Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis

Woonseok Kim* Jihong Kim† Sang Lyul Min
School of Computer Science and Engineering
Seoul National University ENG4190, Seoul, Korea, 151-742
wskim@archi.snu.ac.kr, jihong@davinci.snu.ac.kr, symin@dandelion.snu.ac.kr

ABSTRACT

Dynamic Voltage Scaling (DVS), which adjusts the clock speed and supply voltage dynamically, is an effective technique in reducing the energy consumption of embedded real-time systems. Unlike dynamic-priority real-time scheduling for which highly effective DVS algorithms are available, existing fixed-priority DVS algorithms are less effective in energy efficiency because they are based on inefficient slack estimation methods. This paper describes an efficient on-line slack estimation heuristic for the rate-monotonic (RM) scheduling. The proposed heuristic estimates the slack times using the short term work-demand analysis. The DVS algorithm based on the proposed heuristic is also presented. Experimental results show that the proposed DVS algorithm reduces the energy consumption by 25∼42% over the existing rate-monotonic DVS algorithms.

Categories and Subject Descriptors: D.4.9 [Operating Systems]: Systems Programs and Utilities

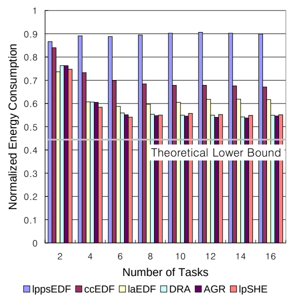
General Terms: Algorithms.

Keywords: Dynamic voltage scaling, low-power systems, real-time systems.

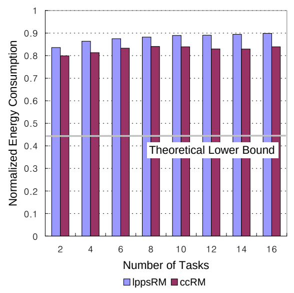
1. INTRODUCTION

Dynamic Voltage Scaling (DVS), which adjusts the supply voltage and its corresponding clock frequency dynamically, is an effective low-power design technique for embedded real-time systems. Since the energy consumption of CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is one of the most effective techniques for reducing the energy consumption.

Since lowering the supply voltage also decreases the maximum achievable clock speed [1], various DVS algorithms for hard real-time systems try to reduce the supply voltage dynamically to the lowest possible level while satisfying the systems' timing constraints. To guarantee the timing requirements of real-time tasks even in the worst case, dynamic voltage scaling can utilize only slack times (or idle times) when adjusting voltage levels. Therefore, the energy efficiency of a DVS algorithm largely depends on how accurately these slack times are estimated.



(a) EDF DVS algorithms



(b) RM DVS algorithms

Figure 1: Energy efficiency of state-of-art DVS algorithms.

Slack time analysis has been extensively investigated in real-time server systems in which aperiodic (or sporadic) tasks are jointly scheduled with periodic tasks [2, 3]. In these systems, the purpose of slack time analysis is to improve the response times of aperiodic tasks or to increase their acceptance ratio. However, since the existing slack analysis methods [2, 3] usually incur high time and/or space overheads, they are not applicable to mobile embedded systems where resources are constrained. For this reason, most existing on-line DVS algorithms for embedded systems use simple heuristics in estimating slack times.

For dynamic-priority hard real-time systems, slack times are efficiently identified with a moderate overhead, contributing to higher energy efficiency of DVS algorithms. For example, the DVS algorithms for the *earliest-deadline first* (EDF) scheduling policies described in [4, 5, 6] are known to be very effective in reducing the energy consumption. Figure 1(a) shows that the energy efficiency of these DVS algorithms, DRA, AGR [4], and lpSHE [6], is quite close to the theoretical lower bound[1] [9].

On the other hand, state-of-art DVS algorithms for *fixed-priority* real-time tasks perform less efficiently, leaving more rooms for better DVS algorithms. Figure 1(b) shows that there is a large gap between the energy efficiency of the *rate-monotonic* (RM) DVS algorithms [5, 10] and the theoretical lower bound. The poor energy efficiency of existing RM DVS algorithms can be attributed to two factors. First, RM DVS scheduling has been less extensively investigated compared to

*This work was supported in part by the Ministry of Education under the BK21 program, and by the Ministry of Science and Technology under the National Research Laboratory program. The ICT at Seoul National University provides research facilities for this study.

†This work was supported by grant No. R01-2001-00360 from the Korea Science & Engineering Foundation.

[1]The theoretical lower bounds were computed with the complete execution trace information using Yao's algorithm [7] and Quan's algorithm [8] for EDF and RM scheduling policies, respectively.

EDF DVS scheduling. We were able to identify two RM DVS algorithms only from existing literatures.

Second, it is more difficult to develop energy-efficient RM DVS algorithms compared to that of EDF DVS scheduling. For example, the *priority-based slack-stealing* method is widely used for EDF DVS algorithms in accurately estimating available slack times [4, 6]. The priority-based slack-stealing method is based on a simple heuristic: *the unused times of completed higher-priority tasks are utilized by the following lower-priority tasks*. However, since each task instance always has the same fixed priority in RM scheduling, this technique does not work as effectively as in EDF scheduling. (In EDF scheduling, the dynamically changing task priorities serve as an efficient slack distributor among the tasks.) Higher-priority tasks tend to have less slack times than lower-priority tasks in RM scheduling. It is likely that the higher the task priority is, the faster the task execution speed is. This unbalance in the execution speeds generally results in the poor energy efficiency.

