# Dynamic Voltage Scaling for Mixed Task Systems in Priority-Driven Systems

Dongkun Shin
School of Computer Science and Engineering
Seoul National University

**Abstract**

We describe 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 the system with both periodic and aperiodic tasks because of arbitrary temporal behaviors of aperiodic tasks. We propose an off-line DVS algorithm and on-line DVS algorithms that are based on existing DVS algorithms. The proposed algorithms utilize the execution behaviors of scheduling server for aperiodic tasks. Experimental results show that the proposed algorithms reduce the energy consumption by 25% and 18% under the RM scheduling policy and the EDF scheduling policy, respectively.

## I. Introduction

Dynamic voltage scaling (DVS) [3] is one of the most effective approaches in reducing the power consumption of real-time systems. When the required performance of the target system is lower than the maximum performance, supply voltage and clock speed can be dynamically reduced to the lowest possible extent that ensures a proper operation of the system. Recently, many voltage scheduling algorithms have been proposed for hard real-time systems [9], [2], [8], [5]. All of these algorithms assume that the system consists of periodic hard real-time tasks only and the task release times are known *a priori*. For periodic tasks, these algorithms assign the proper speed to each task dynamically while guaranteeing all their deadlines. Since the arrival time, the worst case execution time (WCET), and the deadline of each periodic task are known, DVS algorithms can determine how much slack times are generated due to the early completion of a task and how much the speeds of next tasks can be scaled down.

However, many practical real-time applications require aperiodic tasks as well as periodic tasks. For example, consider multimedia applications (e.g., MP3 or MPEG player) in which audio or video data is decoded periodically maintaining consistent output rates. These systems continue accepting user inputs that need prompt responses (e.g., volume control, playback control

or playlist editing). While the decoding tasks are periodic tasks, the tasks to service user inputs are aperiodic tasks. While periodic tasks are time-driven with hard deadlines, aperiodic tasks are usually event-driven (i.e., activated at arbitrary times) with soft deadlines (or best-effort tasks). Generally, the arrival process is Poisson process with parameter $\lambda$ and the service process is exponential process with parameter $\mu$. In this paper, we call a system with periodic and aperiodic tasks as a mixed task system.

In mixed task systems, there are two design objectives. The first objective is to guarantee the schedulability of all periodic tasks under worst-case execution scenarios. That is, aperiodic tasks should not prevent periodic tasks from completing before their deadlines. The second objective is that aperiodic tasks should have "good" average response times. To satisfy these objectives, many scheduling algorithms based on the "server" concept had been proposed [12], [10], [11], [1].

In this paper, we introduce the third design objective for the energy consumption in the mixed task system. That is, the third objective is to minimize the total energy consumption due to *both* periodic tasks and aperiodic tasks. Although the existing DVS algorithms can be effective for optimizing the energy consumption of periodic tasks, they cannot be used for mixed task systems. The arbitrary behaviors of aperiodic tasks prevent the DVS algorithms from identifying the slack times. Therefore, it is necessary to modify the existing DVS algorithms to be applicable to mixed task systems with aperiodic tasks.

In this paper, we propose DVS algorithms that guarantee the first objective (i.e., timing constraints of periodic tasks) while making the best effort of satisfying the third objective (i.e., low energy) with a reasonable performance bound on the second objective (i.e., good average response time). First, we describe an off-line static voltage scaling algorithm which considers the expected workload of aperiodic tasks. Second, we present on-line dynamic voltage scaling algorithms by modifying existing on-line voltage scaling algorithms for a periodic task set.

The modified DVS algorithms utilize the execution behaviors of each scheduling server for aperiodic tasks to apply the key ideas of the existing DVS algorithms such as [9], [2]. The task schedules generated by the proposed DVS algorithms can reduce the energy consumption by 18%$\sim$25% over the task schedules which execute all tasks at full speed and power down at idle intervals (i.e., the power-down mode). To the best of our knowledge, our work is the *first* attempt to develop *on-line* DVS algorithms for the mixed task set.

The rest of this paper is organized as follows. In Section II, we summarize the related works on aperiodic task scheduling and the recent efforts to integrate dynamic voltage scheduling into aperiodic task scheduling. The proposed static DVS algorithm is described in Section III while the dynamic DVS algorithms are presented in Section IV. In Section V, the experimental results are discussed. Section VI concludes with a summary and future works.

## II. Related Works

In this section, we review 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. In this approach, aperiodic tasks are scheduled and executed only at times when there is no periodic task ready for execution. Though this method guarantees the schedulability of 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 which 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 simplest server is the *Polling Server* (PS). PS is ready for execution periodically at integer multiplies 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 same to a periodic task which has the period $T_s$ and the worst case execution time (WCET) $Q_s$, we can test the schedulability of the system using the traditional RM or EDF schedulability test.

The *Deferrable Server* (DS) [12] was introduced to solve the poor performance of background scheduling and PS. Unlike PS, DS can service an aperiodic request at any time as long as the budget is not exhausted. Though this feature of DS provides better performance than that of PS, a lower priority task could miss its deadline even if the task set seemed to be schedulable by the schedulability test because DS can defer its execution. To solve this problem, the *Sporadic Server* (SS) [10] was proposed. SS 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, we can treat a SS exactly like the periodic task $(Q_s, T_s)$ when we check for the schedulability

of the system.

Though there are the 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, the *Total Bandwidth Server* (TBS) [11] 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, there can be overrun when the real execution time is longer than the WCET. (This situation could occur for aperiodic tasks.) Recently, the *Constant Bandwidth Server* (CBS) [1] was proposed to solve the overrun problem of TBS.

A different approach for scheduling aperiodic tasks is the *Slack Stealing* technique [7]. It steals all available slack from periodic tasks and gives it to aperiodic tasks. Though 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 the slack stealing is to give as much as possible time to aperiodic tasks executing periodic tasks at full speed, the slack stealing is improper to be integrated with DVS algorithms. So, we concentrate on the server techniques in this paper.

Despite of many researches on aperiodic task scheduling, there have been few studies to adapt the DVS technique to aperiodic task scheduling. A recent work by W. Yuan and K. Nahrstedt [13] proposed a DVS algorithm for soft real-time multimedia and best-effort applications. They handled only the constant bandwidth server. The target of their algorithm is aperiodic task systems, not mixed task systems.

Y. Doh *et al.* [4] also investigated the problem of allocating both energy and utilization for mixed task sets. They used the total bandwidth server and considered the static 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 the minimal response times. While their algorithm is an off-line static speed assignment algorithm under the EDF scheduling policy, our work in this paper considers both static and dynamic algorithms under both RM and EDF scheduling policies. Another difference is that we concentrate on minimizing the energy consumption under the constraint on the average response time.

## III. Static Scheduling for Mixed Task Sets

Pillai and Shin [8] proposed the static voltage scheduling algorithms for periodic tasks using the RM and EDF schedulability tests. Their static scheduling algorithm 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 TBS, if the worst case utilization of periodic tasks is 0.3 and the utilization of TBS is 0.4 at 100 MHz clock speed, the static scheduling algorithm determines the clock speed as 70 $MHz$ ($= 100\ \mathrm{MHz} \cdot (0.3 + 0.4)$).

However, the scheduling server for aperiodic tasks generally occupies a large utilization compared with the 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 TBS has many idle intervals. However, we cannot use the clock speed 50 MHz ($= 100\ \mathrm{MHz} \cdot (0.3 + 0.2)$) because it can produce deadline misses when the real utilization of aperiodic task is larger than 0.2.

