On-Line Dynamic Voltage Scaling for Hard Real-Time Systems Using the EDF Algorithm *

Cheol-Hoon Lee
System Software Laboratory
Department of Computer Engineering
Chungnam National University
Taejeon 305-764, Korea
chlee@ce.cnu.ac.kr

Kang G. Shin
Real-Time Computing Laboratory
Department of EECS
The University of Michigan
Ann Arbor, MI 48109-2122, U.S.A.
kgshin@eecs.umich.edu

Abstract

Recently, there has been a rapid and wide spread of non-traditional computing platforms, especially mobile and portable computing devices. As applications become sophisticated and computation power increases, the most serious limitation on these devices is the available battery life. Dynamic voltage scaling (DVS) has been a key technique to exploit the hardware characteristics of processors to reduce energy dissipation by lowering the supply voltage and operating frequency. This paper presents a novel on-line DVS algorithm called **OLDVS** *that, when coupled with the underlying OS task management mechanism and real-time scheduler, can make significant energy savings, while preserving timeliness guarantees made by the underlying real-time scheduling algorithm. While most existing DVS algorithms are confined to periodic tasks only,* **OLDVS** *does not assume task periodicity, nor does it require any a priori information on the task set to be scheduled.* **OLDVS** *requires only* $O(1)$ *computation on each task context switch, thus making it fairly easy to be incorporated into a real-time operating system. The* **OLDVS** *algorithm considers a* general *task model which is very difficult, if not impossible, for the existing DVS algorithms to handle. Our simulation results show that* **OLDVS** *achieves great energy savings and outperforms the existing DVS algorithms when the ratio of the computation requirement of aperiodic tasks to the total computation requirement is higher than 40%. The performance advantage becomes much larger as the ratio increases.*

* This work was supported in part by the US AFOSR under Grant No. F49620-01-1-0120.

1. Introduction

As the device technology approaches the limit of scaling in CMOS circuits, power and heat dissipation issues are becoming ever more important. Over the last several years, computation and communication have been steadily moving toward mobile and portable platforms/devices with the increasing popularity of ubiquitous applications. This is evident in the growth of laptop computers and PDAs, but is also occurring in the embedded world. The applications for these domains are typically run on battery-powered embedded systems. With continued miniaturization and increasing computation power, we see ever growing use of powerful microprocessors running sophisticated, intelligent control software in a vast array of embedded systems including digital camcorders, cellular phones, and portable medical devices [24].

In the last decade, significant research and development efforts have been made on Dynamic Voltage Scaling (DVS) for real-time systems to make energy-savings by scaling the voltage and frequency while maintaining real-time deadline guarantees [1–6, 8, 12–17, 23, 25]. This is possible because static CMOS logic, used in the vast majority of microprocessors today, has a voltage-dependent maximum operating frequency, so when used at a reduced frequency, the processor can operate at a lower supply voltage. Since the energy dissipated per cycle with CMOS circuitry scales quadratically to the supply voltage ($E \propto V^2$) [4], DVS can potentially provide a very large net energy savings through frequency and voltage scaling. However, most of them assume that the reference task model consists of periodic tasks, and with relative deadlines equal to their respective periods. For a *mixed* task system with both periodic and aperiodic tasks, researches focus on how to improve the *responsiveness* of aperiodic tasks that do not have explicit deadlines by minimizing their average response time, while meeting the hard deadlines of periodic tasks. To sat-

Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)
1052-8725/04 $20.00 © 2004 **IEEE**

isfy these objectives, many scheduling algorithms had been proposed based on the "server" concept, without taking energy consumption into account [9, 18, 20, 22]. Recently, some algorithms have been proposed for scheduling mixed task sets using DVS to make energy savings. Shin and Kim [16] proposed DVS algorithms that guarantee the schedulability of all periodic tasks with a "good" average response time for each aperiodic task, while doing the best to minimize the total energy consumption. The energy/responsiveness tradeoffs involved in scheduling mixed task sets on DVS-enabled processors have been explored by Aydin and Yang [3]. They proposed the composite metric, energy $\times$ average response time, as a performance measure for the energy-aware scheduling of mixed task sets.

The existing DVS algorithms assume the availability of *a priori* knowledge of the release times and deadlines of all real-time (*i.e.*, periodic) tasks. With this knowledge, they use the processor utilization for schedulability analysis. They also assume that the relative deadline of each task is equal to its period. Therefore, they are not on-line algorithms in a strong sense. They just estimate the slack times on-line. By contrast, we present an *on-line* DVS algorithm for hard real-time systems that attempts to minimize the energy consumed by each task set. The task model considered in this paper is very *general* in that each task has arbitrary release time and deadline. While most existing DVS algorithms focus on periodic tasks only, the proposed algorithm does not assume the periodicity of tasks, nor does it require any *a priori* information (such as periods, deadlines, and their worst-case computation times) on the task set to be scheduled. The algorithm is based on the earliest-deadline-first (EDF) algorithm that is proven to be optimal even under this generalized task model [10, 19]. Unlike the existing algorithms, the proposed algorithm uses the notion of "loading factor" as the feasibility criterion to scale the frequency and voltage for energy savings. The proposed on-line DVS algorithm requires only $O(1)$ computation on each task context switch, so it is fairly easy to incorporate the algorithm into a real-time operating system. Our simulation results show that the proposed algorithm outperforms the existing DVS algorithms when the ratio of the computation requirement of aperiodic tasks to the total computation requirement exceeds a certain threshold. The performance gap becomes much larger as this ratio increases.