In this paper, we focus on improving the on-line slack estimation part of an RM DVS algorithm. Based on the analysis results of the existing RM DVS algorithms, we propose a dynamic slack estimation method based on the short term work-demand analysis method (which significantly improves the efficiency of the slack estimation) and describe a new DVS algorithm for periodic RM real-time tasks. Experimental results show that the proposed DVS algorithm can reduce the energy consumption by 25∼42% over the existing RM DVS algorithms.

2. MOTIVATION

2.1 System model

This paper focuses on a preemptive hard real-time system in which periodic real-time tasks are scheduled under the RM scheduling policy where the shorter the period task, the higher the priority. The target variable voltage processor can scale its supply voltage and clock speed continuously within its operational ranges, $[v_{min}, v_{max}]$ and $[f_{min}, f_{max}]$, respectively. The clock speed is assumed to be adjusted along with the corresponding voltage level at each scheduling point.

A set of n periodic tasks is denoted by $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where tasks are assumed to be mutually independent. τ_i has a shorter period (i.e., a higher priority) than τ_j if $i < j$. Each task τ_i has its own period p_i and worst case execution time (WCET) w_i. The relative deadline d_i of τ_i is assumed to be equal to its period p_i. Each task activates (or releases) its instance periodically, and the j-th instance of τ_i is denoted by $\tau_{i,j}$. The first task instance of each task is assumed to be activated at $t = 0$. A task instance is denoted by a single subscript such as $\tau_{\boldsymbol{\alpha}}$ when no confusion arises. Each task instance $\tau_{\boldsymbol{\alpha}}$ has its own arrival time $r_{\boldsymbol{\alpha}}$ and absolute deadline $d_{\boldsymbol{\alpha}}$. We denote $\alpha < \beta$ if $i < k$ where $\alpha \equiv i, j$ and $\beta \equiv k, l$.

2.2 Motivational example

Consider a periodic task set $\mathcal{T}$ specified in Table 1. In addition to periods and WCETs, Table 1 shows the average case execution time (ACET) of each task.[2] Figure 2(a) shows the execution schedule under the worst case workload using f_{max} in the first hyperperiod.

Suppose that $\mathcal{T}$ is scheduled under the `ccRM` algorithm [5]. This algorithm consists of two phases, off-line and on-line phases. In the off-line phase, using the schedulability condition for RM scheduling [11], `ccRM` computes the *maximum constant speed* f_{mcs} for the given task set, which is the lowest possible clock speed that guarantees the feasible schedule of the task set [10].

[2] We assume that tasks' execution times are based on the maximum clock frequency.

Table 1: An example real-time task set $\mathcal{T}$.

	period (p_i)	WCET (w_i)	ACET (a_i)
τ_1	3	1.0	0.5
τ_2	4	1.0	0.5
τ_3	6	2.0	1.0

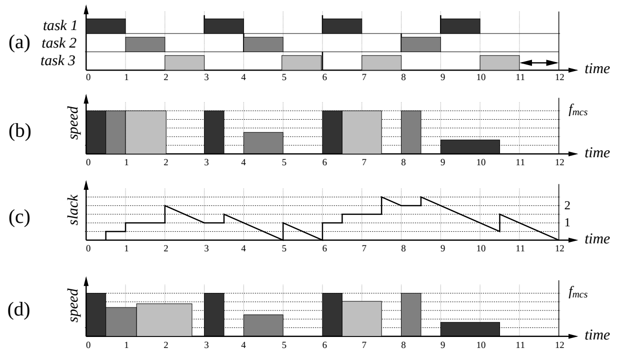


Figure 2: A voltage scheduling example

However, in the case of $\mathcal{T}$ in Table 1, f_{mcs} is computed to be $1.0 \times f_{max}$.

In the on-line phase, under `ccRM`, slack times exist if the earliest arrival time of the next task instance is later than the worst-case completion time of the currently activated task instances. When the slack time exists, the clock speed is adjusted so that the activated task instances complete their execution just before the arrival time of the next task.

Figure 2(b) shows the speed schedule under the `ccRM` algorithm for the example task set $\mathcal{T}$ assuming that the actual execution time of each task is equal to its ACET. This example illustrates that the `ccRM`'s slack estimation heuristic is overly conservative. When the first scheduled task instance $\tau_{1,1}$ completes its execution at $t = 0.5$ with f_{mcs}, there is a slack time of 0.5 time units, which can be used to lower the execution speed for the next scheduled task instance $\tau_{2,1}$. Under the `ccRM` algorithm, however, $\tau_{2,1}$ is scheduled with f_{mcs}, because the amount of imposed work at $t = 0.5$ is 3 ($= \frac{w_{2,1}+w_{3,1}}{1.0}$), which is larger than the length of the interval between $t = 0.5$ and the next task arrival time (NTA $= a_{1,2} = 3$). When $\tau_{2,1}$ is completed and $\tau_{3,1}$ is scheduled at $t = 1.0$, there exists a slack time of 1 time unit. However, $\tau_{3,1}$ cannot be scheduled with a lower speed than f_{mcs} because the arrival time of the next task instance $\tau_{1,2}$ is earlier than the worst case completion time of $\tau_{3,1}$. Similarly, all other task instances, except for two task instances $\tau_{2,2}$ and $\tau_{1,4}$, cannot be scheduled with a speed lower than f_{mcs}.