Therefore, in static voltage scheduling, we should consider both the expected workload and the schedulability condition. Our static voltage scheduling algorithm selects the operating speed $s_p$ of periodic tasks and the operating speed $s_s$ of scheduling server for aperiodic tasks, respectively. $s_p$ and $s_s$ is the relative speed values normalized by the maximum clock speed. $s_p$ 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 the static scheduling can be formulated as follows:

**Static Speed Assignment Problem**

**Given** $U_p, U_s, \omega,$ and $\rho$,

**find** $s_p$ and $s_s$ such that

$E = U_p \cdot \omega \cdot s_p^2 + \rho \cdot s_s^2$ is minimized

**subject to** $\dfrac{U_p}{s_p} + \dfrac{U_s}{s_s} \leq U_{lub}$ and $0 \leq s_p, s_s \leq 1.$

where $U_p$ is the worst case utilization of periodic task set, $U_s$ is the server utilization, $\omega$ is

the average workload ratio of periodic tasks, and $\rho$ is the average workload of aperiodic tasks ($\rho = \lambda/\mu$). $E$ is a metric reflecting energy consumption[1]. $U_{lub}$, which is the least upper bound of schedulable utilization, is 1 at the EDF scheduling and $n(2^{1/n} - 1)$ for $n$ tasks at the RM scheduling[2], respectively. Using the Lagrange transform, we can get a following optimal solution for $s_p$ and $s_s$.

$$s_p = \frac{1}{U_{lub}} \left( U_p + U_s \sqrt[3]{\frac{\rho}{U_s \cdot \omega}} \right), \quad s_s = \frac{1}{U_{lub}} \left( U_p \sqrt[3]{\frac{U_s \cdot \omega}{\rho}} + U_s \right)$$

Under the assumption that we can know the exact $\omega$ and $\rho$ values, we can get the optimal static speeds for periodic and aperiodic tasks. Table I shows the experimental results of the optimal static speed assignment. The results show the reduction of energy consumption and response time varying $U_s$ with fixed values of $U_p$, $\omega$ and $\rho$. Aperiodic tasks are assumed to be serviced by the total bandwidth server. We assumed that if the system is idle it enters into the power-down mode. We compared our optimal speed assignment method (OPT) with Pillai's uniform speed assignment method (UNI) which assigns the same speed to both periodic tasks and aperiodic tasks making the total utilization as $U_{lub}$. The optimal speed assignment method reduced the energy consumption and the average response time up to 11% and 26%, respectively. Since the scheduling server gets a higher speed than the speed for periodic tasks when $\omega > \rho$, the optimal speed assignment reduces the average response time as well as the energy consumption.

From the result, we can see if a higher $U_s$ is used, the average response time of aperiodic tasks decreases and total energy consumption increases. Since two objectives of the response time and the energy consumption conflict with each other, it is recommended to use the constraint on the response time. We can determine the minimum value of $U_s$ which satisfies the constraint minimizing the energy consumption. For example, if we have the constraint that the average response time should be lower than 1 msec, then we can select 0.2 for the server utilization from the results in Table I.

[1]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.

[2]When a deferrable server is used, the utilization bound is 0.6518 [12].

| $U_s$ | Energy consumption (mJ) | | | Response time (msec) | | |
|---|---|---|---|---|---|---|
| | UNI | OPT | Reduction(%) | UNI | OPT | Reduction(%) |
| 0.15 | 52.02 | 50.35 | 3 | 1.65 | 1.38 | 16 |
| 0.20 | 60.73 | 54.50 | 10 | 1.00 | 0.75 | 26 |
| 0.25 | 65.81 | 58.81 | 11 | 0.94 | 0.75 | 20 |
| 0.30 | 71.19 | 63.41 | 11 | 0.89 | 0.75 | 16 |
| 0.35 | 76.66 | 72.32 | 6 | 0.84 | 0.75 | 11 |
| 0.40 | 87.28 | 77.86 | 11 | 0.79 | 0.75 | 5 |

$U_p = 0.4$, $\omega = 0.75$, $\rho = 0.15$

TABLE I

STATIC SPEED ASSIGNMENT FOR TOTAL BANDWIDTH SERVER.

## IV. DYNAMIC SCHEDULING FOR MIXED TASK SETS

### *A. Problem Formulation*

We assume that a mixed task system $\mathcal{T}$ consists of $n$ periodic tasks, $\tau_1, \cdots, \tau_n$, and an aperiodic task, $\sigma$. The aperiodic task $\sigma$ 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. We use $q_s$ 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 within the limit of its budget $q_s$. We denote the release (arrival) time, the start time, and the completion time of $\sigma$ as $r(\sigma)$, $\eta(\sigma)$ and $e(\sigma)$, respectively.

A periodic task $\tau_i$ is specified by $(C_{\tau_i}, T_{\tau_i})$ where $C_{\tau_i}$ and $T_{\tau_i}$ are the worst-case execution cycles (WCEC) and the period of $\tau_i$, respectively. We assume that periodic tasks have relative deadlines equal to their periods. The $j$-th instance of $\tau_i$ and the $k$-th instance of $\sigma$ are denoted by $\tau_{i,j}$ and $\sigma_k$, respectively. We assume that the aperiodic task instances $\sigma_1, \cdots, \sigma_m$ are executed during the hyper period $H$ of periodic tasks. We denote the operating speed of a periodic task $\tau_i$ (an aperiodic task $\sigma_k$) as $s(\tau_i)$ ($s(\sigma_k)$). We assume that the operating speeds are the values between 0 and 1.

If an aperiodic task $\sigma_k$ can be serviced without any interference by periodic tasks or another aperiodic tasks, the response time of the aperiodic task $\sigma_k$ 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 $\sigma_k$, respectively. However, the execution of the aperiodic task $\sigma_k$ is delayed due to the following factors:

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

We denote the delays due to the budget, queueing and preemption as $b(\sigma_k)$, $w(\sigma_k)$, and $p(\sigma_k)$, respectively. Then, the response time of $\sigma_k$ can be represented as

$$c(\sigma_k)/s(\sigma_k) + b(\sigma_k) + w(\sigma_k) + p(\sigma_k). \tag{1}$$

If the response time of $\sigma_k$ is $t$ in the non-DVS scheme, the response time will be increased to $t + D$ by a DVS algorithm because $s(\sigma_k)$, $b(\sigma_k)$, $w(\sigma_k)$ and $p(\sigma_k)$ are changed. We call the increase $D$ in the response time as the *response time delay*.

Our 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. Therefore, the problem of dynamic speed assignment problem for mixed task systems (DSAMTS) can be formulated as follows:

**Dynamic Speed Assignment Problem**

**Given** $\mathcal{T} = \{\tau_1, \cdots, \tau_n, \sigma\}, S$ and $\delta$,

**find** $s(\tau_{1,1}), \cdots, s(\tau_{n,H/T_n})$ and $s(\sigma_1), \cdots, s(\sigma_m)$ 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** $\forall i, j, e(\tau_{i,j}) \leq j \cdot T_{\tau_i}$ 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 $\sigma_k$. $D(\sigma_k)$ represents the *response time delay* of $\sigma_k$.

In this paper, we propose the DVS algorithms which provide solutions for the DSAMTS problem when $\delta = T_s - Q_s$. We plan to develop the DVS algorithms for the DSAMTS problem with an arbitrary value of $\delta$ as a future work.

Existing on-line DVS algorithms such as [9], [2], [8], [5] are not directly applicable for 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. As shown in Figure 1, it stretches the execution time of the periodic task ready for execution to the next arrival time of a periodic task when there is no another periodic task in ready queue. To use the *stretching-to-NTA* technique for a mixed task system, we should know the next arrival time of an aperiodic task as well as a periodic task. Though the arrival times of periodic tasks can be easily computed using their periods, we cannot know the arrival times of aperiodic tasks since they arrive at arbitrary times. If we ignore the arrival of aperiodic tasks, 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.