The paper is organized as follows. In the next section, we present the system model considered in this paper and introduce the notion of loading factor. Section 3 presents details of our on-line DVS algorithm and illustrates how it works. The simulation results are presented in Section 4, and Section 5 concludes the paper with a summary of our results and a discussion of future directions.

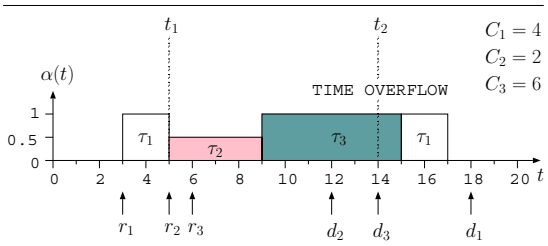


Figure 1. EDF schedule (with time overflow) of an example task set of three tasks.

2. Motivation

2.1. System model

We consider a preemptive hard real-time system in which real-time tasks are scheduled under the EDF scheduling policy [10]. The target variable voltage processor can scale its supply voltage and clock frequnecy continuously within its operational ranges, $[v_{min}, v_{max}]$ and $[f_{min}, f_{max}]$. Associated with each task τ_i are the arbitrary release time r_i, worst-case execution time (WCET) C_i and deadline d_i. Tasks are assumed to be independent. We define the start time, denoted by s_i, as the time at which task τ_i gets the control of CPU and starts to execute. For each task τ_i, the *execution time* e_i is defined as the time spent by CPU to complete the execution of the task when operating at the maximum frequency. Although real-time tasks are specified for worst-case execution environments, they generally use much less than the worst case (*i.e.*, $e_i \leq C_i$). Let $\alpha()$ be a frequency scaling function such that the frequency is scaled down to $\alpha(t) \cdot f_{max}$ at time t ($0 \leq \alpha(t) \leq 1, \forall t$). Also, let α_i be the frequency scaling factor of task τ_i. If the frequency is scaled down, the requested CPU time to complete each task will increase. We define the *effective execution time*, denoted by e_i', as the actual time spent by CPU to complete τ_i under a frequency scaling $\alpha()$ (obviously, $e_i' \geq e_i$).

2.2. The notion of loading factor

This subsection focuses on assessing task set feasibility under EDF scheduling. Two general concepts are used to analyze the feasibility of real-time task sets: *processor demand* and *loading factor*. The processor demand is a focused measure of how much computation is requested, with respect to timing constraints, in a given interval of time, while the loading factor is the maximum of the fraction of processor time possibly demanded by the task set in any interval of time. These two terminologies are originally de-

fined based on the execution times (actually WCETs) of tasks without frequency scaling [21]. In this paper, however, we modify them based on the effective execution times under certain frequency scaling as follows.

Definition 1. Given a set of real-time tasks and an interval of time $[t_1, t_2)$, the effective processor demand of the task set during the interval $[t_1, t_2)$ under a frequency scaling, $\alpha()$ is

$$h'_{[t_1,t_2)} = \sum_{t_1 \le r_k, d_k \le t_2} e'_k.$$

That is, under certain frequency scaling $\alpha()$, the effective processor demand during $[t_1,\ t_2)$ represents the amount of CPU time that is requested by all tasks with releases time at or after t_1 and deadlines before or at t_2.

Definition 2. Given a set of real-time tasks, its effective loading factor during the interval $[t_1,\ t_2)$ is the fraction of the interval needed to execute the tasks under certain frequency scaling $\alpha()$, i.e.,

$$u'_{[t_1,t_2)} = \frac{h'_{[t_1,t_2)}}{t_2 - t_1}.$$

Definition 3. The absolute effective loading factor, or simply effective loading factor, under certain frequency scaling $\alpha()$, is the maximum of all possible intervals, that is,

$$u' = \sup_{0 \le t_1 < t_2} u'_{[t_1,t_2)}.$$

In other words, under frequency scaling $\alpha()$, a task set has an effective loading factor u' if in each interval of time $[t_1,\ t_2)$ the maximum demanded CPU time is at most $u' \cdot (t_2 - t_1)$. For example, the task set of Figure 1 has an effective loading factor $u' = 10/9$, as shown in Table I. Note that only the computation of the effective loading factor during some intervals is shown. It is easy to verify that the effective loading factor over any other interval is less than those shown in Table I.

