Dynamic Voltage Scaling of Mixed Task Sets in Priority-Driven Systems

Dongkun Shin and Jihong Kim, *Member, IEEE*

***Abstract*—This paper describes dynamic voltage scaling (DVS) algorithms for real-time systems with both periodic and aperiodic tasks. Although many DVS algorithms have been developed for real-time systems with periodic tasks, none of them can be used for a system with both periodic and aperiodic tasks because of the arbitrary temporal behaviors of aperiodic tasks. This paper proposes off-line and on-line DVS algorithms that are based on existing DVS algorithms. The proposed algorithms utilize the execution behaviors of scheduling servers for aperiodic tasks. Since there is a tradeoff between the energy consumption and the response time of aperiodic tasks, the proposed algorithms focus on bounding the response time degradation of aperiodic tasks although they delay the response time by stretching the task execution to get high energy savings in mixed task sets. Experimental results show that the proposed algorithms reduce the energy consumption by 48% and 35% over the non-DVS scheme under rate monotonic (RM) scheduling and earliest deadline first (EDF) scheduling, respectively.**

***Index Terms*—Dynamic voltage scaling, low-power design, mixed task systems, real-time systems, task scheduling.**

I. Introduction

MANY practical real-time applications require aperiodic tasks as well as periodic tasks. For example, consider multimedia applications such as a Moving Pictures Expert Group (MPEG) player. While these applications have stringent periodic performance requirements (e.g., 30 frames/s), they also need to serve aperiodic user requests (e.g., volume control and playlist editing) with reasonable response times. The flight system also has both periodic and aperiodic tasks. The system must respond to the pilot's command while continuing to execute the control tasks that fly the airplane. Even when there is no user request, the real-time systems based on automatic memory reclamation (with garbage collection) require aperiodic tasks. The garbage collector (GC) is invoked when the available memory size is below a specific threshold. Generally, periodic tasks are time driven with hard deadlines and aperiodic tasks are event driven (i.e., activated at arbitrary times) with short response times. In this paper, a system with both periodic and aperiodic tasks is called a mixed task system.

In implementing mixed task systems, there are two major design objectives. The first objective is to maintain the schedulability of (feasible) periodic tasks under the presence of aperiodic tasks. That is, aperiodic tasks should not prevent periodic tasks from completing before their deadlines. The second objective is to serve aperiodic tasks with reasonable average response times. To satisfy these two design objectives, many scheduling algorithms such as deferrable servers (DS) [1], sporadic servers (SS) [2], total bandwidth servers (TBS) [3], and constant bandwidth servers (CBS) [4] had been proposed. Since they set aside some portion of the system utilization for aperiodic tasks, they are called bandwidth-preserving servers [5]. In this paper, the authors consider as the third design parameter the energy consumption of mixed task sets. With the added energy consumption requirement, the overall design objective is to minimize the total energy consumption of both periodic tasks and aperiodic tasks while satisfying the previous two requirements.

Among many low-power design approaches, this paper focuses on dynamic voltage scaling (DVS) [6]. Recently, DVS has been accepted as an efficient low-power technique for real-time systems. Under this scheme, when the required performance of the target system is lower than the maximum performance, the supply voltage and the clock speed can be reduced to minimize the energy consumption. For hard real-time systems where timing constraints must be strictly satisfied, a fundamental energy-delay tradeoff makes it more challenging to adjust the supply voltage dynamically while minimizing the energy consumption and guaranteeing the timing requirements. Many on-line voltage-scheduling algorithms exist for hard real-time systems [7]–[10]. Since most of these algorithms assume that the system consists of periodic hard real-time tasks only and the task release times are known *a priori*, they estimate slack times based on the known release times and stretch the task execution using the estimated slack times. Generally, the more slack times a DVS algorithm can estimate, the better energy efficiency the DVS algorithm can have.

However, when a DVS algorithm is applied to mixed task sets, the DVS algorithm should tackle the arbitrary behaviors of aperiodic tasks. Fortunately, since a bandwidth-preserving server limits the execution of aperiodic tasks within its allocated bandwidth, DVS algorithms can estimate slack times considering the characteristics of bandwidth-preserving servers. Although the energy efficiency of a DVS algorithm for mixed task sets is also related to how much slack times it can find, the presence of aperiodic tasks in the mixed task sets raises the tradeoff between the energy consumption of the total system and the response time of aperiodic tasks. If the response time of aperiodic tasks is ignored, the most energy-efficient solution is not to serve aperiodic tasks, thus further reducing the energy

Manuscript received July 25, 2004; revised December 21, 2004. This paper was supported by University IT Research Center Project. This paper was recommended by Associate Editor R. Camposano.

D. Shin is with Samsung Electronics Co., Seoul, Korea (e-mail: dongkun.shin@samsung.com).

J. Kim is with the School of Computer Science and Engineering, Seoul National University, Seoul, Korea (e-mail: jihong@davinci.snu.ac.kr).

Digital Object Identifier 10.1109/TCAD.2005.853706

consumption of periodic tasks. However, it is obvious that such a solution will not be acceptable. Therefore, the main challenge in designing DVS algorithms for the mixed task sets is to bound the response times of aperiodic tasks while reducing the energy consumption of periodic tasks as well as aperiodic tasks.

Recently, DVS algorithms have been proposed for mixed task systems satisfying this requirement for the DS and the TBS [11]. Based on the previous work, the DVS algorithms for mixed task systems were improved in this paper. The main contributions of this paper can be summarized as follows. First, more aggressive DVS algorithms are proposed for DS and SS, which can handle the mixed task sets. The new DVS algorithms can reduce the energy consumption by 16% over the existing DVS algorithm in [11]. Second, DVS algorithms for CBS are proposed. Third, a new slack distribution method for DVS algorithms is also proposed. By assigning slack times only to periodic tasks and executing aperiodic tasks at full speed, the DVS algorithm can make better response times with small increases in the energy consumption. Lastly, the authors prove that the proposed algorithms for DS, SS, and CBS always limit the response time penalties of aperiodic tasks, which were traded for the lower energy consumption of the mixed task set.

The rest of this paper is organized as follows. Section II summarizes related works on aperiodic task scheduling and recent efforts to integrate dynamic voltage scheduling into aperiodic task scheduling. Section III presents the target processor model and the target problem. The proposed dynamic (on-line) DVS algorithms are described in Section IV while the static (off-line) DVS algorithm is presented in Section V. In Section VI, the experimental results are discussed. Section VII concludes with a summary and future works.

II. Related Works

This section reviews the main approaches for scheduling a mixture of aperiodic tasks and periodic hard real-time tasks.

The easiest way to prevent aperiodic tasks from interfering with periodic hard real-time tasks is to schedule them as background tasks executing only at times when there is no periodic task ready for execution. Although this method guarantees the schedulability of a periodic task, the execution of aperiodic tasks may be delayed and their response times are prolonged unnecessarily.

Another approach is to use a dedicated scheduling server that handles aperiodic tasks. The server is characterized by an ordered pair (Q_s, T_s), where Q_s is the maximum budget and T_s is the period of the server. The utilization of a scheduling server was denoted as $U_s(= Q_s/T_s)$. The simplest server is the polling server (PS). PS is ready for execution periodically at integer multiples of T_s and is scheduled together with periodic tasks in the system according to the given priority-driven algorithm. Once PS is activated, it executes any pending aperiodic requests within the limit of its budget Q_s. If no aperiodic requests are pending, PS immediately suspends its execution until the start of its next period. Since PS is exactly identical to a periodic task that has the period T_s and the worst-case execution time (WCET) Q_s, the schedulability of the system can be tested using the traditional rate monotonic (RM) or earliest deadline first (EDF) schedulability test. However, if an aperiodic task arrives after PS examines its aperiodic task queue, it should wait until the next activation time of PS.

For this reason, several bandwidth-preserving servers are proposed. They preserve the execution budget when there are no pending aperiodic tasks and execute later if any aperiodic task arrives. The DS [1] is the simplest of bandwidth-preserving servers. DS also serves aperiodic tasks within the limit of its budget Q_s. At every integer multiplies of T_s, the budget is replenished to Q_s. Unlike PS, DS can service an aperiodic request at any time as long as the budget is not exhausted. Although this feature of DS provides better performance than that of PS, a lower-priority periodic task could miss its deadline even if the total utilization of the system with n tasks is not greater than $n(2^{1/n} - 1)$ because DS can defer its execution. To solve this problem, the SS [2] was proposed. SS has a different replenishment rule for its budget that ensures that each SS with period T_s and budget Q_s never demands more processor time than the periodic task (Q_s, T_s) in any time interval. Consequently, the system designer can treat an SS exactly like the periodic task (Q_s, T_s) when they check for the schedulability of the system.

The parameters of bandwidth-preserving servers T_s and Q_s should be carefully determined. Generally, T_s determines the priority of the scheduling server for aperiodic tasks. As T_s is small, the scheduling server has a higher priority. This can provide the shorter response times of aperiodic tasks if the workloads of aperiodic tasks are smaller than Q_s. However, for the small T_s, a small Q_s should be used to sustain the utilization of the scheduling server below the available utilization $(U_s = Q_s/T_s)$. Moreover, as T_s decreases, the number of task preemptions increases.

Although there are modified DS and SS algorithms for EDF scheduling, DS and SS are mainly used for RM scheduling due to the complexity of the modified algorithms. For EDF scheduling, TBS [3] is more suitable. TBS is characterized by U_s, which is the utilization of TBS. When an aperiodic task arrives, TBS assigns a deadline to the task such that the utilization of the aperiodic task is equal to U_s. Since TBS assigns the deadline using the WCET of the aperiodic task, TBS cannot be used if there are no information on the WCETs of aperiodic tasks.

Recently, CBS [4] was proposed to solve the problem of TBS. CBS, specified by (Q_s, T_s), guarantees that its contribution to the total utilization factor is no greater than $U_s = (Q_s/T_s)$ using its special budget replenishment and deadline assignment rules. The detailed mechanism of CBS is explained in Section IV-B.

A different approach for scheduling aperiodic tasks is the slack-stealing technique [12]. It steals all available slacks from periodic tasks and gives it to aperiodic tasks. Although it provides better performance than the server approaches, i.e., minimizes response times of aperiodic requests, its complexity is very high. In addition, since the main idea of slack stealing is to give as much as possible time to aperiodic tasks executing periodic tasks at full speed, slack stealing is improper to be integrated with DVS algorithms. So, this paper concentrates on server techniques.

Recently, several researchers have proposed DVS algorithms for mixed task sets. Yuan and Nahrstedt [13] proposed DVS

TABLE I
VARIABLE-VOLTAGE PROCESSORS

Processor	Clock Range (megahertz)	Voltage Range (volt)	Transition Time (microsecond)
Transmeta's Crusoe [18]	200 to 700	1.1 to 1.65	300
AMD's Mobile K6 [19]	192 to 588	0.9 to 2.0	200
Intel PXA250 [20]	100 to 400	0.85 to 1.3	500
Compaq's Itsy [21]	59.0 to 206.4	1.0 to 1.55	189

algorithms for another kind of mixed task sets, which consist of sporadic tasks and aperiodic tasks. The sporadic tasks[1] arrive at arbitrary times and have soft deadlines. They only handled CBS.

Doh *et al.* [14] investigated the problem of allocating both energy and utilization for mixed task sets that consist of periodic tasks and aperiodic tasks. They used TBS and considered the static (off-line) scheduling problem only. Given the energy budget, their algorithm finds voltage settings for both periodic and aperiodic tasks such that all periodic tasks are completed before their deadlines and all aperiodic tasks can attain minimal response times. While Doh *et al.*'s algorithm is an off-line static speed assignment algorithm under the EDF scheduling policy, this paper proposes both on-line and off-line algorithms.