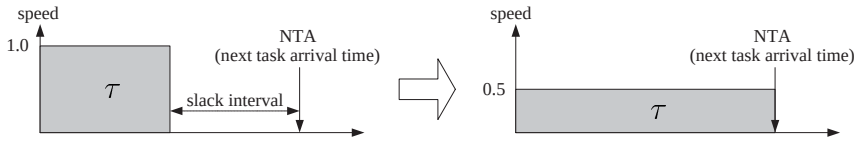



Fig. 1. The stretching-to-NTA algorithm.

To use the *priority-based slack-stealing* method or the *utilization updating* method, we 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 easily be defined as the difference between the WCET and the real execution time of the task. However, for the slack time from aperiodic tasks, we should be concerned about the scheduling server rather than aperiodic tasks because the utilization of scheduling server is

related with the schedulability condition.

Therefore, we need to modify on-line DVS algorithms to utilize the characteristics of scheduling servers. In this paper, we handle deferrable server [12] and sporadic server [10] for the fixed-priority scheduling policy, and total bandwidth server [11] and constant bandwidth server [1] for the dynamic-priority scheduling policy.

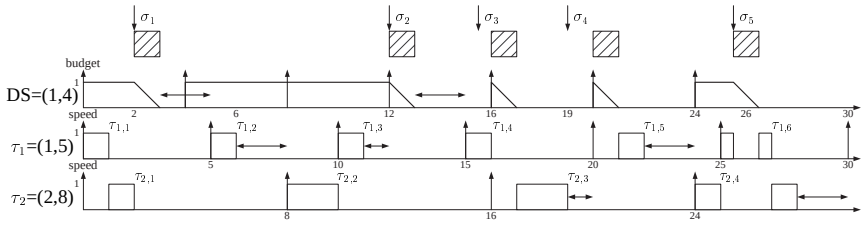
### *B. Scheduling Algorithms in Fixed-Priority Systems*

For fixed-priority systems, we assume the RM scheduling policy. Figure 2(a) shows the task schedule with a deferrable server. There are two periodic tasks, $\tau_1 = (1, 5)$ and $\tau_2 = (2, 8)$, and one DS $= (1, 4)$. Each periodic task and the DS is scheduled by the RM scheduler. The utilization of DS is 0.25 ($= \frac{Q_s}{T_s} = \frac{1}{4}$). DS preserves its budget 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 DS is not exhausted (e.g., task $\sigma_1$). If the budget is exhausted, aperiodic tasks should wait until the next replenishment time. For example, though the task $\sigma_4$ arrived at the time of 19, it is serviced at the time of 20.

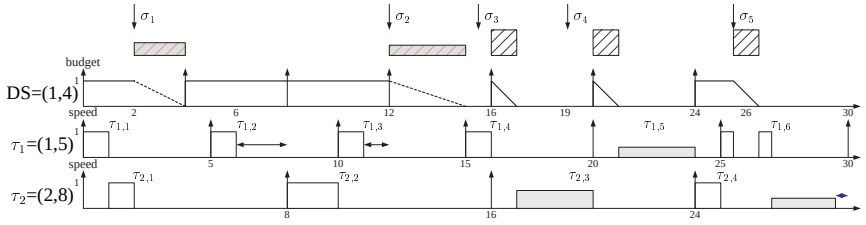
Although we cannot know the arrival times of aperiodic tasks, the *stretching-to-NTA* method can be used if we utilize the execution behavior of DS. There are two cases the current ready task can be stretched:

- **Rule for aperiodic task**: If there is no periodic task in the ready queue, execute an aperiodic task at the speed of $q_s/(min(NTA, R) - t)$ where $NTA$, $R$ and $t$ are the next arrival time of a periodic task, the next replenishment time and the current time, respectively.
- **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 $min(NTA, R)$. This is because the arriving aperiodic task is delayed until the next replenishment time if $q_s$ is 0. If $q_s > 0$, we cannot scale down the speed of the periodic task even though there is only one periodic task in the ready queue.

Using these two rules, we modified existing on-line DVS algorithms. Figure 2(b) shows the task schedule using the `lppsRM/DS` algorithm which is the modified version of `lppsRM` [9] for DS. `lppsRM` uses the *stretching-to-NTA* method. The aperiodic tasks $\sigma_1$ and $\sigma_2$ are stretched to the next arrival times of periodic tasks (5 and 15) because there is no periodic task in ready queue. The periodic tasks $\tau_{1,5}$, $\tau_{2,3}$, and the latter part of $\tau_{2,4}$ are stretched to $min(NTA, R)$ because $q_s$ is 0. We cannot stretch the tasks $\tau_{1,2}$ and $\tau_{1,3}$ because $q_s$ is larger than 0. Though




(a) *DS* without DVS




(b) *lppsRM/DS*

Fig. 2. Task schedules with a deferrable server.

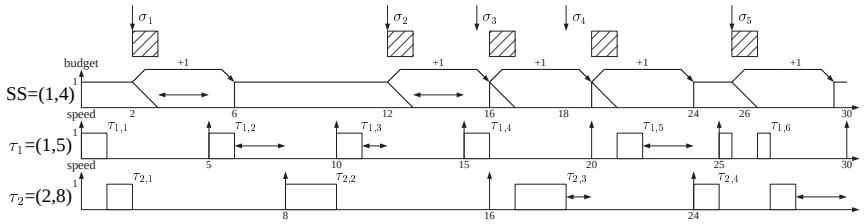
there is no deadline miss even if $\sigma_1$ is stretched to 5, we limit the stretching bound by the replenishment time to bound the delay of response time of aperiodic tasks. Using this policy, we can guarantee that the maximum increase of the average response time is $T_s - Q_s$.

Figure 3(a) shows the task schedule using a sporadic server SS, assuming the same task set. The budget of SS, $q_s$, is set to $Q_s$ at time 0. If an aperiodic task is executed during the time $[t_1, t_2]$, $q_s$ is reduced by $t_2 - t_1$ until the time $t_2$. The budget $q_s$ is replenished by the amount of $t_2 - t_1$ 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 $\sigma_1$). If the budget is exhausted, aperiodic tasks should wait until the next replenishment time. For example, though the task $\sigma_4$ arrived at the time 19, it is serviced at the time 20. Figure 3(b) shows the task schedule using the *lppsRM/SS* algorithm which is the modified version of *lppsRM* [9] for SS.

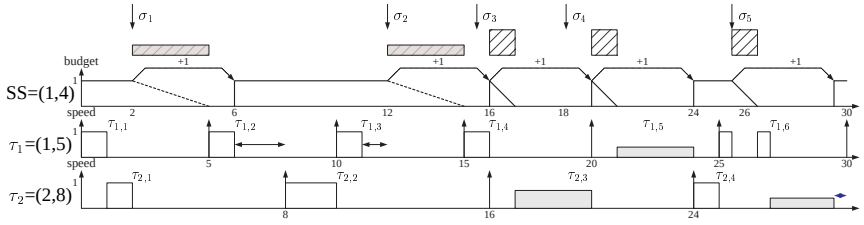
**Definition 1:** The replenishment time $\mathbb{R}(\sigma_k)$ is the last replenishment time among the replenishment times which is earlier than the completion time of $\sigma_k$, $e(\sigma_k)$.