Intuitively, under frequency scaling, a necessary condition for the feasibility of a task set under any scheduling algorithm is that the effective loading factor is not greater than 1. In fact, not only is this claim true, but the condition is also sufficient for the task set feasibility under the EDF scheduling algorithm. In the absence of frequency scaling (*i.e.*, $\alpha(t) = 1, \forall t$), Spuri [19] proved this based on the notion of loading factor u which is the same as the effective loading factor u' when the task set is executed under the worst-case execution scenario without frequency scaling, *i.e.*, $e_i = C_i$ and $\alpha(t) = 1, \forall i, t$. We can directly apply the result of [19] to the case with frequency scaling as follows.

$u'_{[3,18)} = \frac{4+4+6}{15} = \frac{14}{15}$
$u'_{[5,12)} = \frac{4}{7}$
$u'_{[5,14)} = \frac{4+6}{9} = \frac{10}{9}$
$u'_{[6,14)} = \frac{6}{8}$
$u' = \frac{10}{9}$

Table 1. Computation of the effective loading factor for the task set of Figure 1.

Theorem 1. Under certain frequency scaling, each set of real-time tasks is feasibly schedulable by the EDF algorithm if and only if

$$u' \le 1.$$

Proof: "If" part: Assume there is a deadline miss at time t. The miss must be preceded by a CPU busy period, that is, a period of continuous processor utilization, in which only tasks with deadlines earlier than t are executed. Let $t_2 = t$ and t_1 be the last instant preceding t such that there are no pending execution requests of tasks released before t_1 and having deadlines less than or equal to t. Both t_1 and t_2 are well defined. See Figure 1 for example. In particular, after t_1, which must be the release time of some task, the processor is allocated to tasks released after t_1 and having deadlines less than t_2. Since there is a deadline miss at t_2, the amount of CPU time demanded in the interval $[t_1, t_2)$ must be greater than the interval itself, that is,

$$\sum_{t_1 \le r_k, d_k \le t_2} e'_k > (t_2 - t_1).$$

It follows that

$$u'_{[t_1,t_2)} > 1,$$

hence

$$u' > 1,$$

a contradiction.
"Only if" part: Since the schedule is feasible, the amount of CPU time demanded in each interval of time must be less than or equal to the length of the interval, that is,

$$\forall [t_1, t_2), \sum_{t_1 \le r_k, d_k \le t_2} e'_k \le (t_2 - t_1).$$

It follows that

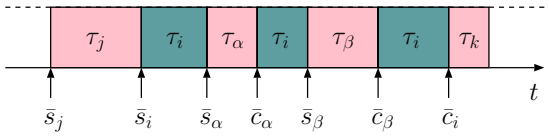


Figure 2. An example schedule under EDF.

$$u'_{[t_1,t_2)} \leq 1,$$

hence

$$u' \leq 1.$$

□

Note that the result shown above also confirms the optimality of the EDF algorithm for uni-processor systems under frequency scaling. An input task set Ψ is called "feasible," if the loading factor u_Ψ is less than or equal to 1 under the worst-case execution scenario. In the remainder of this paper, we assume that the loading factor u_Ψ of task set Ψ is 1 in the worst-case execution scenario. In the case where $u_\Psi < 1$, the maximum clock frequency is assumed to be adjusted to $f'_{max} = u_\Psi \cdot f_{max}$ without violating any timing constraint. With f'_{max}, the loading factor becomes 1, leaving no idle intervals if every task takes the WCET for its execution.

3. The On-Line Dynamic Voltage Scaling (OLDVS)

3.1. The basic idea

Consider a set Ψ of real-time tasks whose loading factor u_Ψ is equal to 1. Since $u_\Psi = 1$, there is no slack time available on the EDF schedule in the worst-case execution scenario in which every task takes the WCET for its execution. Consider task τ_i in the example schedule of Figure 2, where $\bar{s}_i$ and $\bar{c}_i$ are the expected start and completion times of τ_i in the worst-case scenario, respectively. Since we assume preemptive scheduling, if τ_i is released before $\bar{s}_i$ (*i.e.*, $r_i < \bar{s}_i$), task τ_j has higher priority than τ_i and is completed at $\bar{s}_i$ (*i.e.*, $d_j \leq d_i$ and $\bar{c}_j = \bar{s}_i$). If $r_i = \bar{s}_i$, τ_j is completed at $\bar{s}_i$, or τ_j is a lower-priority task and preempted by τ_i at $\bar{s}_i$. Also, tasks τ_α and τ_β in Figure 2 are all higher-priority tasks and preempt τ_i upon their release, respectively. So, the expected completion time for each task τ_i under the worst-case scenario is calculated as follows:

$$\bar{c}_i = \bar{s}_i + C_i + \sum_{\bar{s}_i < r_k, d_k < d_i} C_k.$$

calculate_*$\alpha(t)${*
  if τ_i preempted τ_j *then* /* it means $r_i = s_i = t$ and $d_i < d_j$ */
    $D_i = t + C_i$;
    $R_i = C_i$;
    $R_j = R_j - \alpha_j \cdot (t - l)$; /* l: the previous ctx switch time */
  else if τ_i resumes after some task τ_k *then*
    $D_i = D_i + D_k - t_p$; /* say τ_i was preempted at t_p */
  else /* it means τ_i starts execution after some task τ_k */
    if ($d_k > d_i$ or $D_k < t$) *then* $D_i = t + C_i$;
      else $D_i = D_k + C_i$;
    $R_i = C_i$;
  return ($\alpha_i = \frac{R_i}{D_i - t}$); /* return the scaling factor α_i */
}

Figure 3. Computation of the frequency scaling factor for OLDVS.