Recently, [15] introduced several on-line voltage-scheduling algorithms for a mixed workload. They proposed algorithms similar to the ones here.[2] However, while they handled only TBS under the EDF scheduling policy, this paper proposed algorithms for DS, SS, and CBS under both RM and EDF scheduling policies. In addition, the algorithms here guarantee the constraint on response time.

III. Problem Formulation

A. Target System Model

Recently, many variable-voltage processors have been announced. Table I shows the representative commercial variable-voltage processors. Most variable-voltage processors provide software mechanisms for users to be able to control the voltage and clock level. It was assumed that the voltage and clock levels of the target processor can also be controlled at the operating system level. In this paper, the execution time of a task is assumed to be proportional to the processor speed, ignoring the external memory access time on a cache miss. Since the clock speed of an external memory is not changed by DVS, the memory access time is fixed irrespective of the processor speed. Although it is assumed that there is no cache miss for a simple modeling in this paper, the parametric timing models such as [16] can be used for real systems.

The target processor here is different from existing commercial variable-voltage processors in several aspects. However, the proposed techniques can be extended to support more complex processors. For example, as shown in Table I, real variable-voltage processors have voltage transition overheads. In addition, these processors provide finite numbers of voltage and clock levels within the voltage/clock range specified. The voltage transition overhead has been generally ignored in task-level voltage scheduling because the overhead time can be included into the WCET of a task [7]. The voltage transitions take place at only task boundaries in the OS-level DVS. Although there are additional voltage transitions when a low-priority task is preempted by and resumed from a high-priority task that has a different voltage level, the voltage transition overhead can be included into the WCET of the high-priority task. Thus, a DVS algorithm can be used without modifications even for real variable-voltage processors. So, this paper does not consider the voltage transition time. In addition, it was assumed that the variable-voltage processor provides any clock speed and voltage between the minimum and maximum values ($f_{\min}$ and $f_{\max}$) for a simple modeling. For real variable-voltage processors that provide finite numbers of voltage and clock levels, the authors can use the dithering or the greedy approach to transform a clock speed generated by the proposed DVS algorithms into available speed levels provided by the real processor [17].

In some cases, DVS may increase the energy consumption over a non-DVS version. DVS can increase the number of task preemptions up to 500% over non-DVS executions mainly due to the increased task execution times [22]. The preemption overhead may increase the energy consumption in memory subsystems and the lengthened task lifetime may increase the energy consumption in system devices. However, there exist techniques that take the task preemption into account when adjusting a supply voltage using the delayed preemption technique [22]. The proposed approach can be extended using these techniques to reduce the preemption overhead.

The energy consumption of system devices such as memory and LCD was also not considered. Generally, they have a fixed supply voltage. So, the device energy consumption is not related with a DVS algorithm. However, as shown in Table II, for most benchmark programs, the power consumption of a processor core occupies the largest portion of the total power consumption [23]. Moreover, device power consumption can be reduced by power management techniques that change the power state of an idle device into a low-power state [24]. So, it is important to reduce the energy consumption of the processor core, thus only the processor core was taken into account.

TABLE II
POWER BREAKDOWN OF THE ITSY COMPUTER [23]

Benchmark	P_{core} (percent)	P_{main} (percent)	P_{LCD} (percent)	P_{DRAM} (percent)
MPEG-1	28.8	16.5	0.5	28.4
DECtalk	33	24.7	0.8	23
WAV	18.7	24.6	1.1	5.5
Idle	49	17.6	3.4	15.3

P_{core}	:	StrongARM SA-1100 core
P_{main}	:	Digital logics in mainboard
P_{LCD}	:	LCD and LCD backlight
P_{DRAM}	:	DRAM

[1]Yuan and Nahrstedt [13] say that their target system is a mix of soft real-time (SRT) multimedia and best-effort applications. However, their definition for SRT tasks is same to sporadic tasks.

[2]Their work and the work here are done independently at the same time. Especially, the mutual reclaiming (MRS) scheme in [15] is same to the workload-based slack estimation (WSE) scheme here.

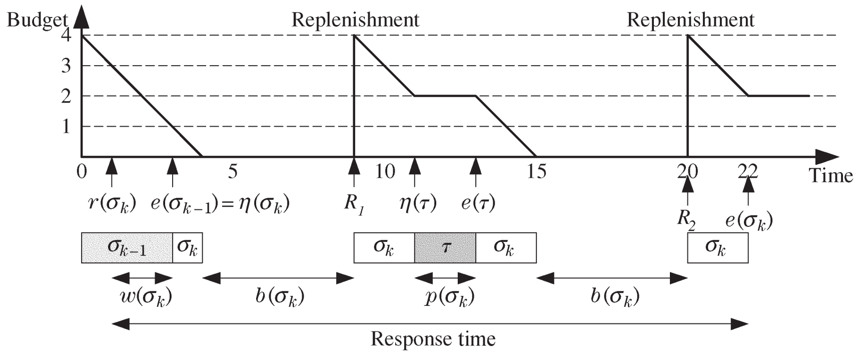


Fig. 1. Response time of aperiodic task.

B. Dynamic Speed Assignment Problem

It was assumed that a mixed task set $\mathcal{T}$ consists of n periodic tasks $\tau_1, \ldots, \tau_n$ and an aperiodic task σ. The aperiodic task σ is serviced by a scheduling server S. The scheduling server S is characterized by an ordered pair (Q_s, T_s). During the execution of aperiodic tasks, the budget of S is consumed. q_s is used to denote the remaining budget of S. The budget q_s is set to Q_s at each replenishment time. S is scheduled together with periodic tasks in the system according to the given priority-driven algorithm. Once S is activated, it executes any pending aperiodic requests in FIFO queue within the limit of its remaining budget q_s. The authors denote the release (arrival) time, the start time, and the completion time of σ as $r(\sigma)$, $\eta(\sigma)$, and $e(\sigma)$, respectively.

A periodic task τ_i is specified by (C_{τ_i}, T_{τ_i}), where C_{τ_i} and T_{τ_i} are the worst-case execution cycles (WCEC) and the period of τ_i, respectively. It was assumed that periodic tasks have relative deadlines equal to their periods. The jth instance of τ_i and the kth instance of σ are denoted by $\tau_{i,j}$ and σ_k, respectively. The authors assume that the aperiodic task instances $\sigma_1, \ldots, \sigma_m$ are executed during the hyper period H of periodic tasks. The operating speed of a periodic task τ_i (an aperiodic task σ_k) is $s(\tau_i)(s(\sigma_k))$. $s(\tau_i)(s(\sigma_k))$ can be any value between $f_{\min}$ and $f_{\max}$. It was assumed that $f_{\max} = 1$ and $f_{\min} > 0$.

If an aperiodic task instance σ_k can be serviced without any interference by periodic tasks or another aperiodic task instance, the response time of the aperiodic task σ_k is $c(\sigma_k)/s(\sigma_k)$, where $c(\sigma_k)$ and $s(\sigma_k)$ are the number of execution cycles and the clock speed of σ_k, respectively. However, the execution of the aperiodic task σ_k is delayed due to the following factors.

1) Budget delay: σ_k should wait until the next replenishment time if q_s of the scheduling server S is 0. The authors define the budget delay formally as the sum of time intervals between $r(\sigma_k)$ and $e(\sigma_k)$ of an aperiodic task σ_k where $q_s = 0$.
2) Queuing delay: σ_k should wait until the completion time of the aperiodic task instances released before σ_k. The authors define the queuing delay formally as the sum of time intervals between $r(\sigma_k)$ and $\eta(\sigma_k)$ of an aperiodic task σ_k where $q_s > 0$.
3) Preemption delay: σ_k should wait until the completion time of the periodic tasks that have higher priorities than the priority of S. The authors define the preemption delay formally as the sum of time intervals between $\eta(\sigma_k)$ and $e(\sigma_k)$ of an aperiodic task σ_k where $q_s > 0$ and σ_k is not executed.

The authors denote the budget delay, the queuing delay, and the preemption delay as $b(\sigma_k)$, $w(\sigma_k)$, and $p(\sigma_k)$, respectively. Then, the response time of σ_k can be represented as $c(\sigma_k)/s(\sigma_k) + b(\sigma_k) + w(\sigma_k) + p(\sigma_k)$. Fig. 1 illustrates these three kinds of delays. When an aperiodic task σ_k, whose execution cycles are seven time units, arrived at the time $r(\sigma_k)$, it waits until the task σ_{k-1} is completed $(e(\sigma_{k-1}))$. When σ_k consumes all the budget, it also waits until the budget replenishment time R_1. At the time $\eta(\tau)$, σ_k is preempted by the periodic task τ until $e(\tau)$. The response time of σ_k is $c(\sigma_k) + b(\sigma_k) + w(\sigma_k) + p(\sigma_k) = 7 + 10 + 2 + 2 = 21$.

If the response time of σ_k is t in the non-DVS scheme, the response time will be increased to $t + D(\sigma_k)$ by the DVS algorithm because $s(\sigma_k)$, $b(\sigma_k)$, $w(\sigma_k)$, and $p(\sigma_k)$ are changed. The increase $D(\sigma_k)$ in the response time is called the response time delay. The authors are to bound $D(\sigma_k)$ by the maximum response time delay δ.

Therefore, the objective is to minimize the total energy consumption of both periodic and aperiodic tasks using a DVS algorithm while satisfying the timing constraints of periodic tasks and bounding the response time delay. Consequently, the problem of dynamic speed assignment problem for mixed task systems (DSAMTS) can be formulated as

Dynamic Speed Assignment Problem

Given $\mathcal{T} = \{\tau_1, \ldots, \tau_n, \sigma\}$, S and δ,
find $s(\tau_{1,1}), \ldots, s\left(\tau_{n,\frac{H}{T_{\tau_n}}}\right)$ and $s(\sigma_1), \ldots, s(\sigma_m)$ such that
$E = \sum_{i=1}^{n} \sum_{j=1}^{H/T_{\tau_i}} E(\tau_{i,j}) + \sum_{k=1}^{m} E(\sigma_k)$ is minimized
subject to $\forall i, j, e(\tau_{i,j}) \leq jT_{\tau_i}$ and $\forall k, D(\sigma_k) \leq \delta$

where $s(\tau_{i,j})$, $E(\tau_{i,j})$, and $e(\tau_{i,j})$ are the clock speed, the energy consumption, and the completion time of the task instance $\tau_{i,j}$, respectively. $E(\sigma_k)$ denotes the energy consumption of the aperiodic task instance σ_k.

This paper proposes DVS algorithms that provide solutions for the DSAMTS problem where δ can be represented with the

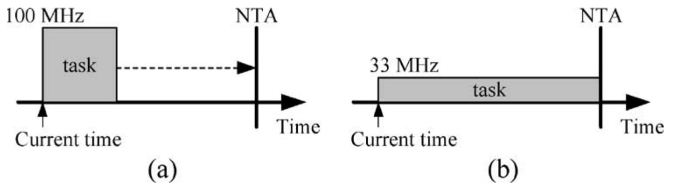


Fig. 2. Stretching-to-NTA DVS algorithm. (a) Before stretching. (b) After stretching.

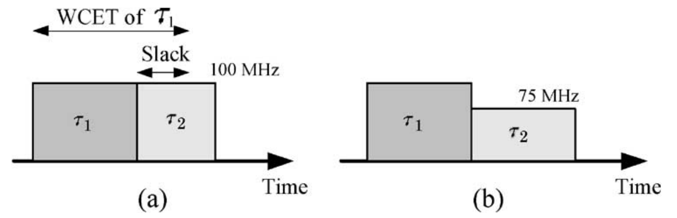


Fig. 3. Priority-based slack-stealing DVS algorithm. (a) Before stretching. (b) After stretching.