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

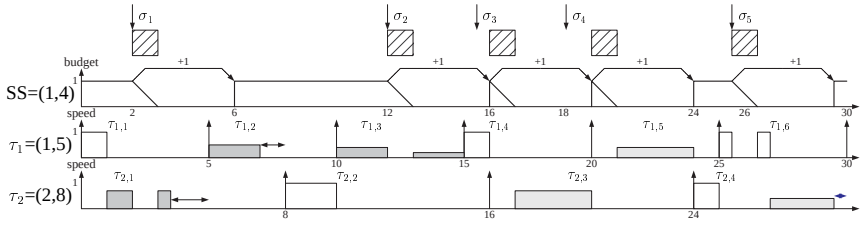
**Proof** The last replenishment time $\mathbb{R}(\sigma_k)$ is determined by the replenishment rule, the server




(a) *SS* without DVS




(b) *lppsRM/SS*




(c) *lppsRM/SS-SE*

Fig. 3. Task schedules with a sporadic 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 *lppsRM/DS* (or *lppsRM/SS*). So, we can conclude that the last replenishment times are same in both algorithms.

**Lemma 2:** The tasks (and the remaining execution cycles) to be executed after the last replenishment time $\mathbb{R}(\sigma_k)$ are same in both *lppsRM/DS* (or *lppsRM/SS*) and *DS* (or *SS*).

**Proof** By the stretching rules of *lppsRM/DS* (or *lppsRM/SS*) which does not stretch tasks over the replenishment time, the tasks executed between the replenishment time are not changed. By **Lemma 1**, the last replenishment time is not changed. Therefore, this lemma is true.

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

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

**case 1** $\eta(\sigma_k) \leq \mathbb{R}(\sigma_k)$.

In this case, there is no queueing delay after $\mathbb{R}(\sigma_k)$ because $\sigma_k$ is 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 the 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 `lppsRM/DS` (or `lppsRM/SS`) by **Lemma 2**. The remaining execution cycles of $\sigma_k$, $c_{rem}(\sigma_k)$ are also same in both algorithms by **Lemma 2**. We can know that $c_{rem}(\sigma_k)$ is not larger than the budget of the scheduling server by the definition of $\mathbb{R}(\sigma_k)$, i.e., $c_{rem}(\sigma_k) \leq Q_s$. We can also know the time interval $[t, min(R, NTA)]$ is smaller than or equal to $T_s$ when $\sigma_k$ is resumed at $t$ after $\mathbb{R}(\sigma_k)$ because $\mathbb{R}(\sigma_k) \leq t$ and $R - \mathbb{R}(\sigma_k) \leq T_s$. Therefore, we can show that $e'(\sigma_k) - e(\sigma_k) \leq T_s - Q_s$ as follows:

$$
\begin{aligned}
e(\sigma_k) &= c_{rem}(\sigma_k) + \mathbb{P}(\sigma_k) - \mathbb{R}(\sigma_k) \\
e'(\sigma_k) &= c_{rem}(\sigma_k) \cdot (min(R, NTA) - t)/Q_s + \mathbb{P}'(\sigma_k) - \mathbb{R}(\sigma_k) \\
e'(\sigma_k) - e(\sigma_k) &= c_{rem}(\sigma_k) \cdot (min(R, NTA) - t)/Q_s - c_{rem}(\sigma_k) \\
&= c_{rem}(\sigma_k)/Q_s \cdot ((min(R, NTA) - t) - Q_s) \\
&\leq ((min(R, NTA) - t) - Q_s) \\
&\leq (T_s - Q_s)
\end{aligned}
$$

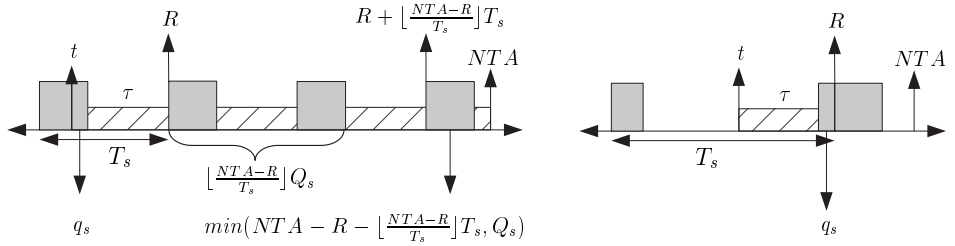
**case 2** $\eta(\sigma_k) > \mathbb{R}(\sigma_k)$.

In this case, we can treat all aperiodic tasks $\sigma_{k-j}, \cdots, \sigma_{k-1}, \sigma_k$ which are completed after $\mathbb{R}(\sigma_k)$ as one aperiodic task $\sigma_{k-j,\cdots,k}$ which has the execution cycles of $c(\sigma_{k-j,\cdots,k}) = \sum_{i=k-j}^{k} c(\sigma_i)$. Then, the proof is same to the case 1.

Consequently, we can conclude that $D(\sigma_k) \leq T_s - Q_s$ for all $\sigma_k$ scheduled by the `lppsRM/DS` and `lppsRM/SS` algorithm. □

Though we can reduce the energy consumption by `lppsRM/SS` algorithm, the algorithm can show poor performance when the workload of aperiodic tasks is small. In this case, since the budget $q_s$ is larger than 0 at most of scheduling points, we cannot use the stretching rule for periodic task. Extremely, when there is no aperiodic request, there is nothing to do for the DVS algorithm. Therefore, we need a more advanced DVS algorithm which can be applicable to the mixed task system with a low aperiodic workload. For this purpose, we propose a new slack estimation method, *bandwidth-based slack-stealing*, which identifies the maximum slack time for a periodic task considering the bandwidth of scheduling server. Figure 3(c) shows the `lppsRM/SS-SE` algorithm, which is based on `lppsRM/SS` but uses the *bandwidth-based slack-stealing* method. 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.

Figure 4 shows the *bandwidth-based slack-stealing* method. In Figure 4, $T_\tau$ is the period of $\tau$, $t$ is the current time, $NTA$ is the next periodic task arrival time and $R$ is the next replenishment time of SS. We should consider two different cases depending on the priority of SS. Figure 4(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. Figure 4(b) shows the case when $T_\tau < T_s$. In this case, the task $\tau$ is stretched to $min(R, NTA) - q_s$. Although there is no deadline miss even when the periodic task $\tau$ is completed after $R$, the proposed DVS algorithm is designed to bound the response time delay. Under this policy, the *preemption delay* is increased but we can guarantee that $D(\sigma_k) \leq T_s - Q_s$ for all $\sigma_k$ because $\sigma_k$ is not delayed above the replenishment time $R$.




(a) $T_\tau > T_s$ (b) $T_\tau < T_s$

Fig. 4. Bandwidth-based slack stealing in `lppsRM/SS-SE`.

From Figure 4, the maximum available time $MAT$ of a task $\tau$ can be calculated as follows:

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

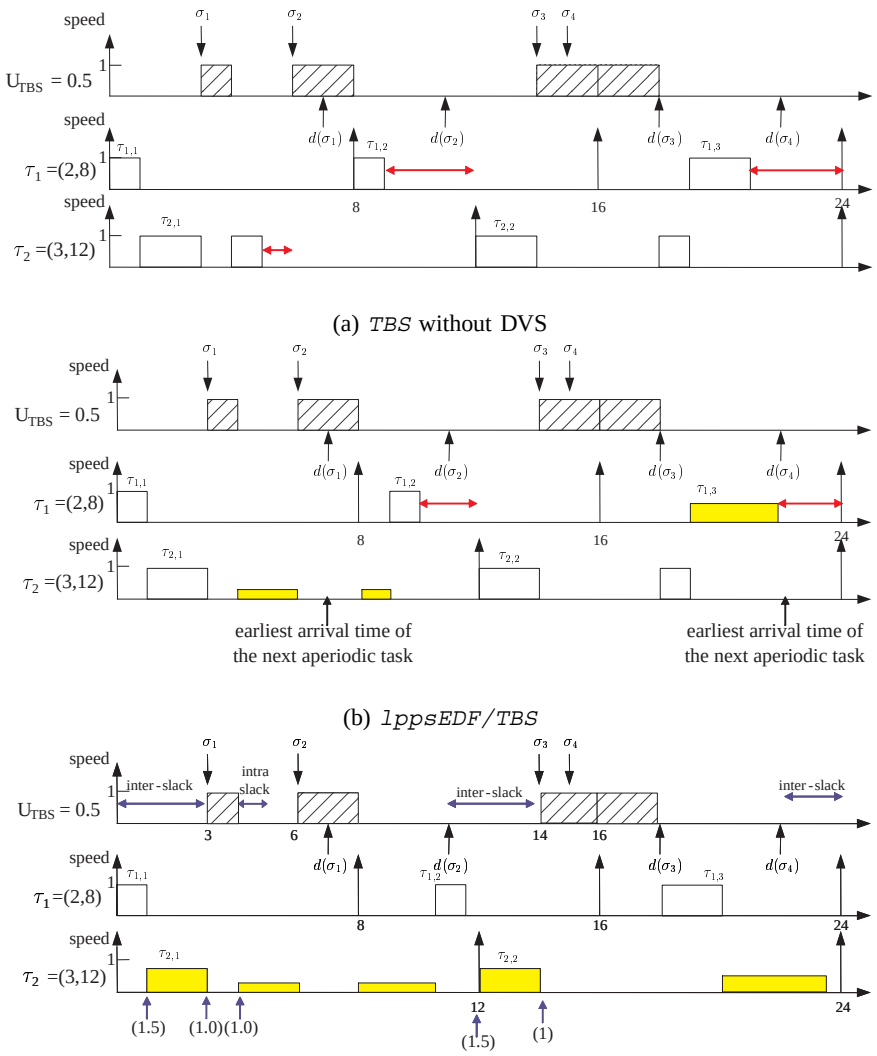
In Figure 3(c), the periodic tasks $\tau_{1,2}$, $\tau_{1,3}$ and $\tau_{2,1}$ are stretched by the *bandwidth-based slack-stealing* method. For example, at the time 5, the task $\tau_{1,2}$ has the available time 2 ($= NTA - t - q_s = 8 - 5 - 1$). A side effect of the *bandwidth-based slack-stealing* 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.

### *C. Scheduling Algorithms in Dynamic-Priority Systems*

For dynamic-priority systems, we assume the EDF scheduling policy. Figure 5(a) shows the task schedule with a TBS. There are two periodic tasks, $\tau_1 = (2, 8)$ and $\tau_2 = (3, 12)$, and one TBS with $U_s = 0.5$. $U_s$ is the utilization of TBS. If $U_p + U_s \leq 1$, the periodic tasks are schedulable. When an aperiodic task $\sigma_k$ arrives, TBS sets the deadline of $\sigma_k$ to $d(\sigma_k) = max(r(\sigma_k), d(\sigma_{k-1})) + C/U_s$, where $C$, $r(\sigma_k)$ and $d(\sigma_k)$ is the WCEC, the release time, and the deadline of $\sigma_k$, respectively. For example, when an aperiodic task $\sigma_2$, whose WCEC is 2, arrives at 6, TBS sets $\sigma_2$'s deadline to 11 ($= max(6, 7) + 2/0.5$). When a task $\sigma_3$ arrives at 14, it preempts the task $\tau_{2,2}$ because $\sigma_3$'s deadline is 18 and $\tau_{2,2}$'s deadline is 24.

With TBS, we can employ a stretching rule for aperiodic tasks similar to the stretching rule used in `lppsRM/DS`. An aperiodic task $\sigma_k$ can be stretched to $min(d(\sigma_k), NTA)$. (Note that $d(\sigma_k)$ is used instead of $R$ in `lppsRM/DS`.) However, we cannot employ the stretching rule for periodic task used `lppsRM/DS` because TBS is not controlled by the budget. Instead, we can make use of the fact that TBS sustains the utilization of aperiodic tasks as $U_s$. If we can endure a little degradation of aperiodic task response time, we can delay an aperiodic task $\sigma_i$ until $d_{i-1}$ when $r_i < d_{i-1}$. This delay does not affect the utilization of TBS and does not cause the deadline miss of periodic task. Delaying an aperiodic task until $d_{i-1}$ is identical with assuming the earliest arrival time of the aperiodic task $\sigma_i$ as $d_{i-1}$.

Figure 5(b) shows the task schedule using the `lppsEDF/TBS` algorithm which is a modified version of `lppsEDF` [9] using the *stretching-to-NTA* method. For example, the remaining part




(a) *TBS* without DVS

(b) *lppsEDF/TBS*

(c) *DRA/TBS*

Fig. 5. Task schedules with a total bandwidth server.

of the task $\tau_{2,1}$ at the time of 4 can be stretched to $d_1 = 7$. When an aperiodic task $\sigma_2$ arrives at the time of 6, it preempts $\tau_{2,1}$ because its priority (i.e., deadline) is higher than $\tau_{2,1}$ but produces no deadline miss. If the priority of $\tau_{2,1}$ is higher than $\sigma_2$, the start of $\sigma_2$ will be delayed until 7. Unfortunately, we cannot bound the maximum response time delay in *lppsEDF/TBS* because it is not controlled by the budget.

To use the *priority-based slack-stealing* method for TBS, we should identify the slack times of TBS. There are two types of slack times available when TBS is used:

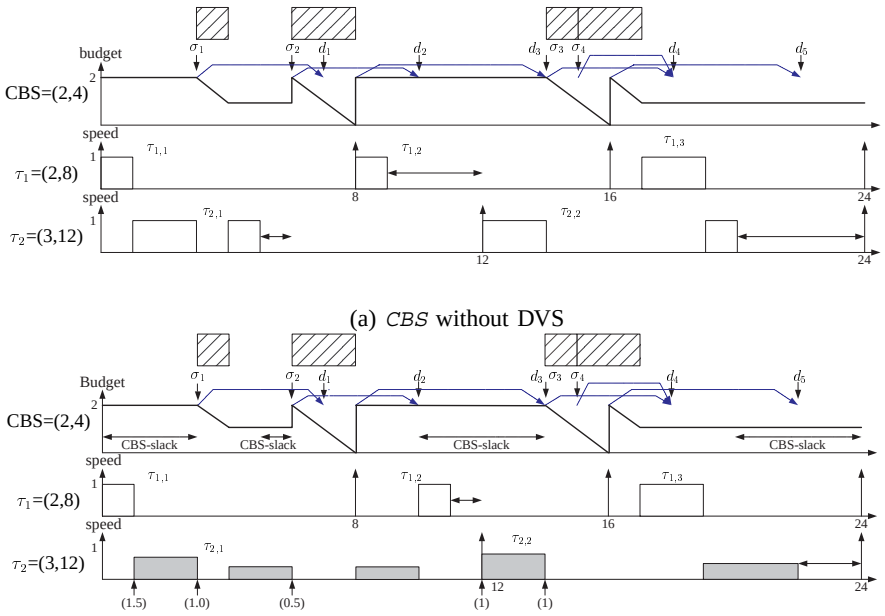
- **Inter-slack**: If an interval $[t_1, t_2]$ in TBS is not overlapped with any active interval of aperiodic tasks $[r_k, d_k]$, there is $(t_2 - t_1) \cdot U_s$ amount of slack time. This is because the total utilization does not exceed 1 even if an aperiodic task with the execution time of $(t_2 - t_1) \cdot U_s$ is executed during the interval $[t_1, t_2]$.
- **Intra-slack**: When an aperiodic task, whose WCET is $C$, consumes only the time of $c$, there is $(C - c)$ amount of slack time.