To guarantee the feasible execution of all the upcoming tasks under the worst-case scenario, each task τ_i must complete before or at $\bar{c}_i$. So, each task τ_i must also start before or at $\bar{s}_i$. We define the *worst-case completion time* of task τ_i, denoted by D_i, as the latest time to complete, that guarantees the feasible execution of all the upcoming tasks. If an on-line algorithm estimates the worst-case completion time of each task to be less than or equal to its expected completion time under the worst-case scenario (*i.e.*, $D_i \leq \bar{c}_i, \forall i$) and schedules it to complete before that, the algorithm guarantees the feasible execution. For each context switch to task τ_i, say at time t, its worst-case completion time D_i can be calculated incrementally as follows: (i) *if* task τ_i preempted some task τ_j (*i.e.*, $s_i = t$ and $d_i < d_j$), *then* $D_i = t + C_i$; (ii) *else if* τ_i resumes right after the completion of some task τ_k (meaning that τ_i was previously preempted by other task(s), say at time t_p), *then* $D_i = D_i + D_k - t_p$; (iii) *else* (meaning that τ_i starts after some task τ_k) *if* $d_i < d_k$ or $D_k < t$, *then* $D_i = t + C_i$, *else* $D_i = D_k + C_i$. Initially, the worst-case completion time is set to 0 (*i.e.*, $D_0 = 0$). It is shown in the next subsection that the worst-case completion time for each task estimated as above is less than or equal to its expected completion time under the worst-case scenario. If each task completes earlier than its expected worst-case completion time, the next task can use this unused execution time, effectively moving its start time ahead. These slack times can be exploited for energy savings by lowering the frequency and the supply voltage accordingly. For each task τ_i, we define the *worst-case remaining time*, denoted by R_i, to be the remaining CPU time to complete the task when operating at the maximum frequency under the worst-case scenario. At the start time of each task τ_i, it is initially set to its WCET

(*i.e.*, $R_i = C_i$). When a task is preempted by another task, its worst-case remaining time is reduced by the amount converted into maximum-frequency CPU time from the previous context switch. For example, if task τ_j is preempted by another task at t, then its worst-case remaining time R_j is recalculated such that $R_j = R_j - \alpha_j \cdot (t - l)$, where l is the previous context switch time. Each task τ_i must be completed before or at D_i to guarantee the feasible execution of all the remaining tasks, and it takes R_i to complete at the maximum frequency in the worst case. Therefore, at each context switch to task τ_i at time t, we can set the frequency scaling factor α_i as follows:

$$\alpha_i = \frac{R_i}{D_i - t}.$$

By scaling the frequency as above, each task is guaranteed to complete before or at its worst-case completion time, thus guaranteeing the feasible execution of all the upcoming tasks under the worst-case scenario. If each task τ_i takes the WCET for its execution, it completes exactly at D_i. Note that $\alpha_i \leq 1$, since $R_i \leq D_i - t$, for all i and t. The complete algorithm for calculation of the frequency scaling factor, called *calculate_*$\alpha()$, is shown in Figure 3. It is called on each context switch, and calculates both the worst-case completion and the worst-case remaining times of the task, then returns the value of the frequency scaling factor for the task. Therefore, no more than two frequency (and voltage) transitions can occur per task. Since the time complexity of the algorithm is $O(1)$, we can calculate the scaling factor on-line at each context switch time with negligible overhead. Using the function *calculate_*$\alpha()$, next we present the on-line dynamic voltage scaling algorithm, called *OLDVS*.

3.2. The OLDVS algorithm

Although real-time tasks are specified with worst-case computation requirements, they generally use much less than the worst-case values for most of their invocations. To exploit this, the DVS mechanism could reduce the operating frequency and voltage when tasks use less than their worst-case time allotment, and, if needed, increase frequency to meet the worst-case needs. When a task is released, we cannot know how much computation it will actually require, so we must assume that it will need its specified worst-case execution time. In other words, at each start time of task τ_i, we must set its worst-case remaining time to its worst-case execution time (*i.e.*, $R_i = C_i$).

The OLDVS algorithm itself (Figure 4) is quite simple. At each task context switch time, it first calculates the frequency scaling factor using the function *calculate_*$\alpha()$, then it scales the voltage and frequency. The algorithm is tightly-coupled with the operating system's task management services, since they may need to reduce or increase frequency on each task context switch. The main challenge in designing such algorithms is to ensure that deadline guarantees are not compromised when the operating frequencies are reduced. If the input task set Ψ is feasible, the algorithm OLDVS guarantees the deadlines of all the tasks. This is proved in the following theorem.

upon context switch to each task τ_i at time t:
    $\alpha_i = calculate\_\alpha(t)$;
    scale_voltage_and_frequency(α_i);

Figure 4. The algorithm OLDVS.