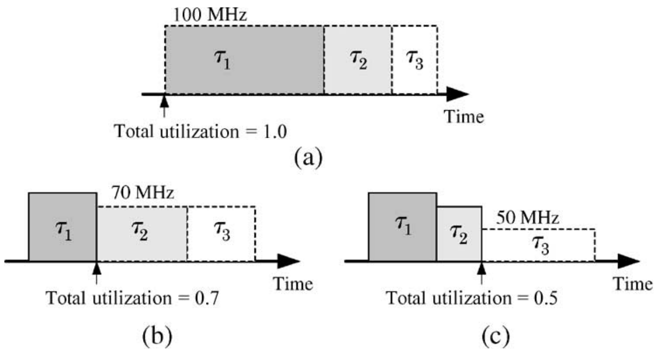


Fig. 4. Utilization-updating DVS algorithm. (a) Before the execution of $\mathcal{T}_1$. (b) After the completion of $\mathcal{T}_1$. (c) After the completion of $\mathcal{T}_2$.

parameters of the scheduling server such as T_s and Q_s in order that a system designer can control the value of δ.

Existing on-line DVS algorithms such as [7]–[10] are not directly applicable to the DSAMTS problem. As discussed in [6], most existing heuristics are based on three techniques: 1) stretching-to-NTA; 2) priority-based slack stealing; and 3) utilization updating. For example, consider the stretching-to-NTA technique. The technique stretches the execution time of the periodic task ready for execution to the next arrival time of a periodic task (NTA) when there is no other periodic task in ready queue as shown in Fig. 2 [7]. To use the stretching-to-NTA technique for a mixed task system, the DVS algorithm should know the next arrival time of an aperiodic task as well as a periodic task. Although the arrival times of periodic tasks can be easily computed using their periods, the arrival times of aperiodic tasks cannot be known since they arrive at arbitrary times. If the arrival of aperiodic tasks is ignored, there will be a deadline miss of periodic hard real-time task when an aperiodic task arrives before the next arrival time of a periodic task. Consequently, the stretching-to-NTA technique should assign the full speed to all tasks in the mixed task system.

The priority-based slack-stealing method exploits the basic properties of priority-driven scheduling such as RM and EDF scheduling policies. The basic idea is that when a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task [8]. It is also possible for a higher-priority task to utilize the slack times from completed lower-priority tasks [10]. However, the latter type of slack stealing is computationally expensive to implement precisely. Fig. 3 shows an example of a priority-based slack-stealing method. When the higher-priority task τ_1 completes before its WCET, the lower-priority task τ_2 uses the slack time and starts with a lower clock speed.

The utilization-updating technique estimates the required processor performance at the current scheduling point by recalculating the expected worst-case processor utilization using the actual execution times of completed task instances [9]. When the processor utilization is updated, the clock speed can be adjusted accordingly. Fig. 4 shows a utilization-updating technique. Before the execution of τ_1, the DVS scheduler should assign the clock speed based on the worst case processor utilization, 1.0. But after the completion of τ_1, the DVS scheduler can know that the processor utilization is reduced to 0.7 by the early completion of τ_1. So, the clock speed was reduced to 70 MHz. After the completion of τ_2, the clock speed can also be reduced to 50 MHz because the processor utilization is changed to 0.5. The main merit of this method is its simple implementation, since only the processor utilization of completed task instances has to be updated at each scheduling point.

To use the priority-based slack-stealing method or the utilization-updating method, the DVS scheduler should be able to identify a slack time due to aperiodic tasks as well as periodic tasks. The slack time of a periodic task can be easily defined as the difference between the WCET and the real execution time of the task. However, for aperiodic tasks, the slack time cannot be identified from the early completion of an aperiodic task because the scheduling server can service aperiodic tasks if the remaining budget is larger than 0. Therefore, the DVS algorithm should be concerned about the scheduling server rather than aperiodic tasks because the utilization of scheduling server is related with the schedulability condition. Therefore, on-line DVS algorithms need to be modified to utilize the characteristics of scheduling servers.

IV. Dynamic Scheduling for Mixed Task Sets

A. Scheduling Algorithms in Fixed-Priority Systems

1) DS and SS: For fixed-priority systems, the RM scheduling policy is targeted in this paper. Fig. 5(a) shows the task schedule using an SS. There are two periodic tasks $\tau_1 = (1, 5)$ and $\tau_2 = (2, 8)$ and one SS $= (1, 4)$. The RM scheduler schedules each periodic task and the SS. The utilization of the SS is 0.25 $(= Q_s/T_s = 1/4)$. The budget of SS q_s is set to Q_s at time 0. The budget is changed by the following rules.

1) Consumption rule: When SS executes an aperiodic task σ_k, the execution budget of the server is consumed at a rate of $s(\sigma_k)$ per unit time.
2) Replenishment rule: If SS consumes θ amount of budget during the time [t_1, t_2], the execution budget of the

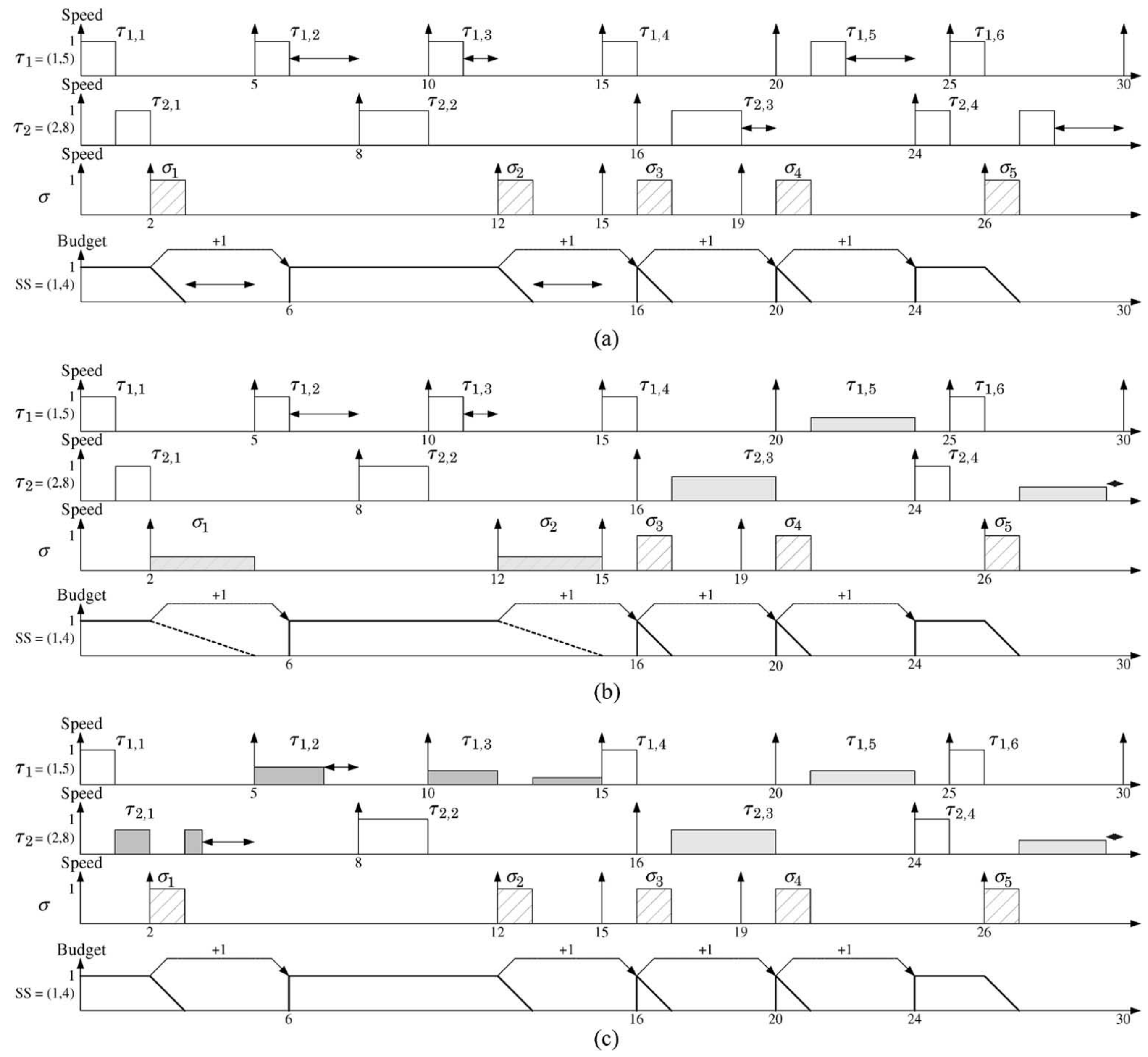


Fig. 5. Task schedules with an SS: (a) SS without DVS, (b) SS/lppsRM-S, and (c) SS/lppsRM-B.

server q_s is replenished by the amount of θ at the time $t_1 + T_s$.

SS preserves its budget q_s if no requests are pending when released. An aperiodic request can be serviced at any time (at server's priority) as long as the budget of SS is not exhausted (e.g., task σ_1). If the budget is exhausted, aperiodic tasks should wait until the next replenishment time. For example, although the task σ_4 arrived at the time of 19, it is serviced at the time of 20.

DS uses the same consumption rule to SS, but has a following replenishment rule: The execution budget of the server is set to Q_s at time instants kT_s, for $k = 0, 1, 2, \ldots$.

2) Stretching-To-NRT[3] *(SNRT) Scheme:* A modified stretching-to-NTA algorithm SNRT for DS and SS is first proposed. Since the arrival times of aperiodic tasks cannot be known, the stretching-to-NTA method should be modified such that the execution behavior of DS and SS is utilized. So, the following two stretching rules for DS and SS are proposed.

1) Stretching rule for aperiodic task: If there is no periodic task in the ready queue, execute an aperiodic task at the speed of $\max(1, q_s/(\min(\mathrm{NTA}, R) - t))$, where NTA, R, and t are the next arrival time of a periodic task, the next replenishment time of the scheduling server, and the current time (the start time of aperiodic task), respectively.
2) Stretching rule for periodic task: If there is only one periodic task in the ready queue and q_s is 0, stretch the periodic task to $\min(\mathrm{NTA}, R)$. This is because the arriving aperiodic task is delayed until the next replenishment

[3]NRT means next replenishment time.

time if q_s is 0. If $q_s > 0$, the speed of the periodic task cannot be scaled down even although there is only one periodic task in the ready queue.

Using these two rules, the authors modified existing on-line DVS algorithms. Fig. 5(b) shows the task schedule using the SS/lppsRM-S algorithm, which is the modified version of lppsRM [7] for SS. lppsRM uses the stretching-to-NTA method. The aperiodic tasks σ_1 and σ_2 are stretched to the next arrival times of periodic tasks (5 and 15), respectively, because there is no periodic task in the ready queue. The periodic tasks $\tau_{1,5}$, $\tau_{2,3}$, and the latter part of $\tau_{2,4}$ are stretched to $\min(\mathrm{NTA}, R)$ because q_s is 0. The tasks $\tau_{1,2}$ and $\tau_{1,3}$ cannot be stretched because q_s is larger than 0. Under the task stretching rules, it can be guaranteed that the maximum increase of the average response time is $T_s - Q_s$. This paper provides the proof. Although DS has a different replenishment rule with SS, both DS and SS can use the same DVS algorithm.

Definition 1: The replenishment time $\mathbb{R}(\sigma_k)$ is the last replenishment time of a scheduling server among the replenishment times, which is earlier than the completion time of σ_k, $e(\sigma_k)$.

Definition 2: The execution cycles $\mathbb{C}(\sigma_k)$ are the execution cycles of σ_k that are executed after the last replenishment time $\mathbb{R}(\sigma_k)$.