The inability of using lower speeds for the example task set $\mathcal{T}$ under `ccRM` results from an inefficient slack estimation method used in this algorithm. As shown in Figure 2(c), there exist slack times at most scheduling points. To improve the efficiency of slack estimation in RM scheduling, we may extend the `ccRM` algorithm by adopting the priority-based slack stealing method [4, 6]. However, this method may not work well either in RM scheduling where the priority of a task is fixed. Since each task instance takes its slack time mainly from completed higher-priority tasks under the priority-based slack stealing method, high-priority tasks have less sources for slack times than low-priority tasks. Thus, there exist a large unbalance in slack usage among tasks. Consequently, such an unbalance in slack usage results in an uneven voltage schedule, further limiting the energy efficiency in RM scheduling. Figure 2(d) shows the speed schedule when the priority-based slack stealing method is used for the slack estimation in RM scheduling for the same example. As shown in the figure, the highest-priority task, which has no high-priority task, runs with high speeds in three instances (out of four activations).

The main goal in this paper is to devise an efficient and accurate slack estimation method for RM scheduling. Our key observation leading to the new slack estimation technique is that, for RM scheduling, the priority-based slack stealing method should be augmented with the work-demand analysis for solving the uneven slack source problem.

3. BASIC IDEA

At time t, the *slack* (or *laxity*) of a task $\tau_{i,j}$ with deadline at $d_{i,j}$ is equal to $d_{i,j} - t - w_{i,j}$ [12]. If additional work w_{other} from other tasks should be done before $d_{i,j}$, the slack time of $\tau_{i,j}$ is reduced by w_{other}.

In order to accurately estimate the slack time, w_{other} should be computed correctly. However, as shown in [2, 3], identifying the exact w_{other} values requires to consider all the timing constraints of subsequent task instances in a hyperperiod. Since such a procedure incurs high time and space overhead, it cannot be used for the on-line slack estimation. Instead, we calculate w_{other} approximately. The goal of short-term work-demand analysis is to enlarge the available slack time of the scheduled task by delaying the schedule of lower-priority tasks in near future as late as possible.

Suppose that there are n periodic tasks, and $\tau_{i,j}$ $(1 < i < n)$ is scheduled at t. The required work to be processed before $d_{i,j}$ can be classified into the following three types:

- $w_{i,j}$: the work required by the scheduled task itself.
- $H_{i,j}(t)$: the work required by the higher-priority tasks that are activated during $[t, d_{i,j}]$. ($H_{i,j}(t)$ should be processed before $d_{i,j}$ because these tasks will preempt $\tau_{i,j}$.)
- $L_{i,j}(t)$: the work required by the lower-priority tasks that were activated before t or will be activated during $[t, d_{i,j}]$. ($L_{i,j}(t)$ includes part of work from the lower-priority tasks.)

Once $H_{i,j}(t)$ and $L_{i,j}(t)$ are estimated, the available execution time for the scheduled task $\tau_{i,j}$ can be computed.

Consider the example task set in Table 1 again. When the first scheduled task instance $\tau_{1,1}$ completes its execution at $t = 0.5$, $\tau_{2,1}$ begins its execution at $t = 0.5$. Since the higher-priority task $\tau_{1,2}$ will be activated at $t = 3$ (i.e., $H_{2,1}(0.5) = w_{1,2} = 1$), 2 time units of work $(w_{2,1} + H_{2,1}(0.5))$ should be scheduled during [0.5, 4]. In addition, since $\tau_{3,1}$ cannot have 2 time units of available execution time during [4, 6] (due to $\tau_{2,2}$), part (i.e., 1 time unit) of $w_{3,1}$ should be processed before $d_{2,1}$ (i.e., $L_{2,1}(0.5) = 1$). Thus, total of 3 time units of work must be processed before $d_{2,1}$ (i.e., during [0.5, 4]). In other words, 1.5 time units are available to $\tau_{2,1}$, so that $\tau_{2,1}$ can be scheduled with lowered clock speed $\frac{1}{1.5}f_{max}$ as shown in Figure 3(a).

When $\tau_{2,1}$ completes its execution at $t = 1.25$ and $\tau_{3,1}$ is scheduled, 4 time units of work are required to be processed before $d_{3,1}$. That is, during [1.25, 6], 2 time units for $w_{2,1}$ and 2 time units for $H_{3,1}(1.25)$ should be allocated, respectively. Thus, 2.75 time units are available for $\tau_{3,1}$, and $\tau_{3,1}$ can also be scheduled with a lowered clock speed $\frac{2}{2.75}f_{max}$ as shown in Figure 3(b).

The remaining task instances can be scheduled in a similar manner. The final voltage schedule is shown in Figure 3(c). Assuming that the power consumption is proportional to the square of the clock speed, the schedule in Figure 3(c) consumes 30% less energy than the schedule in Figure 2(b) under the `ccRM` algorithm.

As shown in Figure 3(c), the slack times are more evenly distributed under the proposed method (cf., Figure 2(d)). Under

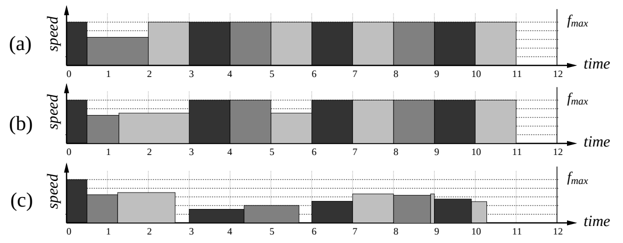


Figure 3: A voltage scheduling example.

the priority-based slack stealing method, more slack times are given to low-priority tasks than to high-priority tasks. However, in the proposed method, slack times are more evenly distributed to all tasks. For a high-priority task τ_h, small slack times are inherited from higher-priority tasks but few tasks preempt the allocated time interval for the task τ_h. On the other hand, for a low-priority tasks τ_l, large slack times are passed over to τ_l from higher-priority tasks. However, τ_l's allocated interval is frequently preempted by higher-priority tasks. Therefore, we can expect an even slack distribution, compared to the voltage schedule from the priority-based slack stealing method.