Figure 5(c) shows the task schedule using the *DRA/TBS* algorithm which is the modified version of *DRA* [2] using the *priority-based slack-stealing*. Originally, in the *DRA* algorithm, when a task $\tau$ is to be executed, the slack times due to the early completions of tasks which have the higher priorities than the priority of $\tau$ are computed and the speed of $\tau$ is determined using the slack times. The *DRA/TBS* algorithm uses the same technique except that it considers the inter-slack as well as the intra-slack of TBS.

For example, in Figure 5(c), when a task $\tau_{2,1}$ is scheduled at the time of 1, there is a slack time 1.5 (1 from the early completion of $\tau_{1,1}$ and 0.5 from the inter-slack of TBS 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 $\sigma_1$ is completed consuming only the 1 time unit, the intra-slack 1.0 is transferred to the remaining part of the task $\tau_{2,1}$ lowering its clock speed. Using the *DRA/TBS* algorithm, we can get a better energy efficiency than that of the *lppsEDF/TBS* algorithm because *DRA/TBS* exploits more slack times.

Figure 6(a) shows the task schedule using a constant bandwidth server 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 $U_p + U_s \leq 1$, where $U_p$ is the maximum utilization of periodic tasks, the task set is schedulable.

At each instant, a CBS deadline $d_k$ is associated with CBS. At the beginning $d_0 = 0$. Each served aperiodic task $\sigma_i$ is assigned a dynamic deadline equal to the current server deadline $d_k$. Whenever a served task executes, the budget $q_s$ is decreased by the same amount. When $q_s = 0$, the server budget is replenished to the maximum value $Q_s$ and a new server deadline is generated as $d_{k+1} = d_k + T_s$. A CBS is said to be active at time $t$ if there are pending jobs; that is, if there exists a served task $\sigma_i$ such that $r(\sigma_i) \leq t < e(\sigma_i)$, where $r(\sigma_i)$ and $e(\sigma_i)$ are the arrival time and the completion time of the task $\sigma_i$. A CBS is said to be idle at time $t$ if it is not active. When a task $\sigma_i$ arrives and the server is active the request is enqueued in a queue




(a) *CBS* without DVS




(b) *DRA/CBS*

Fig. 6. Task schedules with a constant bandwidth server.

of pending jobs according to a given (arbitrary) non-preemptive discipline (e.g., FIFO).

When an aperiodic task $\sigma_i$ arrives at $r(\sigma_i)$ and the server is idle (when CBS does not service aperiodic tasks), if $q_s \geq (d_k - r(\sigma_i))U_s$ 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. When a job finishes, the next pending job, if any, is served using the current budget and deadline. If there are no pending jobs, the server becomes idle. At any instant, a job is assigned the last deadline generated by the server.

For example, when an aperiodic task $\sigma_1$ arrives at time 3, CBS sets its deadline $d_1$ to 7 $(= r(\sigma_1) + T_s = 3 + 4)$ and $\sigma_1$ uses the deadline. When an aperiodic task $\sigma_2$ arrives at time 6, CBS sets $\sigma_2$'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 $\sigma_3$ arrives at 14, CBS sets $\sigma_3$'s deadline to 18 and $\sigma_3$ preempts the task $\tau_{2,2}$. When an aperiodic task $\sigma_4$ arrives at 15, CBS sets $\sigma_4$'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 $\sigma_4$'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 $U_s$.

For CBS, we cannot employ the stretching rule for periodic tasks used for DS and SS because there are no finite intervals of time in which the budget is equal to zero. But, we can modify the stretching rule like `lppsEDF/TBS`.

To use the *priority-based slack-stealing* [6] method for CBS, we should identify the slack times of CBS. We can estimate the slack time using the *workload-based slack-estimation* method. When the workload of CBS is lower than $U_s$, we can identify slack times.

Figure 7 shows the *workload-based slack-estimation* algorithm for CBS. The algorithm uses four variables, *release*, $C_{slack}$, $C_{idle}$ and $C_{active}$. The *release* is a flag variable to know whether an aperiodic task is released. The $C_{active}$ contains the number of execution cycles of the completed aperiodic tasks. When an aperiodic task is completed, $C_{idle}$, which is the number of idle cycles required to make the workload of CBS to be same to $U_s$, is calculated. During the idle period, the $C_{idle}$ is decreased. When $C_{idle}$ becomes to 0, the workload of CBS is equal to $U_s$. If the idle interval of CBS continues, the workload of CBS becomes to be smaller than $U_s$ and $C_{slack}$ is increased. The $C_{slack}$ can be used for periodic tasks to stretch the execution time.

---

Initiation:
  *release*=F; $C_{slack} = 0$; $C_{idle} = 0$; $C_{active} = 0$;

upon aperiodic_task_release:
  *release* = T;

upon aperiodic_task_completion:
  $C_{idle}$ += $C_{active} \cdot (1 - U_s)$ / $U_s$;
  *release* = F; $C_{active} = 0$;

during aperiodic_task_execution($t$):
  increase $C_{active}$ by $t$;

during CBS_idle($t$):
  **if** ( *release*==F and $C_{idle} == 0$) increase $C_{slack}$ by $t \cdot U_s$;
  **else** decrease $C_{idle}$ by $t$;

---

Fig. 7. Workload-based slack estimation in CBS.

Figure 6(b) shows the task schedule using the `DRA/CBS` algorithm which is modified from the `DRA` algorithm [2]. In Figure 6(b), the time intervals, where $C_{slack} > 0$, 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 $\tau_{2,1}$.

`DRA/CBS` generally increases the *preemption delay* of aperiodic task. However, we can

guarantee that $D(\sigma_k) \leq T_s - Q_s$ for all $\sigma_k$.

**Lemma 3:** The deadline $d'_k$ assigned by *DRA/CBS* algorithm is same to the deadline $d_k$ assigned by *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 *DRA/CBS* and *CBS*, $d_{k+1}$ and $d'_{k+1}$ are also same. Second, when the CBS server is inactive and $q_s \geq (d_k - r(\sigma_i))U_s$, a new deadline $d_{k+1} = r(\sigma_i) + T_s$ is assigned. The new deadline assignment occurs to sustain the utilization of CBS below $U_s$. Since *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 aperiodic tasks (and the remaining execution cycles) to be executed after the last replenishment time $\mathbb{R}(\sigma_k)$ are same in both *CBS* and *DRA/CBS*.

**Proof** *CBS/DRA* does not change the replenishment rule of 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 *DRA/CBS* algorithm, $D(\sigma_k) \leq T_s - Q_s$ for all $\sigma_k$.

**Proof** When we denote the completion times of $\sigma_k$ in *DRA* and *DRA/CBS* as $e(\sigma_k)$ and $e'(\sigma_k)$ respectively, we should show $e'(\sigma_k) - e(\sigma_k) \leq T_s - Q_s$. We prove the theorem for two cases.

**case 1** During the execution of $\sigma_k$, there is no replenishment of the budget of CBS.

In this case, we can know the execution cycle of $\sigma_k$, $c(\sigma_k)$, is smaller than $Q_s$. By the **Lemma 3**, $e'(\sigma_k) \leq d'(\sigma_k) = d(\sigma_k)$. From the deadline assignment rule, $d(\sigma_k) - r(\sigma_k) \leq T_s$. Therefore, we can show $e'(\sigma_k) - e(\sigma_k) \leq T_s - Q_s$ as follows:

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

**case 2** During the execution of $\sigma_k$, there is one or more replenishments of the budget of CBS. From the **Lemma 4**, we can know that the remaining execution cycles of $\sigma_k$ before the last replenishment time, $c_{rem}(\sigma_k)$, are same in both *DRA* and *DRA/CBS*. Therefore, we can treat the remaining part of $\sigma_k$ as another aperiodic task which has the execution cycles of $c_{rem}(\sigma_k)$. Then, we can prove the **case 2** with the proof for **case 1**.