Lemma 1: The last replenishment times of σ_k, $\mathbb{R}(\sigma_k)$, are the same in both DS/lppsRM-S (or SS/lppsRM-S) and DS (or SS).

Proof: The last replenishment time $\mathbb{R}(\sigma_k)$ is determined by the replenishment rule, the server budget Q_s, the server period T_s, the release times of aperiodic tasks, and the execution cycles of aperiodic tasks. However, none of them is changed by DS/lppsRM-S (or SS/lppsRM-S). So, it can be concluded that the last replenishment times are the same in both algorithms. ■

Lemma 2: The numbers of $\mathbb{C}(\sigma_k)$ are the same in both DS/lppsRM-S (or SS/lppsRM-S) and DS (or SS).

Proof: Since the stretching rules of DS/lppsRM-S (or SS/lppsRM-S) do not stretch tasks over the replenishment time, the number of execution cycles of a task executed between two replenishment times is not changed by the DVS algorithms. By Lemma 1, the last replenishment time is unchanged. Therefore, this lemma is true. ■

Theorem 1: By the DS/lppsRM-S and SS/lppsRM-S algorithms, $D(\sigma_k) \le T_s - Q_s$ for all σ_k.

Proof: The authors prove that the completion time of σ_k in DS/lppsRM-S (or SS/lppsRM-S), $e'(\sigma_k)$, is smaller than $e(\sigma_k) + T_s - Q_s$ when σ_k is completed at $e(\sigma_k)$ in DS (or SS). The authors prove the theorem for two cases.

Case 1 $\eta(\sigma_k) \le \mathbb{R}(\sigma_k)$: In this case, there is no queuing delay after $\mathbb{R}(\sigma_k)$ because σ_k has already started. The budget delay after $\mathbb{R}(\sigma_k)$ is also 0 by the definition of $\mathbb{R}(\sigma_k)$. Since $q_s > 0$ between $\mathbb{R}(\sigma_k)$ and $e(\sigma_k)$ (or $e'(\sigma_k)$), all periodic tasks are executed at full speed by the stretching rule for periodic tasks. So, the preemption delay after $\mathbb{R}(\sigma_k)$, $\mathbb{P}(\sigma_k)$, in DS (or SS) is same to the preemption delay $\mathbb{P}'(\sigma_k)$ in DS/lppsRM-S (or SS/lppsRM-S) by Lemma 2. The remaining execution cycles of σ_k, $\mathbb{C}(\sigma_k)$, are also the same in both algorithms by Lemma 2. $\mathbb{C}(\sigma_k)$ is not larger than the budget of the scheduling server by the definition of $\mathbb{R}(\sigma_k)$, i.e., $\mathbb{C}(\sigma_k) \le Q_s$. When σ_k is resumed at t ($t \ge \mathbb{R}(\sigma_k)$) and R is the next replenishment time after the completion of σ_k, it can be said that the time interval $[t, \min(R, \mathrm{NTA})]$ is smaller than or equal to T_s because $R - \mathbb{R}(\sigma_k) \le T_s$. Therefore, the authors can show that $e'(\sigma_k) - e(\sigma_k) \le T_s - Q_s$ as

$$\begin{aligned}
e(\sigma_k) &= \mathbb{R}(\sigma_k) + \mathbb{C}(\sigma_k) + \mathbb{P}(\sigma_k)\\
e'(\sigma_k) &= \mathbb{R}(\sigma_k) + \mathbb{C}(\sigma_k)\\
&\quad \times \frac{(\min(R, \mathrm{NTA}) - t)}{Q_s} + \mathbb{P}'(\sigma_k)\\
e'(\sigma_k) - e(\sigma_k) &= \mathbb{C}(\sigma_k)\frac{(\min(R, \mathrm{NTA}) - t)}{Q_s} - \mathbb{C}(\sigma_k)\\
&= \frac{\mathbb{C}(\sigma_k)}{Q_s}\left((\min(R, \mathrm{NTA}) - t) - Q_s\right)\\
&\le \left((\min(R, \mathrm{NTA}) - t) - Q_s\right)\\
&\le (T_s - Q_s).
\end{aligned}$$

Case 2 $\eta(\sigma_k) > \mathbb{R}(\sigma_k)$: In this case, the authors can treat all aperiodic tasks $\sigma_{k-j}, \ldots, \sigma_{k-1}, \sigma_k$ that are completed after $\mathbb{R}(\sigma_k)$ as one aperiodic task $\sigma_{k-j,\ldots,k}$ that has the execution cycles of $c(\sigma_{k-j,\ldots,k}) = \sum_{i=k-j}^{k} c(\sigma_i)$. Then, the proof is equal to that in Case 1. ■

Consequently, it can be concluded that $D(\sigma_k) \le T_s - Q_s$ for all σ_k scheduled by the DS/lppsRM-S and SS/lppsRM-S algorithm.

3) Bandwidth-Based Slack-Stealing (BSS) Scheme: Although the energy consumption can be reduced by the SNRT algorithm, the algorithm can show poor energy performance when the workload of aperiodic tasks is small. In this case, since the budget q_s is larger than 0 at most scheduling points, the DVS algorithm cannot use the stretching rule for a periodic task. Extremely, when there is no aperiodic request, there is nothing to do for the DVS algorithm. Therefore, a more advanced DVS algorithm that can be applicable to the mixed task system with a low aperiodic workload is needed. For this purpose, the authors propose a new slack estimation method, bandwidth-based slack stealing, which identifies the maximum slack time for a periodic task considering the bandwidth of a scheduling server. Fig. 5(c) shows the SS/lppsRM-B algorithm, which is based on SS/lppsRM-S but uses the bandwidth-based slack-stealing method additionally. When q_s is larger than 0 and there is only one periodic task in the ready queue, the slack estimation method calculates the maximum available time before the arrival time of next periodic task.

Fig. 6 shows the bandwidth-based slack-stealing method. Assume that a periodic task τ is released at the time t and the remaining budget of the scheduling server is q_s. T_τ is the period of τ, NTA is the next periodic task arrival time, and R is the next replenishment time of a scheduling server. Two different cases should be considered depending on the priority of the scheduling server. Fig. 6(a) shows the case when $T_\tau > T_s$. In this case, the maximum blocking time by aperiodic tasks before the next task arrival time (NTA) should be identified. Before R, the scheduling server can use q_s at most.

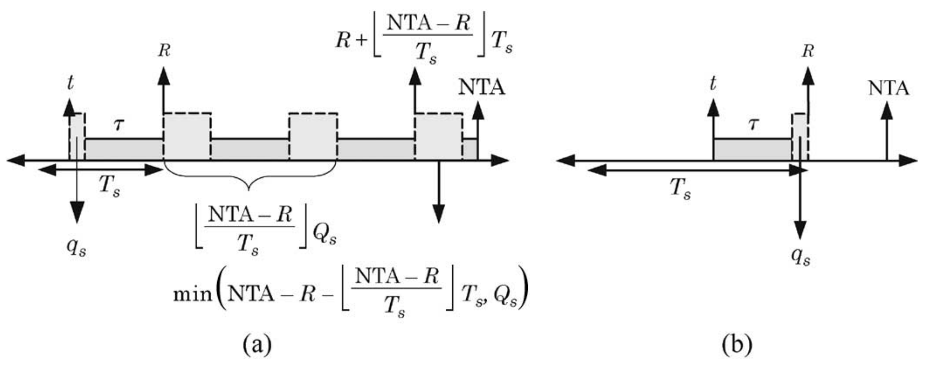

$R+\left\lfloor \frac{\text{NTA}-R}{T_s} \right\rfloor T_s$
R
t
NTA
τ
T_s
q_s
$\left\lfloor \frac{\text{NTA}-R}{T_s} \right\rfloor Q_s$
$\min\left(\text{NTA}-R-\left\lfloor \frac{\text{NTA}-R}{T_s} \right\rfloor T_s, Q_s\right)$
(a)
(b)

Fig. 6. Bandwidth-based slack stealing in SS/lppsRM-B. (a) $T_{\mathcal{T}} > T_s$. (b) $T_{\mathcal{T}} < T_s$.

And, from R to NTA, the server can occupy $\lfloor(\text{NTA} - R)/T_s\rfloor Q_s + \min(\text{NTA} - R - \lfloor(\text{NTA} - R)/T_s\rfloor T_s, Q_s)$ at maximum (dotted squares).

Fig. 6(b) shows the case when $T_\tau < T_s$. In this case, the task τ is stretched to $\min(R, \text{NTA}) - q_s$. Although there is no deadline miss even when the periodic task τ is completed after R, the proposed DVS algorithm is designed to limit the response time delay. Under this policy, the preemption delay increases, but the DVS algorithm can guarantee that $D(\sigma_k) \le T_s - Q_s$ for all σ_k because σ_k is not delayed above the replenishment time R.

From Fig. 6, the maximum available time MAT of a task τ can be calculated as

$$\begin{aligned}
&\text{if } (T_\tau > T_s)\ \text{MAT} \\
&\quad = \text{NTA} - t - q_s - \left\lfloor \frac{\text{NTA} - R}{T_s} \right\rfloor Q_s \\
&\qquad - \min\left(\text{NTA} - R - \left\lfloor \frac{\text{NTA} - R}{T_s} \right\rfloor T_s, Q_s\right) \\
&\text{if } (T_\tau < T_s)\ \text{MAT} \\
&\quad = \min(R, \text{NTA}) - t - q_s.
\end{aligned}$$

In Fig. 5(c), the periodic tasks $\tau_{1,2}$, $\tau_{1,3}$, and $\tau_{2,1}$ are stretched by the bandwidth-based slack-stealing method. For example, at the time 5, the task $\tau_{1,2}$ has the available time 2 $(= \text{NTA} - t - q_s = 8 - 5 - 1)$. A side effect of the bandwidth-based slack-stealing method is that aperiodic tasks tend to be executed at full speed. Due to the side effect, the DVS algorithm using the bandwidth-based slack-stealing method generates better average response times.

4) Periodic Only Slack Distribution (POSD) Scheme: In the previous two schemes, both periodic and aperiodic tasks use the identified slack times. However, if all the slack times are given to only periodic tasks, a better response time can be obtained. In this case, aperiodic tasks are always executed at full speed (or the initial clock speed assigned by the off-line algorithm). The authors call such a slack distribution technique as a POSD scheme. A POSD scheme shows better response time than SNRT and BSS schemes with a slight degradation on energy performance. The POSD scheme can be applied to both fixed-priority and dynamic-priority systems.

B. Scheduling Algorithms in Dynamic-Priority Systems

1) CBS: For dynamic-priority systems, the EDF scheduling policy was assumed. The authors propose the slack estimation algorithm for CBS. Fig. 7(a) shows the task schedule using CBS, assuming two periodic tasks $\tau_1 = (2, 8)$ and $\tau_2 = (3, 12)$ and one CBS $= (2, 4)$. The maximum utilization of CBS (U_s) is $0.5 (= 2/4)$. If $U_p + U_s \le 1$, where U_p is the maximum utilization of periodic tasks, the task set is schedulable.

For the description of CBS, the following terms were defined.

1) A CBS is said to be active at time t if there are pending aperiodic tasks; that is, if there exists a served task σ_i such that $r(\sigma_i) \le t < e(\sigma_i)$, where $r(\sigma_i)$ and $e(\sigma_i)$ are the arrival time and the completion time of the task σ_i. When a task σ_i arrives and the server is active, the request is enqueued in a queue of pending tasks according to a given (arbitrary) nonpreemptive discipline (e.g., FIFO).
2) A CBS is said to be idle at time t if it is not active.
3) At each instant, a server deadline d_k is associated with CBS. The server deadline is updated by the following deadline assignment rules. Each served aperiodic task σ_i is assigned a dynamic deadline equal to the current server deadline d_k. At the beginning, $d_0 = 0$.