Theorem 2. Each feasible set Ψ of real-time tasks is feasibly schedulable by OLDVS.

Proof: By definition, the loading factor u_Ψ of a feasible task set Ψ is less than or equal to 1 under the worst-case scenario, *i.e.*,

$$u_\Psi = \frac{\sum_{t_1 \leq r_k, d_k \leq t_2} C_k}{t_2 - t_1} \leq 1, \forall [t_1, t_2).$$

Let the tasks be sorted by their deadlines, *i.e.*, $\Psi = \{\tau_i | 1 \leq i, d_i \leq d_{i+1}\}$. Also, let s_i and c_i be the start and completion times of each task τ_i, respectively, in the schedule yielded by OLDVS. Assume there is a deadline miss at time t while executing some task, say τ_i. The miss must be preceded by a CPU busy period, that is, a period of continuous processor utilization, in which only tasks with deadlines less than t are executed. Let again, as in the proof of Theorem 1, $t_2 = t$ and t_1 be the last instant preceding t such that there are no pending execution requests of tasks released before t_1 and having deadlines less than or equal to t. In particular, after t_1, which must be the release time of some task, the processor is allocated to tasks released after t_1 and having deadlines less than t_2, *i.e.*, τ_k's with $t_1 \leq r_k$ and $k \leq j$. Let τ_j be the task to be scheduled at t_1, *i.e.*, $s_j = t_1$. Then, by the algorithm *calculate_*$\alpha()$, $D_j = t_1 + C_j$. Also each task τ_k following τ_j has an effect of adding the amount of C_k on the computation of the worst-case execution time D_i of task τ_i. Therefore, $D_i = t_1 + \sum_{t_1 \leq r_k, k \leq i} C_k$. By applying the algorithm, task τ_i completes before or at D_i (*i.e.*, $c_i \leq D_i$). Since there is a deadline miss at t_2, the amount of CPU time demanded in the interval $[t_1, t_2)$ must be greater than the interval itself, that is,

$$\sum_{t_1 \leq r_k, d_k \leq t_2} e'_k > (t_2 - t_1). \quad (1)$$

Since τ_i completes before or at its worst-case computation time D_i, $c_i \leq D_i$, *i.e.*,

$$t_1 + \sum_{t_1 \le r_k, d_k \le t_2} e'_k \le t_1 + \sum_{t_1 \le r_k, d_k \le t_2} C_k \quad (2)$$

From Eqs. (1) and (2), we get

$$\sum_{t_1 \le r_k, d_k \le t_2} C_k > (t_2 - t_1).$$

It follows that

$$u_{[t_1,t_2)} > 1,$$

hence

$$u_\Psi > 1,$$

a contradiction. Therefore, the algorithm follows. □

The OLDVS algorithm presented above should be fairly easy to be incorporated into a real-time operating system, and does not incur any significant processing cost. The dynamic schemes require $O(1)$ computation, and should not require significant processing for the scheduler. The most significant overhead may be the hardware voltage switching times. However, no more than two switches can occur per task, so this overhead can easily be accounted for, and added to, the worst-case task execution times.

3.3. An illustrative example

Consider a task set Ψ composed of $\tau_1(0, 4, 7)$, $\tau_2(6, 2, 9)$, $\tau_3(3, 6, 15)$, $\tau_4(10, 4, 18)$, $\tau_5(20, 4, 26)$, $\tau_6(11, 7, 30)$, $\cdots$, where each task is denoted by a three-tuple $\tau_i(r_i, C_i, d_i)$. Note that tasks τ_1, τ_4, τ_5, $\cdots$ are all periodic with the period of 10 but with arbitrary deadlines. Figure 5(a) shows the EDF schedule under the worst-case execution scenario in which every task takes the WCET for its execution. Since u_Ψ is 1, there is no slack time available on the schedule. Although real-time tasks are specified with worst-case execution environments, they generally use much less than the worst case. Suppose that actual execution times are $e_1 = 2$, $e_2 = 1$, $e_3 = 5$, $e_4 = 2$, $e_5 = 2$, $e_6 = 4$, $\cdots$. Then, the resulting EDF schedule is shown in Figure 5(b). We illustrate how the OLDVS algorithm works with this example task set in Figure 6. At time $t = 0$, the highest-priority task τ_1 starts to run with $\alpha_1 = 1$, since $D_1 = R_1 = 4$ (Figure 6(a)). At time $t = 3$, task τ_3 releases and starts to run with $\alpha_3 = \frac{6}{7}$, since $D_3 = D_1 + C_3 = 10$ and $R_3 = C_3 = 6$ (Figure 6(b). At time $t = 6$, task τ_2 releases (Figure 6(c)). Since it has higher priority than τ_3, it preempts τ_3, and starts to execute with $\alpha_2 = 1$. At $t = 7$, τ_3 resumes with $\alpha_3 = \frac{24}{35}$ since $D_3 = D_3 + D_2 - s_2 = 12$ and $R_3 = R_3 - \alpha_3 \cdot 3 = \frac{24}{7}$ (Figure 6(d)). In this way, tasks τ_4, τ_5 and τ_6 execute with their scaling factors of 0.73, 0.72 and 1, respectively. The complete schedule is shown in Figure 6(f). The algorithm dynamically adjusts frequency and voltage on each context switch, reacting to the actual computation requirements of the real-time tasks, to attempt to minimize the energy consumed while guaranteeing the feasible execution of all the upcoming tasks under the worst-case scenario. Compared with the schedule without voltage scaling (Figure 5(b)), using the OLDVS algorithm could save energy by more than 30% for this specific example.