## V. Experimental Results

We have evaluated the performance of our DVS algorithms for scheduling servers using simulations. 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. In the experiments, BCET is assumed to be 10% of WCET.

The interarrival times and service times of aperiodic tasks were generated from the exponential distribution using the parameters $\lambda$ and $\mu$ where $1/\lambda$ is the mean interarrival time and $1/\mu$ is the mean service time. Then, the 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 queueing model.

Varying the server utilization $U_s$ and the workload of aperiodic tasks $\rho$ under a fixed utilization $U_p$ of periodic tasks, we observed the energy consumption of the total system and the average response time of aperiodic tasks. We present only the experimental results where $U_s$ is controlled by changing the value of $T_s$ with a fixed $Q_s$ value and $\rho$ is controlled by a varying $\lambda$ with a fixed $\mu$ value.

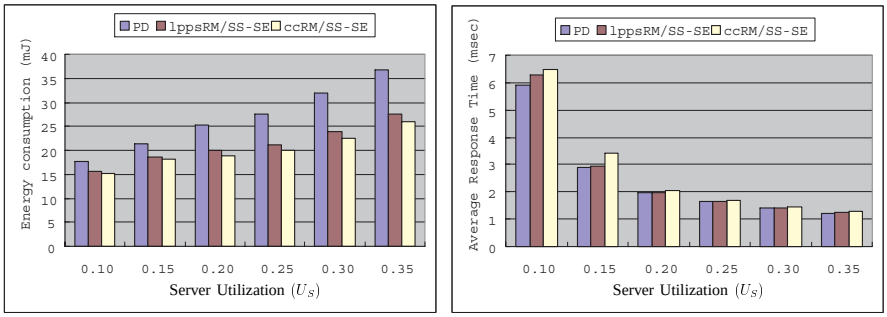
The periodic task set has three tasks with $U_p = 0.3$ and four tasks with $U_p = 0.4$ in the experiments of fixed-priority systems and dynamic-priority systems, respectively. For all experiments including the non-DVS scheme, both periodic tasks and aperiodic tasks were given an initial clock speed $s_0 = (U_p + U_s)s_m/U_m$, where $s_m$ is the maximum clock speed and $U_m$ is the upper bound of the schedulable utilization (1 in the EDF policy and $n(2^{1/n} - 1)$ for $n$ tasks in the RM policy). During run time, the speed is further reduced by on-line DVS algorithms exploiting the slack times.

First, we observed how the server utilization ($U_s$) is related with the energy consumption and the average response time. Figure 8(a) shows the energy consumptions of the power-down

method, *lppsRM/SS-SE* and *ccRM/SS-SE* when a sporadic server is used. *ccRM* [8] also use the *stretching-to-NTA* method. *ccRM/SS-SE* uses the *bandwidth-based slack-stealing* method additionally. For the workload of aperiodic tasks ($\rho$), 0.1 was used. As $U_s$ increases, the energy consumption also increases because the initial clock speed $s_0$ increases.

Figure 8(b) shows how the average response time of aperiodic tasks changes. As $U_s$ increases, the response time decreases, converging on the average response time of M/M/1 because the number of interferences by periodic tasks is reduced. *lppsRM/SS-SE* and *ccRM/SS-SE* do not significantly increase the response time. This is because the response time delays due to the DVS algorithms are smaller than $T_s - Q_s$.

From these results, we can observe that the server utilization should be carefully selected to satisfy the response time requirement. For example, assume that $\rho$ is 0.05 and SS is used for aperiodic task scheduling. If the average response time should be less than 2 msec, the server utilization 0.2 is the best choice because it minimizes the energy consumption while satisfying the response time constraint.




(a) Energy Consumption (b) Response Time

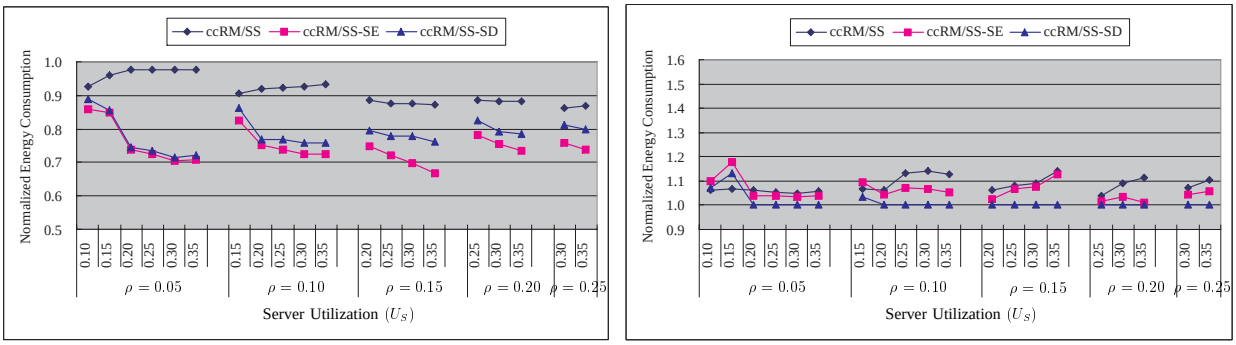
Fig. 8. Experimental results using a sporadic server ($\rho = 0.05$).

Figure 9(a) shows the energy consumptions of the *ccRM/SS* algorithm and the *ccRM/SS-SE* algorithm normalized by that of the power-down method. We also evaluated the modified version of *ccRM/SS-SE* called *ccRM/SS-SD*. The *ccRM/SS-SD* algorithm uses a different slack distribution method. When slack times are identified, *ccRM/SS-SD* gives the slack times to only periodic tasks. Therefore, aperiodic tasks are always executed at the initial clock speed $s_0$. *ccRM/SS-SD* is good for a better response time.

The difference between the energy savings of *ccRM/SS* and *ccRM/SS-SE* decreases as $\rho$

increases. This is because there are more chances for SS to have the zero budget when $\rho$ is large. As $U_s$ increases, *ccRM/SS-SE* shows a larger energy saving compared with *ccRM/SS* because *ccRM/SS-SE* performs well in the low aperiodic workload (over $U_s$). The *ccRM/SS* and *ccRM/SS-SE* reduced the energy consumption on average by 9% and 25% over the power-down method, respectively.

As shown in Figure 9(b), *ccRM/SS* and *ccRM/SS-SE* increase the response time on average by 8% and 6% over the power-down method, respectively. Due to the side effect on aperiodic tasks explained at Section IV, *ccRM/SS-SE* shows better average response times. *ccRM/SS-SD* shows almost the same response time to that of power-down method because the execution speed of aperiodic task is always $s_0$ and the *preemption delay* is not increased except the case when $T_s$ is larger than the periods of periodic tasks. However, it shows better energy performances than *ccRM/SS*.