When an aperiodic task finishes, the next pending task, if any, is served using the current budget and deadline. If there are no pending tasks, the server becomes idle. At any instant, an aperiodic task is assigned the last server deadline generated by the server.

CBS uses following special rules to sustain its utilization below U_s.

1) Budget replenishment and deadline assignment rule 1: When $q_s = 0$, the server budget is replenished to the maximum value Q_s and a new server deadline is generated as $d_{k+1} = d_k + T_s$.
2) Budget replenishment and deadline assignment rule 2: When an aperiodic task σ_i arrives at $r(\sigma_i)$ and the server is idle (when CBS does not service aperiodic tasks), if $q_s \ge (d_k - r(\sigma_i))U_s$ the server generates a new deadline $d_{k+1} = r(\sigma_i) + T_s$ and q_s is replenished to the maximum value Q_s, otherwise the task is served with the last server deadline d_k using the current budget.

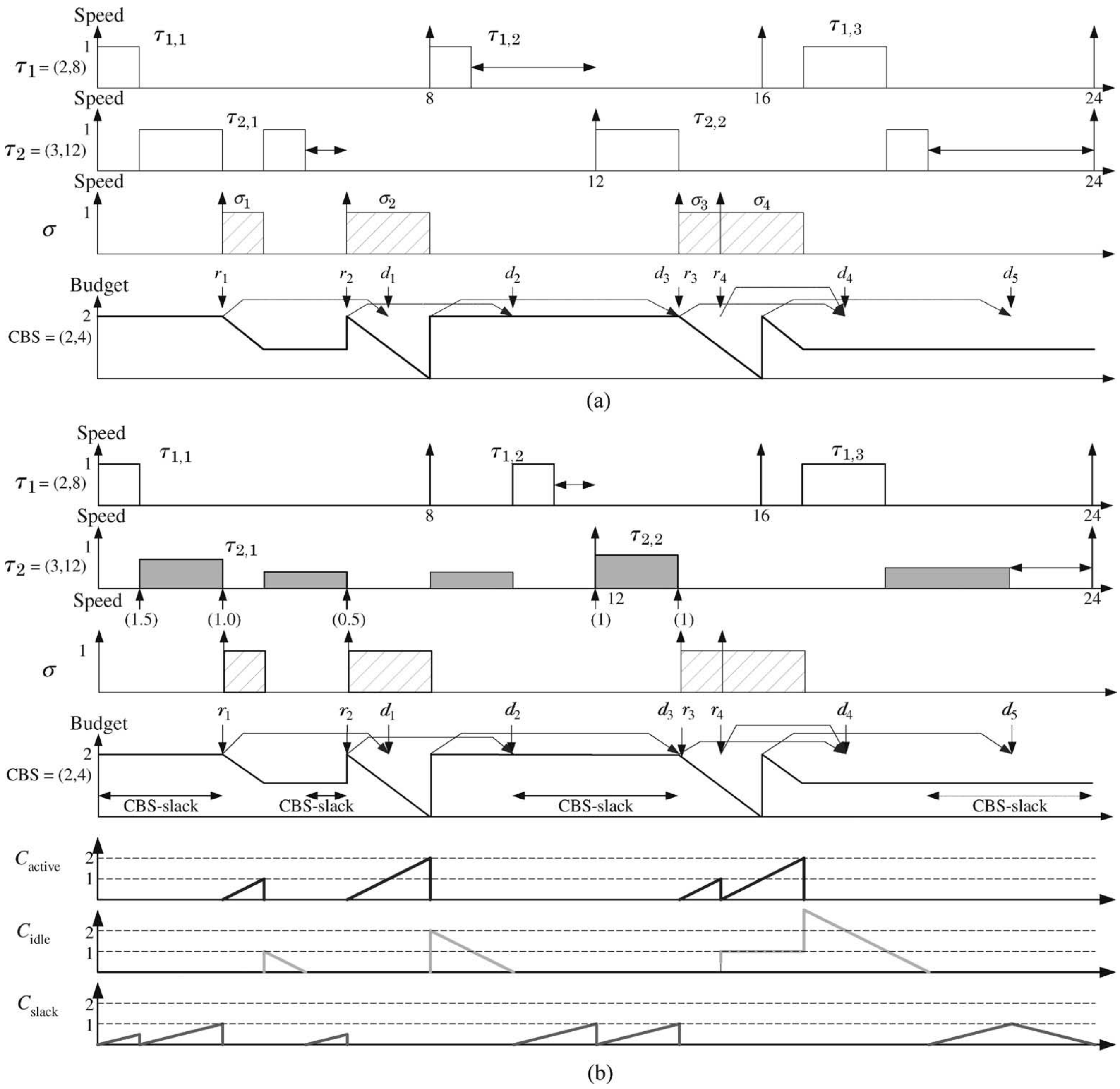


Fig. 7. Task schedules with a CBS: (a) CBS without DVS and (b) CBS/DRA-W.

For example, in Fig. 7(a), when an aperiodic task σ_1 arrives at time 3, CBS sets its deadline d_1 to 7 $(= r(\sigma_1) + T_s = 3 + 4)$ and σ_1 uses the deadline. When an aperiodic task σ_2 arrives at time 6, CBS sets σ_2's deadline to 10 $(= r(\sigma_2) + T_s = 6 + 4)$ and q_s is replenished to 2 because $q_s = 1$ is greater than $(d_1 - r(\sigma_2))\ U_s = (7 - 6)\ 0.5 = 0.5$. When a task σ_3 arrives at 14, CBS sets σ_3's deadline to 18 and σ_3 preempts the task $\tau_{2,2}$. When an aperiodic task σ_4 arrives at 15, CBS sets σ_4's deadline to 18 $(= d_4)$ because $q_s = 1$ is smaller than $(d_4 - r(\sigma_4))\ U_s = (18 - 15)\ 0.5 = 1.5$. When $q_s = 0$ at time 16, CBS changes σ_4's deadline to a new deadline $d_5 = d_4 + T_s = 22$ and q_s is replenished to 2. In this manner, CBS maintains its bandwidth under U_s.

2) Workload-Based Slack Estimation (WSE) Scheme: For CBS, the stretching rule for periodic tasks used for DS and SS cannot be employed because there are no finite intervals of time in which the budget is equal to zero. To use the priority-based slack-stealing [6] method for CBS, the slack times of the CBS should be identified. The slack time can be estimated using the WSE method. When the workload of CBS is lower than U_s, the slack times can be identified.

Fig. 8 shows the WSE algorithm for CBS. The algorithm uses four variables: aperiodic_run, C_{slack}, C_{idle}, and C_{active}. The aperiodic_run is a flag variable to know whether an aperiodic task is being executed. C_{active} contains the number of execution cycles of the completed aperiodic tasks. When an aperiodic task

```
Initiation:
        aperiodic_run = F; $C_{slack}$ = 0; $C_{idle}$ = 0; $C_{active}$ = 0;
upon aperiodic_task_start:
        aperiodic_run = T;
upon aperiodic_task_completion:
        $C_{idle}$ += $C_{active}$ * (1-$U_s$) / $U_s$;
        aperiodic_run = F; $C_{active}$ = 0;
during aperiodic_task_execution($t$):
        increase $C_{active}$ by $t$;
during aperiodic_idle($t$):
        if (aperiodic_run is T)
        if ($C_{idle}$ > 0) decrease $C_{idle}$ by $t$; else increase $C_{slack}$ by $t \cdot U_s$;
```

Fig. 8. WSE in CBS.

is completed, C_{idle}, which is the number of idle cycles required to make the workload of CBS equal to U_s, is calculated. When an aperiodic task is not executed, C_{idle} is decreased. When C_{idle} becomes 0, the workload of CBS is equal to U_s. If the idle interval of CBS continues, the workload of CBS becomes smaller than U_s and C_{slack} is increased. C_{slack} can be used for periodic tasks to stretch the execution time.

Fig. 7(b) shows the task schedule using the CBS/DRA-W algorithm, which is the modified version of DRA [8] for CBS using the WSE scheme. In Fig. 7(b), the time intervals, where $C_{\text{slack}} > 0$, are marked with arrow lines. For example, when a task $\tau_{2,1}$ is scheduled at time 1, there is a slack time 1.5 (1 from the early completion of $\tau_{1,1}$ and 0.5 from CBS during the time interval [0,1]). Using the slack time, the task $\tau_{2,1}$ is scheduled with the speed of 0.67 $(= 3/(3+1.5))$. When the task $\tau_{2,1}$ is preempted at time 3, the slack time 1.0 from CBS is transferred to the remaining part of $\tau_{2,1}$.

CBS/DRA-W generally increases the preemption delay of an aperiodic task. However, the authors can guarantee that $D(\sigma_k) \le T_s - Q_s$ for all σ_k.

Lemma 3: The deadline d'_k assigned by the CBS/DRA-W algorithm is identical to the deadline d_k assigned by the CBS algorithm.

Proof: There are two cases when a new deadline is assigned in CBS. First, when all the budget is consumed $(q_s = 0)$, a new deadline $d_{k+1} = d_k + T_s$ is generated. Since T_s is same in both CBS/DRA-W and CBS, d_{k+1} and d'_{k+1} are also same. Second, when CBS is inactive and $q_s \ge (d_k - r(\sigma_i))U_s$, a new deadline $d_{k+1} = r(\sigma_i) + T_s$ is assigned if an aperiodic task σ_k arrives at $r(\sigma_k)$. The new deadline assignment occurs to sustain the utilization of CBS below U_s. Since the DRA algorithm transfers only the unused slack times of higher-priority task to low-priority task, there is no change of the utilization of CBS at $r(\sigma_i)$. Therefore, d_k and d'_k for all k are same. ■

Lemma 4: The numbers of $\mathbb{C}(\sigma_k)$, which is the remaining execution cycle after $\mathbb{R}(\sigma_k)$, are same in both CBS and CBS/DRA-W.

Proof: CBS/DRA does not change the replenishment rule of the scheduling server. The budget of CBS is only consumed by aperiodic tasks. Therefore, the amount of executed aperiodic tasks before $\mathbb{R}(\sigma_k)$ is not changed. ■

Theorem 2: By the CBS/DRA-W algorithm, $D(\sigma_k) \le T_s - Q_s$ for all σ_k.

Proof: When the completion times of σ_k in DRA and CBS/DRA-W are denoted as $e(\sigma_k)$ and $e'(\sigma_k)$, respectively, the authors should show that $e'(\sigma_k) - e(\sigma_k) \le T_s - Q_s$. The theorem is proven for two cases.

TABLE III
SUMMARY OF DVS ALGORITHMS FOR MIXED TASK SETS

Scheduling Server	DVS Algorithms		Proposed Techniques
DS/SS (RM)	lppsRM	Stretching-to-NTA	SNRT
	ccRM		BSS
CBS (EDF)	lppsEDF		POSD
	ccEDF	Utilization Updating	WSE
	DRA	Priority-based Slack Stealing	POSD

Case 1: During the execution of σ_k, there is no replenishment of the budget of CBS.

In this case, it can be known that the execution cycle of σ_k, $c(\sigma_k)$, is smaller than Q_s. By Lemma 3, $e'(\sigma_k) \le d'(\sigma_k) = d(\sigma_k)$. From the deadline assignment rule, $d(\sigma_k) - r(\sigma_k) \le T_s$. Therefore, it can be shown that $e'(\sigma_k) - e(\sigma_k) \le T_s - Q_s$ as

$$\begin{aligned} e'(\sigma_k) - e(\sigma_k) &\le d(\sigma_k) - e(\sigma_k) \\ &\le d(\sigma_k) - (r(\sigma_k) + c(\sigma_k)) \\ &= (d(\sigma_k) - r(\sigma_k)) - c(\sigma_k) \\ &\le T_s - c(\sigma_k) \\ &\le T_s - Q_s. \end{aligned}$$

Case 2: During the execution of σ_k, there are one or more replenishments of the budget of CBS.