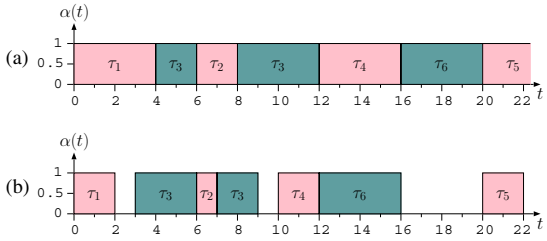


Figure 5. An example schedule under EDF.

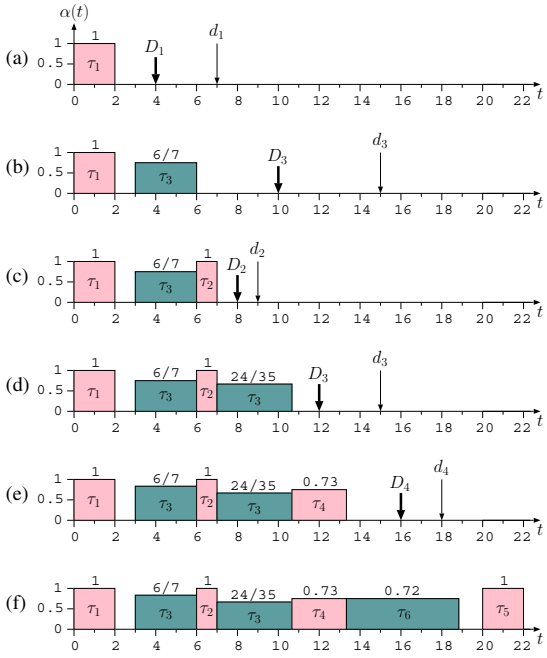
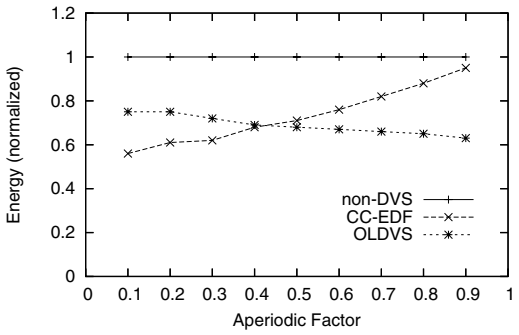


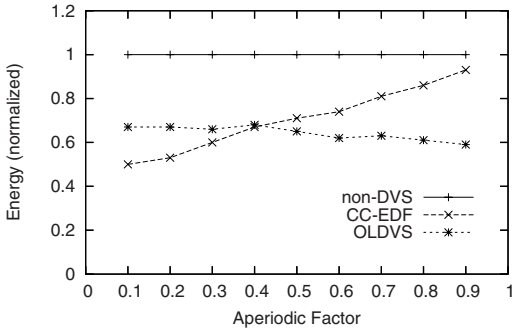
Figure 6. An example of OLDVS.

4. Simulation Results

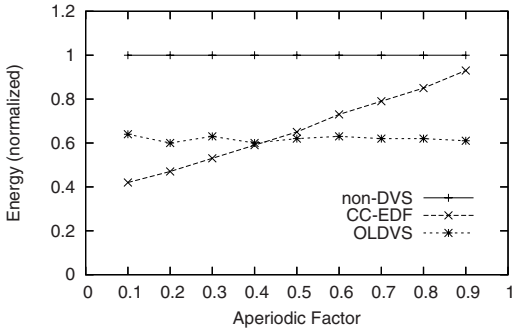
To evaluate the potential energy savings from voltage scaling in a real-time scheduling system, we have



(a) 5 tasks.



(b) 10 tasks.



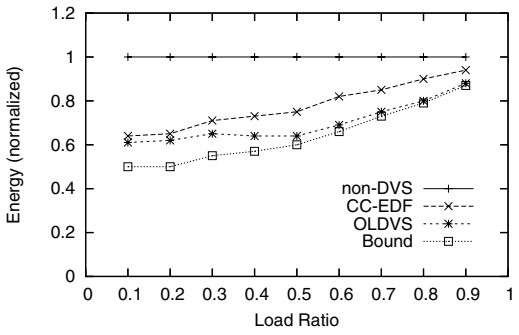
(c) 15 tasks.

