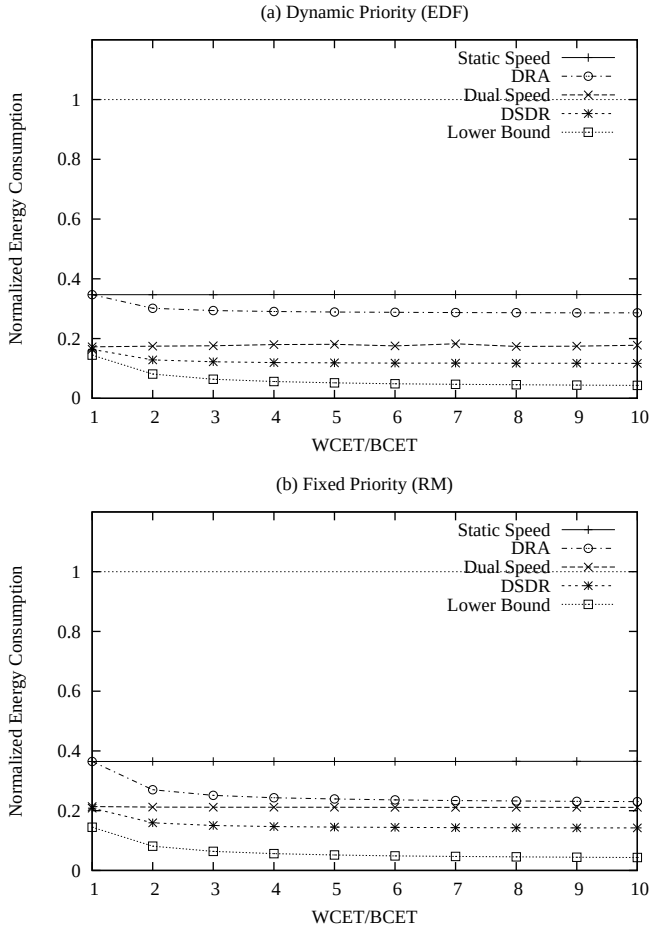

Figure 9: Normalized energy consumption with varying WCET/BCET ratio.

saved more than 90 percent of energy when the system load was low. Moreover, the two dynamic speed algorithms demonstrated better energy saving capabilities among the proposed algorithms. Thanks to blocking-awareness, the dual speed algorithm and DSDR were able to adjust the processor speed more flexibly and aggressively, and saved more energy than the existing DRA algorithm in all the experiments performed. When jobs did not always execute at their WCETs, DSDR was able to save additional energy compared with the dual speed algorithm owing to its dynamic reclaiming mechanism. The energy consumptions of the two dynamic speed algorithms were close to the theoretical lower bound in dynamic priority scheduling using EDF. However, the results under rate-monotonic scheduling suggest room for further energy reduction.

7 Conclusion

Voltage scheduling is an effective way in reducing energy consumption of the processor. In this work, we have modelled and studied the performance of voltage scheduling of real-time tasks with non-preemptible