From Lemma 4, it can be known that $\mathbb{C}(\sigma_k)$ is same in both DRA and CBS/DRA-W. Therefore, the remaining part of σ_k can be treated as another aperiodic task that has the execution cycles of $\mathbb{C}(\sigma_k)$. Then, Case 2 can be proven with the proof for Case 1. ■

3) POSD Scheme: The POSD scheme where aperiodic tasks are always executed at full speed can be applied to both fixed-priority and dynamic-priority systems. The POSD scheme shows a better response time than the pure WSE scheme with a slight degradation on energy performance in dynamic-priority systems.

4) Summary for DVS Algorithms: Table III summarizes the proposed DVS techniques for mixed task sets. The authors proposed SNRT, BSS, and POSD schemes for the stretching-to-NTA algorithm and proposed WSE and POSD for the utilization-updating algorithm and the priority-based slack-stealing algorithm.

Fig. 9 shows an evolution tree of DVS algorithms for CBS. If the authors are to take the lppsEDF technique for CBS, three kinds of DVS algorithms can be used. While CBS/lppsEDF-B uses both SNRT and BSS techniques, CBS/lppsEDF-P uses all of the SNRT, BSS, and POSD techniques.

V. STATIC SCHEDULING FOR MIXED TASK SETS

This section proposes a static scheduling algorithm that is suitable for mixed task sets. Pillai and Shin [9] proposed static voltage-scheduling algorithms for periodic tasks using RM and EDF schedulability tests. Their static scheduling algorithm

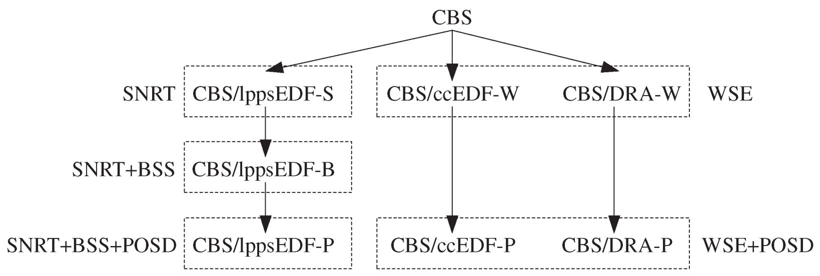


Fig. 9. Evolution tree of DVS algorithms for CBS.

finds a clock speed of periodic tasks for a hard real-time system. The clock speed is set statically and is not changed unless the task set is changed. For the mixed task set using a scheduling server, Pillai's static scheduling algorithms can also be used with the utilization of the scheduling server. For example, in EDF scheduling using CBS, if the worst-case utilization of periodic tasks is 0.3 and the utilization of CBS is 0.4 at a 100-MHz clock speed, the static scheduling algorithm determines the clock speed as 70 MHz $(= 100 \text{ MHz}(0.3 + 0.4))$ because the total utilization is 0.7.

However, the scheduling server for aperiodic tasks generally occupies a large utilization compared with the real workload of aperiodic tasks to provide a good responsiveness. If the real utilization of aperiodic tasks is 0.2 rather than 0.4, it is better to use a lower clock speed for periodic tasks and a higher clock speed for aperiodic tasks than 70 MHz. This is because CBS has many idle intervals. However, the DVS scheduler cannot use the clock speed of 50 MHz $(= 100 \text{ MHz}(0.3 + 0.2))$ because it can produce deadline misses when the real utilization of an aperiodic task is larger than 0.2.

Therefore, in static voltage scheduling, both the expected workload and the schedulability condition should be considered. The static voltage-scheduling algorithm selects the static operating speed $s(\tau_i)$ of a periodic task τ_i and the operating speed s_s of scheduling server for aperiodic tasks, respectively. $s(\tau_i)$ and s_s should allow a real-time scheduler to meet all the deadlines for a given periodic task set minimizing the total energy consumption. Consequently, the problem of static scheduling can be formulated as

Static Speed Assignment Problem

Given $\tau_1, \ldots, \tau_n, \omega_1, \ldots, \omega_n$, and ρ,
find $s(\tau_i), \ldots, s(\tau_n)$ and s_s such that
$E = \sum_{1}^{n} U_i \times \omega_i \times s(\tau_i)^2 + \rho \times s_s^2$ is minimized
subject to $\sum_{1}^{n} \frac{U_i}{s(\tau_i)} + \frac{U_s}{s_s} \leq U_{\text{lub}}$ and $0 \leq s(\tau_i), s_s \leq 1$

where U_i is the worst-case utilization of a periodic task τ_i and U_s is the utilization of a scheduling server. E is a metric reflecting energy consumption.[4] U_{lub}, which is the least upper bound of schedulable utilization, is 1 with EDF scheduling and $n(2^{1/n} - 1)$ for n tasks with RM scheduling,[5] respectively. ω_i is the ratio between the average-case execution time and the WCET of a periodic task τ_i. For example, when the WCET of τ_i is 10 ms and the average execution time is 5 ms, ω_i is 0.5. ρ is the average workload of aperiodic tasks. The authors assume that the interarrival times and service times of aperiodic tasks follow the exponential distribution using the parameters λ and μ, where $1/\lambda$ is the mean interarrival time and $1/\mu$ is the mean service time. Then, the average workload of aperiodic tasks can be represented by $\rho = \lambda/\mu$. If there is no interference between aperiodic tasks and periodic tasks, the average response time of aperiodic tasks is given by $(\mu - \lambda)^{-1}$ from the M/M/1 queuing model. For example, when the mean interarrival time is 5 ms and the mean service time is 2 ms, λ, μ, and ρ are 0.2, 0.5, and 0.4, respectively.

Using the Lagrange transform, the optimal solution for $s(\tau_i)$ and s_s can be obtained as

$$s(\tau_i) = \frac{1}{U_{\text{lub}}} \left(\sum_{j=1}^{n} U_j \sqrt[3]{\frac{\omega_j}{\omega_i}} + U_s \sqrt[3]{\frac{\rho}{U_s \omega_i}} \right)$$
$$s_s = \frac{1}{U_{\text{lub}}} \left(\sum_{j=1}^{n} U_j \sqrt[3]{\frac{U_s \omega_j}{\rho}} + U_s \right). \quad (1)$$

Under the assumption that the exact ω_i and ρ values can be known, the optimal static speeds for periodic and aperiodic tasks can be obtained.

Doh *et al.*'s algorithm handled a different problem to that proposed here [14]. It finds a schedule that minimizes the average response time for a given energy budget. However, the static schedule based on the energy budget can be useless if the energy budget is changed during run time. Generally, since the run time workload is smaller than the worst-case workload, the energy budget decreases slowly than expected. So, this paper takes the approach that minimizes the energy consumption, limiting the response time delay. Moreover, Doh *et al.*'s technique has a high complexity because it uses an iterative search algorithm while that here requires only to solve simple mathematics.

[4]Assuming the supply voltage and clock speed are proportional in DVS, the energy consumption is represented to be proportional to the square of clock speed.

[5]When a deferrable server is used, the utilization bound is 0.6518 [1].

TABLE IV
TASK SET DESCRIPTION

$\mathcal{T}_1$ (EDF) (millisecond)			$\mathcal{T}_2$ (RM) (millisecond)		
Task	Period	WCET	Task	Period	WCET
τ_1	6	0.5	τ_1	6	0.5
τ_2	8	1.0	τ_2	8	1.0
τ_3	14	2.1	τ_3	14	1.283
τ_4	18	3.1			
U_p	0.4		U_p	0.3	

VI. EXPERIMENTAL RESULTS

A. *Static Speed Assignment*

The performances of the proposed DVS algorithms have been evaluated for scheduling servers using simulations. For static scheduling, the periodic task set $\mathcal{T}_1$ in Table IV is used. The execution time of each periodic task instance was randomly drawn from a Gaussian distribution in the range of [BCET, WCET], where BCET is the best-case execution time.

The interarrival times and service times of aperiodic tasks were generated from the exponential distribution using the parameters λ and μ, where $1/\lambda$ is the mean interarrival time and $1/\mu$ is the mean service time ($\rho = \lambda/\mu$).

Varying the server utilization U_s and the workload of aperiodic tasks ρ under a fixed utilization U_p of periodic tasks, the authors observed the energy consumption of the total system and the average response time of aperiodic tasks. The authors present only the experimental results where U_s is controlled by changing the value of T_s with a fixed Q_s value and ρ is controlled by a varying λ with a fixed μ value. ρ ranging from 0.05 to 0.25 ($\lambda = 0.05 \sim 0.25$ and $\mu = 1.0$) was used.

Fig. 10 shows the experimental results of the static speed assignment. The energy consumptions and response times are normalized by the values of Pillai's uniform speed assignment method, which assigns the same speed to both periodic tasks and aperiodic tasks, making the total utilization as U_{lub}. Aperiodic tasks are assumed to be serviced by the SS. It was assumed that if the system is idle, it enters into the power-down mode (PD). The power consumption in the PD mode is assumed to be zero.

A fixed value of 0.05 was used for the workload of aperiodic tasks (ρ) and the server utilization was varied from 0.1 to 0.35. For each U_s, the authors experimented with two workload ratios of periodic tasks ($\omega = 0.55$ and $\omega = 0.9$). The speed assignment method reduced the energy consumption up to 12% and 15% compared to the uniform speed assignment method when ω is 0.55 and 0.9, respectively. Since aperiodic tasks get higher speeds than the speeds for periodic tasks, the optimal speed assignment reduces the average response time as well as the energy consumption.

From the result, it can be seen that as U_s and ω are large, the energy consumption decreases. This is because the static scheduling algorithm is advantageous when the workload of aperiodic tasks (ρ) is small compared to the server utilization (U_s) and the workload of periodic tasks (ω).

B. *Dynamic Speed Assignment*

For the dynamic speed assignment algorithms, the periodic task sets $\mathcal{T}_1$ and $\mathcal{T}_2$ in Table IV were used for the experiments of fixed-priority systems and dynamic-priority systems, respectively. In the experiments, BCET is assumed to be 10% of WCET. For all experiments, both periodic tasks and aperiodic tasks were given an initial clock speed $s_0 = (U_p + U_s)s_m/U_{\text{lub}}$, where s_m is the maximum clock speed. During run time, the speed is further reduced by on-line DVS algorithms exploiting the slack times. In the experiments, the voltage scaling overhead is assumed negligible both in the time delay and power consumption.

Fig. 11(a) shows the energy consumptions of the SS/lppsRM-S algorithm, the SS/lppsRM-B algorithm, and the SS/lppsRM-P algorithm normalized by that of the PD method.

As U_s increases, the energy consumption increases because the initial clock speed s_0 increases and the response time decreases, converging on the average response time of M/M/1 because the number of interferences by periodic tasks is reduced.

The difference between the energy savings of SS/lppsRM-S and SS/lppsRM-B increases as ρ decreases. This is because there are more chances for SS to have the zero budget when ρ is large. As U_s increases, SS/lppsRM-B shows a larger energy saving compared with SS/lppsRM-S because SS/lppsRM-B performs well in the low aperiodic workload (over U_s). SS/lppsRM-S and SS/lppsRM-B reduced the energy consumption on average by 38% and 48% over the PD method, respectively. SS/lppsRM-B reduced the energy consumption on average by 16% over SS/lppsRM-S.