Figure 7. Normalized energy consumption with 5, 10, and 15 tasks.

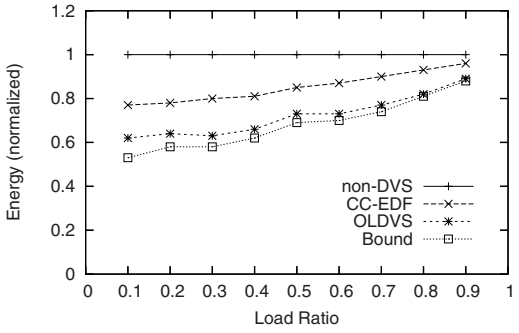
developed a simulator for the operation of hardware capable of voltage and frequency scaling. The simulation assumes that a constant amount of energy is required for each cycle of operation at a given voltage. This quantum is scaled by the square of the operating voltage, consistent with energy dissipation in CMOS circuits ($E \propto V^2$). Only the energy consumed by the processor is computed, and variations due to different types of instructions executed are not taken into account. With this simplification, the task execution modeling can be reduced to computing cycles of execution, and execution traces are not needed. The simulation also assumes a perfect machine in that (i) the clock speed and the supply voltage can be varied continuously within its operational ranges $[f_{min}, f_{max}]$ and $[v_{min}, v_{max}]$, respectively; (ii) a perfect software-controlled halt feature is provided by the processor, so idle time consumes no energy. To evaluate the energy efficiency of the proposed OLDVS algorithm, we explored various RT-DVS algorithms proposed in [15] that is shown to be close to optimal with periodic tasks in real-time systems [7]. Pillai and Shin [15] observed that, if the machine has a large number of voltage settings like the machine considered here, the cycle-conserving EDF algorithm (CC-EDF) outperforms the others and very closely approximates the theoretical lower bound over the entire range of utilizations. In the following simulations, we compare our OLDVS algorithm to CC-EDF and to a non-DVS system.

The periodic real-time task sets are specified using the period and worst-case execution time of each task. The task sets are generated randomly as follows. Each task has an equal probability of having a short (1-10ms), medium (10-100ms), or long (100-1000ms) period. Within each range, task periods are uniformly distributed. Aperiodic task sets are also generated randomly such that their worst-case computation requirements and deadlines are uniformly distributed in the ranges. This simulates a varied mix of short and long period tasks commonly found in real-time systems. The computation requirements of the tasks are assigned randomly using a similar 3 range uniform distribution. For a fair and efficient comparison, for each test with CC-EDF, all aperiodic tasks are converted into a single periodic task such that its period and worst-case execution time are chosen so as to minimize the processor utilization while guaranteeing the deadlines of all tasks in each period. We call the converted task a *periodic server* that behaves like a periodic task and is created for the purpose of executing aperiodic tasks [11]. Finally, the computation requirements of the aperiodic tasks are chosen such that the total processor utilization of the periodic tasks (including the periodic server) becomes 1.

First, we have performed simulations while varying the aperiodic factor, the ratio of the computation requirement of aperiodic tasks to the total computation requirement. Figure 7 shows the normalized energy consumption for tasks sets with 5, 10, and 15 tasks for our OLDVS algorithm and CC-EDF. All of these simulations assume that the average actual computation required by the tasks are 30% of their worst-case computation requirements. As expected, when the aperiodic factor passes a certain point (around 0.4 in these simulations), the proposed on-line algorithm outperforms CC-EDF in all simulations. The performance gap becomes much larger as the factor increases. Since the CC-EDF algorithm (as well as most other existing DVS algorithms) is focused on periodic tasks, the energy efficiency deteriorates as the aperiodic factor increases. On the other hand, since the OLDVS al-



(a) Aperiodic factor = 0.5.



(b) Aperiodic factor = 0.7.

Figure 8. Normalized energy consumption with aperiodic factors 0.5 and 0.7.

gorithm does not assume the periodicity of real-time tasks, the aperiodic factor does not affect its energy efficiency. Instead, the simulation results show that the energy efficiency of OLDVS becomes slightly better as the aperiodic factor increases. This is because the deadlines of the aperiodic tasks are uniformly distributed in the ranges, while the deadlines of the periodic tasks are set to their periods for a fair comparison with CC-EDF.

The proceeding simulations assumed that the average actual computation required by the tasks are 30% of their worst-case computation requirements. To see how well the OLDVS algorithm (along with CC-EDF) takes advantage of task sets that do not consume their worst-case execution times, we performed simulations while varying the ratios of the actual computation required by the tasks to their worst-case execution times. In these simulations, each task set is composed of 10 tasks. Figure 8 shows the simulation results for task sets with the aperiodic factors of 0.5 and 0.7. In this figure, we also include the result for a *clairvoyant* algorithm, named **Bound**, that knows the exact actual computation requirement of each task in advance and adopts an optimal frequency accordingly. Although **Bound** is not a practical algorithm (since no algorithm can predict the exact computation requirement beforehand), it is included as a yardstick in our simulations. Clearly, no real DVS algorithm can offer a better performance than that of **Bound**. In Figure 8, we can observe that both the OLDVS and CC-EDF algorithms show great reductions in relative energy consumption as the actual computation performed decreases.