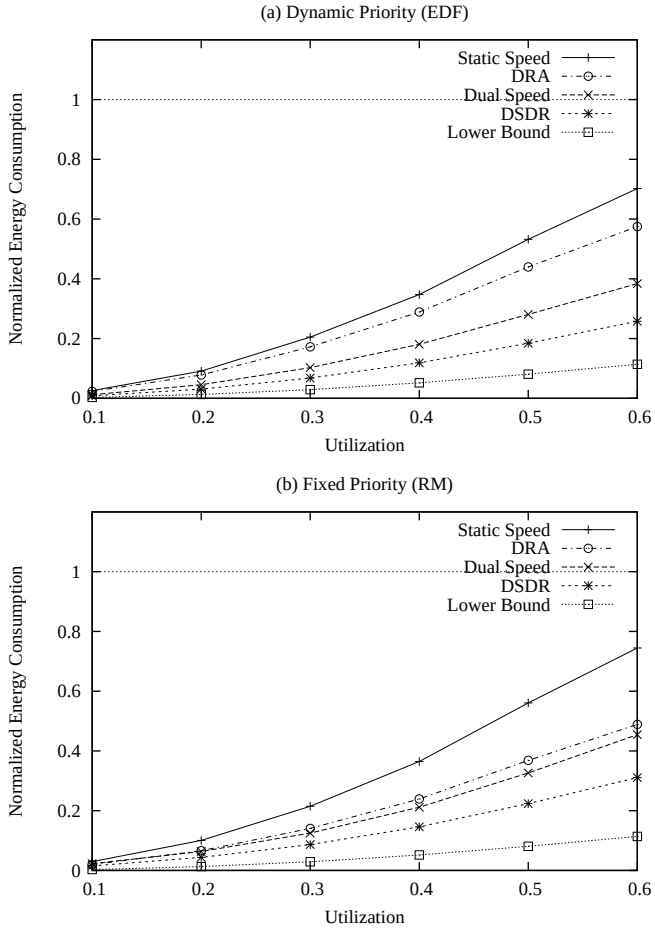

Figure 10: Normalized energy consumption with varying processor utilization.

blocking sections. Three algorithms are proposed for dynamic priority scheduling (EDF) and also for fixed priority scheduling (RM). The static speed scheme maintains a minimal feasible static speed based on a result from the stack resource policy. In the dual speed algorithm, the high-speed interval is defined as the interval that begins when a job is blocked and terminates when a job with priority equal to or lower than that of the blocking job starts execution. Except during the high speed intervals, the processor runs at a much lower utilization speed in the dual speed algorithm. The DSDR algorithm extends the dual speed algorithm by reserving time budget for each job. A reclaiming mechanism is used to collect the unused time budget from completed jobs and redistribute it to subsequent jobs whenever possible. The feasibility conditions for the algorithms have been derived and proved. Extensive simulations have shown that the proposed algorithms can significantly reduce energy consumption in all scenarios compared to the case where voltage scheduling is not used. Furthermore, our blocking-aware algorithms (dual speed and DSDR) excelled the existing DRA algorithm in all the experiments. Moreover, the DSDR algorithm achieved performance close to the theoretical lower bound and consistently outperformed the static speed algorithm by up to 60% in energy consumption.

References

[1] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole. A Measurement-Based Analysis of the Real-Time Performance of Linux. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 133–142, September 2002.

[2] 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.

[3] T. P. Baker. Stack-based scheduling of real-time processes. In *Advances in Real-Time Systems*. IEEE Computer Society Press, 1993.

[4] A. Bavier, B. Montz, and L. Peterson. Predicting mpeg execution times. In *SIGMETRICS'98*, pages 131–140, June 1998.

[5] L. Benini, A. Bogliolo, and G.D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.

[6] T. D. Burd and R. W. Brodersen. *Energy Efficient Microprocessor Design*. Kluwer Academic Publishers, 2002.

[7] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J. M. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proceedings of SIGMETRICS*, pages 252–263, June 2000.

[8] Dirk Grunwald, Philip Levis, and Keith I. Farkas. Policies for dynamic clocking scheduling. In *Proceedings of the Symposium on Operating system Design and Implementation (OSDI)*, pages 73–86, October 2000.

[9] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(12):1702–1714, 1999.

[10] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, December 1989.

[11] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[12] J. R. Lorch and A. J. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications*, 5(3):60–73, 1998.

[13] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of SIGMETRICS*, pages 50–61, June 2001.

[14] Y.H. Lu, L. Benini, and G.D. Micheli. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1284–1305, November 2002.

[15] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

[16] 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.

[17] 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.

[18] T. Simunic, L. Benini, and G. De Micheli. Energy-efficient design of battery-powered embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):15–28, 2001.

[19] M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications, Special Issue on Mobile Computing*, E80-B(8):1125–1131, 1997.

[20] O.S. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE*, 91(7):1055–1069, July 2003.

[21] 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.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE Annual Symposium on Foundations of Computer Science*, pages 374–382, October 1995.

[23] 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.