As shown in Fig. 11(b), SS/lppsRM-S and SS/lppsRM-B increase the response time on average by 10% and 8% over the PD method, respectively. Due to the side effect on aperiodic tasks explained at Section IV-A, SS/lppsRM-B shows better average response times. SS/lppsRM-P shows almost the same response time to that of the PD method because the execution speed of the aperiodic task is always s_0 and the preemption delay is not increased except in the case when T_s is larger than the periods of periodic tasks. However, it shows better energy performances than SS/lppsRM-S. Consequently, it can be said that SS/lppsRM-B and SS/lppsRM-P show better results in both the energy consumption and the response time than SS/lppsRM-S by assigning the most slack times to periodic tasks.

There are similar results for the ccRM [9] algorithm as shown in Fig. 11(c) and (d). The experimental results of DS showed similar results with that of SS. DS/lppsRM-B reduced the energy consumption on average by 24% over DS/lppsRM-S.

For CBS, the authors observed the performances of ccEDF and DRA. Fig. 12(a) and (c) shows the energy consumptions by CBS/ccEDF and CBS/DRA normalized by that of the PD method. The average energy reductions by CBS/ccEDF-W and CBS/ccEDF-P are 44% and 39%, respectively. Since most of

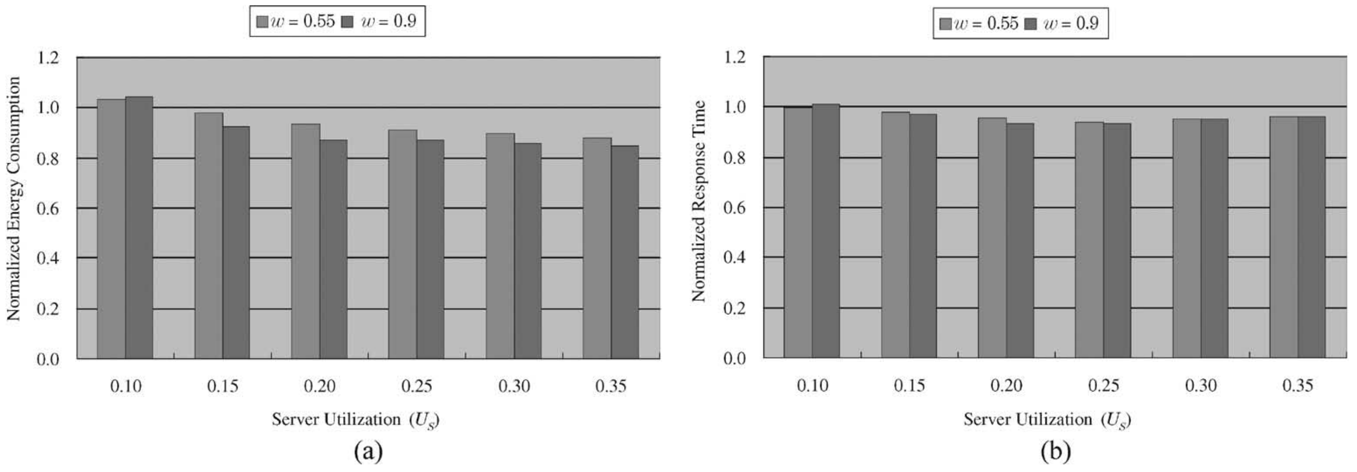


Fig. 10. Experimental results using static optimal algorithm. (a) Energy consumption. (b) Response time.

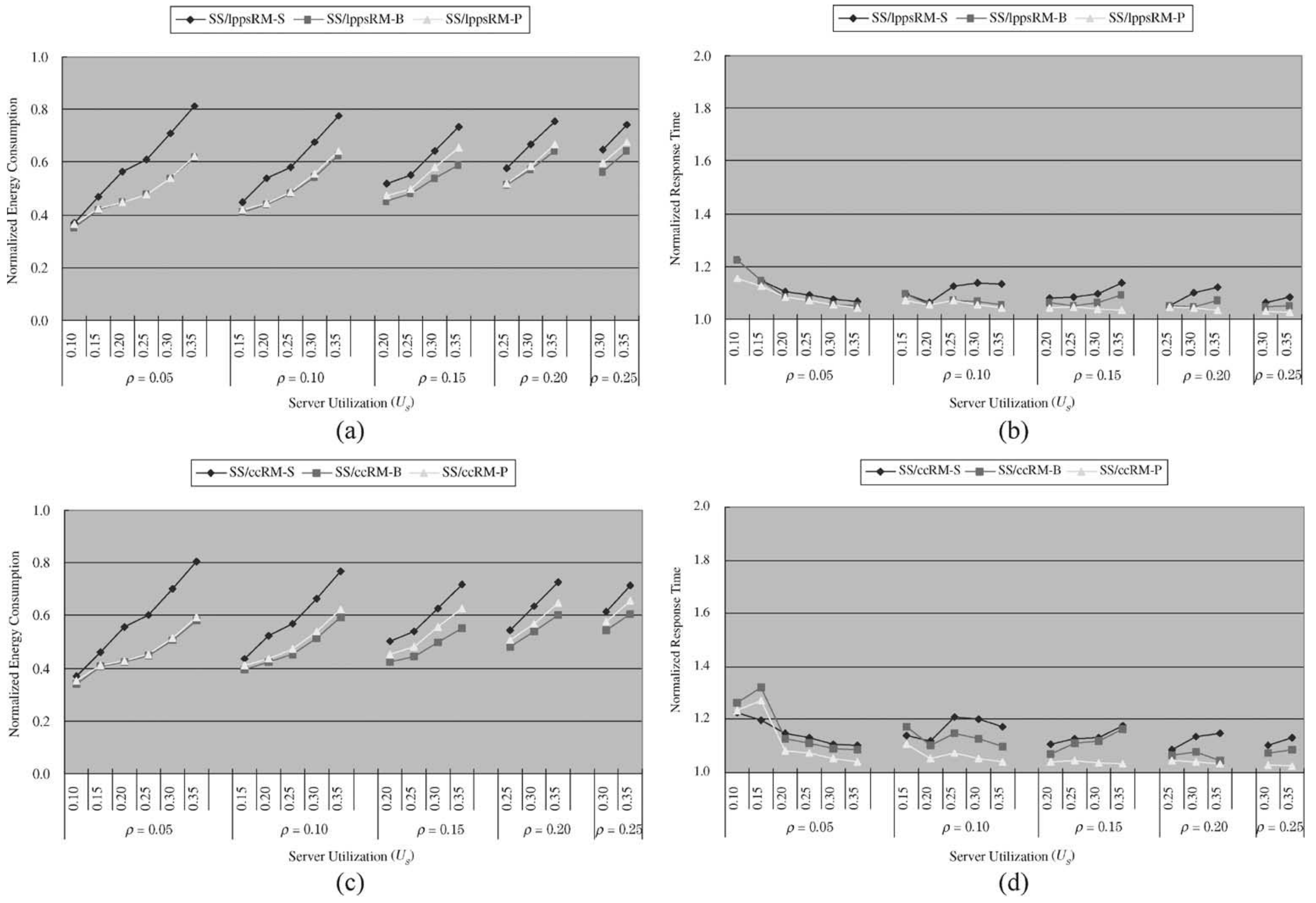


Fig. 11. Experimental results using SS. (a) Energy consumption. (b) Response time. (c) Energy consumption. (d) Response time.

the slack times are generated by CBS and used by periodic tasks in the DRA algorithm, CBS/DRA-W and CBS/DRA-P show similar energy performances. The average energy reductions by CBS/DRA-W and CBS/DRA-P are 35%.

Generally, the utilization-updating algorithm and the priority-based slack-stealing algorithm used in ccEDF and DRA find more slack times than the stretching-to-NTA algorithm. So, CBS/ccEDF-W and CBS/DRA-W increase the average response time significantly. CBS/DRA-W increased the average response time on average by 50%. As U_s decreases (T_s increases),[6] the response time increases because the maximum response time delay is $T_s - Q_s$. However, the response time delay of an aperiodic task is still smaller than $T_s - Q_s$. Fortunately, CBS/ccEDF-P and CBS/DRA-P increased the response time on average by 37%. Since CBS/DRA-P is similar

[6]Note that the authors varied T_s to change U_s.

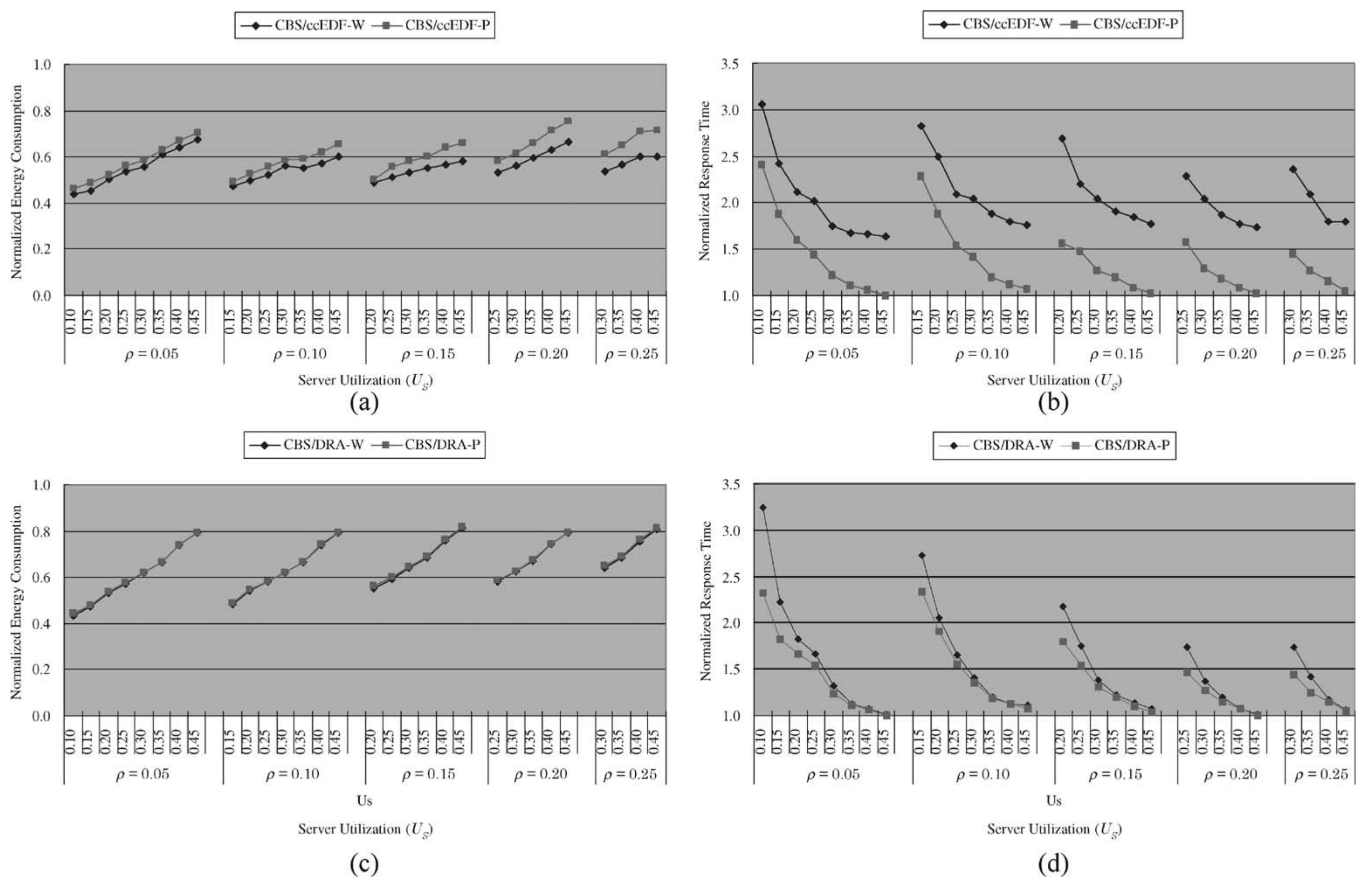


Fig. 12. Experimental results using CBS. (a) Energy consumption. (b) Response time. (c) Energy consumption. (d) Response time.