In this example, the required work needed to be processed before the scheduled task's deadline is estimated by examining tasks in the future on instance-by-instance basis. Although such a procedure might be easy to compute in this simple example, it becomes complex as the number of task instances increases. Therefore, we present a heuristic for this slack estimation procedure in the following section.

4. VOLTAGE / CLOCK SCALING USING SHORT TERM WORK DEMAND ANALYSIS

In describing the slack analysis method using the short term work-demand analysis, the following three notations are defined:

- $w_\alpha^{rem}(t)$: the remaining WCET of τ_α at t.
- $load_\alpha(t)$: the amount of work required to be processed in $[t,\ d_\alpha]$.
- $A_\alpha(t)$: the available execution time of τ_α which is scheduled at t.

The goal is to estimate the available execution time $A_\alpha(t)$ for the task τ_α scheduled at t by identifying $load_\alpha(t)$.

We assume that a real-time scheduler has two queues: *waitQueue* and *readyQueue*. The *waitQueue* and the *readyQueue* contain the completed tasks and the currently activated tasks, respectively. All the tasks are initially queued in *waitQueue*. When a task is activated, the task is moved from *waitQueue* to *readyQueue*, and the remaining WCET of τ_α is set to w_α, i.e., $w_\alpha^{rem}(t) = w_\alpha$.

Among the tasks in *readyQueue*, the active task τ_α with the shortest period is scheduled to run under the RM scheduling policy. As τ_α executes, its $w_\alpha^{rem}(t)$ decreases and consumes its available execution time. When τ_α completes its execution, its $w_\alpha^{rem}(t)$ is reset to 0.

Suppose that there are n periodic tasks (τ_1 and τ_n have the highest priority and the lowest priority, respectively). When τ_α is executed at t, if $load_\alpha(t)$ amount of work should be completed before d_α, the slack time $slack_\alpha(t)$ can be computed as $d_\alpha - t - load_\alpha(t)$. In this equation, $load_\alpha(t)$ consists of three types of work: (1) $w_\alpha^{rem}(t)$ for τ_α itself, (2) $H_\alpha(t)$ from the higher-priority tasks, and (3) $L_\alpha(t)$ from the lower-priority tasks. While $w_\alpha^{rem}(t)$ is the known value at each scheduling point, both $H_\alpha(t)$ and $L_\alpha(t)$ should be computed from a complex analysis. In the proposed heuristic, we compute approximate estimates of $H_\alpha(t)$ and $L_\alpha(t)$, $\widetilde{H}_\alpha(t)$ and $\widetilde{L}_\alpha(t)$, where $\widetilde{H}_\alpha(t) \geq H_\alpha(t)$ and $\widetilde{L}_\alpha(t) \geq L_\alpha(t)$ for a safe estimation on available slack times. Based on the periodicity of tasks, we compute $\widetilde{H}_\alpha(t)$ as follows:

$$\widetilde{H}_\alpha(t) = \widetilde{H}_\alpha^{past}(t) + \widetilde{H}_\alpha^{future}(t), \tag{1}$$

where $\widetilde{H}_\alpha^{past}(t)$ represents the work required by uncompleted higher-priority tasks activated before t and $\widetilde{H}_\alpha^{future}(t)$ represents the work required by higher-priority tasks activated during $[t, d_\alpha]$. $\widetilde{H}_\alpha^{past}(t)$ and $\widetilde{H}_\alpha^{future}(t)$ are computed as follows:

$$\widetilde{H}_\alpha^{past}(t) = \sum_{\tau_\kappa \in \mathcal{T}_\beta^{ACT}(t)} w_\kappa^{rem}(t)$$

and

$$\widetilde{H}_\alpha^{future}(t) = \sum_{i=1}^{\alpha-1} (\lfloor \frac{d_\alpha - \epsilon}{p_i} \rfloor - \lceil \frac{t+\epsilon}{p_i} \rceil + 1) \cdot w_i,$$

where

$$\mathcal{T}_\beta^{ACT}(t) = \{\tau_\kappa | \kappa < \beta \ and \ \tau_\kappa \in readyQueue(t)\}$$

and ϵ is the infinitesimal.

In Equation 1, since $\widetilde{H}_\alpha(t)$ is computed based on the task arrival times, it is obvious that $\widetilde{H}_\alpha(t) \geq H_\alpha(t)$.[3]

For a correct estimation of $L_\alpha(t)$, it is necessary to estimate how much of works from lower-priority tasks can be delayed beyond d_α. Unfortunately, such an estimation requires to examine all the lower-priority task instances in the future. In the proposed heuristic, we approximately estimate $L_\alpha(t)$ by examining only the current or next instance of each periodic task, reducing the computational complexity to a reasonable level.

First, in order to set the analysis scope, we compute the *upcoming deadline* ud_κ of each task τ_κ as follows:

$$ud_\kappa(t) = \begin{cases} \lceil \frac{t+\epsilon}{p_\kappa} \rceil p_\kappa & \text{if } \tau_\alpha \text{ is active at } t \\ (\lceil \frac{t+\epsilon}{p_\kappa} \rceil + 1) p_\kappa & \text{otherwise.} \end{cases}$$

The task instances which are active or activated during $[t, max\{ud_\kappa(t)\}]$ will be examined for slack estimation. For instance, suppose that there are three periodic tasks and their periods are 5, 6, and 8, respectively. If WCETs of tasks are 1, 1, and 2 time units, respectively, Figure 4(a) shows the worst case execution schedule of these tasks. When $\tau_{1,1}$ is scheduled at $t = 0$, the latest upcoming deadline at $t = 0$ is 8, and we will examine task instances which are active in [0,8].