5. Conclusions and Future Directions

In this paper, we have presented a novel algorithm for on-line real-time dynamic voltage scaling that, when coupled with the underlying OS task management mechanism and real-time scheduler, can achieve significant energy savings, while simultaneously preserving timeliness guarantees. While most existing DVS algorithms are designed for periodic tasks only, the proposed algorithm does not assume the periodicity of tasks, and thus, does not require any *a priori* information on the task set to be scheduled. The proposed algorithm requires only $O(1)$ computation on each task context switch, so it is fairly easy to incorporate the algorithm into a real-time operating system. The main contribution of this paper is that the proposed on-line DVS algorithm considers a general task model which is difficult to deal with by using the existing DVS algorithms. The simulation results show that the proposed algorithm achieves great energy savings and outperforms the existing DVS algorithms when the ratio of the computation requirement of aperiodic tasks to the total computation requirement exceeds a certain point. The performance gap becomes much larger as the ratio increases.

In future, we would like to expand this work beyond the deterministic/absolute real-time paradigm presented here. In particular, we would like to investigate DVS with probabilistic or statistical deadline guarantees. We will also explore integration with other energy-conserving mechanisms, including application energy adaptation and energy-adaptive (both real-time and best-effort) communication.

References

[1] H. Aydin, et. al., "Dynamic and aggressive scheduling techniques for power-aware real-time systems," In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'01)*, 2001.

[2] H. Aydin, et. al., "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, 53 (5), pp. 584-600, May 2004.

[3] H. Aydin and Q. Yang, "Energy-responsiveness tradeoffs for real-time systems with mixed workload," In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, 2004.

[4] T. D. Burd and R. W. Brodersen,"Energy efficient CMOS microprocessor design," In *Proceesings of the 28th Annual Hawaii International Conference on System Sciences. Volumn 1: Architecture* (Los Alamitos,

CA, USA, Jan. 1995), T. N. Mudge and B. D. Shriver, Eds., IEEE Computer Society Press, pp. 288-297.

[5] F. Gruian, "Hard real-time scheduling for low energy using stochastic data and DVS processors," In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01* (Huntington Beach, CA, Aug. 2001).

[6] W. Kim, J. Kim, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," In *Proceedings of the Design Automation and Test in Europe (DATE'02)* (Paris, France, Mar. 2002), pp. 788-794.

[7] W. Kim, et. al., "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," In *Proceedings Real-Time and Embedded Technology and Applications Symposium RTAS'02* (June 2002).

[8] C. M. Krishna and Y. -H. Lee, "Voltage-clock-scaling techniques for low power in hard real-time systems," In *Proceedings of the IEEE Real-Time Technology and Applications Symposium* (Washington, D.C., May 2000), pp. 156-165.

[9] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," In *Proceedings of the 8th IEEE Real-Time Systems Symposium* (Los Alamitos, CA, Dec. 1987), pp. 261-270.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming for a hard real-time environment," J ACM 20, 1 (Jan. 1973), 46-61.

[11] Jane W.S. Liu, *Real-Time Systems*, Prentice-Hall, 2000.

[12] D. Mosse, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," In *Workshop on Compilers and Operating Systems for Low-Power (COLP'00)* (Philadelphia, PA, Oct. 2000).

[13] T. Pering and R. Broderson, "Energy efficient voltage scheduling for real-time operating systems," In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium RTAS'98, Work in Progress Session* (Denver, CO, June 1998).

[14] T. Pering, T. Burd, and R. Brodersen, "Voltage scheduling in the lpARM microprocessor system," In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'00* (Rapallo, Italy, July 2000).

[15] 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 System Principles SOSP'01)*, Oct. 2001, pp. 89-102.

[16] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," In *Proceedings of the Asia Pacific Design Automation Conference,* 2004.

[17] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," In *Proceedings of the 36th Design Automation Conference (DAC'99),* 1999.

[18] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *Journal of Real-Time Systems,* 1 (1), pp. 27-60, 1989.

[19] M Spuri, *Earliest Deadline Scheduling in Real-Time Systems*, Doctorate Dissertation, Scuola Superiore S. Anna, Pisa, Italy, 1995.

[20] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Journal of Real-Time Systems,* 10 (2), pp. 179-210, 1996.

[21] J. Stankovic, et. al., *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, 1998.

[22] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Transactions on Computers,* 44 (1), pp. 73-91, 1995.

[23] V. Swaminathan and K. Chakrabarty, "Real-time task scheduling for energy-aware embedded systems," In *Proceedings of the IEEE Real-Time Systems Symposium (Work-in-Progress Session)* (Orlando, FL, Nov. 2000).

[24] O. S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE*, Vol. 91, No. 7, July 2003, pp. 1-15.

[25] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," In *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI)* (Monterey, CA, Nov. 1994), pp. 13-23.