TABLE V
POWERPC 405LP SPEEDS, VOLTAGES, AND POWER RANGES

Speed Level	Speed (megahertz)	Voltage (volt)	Power Range (milliwatt)
1	100	1.0	46 - 82
2	200	1.4	154 - 300
3	266	1.7	307 - 630
4	333	1.9	429 - 881

to CBS/DRA-W in energy performances despite of its good response times, it can be known that it is better to give slack times only to periodic tasks when the short response times are required.

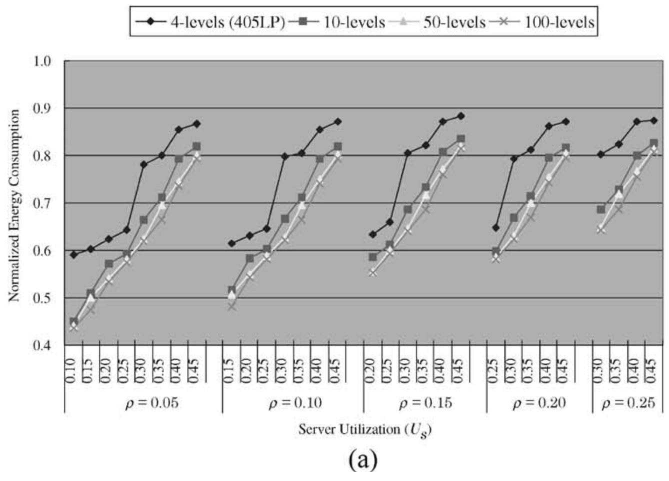
C. Practical Processor Model and Task Set

The performance of proposed DVS algorithms was also experimented using a real processor model and a practical task set. Although it was assumed that any speed (and voltage) between the minimum speed and the maximum speed can be used, real variable-voltage processors provide only finite numbers of speed levels. For example, Table V shows the speeds, voltages, and power ranges of PowerPC 405LP [25]. So, for various processor models, the proposed DVS algorithms are evaluated. Fig. 13 shows the experimental results. The experiment used four kinds of processor models, which provide 100, 50, 10, and 4 voltage levels, respectively. The processor with four levels of voltage is the PowerPC 405LP.

As shown in Fig. 13, as the number of available voltage levels decreases, the energy consumption increases but the response time decreases. However, even in the PowerPC 405LP model, the energy consumption is reduced by 23%.

The authors also experimented with a practical application. For a mixed task set, a videophone application implemented with a Java language was selected. The videophone application has four periodic tasks for video/audio encoding/decoding as shown in Table VI. Since it is implemented with Java, the GC is invoked aperiodically when the available memory size is below a given threshold. The mean interarrival time and the mean execution time of GC are 600 and 3.732 ms, respectively. To service the GC, the authors used a CBS whose period (T_s) and budget (Q_s) are 300 and 4 ms, respectively.

Table VII shows the experimental results. The authors experimented using CBS/DRA-W and CBS/DRA-P. Under the processor model with 100 levels of voltages, the energy consumption was reduced by 14% and the response time increased by 35% to 57%. Under the PowerPC 405LP processor model, the energy consumption was reduced by 7% and the response time was increased by 18% to 40%. From these results, it can be known that the proposed DVS algorithm shows good results for practical application and the real processor model.



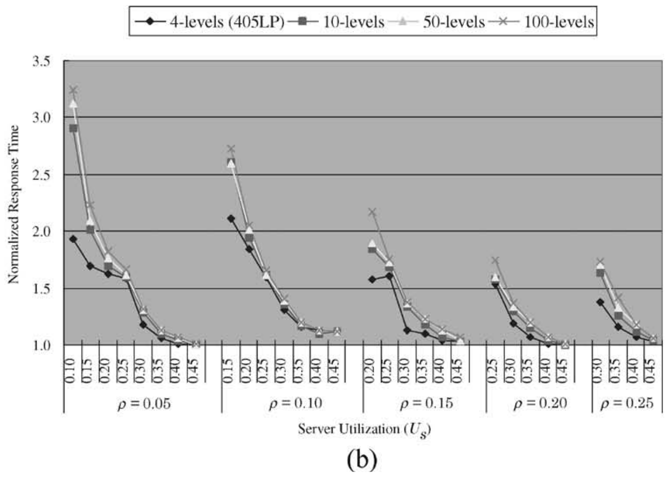


Fig. 13. Experimental results varying the voltage levels. (a) Energy consumption. (b) Response time.

TABLE VI
JAVA-BASED VIDEOPHONE APPLICATION MODEL

Task	Period (millisecond)	WCET (millisecond)	Utilization (millisecond)
Video Encoding	66.667	50.386	0.756
Video Encoding	66.667	9.826	0.147
Speech Encoding	40	1.844	0.046
Speech Encoding	40	1.383	0.035
U_p			0.984

GC Task Model	$1/\lambda = 600$	$1/\mu = 3.732$	$\rho = 0.006$
CBS	$T_s = 300$	$Q_s = 4$	$U_s = 0.012$

TABLE VII
NORMALIZED ENERGY CONSUMPTION AND AVERAGE RESPONSE TIME OF VIDEOPHONE APPLICATION

Voltage Levels	DVS Algorithm	Energy Consumption	Response Time
100 levels	CBS/DRA-W	0.86	1.57
	CBS/DRA-P	0.86	1.35
4 levels	CBS/DRA-W	0.93	1.40
(405LP)	CBS/DRA-P	0.93	1.18

VII. CONCLUSION

This paper proposed on-line dynamic voltage scaling (DVS) algorithms for mixed task systems. Considering the trade-off between the energy consumption and the response time, the authors modified the existing on-line DVS algorithms for periodic task sets to utilize the execution behaviors of various bandwidth-preserving servers. It was proved that the proposed algorithms guarantee that the response time delay is no greater than $T_s - Q_s$. By using a more aggressive slack estimation method than the existing algorithms for the mixed task sets, the proposed algorithms reduced the energy consumption by 35% to 48% over the non-DVS scheme. The authors also proposed a new slack distribution method that provides better response times with slight energy overheads. For the experiments using a practical application and a real processor model, the proposed DVS algorithms showed a good energy performance.

This work can be extended in several directions. Although the proposed algorithms only guarantee that the maximum response time delay is $T_s - Q_s$, it will be more useful if the authors can control the maximum response time delay with an arbitrary δ value. Furthermore, it will be interesting to use the DVS algorithm to utilize the temporal locality of aperiodic requests. When aperiodic requests are sparse, the authors could use a larger δ value for a more energy-efficient schedule.

ACKNOWLEDGMENT

The authors would like to acknowledge the anonymous reviewers whose helpful comments improved both the contents and the presentation of the manuscript.

REFERENCES

[1] J. K. Strosnider, J. P. Lehoczky, and L. Sha, “The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments,” *IEEE Trans. Comput.*, vol. 44, no. 1, pp. 73–91, Jan. 1995.
[2] B. Sprunt, L. Sha, and J. P. Lehoczky, “Aperiodic task scheduling for hard real-time systems,” *J. Real-Time Syst.*, vol. 1, no. 1, pp. 27–60, 1989.
[3] M. Spuri and G. Buttazzo, “Scheduling aperiodic tasks in dynamic priority systems,” *J. Real-Time Syst.*, vol. 10, no. 2, pp. 179–210, 1996.
[4] L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” in *Proc. IEEE Real-Time Systems Symp.*, Madrid, Spain, 1998, pp. 4–13.
[5] J. W. S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice-Hall, 2000.
[6] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, “Performance comparison of dynamic voltage scaling algorithms for hard real-time systems,” in *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.*, San Jose, CA, 2002, pp. 219–228.
[7] Y. Shin and K. Choi, “Power conscious fixed priority scheduling for hard real-time systems,” in *Proc. Design Automation Conf.*, New Orleans, LA, 1999, pp. 134–139.
[8] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez, “Dynamic and aggressive scheduling techniques for power-aware real-time systems,” in *Proc. IEEE Real-Time Systems Symp.*, London, U.K., 2001, pp. 95–106.
[9] P. Pillai and K. G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in *Proc. ACM Symp. Operating Systems Principles*, Banff, Canada, 2001, pp. 89–102.
[10] 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 *Proc. Design Automation and Test Europe*, Paris, France, 2002, pp. 788–794.

[11] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *Proc. Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, 2004, pp. 653–658.
[12] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed priority preemptive systems," in *Proc. IEEE Real-Time Systems Symp.*, Phoenix, AZ, 1992, pp. 110–123.
[13] W. Yuan and K. Nahrstedt, "Integration of dynamic voltage scaling and soft real-time scheduling for open mobile systems," in *Proc. Int. Workshop Network and Operating Systems Support Digital Audio and Video*, Miami Beach, FL, 2002, pp. 105–114.
[14] Y. Doh, D. Kim, Y.-H. Lee, and C. M. Krishna, "Constrained energy allocation for mixed hard and soft real-time tasks," in *Proc. Int. Conf. Real-Time and Embedded Computing Systems and Applications*, Tainan, Taiwan, 2003, pp. 533–550.
[15] H. Aydin and Q. Yang, "Energy-responsiveness tradeoffs for real-time systems with mixed workload," in *Proc. 10th IEEE Real-time and Embedded Technology and Applications Symp.*, Toronto, Canada, 2004, pp. 74–83.
[16] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "FAST: Frequency-aware static timing analysis," in *Proc. 24th IEEE Int. Real-Time Systems Symp.*, Cancun, Mexico, 2003, pp. 40–51.
[17] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," in *Proc. 40th Design Automation Conf.*, Anaheim, CA, 2003, pp. 125–130.
[18] Transmeta Crusoe. [Online]. Available: http://www.transmeta.com/crusoe/longrun.html
[19] AMD, Inc., AMD PowerNow Technology, 2000.
[20] Intel, Inc., The Intel(R) XScale(TM) Microarchitecture Technical Summary, 2000.
[21] R. Hamburgen, D. Wallach, M. Viredaz, L. Brakmo, C. Waldspurger, J. Bartlett, T. Mann, and K. Farkas, "Itsy: Stretching the bounds of mobile computing," *IEEE Computer*, vol. 34, no. 4, pp. 28–36, Apr. 2001.
[22] W. Kim, J. Kim, and S. L. Min, "Preemption-aware dynamic voltage scaling in hard real-time systems," in *Proc. Int. Symp. Low Power Electronics and Design (ISLPED)*, Newport Beach, CA, 2004, pp. 393–398.
[23] M. A. Viredaz and D. A. Wallach, "Power evaluation of a handheld computer," *IEEE Micro*, vol. 23, no. 1, pp. 66–74, Jan./Feb. 2003.
[24] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale (VLSI) Integr. Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.
[25] C. Rusu, R. Melhem, and D. L. Mosse, "Maximizing the system value while satisfying time and energy constraints," *IBM J. Res. Develop.*, vol. 47, no. 5/6, pp. 689–702, 2003.

Dongkun Shin received the B.S. degree in computer science and statistics, the M.S. degree in computer science, and the Ph.D. degree in computer science and engineering, all from the Seoul National University, Seoul, Korea.

He is a Senior Engineer at Samsung Electronics Co., Seoul, Korea. His research interests include low-power systems, computer architecture, and embedded and real-time systems.

Dr. Shin is a Member of the Association for Computing Machinery (ACM).

Jihong Kim (M'00) received the B.S. degree in computer science and statistics from the Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, WA.

He is an Associate Professor at the School of Computer Science and Engineering, Seoul National University, Seoul, Korea. His research interests include embedded systems, computer architecture, Java computing, and multimedia and real-time systems.

Dr. Kim is a Member of the ACM.