Next, we choose task τ_β which has the earliest upcoming deadline among tasks whose priorities are lower than that of τ_α, e.g., $\tau_{2,1}$ in Figure 4(a). For τ_β, the amount of work required to be processed before τ_β's upcoming deadline $ud_\beta(t)$ can be expressed as

$$\widetilde{load}_\beta(t) = w_\beta^{rem}(t) + \widetilde{H}_\beta(t) + \widetilde{L}_\beta(t).$$

Once $\widetilde{load}_\beta(t)$ is computed, we can estimate $\widetilde{load}_\alpha(t)$ as follows. We separate two cases, when $ud_\beta(t) > d_\alpha$ (case I) and when $ud_\beta(t) \leq d_\alpha$ (case II).

Case I : When $ud_\beta(t)$ is later than d_α, $\widetilde{load}_\beta(t)$ is greater than $w_\alpha^{rem}(t) + \widetilde{H}_\alpha(t)$, i.e., the work in $w_\alpha^{rem}(t) + \widetilde{H}_\alpha(t)$ is the subset of the work in $\widetilde{load}_\beta(t)$. Defining $\Delta = \widetilde{load}_\beta(t) - w_\alpha^{rem}(t) - \widetilde{H}_\alpha(t)$, if $\Delta \leq ud_\beta(t) - d_\alpha$, Δ amount of work can be processed in $[d_\alpha, ud_\beta(t)]$, and only $w_\alpha^{rem}(t) + \widetilde{H}_\alpha(t)$ is required to be processed before d_α. Otherwise, $\Delta - (ud_\beta(t) - d_\alpha)$ amount of work should be processed before d_α. That is,

[3] If all the higher-priority tasks $\tau_{k,l}$ arrives before $d_\alpha - w_{k,l}$ or after d_α, $\widetilde{H}_\alpha(t)$ is equal to $H_\alpha(t)$. Otherwise, $\widetilde{H}_\alpha(t) \geq H_\alpha(t)$. For example, when the arrival time $r_{k,l}$ of a higher-priority task $\tau_{k,l}$ is earlier than d_α and $r_{k,l} + w_k$ is later than τ_α, only part of w_k can preempt the execution of τ_α. Moreover, when such higher-priority tasks arrive simultaneously, the degree of overestimation increases.

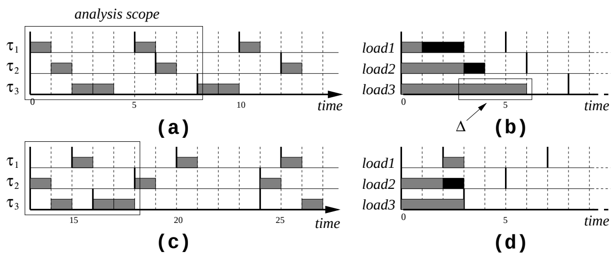


Figure 4: Short-term work-demand analysis example: (a) and (c) show RM scheduling examples at $t = 0$ and $t = 13$, respectively. (b) and (d) show the load estimation of each task at $t = 0$ and $t = 13$, respectively. (the gray-boxes mean the amount of work from higher or same priority tasks, and the black-boxes mean the amount of work from lower priority tasks.)

$$\widetilde{L}_\alpha(t) = max(0, \widetilde{load}_\beta(t) - w_\alpha^{rem}(t) - \widetilde{H}_\alpha(t) - (ud_\beta(t) - d_\alpha)),$$

$$\widetilde{load}_\alpha(t) = w_\alpha^{rem}(t) + \widetilde{H}_\alpha(t) + \widetilde{L}_\alpha(t),$$

and

$$slack_\alpha(t) = d_\alpha - t - \widetilde{load}_\alpha(t). \tag{2}$$

The remaining problem in estimating $\widetilde{load}_\alpha(t)$ is how to compute $\widetilde{load}_\beta(t)$; $L_\beta(t)$ is still unknown. $\widetilde{L}_\beta(t)$ can be estimated by the same procedure as $\widetilde{L}_\alpha(t)$. First, we identify a task τ_γ which has the earliest upcoming deadline among tasks whose priorities are lower than that of τ_β. If τ_γ is τ_n, $\widetilde{load}_\gamma(t)$ is equal to

$$\widetilde{load}_\gamma(t) = w_\gamma^{rem}(t) + \widetilde{H}_\gamma(t)$$

because there is no lower-priority tasks to consider. Otherwise $\widetilde{load}_\gamma(t)$ is given by:

$$\widetilde{load}_\gamma(t) = w_\gamma^{rem}(t) + \widetilde{H}_\gamma(t) + \widetilde{L}_\gamma(t_c).$$

Above procedure is recursively repeated until there is no lower-priority task to consider. Consequently, $\widetilde{L}_\beta(t)$ can be computed as

$$\widetilde{L}_\beta(t) = max \begin{cases} 0, \\ \widetilde{load}_\gamma(t) - w_\beta^{rem}(t) - \widetilde{H}_\beta(t) \\ -(ud_\gamma(t) - ud_\beta(t)). \end{cases}$$

Since $\widetilde{L}_\alpha(t)$ is estimated based on $\widetilde{H}_\beta(t)$, it is also true that $\widetilde{L}_\alpha(t) \geq L_\alpha(t)$. Further, the available execution time for the scheduled task τ_α can be estimated as

$$A_\alpha(t) = max(0, slack_\alpha(t)) + w_\alpha^{rem}(t). \tag{3}$$

Since $slack_\alpha(t)$ is estimated through the computation of $\widetilde{H}_\alpha(t)$ and $\widetilde{L}_\alpha(t)$, $slack_\alpha(t)$ may have a negative value when $\widetilde{H}_\alpha(t)$ and $\widetilde{L}_\alpha(t)$ are highly overestimated. In this case, we set the $slack_\alpha(t)$ as zero, and schedule τ_α with f_{max}.