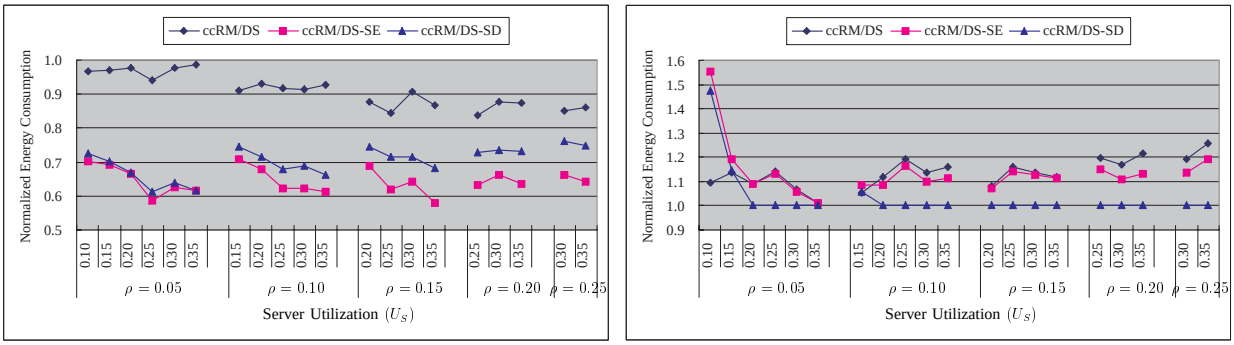



(a) Energy Consumption (b) Response Time

Fig. 9. Experimental results using a sporadic server.

Figures 10(a) and (b) are the experimental results of deferrable server. They show similar results with the results of sporadic server.

For CBS, we observed the performances of *lppsEDF/CBS*, *lppsEDF/CBS-SD*, *DRA/CBS* and *DRA/CBS-SD*. *lppsEDF/CBS-SD* and *DRA/CBS-SD* assigns all aperiodic tasks the initial clock speed $s_0$. Figure 11(a) shows the energy consumption by each algorithm normalized by that of power-down method. The energy reductions are not significantly changed as $\rho$ changes. This is because *DRA/CBS* does not utilize the zero budget of server as *ccRM/SS*. The average energy reductions by *DRA/CBS* and *DRA/CBS-SD* are 18%. Since most of slack times are

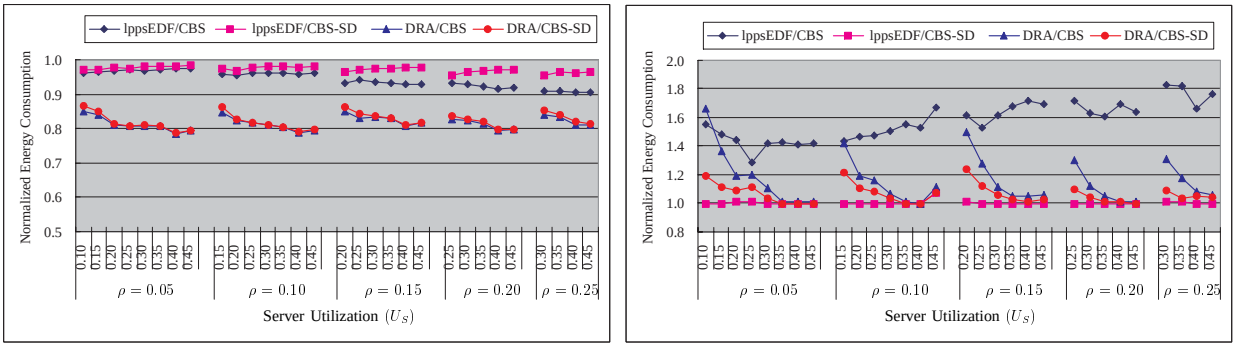



(a) Energy Consumption (b) Response Time

Fig. 10. Experimental results using a deferrable server.

generated by CBS and used by periodic tasks, *DRA/CBS* and *DRA/CBS-SD* show similar energy performances.

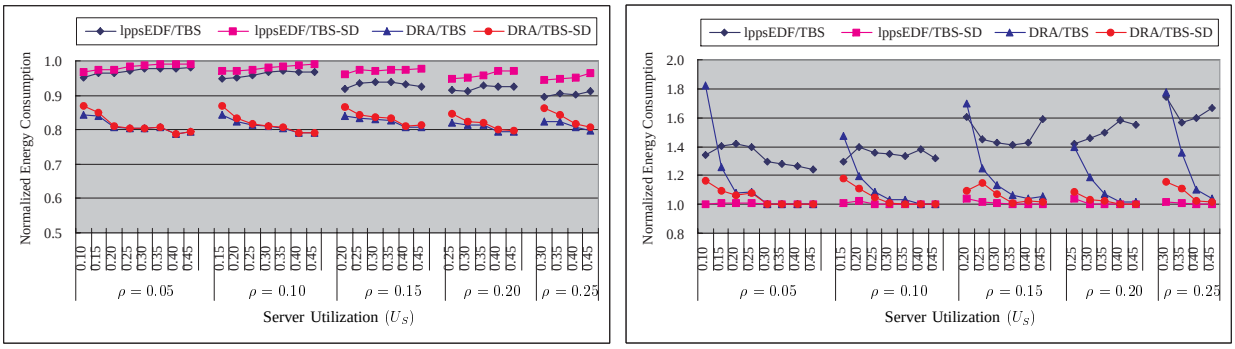
*DRA/CBS* increased the average response time on average by 16%. As $U_s$ decreases ($T_s$ increases)[3], the response time increases because the maximum response time delay is $T_s - Q_s$. However, the response time delay of aperiodic task is still smaller than $T_s - Q_s$. *lppsEDF/CBS* shows long response times because it does not bound the maximum response time delay. Since *DRA/CBS-SD* is similar to *DRA/CBS* in energy performances despite of its good response times, we can know that it is better to give slack times only to periodic tasks when the short response times are required.




(a) Energy Consumption (b) Response Time

Fig. 11. Experimental results using a constant bandwidth server.

[3]Note that we varied $T_s$ to change $U_s$.




(a) Energy Consumption

(b) Response Time

Fig. 12. Experimental results using a total bandwidth server.

Figures 12(a) and (b) are the experimental results of total bandwidth server. They show similar results with the results of constant bandwidth server.

## VI. Conclusions

We have proposed DVS algorithms for mixed task systems which have both periodic and aperiodic tasks. We presented the slack estimation methods for the scheduling servers. Existing on-line DVS algorithms, which cannot be used for mixed task systems, were modified to use the proposed slack estimation methods. The modified DVS algorithms reduced the energy consumption by 18$\sim$25% over the power-down method. We also showed the effects of the slack distribution methods on the energy and the response time.

Our work in this paper can be extended in several directions. Though the proposed algorithm only guarantees that the response time delay is smaller than $T_s - Q_s$, it will be more useful if we can control the maximum response time delay with an arbitrary $\delta$ value. Furthermore, it will be interesting to use the DVS algorithm to utilize the temporal locality of aperiodic requests. When the aperiodic requests are sparse, we could use a larger $\delta$ value for a more energy-efficient schedule.

## References

[1] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pages 4–13, 1998.

[2] H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proc. of IEEE Real-Time Systems Symp.*, pages 95–106, 2001.

[3] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A Dynamic Voltage Scaled Microprocessor System. In *Proc. of IEEE Int. Solid-State Circuits Conf.*, pages 294–295, 2000.

[4] Y. Doh, D. Kim, Y.-H. Lee, and C. M. Krishna. Constrained Energy Allocation for Mixed Hard and Soft Real-Time Tasks. In *Proc. of Int. Conf. on Real-Time and Embedded Computing Systems and Applications*, pages 533–550, 2003.

[5] 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. of Design Automation and Test in Europe*, pages 788–794, 2002.

[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. of IEEE Real-Time and Embedded Technology and Applications Symp.*, pages 219–228, 2002.

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

[8] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of ACM Symp. on Operating Systems Principles*, pages 89–102, 2001.

[9] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proc. of Design Automation Conf.*, pages 134–139, 1999.

[10] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Journal of Real-Time Systems*, 1(1):27–60, 1989.

[11] M. Spuri and G. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Journal of Real-Time Systems*, 10(2):179–210, 1996.

[12] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[13] W. Yuan and K. Nahrstedt. Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems. In *Proc. of Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 105–114, 2002.