Consider the example shown in Figure 4(a). When $\tau_{1,1}$ is scheduled at $t = 0$, there is no high-priority task which can preempt $\tau_{1,1}$'s execution. Even if $\tau_{1,1}$ fully utilizes the time interval [0,5] for its execution, it does not violate its timing constraint. However, in order to guarantee the feasible schedule of the lower-priority tasks, we should estimate how much lower-priority work should be done before $ud_{1,1} = 5$. In order to determine $L_{1,1}(0)$, the proposed heuristic examines the lower-priority tasks in a top-down fashion from $\tau_{2,1}$ to

$\tau_{3,1}$, and computes $L_{3,1}(0)$ and $L_{2,1}(0)$ in a bottom-up fashion from $\tau_{3,1}$ to $\tau_{2,1}$. Figure 4(b) shows the computed results. In the case of $\tau_{3,1}$, $\widetilde{load}_{3,1}(0)$ is set to the sum of $w_{3,1}$ and $\widetilde{H}_{3,1}(0)$,[4] which is shown in the gray box of Figure 4(b); 6 time units are required to be processed before $ud_{3,1} = 8$. In the case of $\tau_{2,1}$, only 3 time units of work should be processed before $ud_{2,1} = 6$ for $\tau_{2,1}$ and its higher-priority tasks, $\tau_{1,1}$ and $\tau_{1,2}$. However, during $[ud_{2,1}, ud_{3,1}]$, because only 2 time units are available, 1 time unit of additional lower-priority work ($\widetilde{L}_{2,1}(0) = \Delta - (ud_{3,1} - ud_{2,1})$), which is shown in the black-box of Figure 4(b), should be processed before $ud_{2,1}$ in order to guarantee the feasible schedule of $\tau_{3,1}$. Therefore, $\widetilde{load}_{2,1}(0)$ becomes 4. In a similar way, we can compute $\widetilde{load}_{1,1}(0)$. Consequently, 2 time units of slack time is available for $\tau_{1,1}$ and it can be scheduled with a lower clock speed $f_{clk} = \frac{w_{1,1}}{slack_{1,1}(0)+w_{1,1}} f_{max} = \frac{1}{3} f_{max}$.

Case II : When $ud_\beta(t)$ is earlier than d_α, $\widetilde{load}_\beta(t)$ must be processed before d_α. Otherwise, τ_β violates its timing constraint because τ_β can be executed after the execution of its higher-priority tasks including τ_α. Thus, in this case, we modify the execution interval for τ_α by changing the τ_α's deadline to $ud_\beta(t)$. Therefore, the slack time of τ_α is given by :

$$slack_\alpha(t) = ud_\beta(t) - t - \widetilde{load}_\beta(t). \tag{4}$$

For example, assume that, in the above example, all tasks are active except for τ_1 at $t = 13$ (Figure 4(c)) and only 1 time unit of work is left for $\tau_{3,2}$. In this case, $ud_{2,3}$(=18) is later than $ud_{3,2}$(=16), and the slack time of $\tau_{2,3}$ is estimated relative to $ud_{3,2}$. Since $\tau_{3,2}$ can be scheduled after $\tau_{2,3}$'s completion, $\tau_{2,3}$'s effective deadline is earlier than its original deadline. In Figure 4(c), $\tau_{2,3}$ should complete before $t = 14$ and $\tau_{2,3}$ cannot have any slack time (cf. Figure 4(d)).

Figure 5 summarizes the proposed slack estimation procedure. During run time, this algorithm can be executed at every scheduling point such as the activation, resumption, and completion of task instances. In estimating the available execution time for the scheduled task, only one instance is examined per a periodic task,thus the estimation algorithm having the $O(n)$ time complexity[5]. When the available execution time $A_\alpha(t)$ for τ_α at t, the clock speed can be adjusted to

$$f_{clk} = (w_\alpha^{rem}(t)/A_\alpha(t)) \times f_{max}, \tag{5}$$

and the supply voltage is adjusted accordingly.

5. EXPERIMENTAL RESULTS

To evaluate the energy efficiency of the proposed voltage scheduling algorithm, several experiments were performed using three DVS algorithms: (1) the `lppsRM` algorithm [10], (2) the `ccRM` algorithm [5], and (3) the `lpWDA` algorithm proposed in this paper. Experiments were performed using SimDVS, an integrated DVS simulation environment [9, 13]. The energy simulator in SimDVS is based on the ARM8 microprocessor core. The clock speed is scaled in the range of [8, 100] MHz with a step size of 1 MHz and the supply voltage is scaled in the range of [1.1, 3.3] V. It is assumed that the system enters into a power-down mode when the system is idle. (The power consumption in the power-down mode is assumed to be zero.) In the experiments, the voltage scaling overhead is assumed negligible both in the time delay and power consumption.

[4] $\widetilde{H}_{3,1}(0)$ includes the works from $\tau_{1,1}$, $\tau_{1,2}$, $\tau_{2,1}$, and $\tau_{2,2}$.

[5] The time complexity of the proposed heuristic varies from $O(n)$ (in the worst case) to $O(1)$ (in the best case) according to the priority of the scheduled task. When τ_i is scheduled, the time complexity is $O(n - i)$ in the worst case.

Table 2: Task sets for experiments.

	♯ tasks	WCETs (ms)	Periods (ms)	Utilization
CNC	8	0.035 ∼ 0.72	2.4∼9.6	0.489
Avionics	17	1 ∼ 9	25∼1,000	0.848
Videophone	4	1.4 ∼ 50.4	40∼66.7	0.986

Figure 6 shows the experimental results for three real-world application task sets and a number of synthesized task sets. The three real-world application task sets are derived from the Computerized Numerical Control (CNC) machine controller application, the Avionics application, and the VideoPhone application, which were used for the experiments in [10, 6]. The parameters of these applications are summarized in Table 2. In each experiment, the execution time of each task instance was randomly drawn from a Gaussian distribution[6] in the range of [BCET, WCET], where BCET is the best case execution time. The experiments were performed by varying BCET from 10% to 90% of WCET for each application. In each figure, the x-axis represents the ratio of BCET to WCET while the y-axis represents the normalized energy consumption ratio to the energy consumption of the same application running on a DVS-unaware system with a power-down mode only.

As shown in Figures 6(a)∼(c), `lpWDA` reduces the energy consumption up to 48% and 42% over `lppsRM` and `ccRM`, respectively. Since the average execution times of task instances decrease as BCET becomes smaller, the slack times of task instances increase as BCET decreases. Thus, as shown in the figures, the energy efficiency of each DVS algorithm increases as the ratio of BCET to WCET decreases. Note that the energy efficiency of the proposed `lpWDA` algorithm increases much faster than `lppsRM` and `ccRM` because the proposed `lpWDA` algorithm is more efficient in exploiting the slack times of tasks than the others.

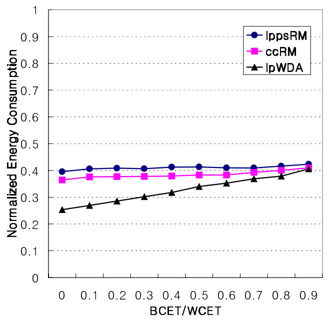
We also performed extensive experiments using synthesized application sets by varying the number of tasks in a task set whose results are given in Figure 6(d). For a given number of tasks, 100 random task sets[7] were generated, whose utilization is 0.9. In these experiments, another RM DVS algorithm `lpSHR` is evaluated together. `lpSHR` estimates the slack times using the RM extension of *priority-based slack stealing* [6, 4], and adjusts the clock speed and voltage accordingly.

The experimental results in Figure 6(d) show that `lpWDA` achieves 25∼42% more energy savings compared to the others. Figure 6(d) also shows that as the number of tasks increases, the energy efficiency of `lpWDA` increases while those of `lppsRM` and `ccRM` are not changed. This can be explained by the fact that as the number of tasks increases, `lpWDA` has more task instances from which slack times are taken. On the other hand, in `lppsRM` and `ccRM`, the slack estimation is limited to the time between the completion of a task instance and the arrival of the next task instance, which is largely independent of the number of tasks in the system.

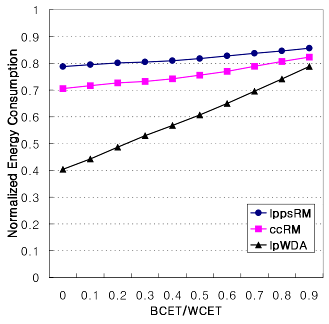
Although the energy efficiency of `lpSHR` is much better than those of `lppsRM` and `ccRM` (by 16% ∼ 22%), it is still worse than that of `lpWDA` (by 25%). As pointed out earlier, this is because higher-priority tasks tend to have less slacks than lower-priority tasks in RM scheduling. Due to such an unbalance in the amount of available slack times among the tasks, `lpSHR` cannot achieve a high energy efficiency as when the priority-based slack stealing method was used in EDF DVS algorithms.

[6] With the mean $m = \frac{\text{BCET+WCET}}{2}$ and the standard deviation $\sigma = \frac{\text{WCET-BCET}}{6}$.

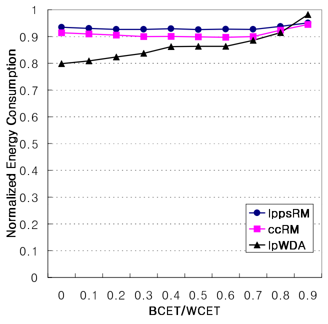
[7] The period and WCET of each task were randomly generated using the uniform distribution within the ranges of [10, 100] ms and [1, period) ms.



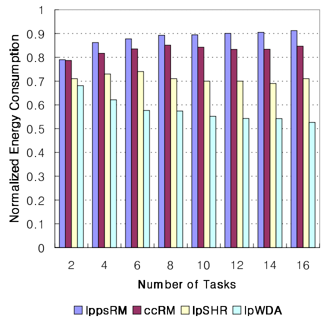
(a) CNC



(b) Avionics



(c) VideoPhone



(d) Synthesized applications

Figure 6: Experimental results.

6. CONCLUSIONS

We have presented a novel voltage scheduling algorithm for RM scheduling based on an efficient slack estimation heuristic. We have discussed why existing RM DVS algorithms do not perform well and explained how we can overcome such limitations. As a heuristic, a slack estimation method using the short term work-demand analysis was introduced, and a new RM DVS algorithm, `lpWDA`, was proposed. Experimental results show that the `lpWDA` algorithm reduces the energy consumption up to 40% over the existing algorithms.

7. REFERENCES

[1] T. Sakurai and A. Newton. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas. *IEEE Journal of Solid State Circuits*, 25(2):584–594, 1990.

[2] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 110–123, December 1992.

[3] T. S. Tia, J. W. S., and M. Shankar. Algorithms and Optimality of Scheduling of Soft Aperiodic Requests in Fixed-Priority Preemptive Systems. *Journal of Real-Time Systems*, 10(1):23–43, 1996.

[4] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 95–105, December 2001.

[5] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of 18th ACM Symposium on Operating Systems Principles*, pages 89–102, October 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 Design, Automation and Test in Europe*, pages 788–794, March 2002.

[7] F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In *Proceedings of the IEEE Foundations of Computer Science*, pages 374–382, 1995.

[8] G. Quan and X. S. Hu. An Optimal Voltage Schedule for Real-Time Systems on a Variable Voltage Processor. In *Proceedings of the Design, Automation and Test in Europe*, pages 782–787, March 2002.

[9] W. Kim, D. Shin, J. Jeon, J. Kim, and S. L. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, pages 219–228, September 2002.

[10] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 365–368, November 2000.

[11] J. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm : Exact Characterization and Average Case Behavior. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[12] W.-S. Liu. *Real-Time Systems*. Prentice Hall, Englewood Cliffs, NJ, June 2000.

[13] D. Shin, W. Kim, J. Jeon, J. Kim, and S. L. Min. SimDVS: An Integrated Simulation Environment for Performance Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceedings of Workshop on Power-Aware Computer Systems*, February 2002.

```
Algorithm :   Estimate the available execution time
              and set the voltage/clock speed for $\tau_{\alpha}$

1.  Initially, put all tasks into readyQ and, for each task $\tau_\alpha$,
2.      set $ud_\alpha = p_\alpha$ and
3.      $H_\alpha(t) = \sum_{\tau_\kappa \in \mathcal{T}_\alpha^{ACT}(t)} w_\kappa^{rem}(t)$
            $+ \sum_{i=1}^{\alpha-1} (\lfloor \frac{ud_\alpha - \epsilon}{p_i} \rfloor - \lceil \frac{t+\epsilon}{p_i} \rceil) \cdot w_i$
4.      where, $\mathcal{T}_\alpha^{ACT}(t) = \{\tau_\kappa | \kappa < \alpha$ and $\tau_\kappa \in$ readyQ(t) $\}$
5.  When a task $\tau_\alpha$ is activated, set $w_\alpha^{rem}(t) = w_\alpha$
6.  When a task $\tau_\alpha$ is completed or preempted, call UpdateLoadInfo()
7.  When a task $\tau_\alpha$ is scheduled for execution,
8.  (1) call CalcSlackTime() to get slack time $slack_\alpha(t)$
9.  (2) set the clock frequency as
        $f_{clk} = \frac{w_\alpha^{rem}(t)}{slack_\alpha(t) + w_\alpha^{rem}(t)} \cdot f_{max}$, and
10. (3) set the voltage accordingly.
    ------------------------------------------------
11. Function CalcSlackTime()
12. Input : the active task $\tau_\alpha$, waitQ, readyQ, and current time $t$
13. Output : the slack time $slack_\alpha(t)$ for $\tau_\alpha$
14. Identify the task $\tau_\beta$ that has the earliest upcoming deadline
    among tasks whose priorities are not higher than that of $\tau_\alpha$
15. $L_\beta(t)$ = CalcLowerPriorityWork($\tau_\beta$, $t$)
16. $load_\beta(t) = w_\beta^{rem}(t) + H_\beta(t) + L_\beta(t)$
17. $slack_\alpha(t) = max(0, ud_\beta - t - load_\beta)$
18. return $(slack_\alpha(t))$
    ------------------------------------------------
19. Function CalcLowerPriorityWork()
20. Input : a reference task $\tau_\beta$ and current time $t$
21. Output : the amount of lower-priority work needed to be
             done before $ud_\beta$
22. if $\tau_\beta$ is identical to $\tau_n$ then return 0 end if
23. Identify the task $\tau_\gamma$ that has the earliest upcoming deadline
    among tasks whose priorities are lower than that of $\tau_\beta$
24. $L_\gamma(t)$ = CalcLowerPriorityWork($\tau_\gamma, t$)
25. $load_\gamma(t) = w_\gamma^{rem}(t) + H_\gamma(t) + L_\gamma(t)$
26. $L_\beta(t) = max(0, load_\gamma(t) - w_\beta^{rem}(t) - H_\beta(t) - ud_\gamma + ud_\beta)$
27. return $L_\beta(t)$
    ------------------------------------------------
28. Function UpdateLoadInfo()
29. Input : the completed or preempted task $\tau_\alpha$, and the
    amount of work $w_{done}$ done for $\tau_\alpha$ in the previous schedule
30. if (COMPLETION) then
31.     $ud_\alpha = ud_\alpha + p_\alpha$
32.     $H_\alpha(t) = \sum_{i=1}^{\alpha-1} (\lfloor \frac{ud_\alpha - \epsilon}{p_i} \rfloor - \lceil \frac{t+\epsilon}{p_i} \rceil) \cdot w_i$
33.     loop from $\kappa = \alpha - 1$ until $\kappa = n$ by increasing $\kappa$
34.         $H_\kappa(t) = H_\kappa(t) - w_\alpha^{rem}(t)$
35.     end loop
36.     $w_\alpha^{rem}(t) = 0$
37. else (PREEMPTION)
38.     $w_\alpha^{rem}(t) = w_\alpha^{rem}(t) - w_{done}$
39.     loop from $\kappa = \alpha - 1$ until $\kappa = n$ by increasing $\kappa$
40.         $H_\kappa(t) = H_\kappa(t) - w_{done}$
41.     end loop
42. end if
```

Figure 5: Voltage scaling algorithm.