Electronic Preprint for Journal of Information Processing Vol.23 No.5

Regular Paper

Real-Time Scheduling for Reducing Jitters of Periodic Tasks

Kiyofumi Tanaka[1,a)]

Received: November 21, 2014, Accepted: May 9, 2015

Abstract: For control applications implemented with periodic tasks, fluctuations in start or response timing cause jitters which may disturb periodicity and result in performance degradation or instability. This study proposes real-time scheduling techniques that reduce jitters and average response times of periodic tasks. In the proposed scheduling, two techniques are employed: one which adaptively extends deadlines according to varying execution times, and the other which further obtains short deadlines by virtually advancing release times. These two techniques aim at scheduling target tasks as early as possible by giving them short enough deadlines. The evaluation shows that the former shortens average response times of the target tasks by up to 20.5% and the latter mitigates jitters by up to 35.4%, compared to the existing scheduling algorithm, Total Bandwidth Server.

Keywords: Real-time scheduling, jitter, response time, periodic tasks, deadline

1. Introduction

In various computer-controlled systems, it is common for sampling/sensing and actuating to be done periodically [1]. Especially in closed-loop control systems where tasks are required to be executed at a constant interval, jitters, which is defined as the fluctuation in response times, may influence periodicity and cause performance degradation or instability [2], [3]. While small jitters can be expected in systems with a single controlled object, unpredicted and unacceptable delays or jitters can occur in multitask/mixed-criticality systems [4], [5], which results in failure in periodical control [6].

Not only jitters, but response times are important for some applications. For example, if a task that occupies many resources has long response times, conflicts would often occur for the resources, and other tasks would have to wait long for the resources to be released. To alleviate this kind of conflict, techniques that can shorten response times of particular tasks are desired.

For hard real-time systems, in addition to reducing jitters and response times, satisfying deadline requirements is most important. There are representative scheduling algorithms, Rate Monotonic (RM) and Earliest Deadline First (EDF), that can guarantee the schedulability of periodic tasks where all deadline constraints are satisfied [7]. RM is one of the fixed-priority algorithms which always gives higher priorities to tasks with shorter periods. Although it exhibits small jitters and short response times for high-priority (short-period) tasks, it cannot utilize 100% of a processing resource while keeping the schedulability[*1]. EDF is one of the dynamic-priority algorithms which gives preference to tasks with earliest absolute deadlines. While it guarantees the schedulability for the processor utilization of up to 100%, it cannot assign fixed-priorities to particular (important) tasks for which small jitters or short response times are preferable.

The scheduling method proposed in this paper aims to shorten jitters and response times of particular periodic tasks that are important for the systems, independent of their periods, while maintaining schedulability with up to 100% utilization. This method consists of two techniques to make deadlines as early as possible: applying *adaptive total bandwidth server* [9], [10] to periodic tasks, which adaptively extends deadlines according to varying execution times, and *virtual release advancing*, which obtains short deadlines by virtually assuming retroactive release times.

This paper consists of five sections. Section 2 describes related works for mitigating jitters. Then, Section 3 proposes two scheduling techniques that obtain short deadlines for the target tasks and reduce jitters for them. Evaluation of the proposed method is shown in Section 4. Finally, Section 5 concludes the paper with a summary and future directions.

2. Related Works

2.1 Mitigation of Jitters

There are various researches which aim to mitigate jitters. They are divided into three types according to the policies [11]. The first policy is to divide tasks' codes into three phases and reduce jitters by executing them as subtasks. The second is to obtain small jitters by making deadlines early and scheduling by EDF. The last one is to try to make execution times constant by prohibiting preemption.

As one belonging to the first policy, Balbastre, et al. proposed a task set model which divides a task's code into three phases: data acquisition (input), computation of the control action, and output

[1] Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1292, Japan
[a)] kiyofumi@jaist.ac.jp

[*1] The sufficient condition for the schedulability is that the processor utilization should be less than 69% [8].

of the control action [12]. Jitters can be reduced by giving short deadlines to the input and output subtasks to raise their priorities. However, this method forces application designers to divide the task codes into three types of sub codes. In addition, the schedulability analysis requires high complexity of computation based on processor demands/response times, instead of simple estimation based on processor utilization, since the deadlines of the subtasks would become earlier than the next release times [8].

As a similar technique, Buttazzo, et al. proposed to implement input and output phases as low-level handlers and execute them just at period timing, drastically reducing jitters [11]. To be more precise, the output phase of the previous job and the input phase of the current job are processed at every period timing, so that jitters become almost zero. This technique needs explicit division of task codes and implementation of input/output subtasks as OS handlers. In addition, the processing of the handlers is regarded as fixed(highest)-priority execution, which means pure EDF cannot be employed and therefore the processor utilization has to be low enough. Moreover, when different tasks have the same period timing (which is a kind of hyper-period timing), processing two or more handlers of their subtasks would cause large overheads. Because of these problems, this paper does not target this kind of static phase-division technique.

As the second policy, Baruah, et al. proposed a deadline calculation to keep the worst jitter smallest across all periodic tasks [13]. In this method, each task (τ_i) is supposed to originally have a deadline equal to its period timing. Given that C_i and T_i are the task's worst-case execution time and length of period, respectively, its processor utilization factor becomes C_i/T_i. This proposal tries to give τ_i a bandwidth, θ_i, which is larger than their processor utilization (i.e. $\theta_i > C_i/T_i$), and assign a new relative deadline, D_i, shorter than the period length (i.e. $D_i = C_i/\theta_i$). This technique can be regarded as applying Total Bandwidth Server (TBS) [14] to periodic tasks (although this fact is not explicitly described in Ref. [13]). This is effective when the total utilization, U_p, is relatively small and the residual bandwidth $(1 - U_p)$ can be distributed among tasks for large θ_i values. However, when U_p is near 100%, enough effect cannot be expected. In addition, this technique supposes each job to always spend the worst-case execution time and therefore the calculated deadlines are fixed, which might not fit the actual execution time of each job. (On the other hand, the technique proposed in Section 3.2 tries to calculate deadlines appropriate for each job the execution time of which can vary). Moreover, relative deadlines shorter than the periods' lengths makes it necessary to perform an elaborate schedulability analysis based on processor demands/response times. The aim of this technique is to shorten the worst jitter over the task set, while this paper tries to reduce jitters of particular (important) tasks. Although the aims are different, the basic policy of applying TBS is the same.

The third policy is to prohibit preemption to reduce jitters where jobs are not interrupted from the beginning to the end. The most serious problem with this policy is that it is impossible to ensure schedulability [11]. Since this paper puts importance on schedulability, this policy is left outside the scope.

This paper proposes solutions to the problems above. That is, the proposed method achieves the following while keeping small jitters and short response times: designing tasks with divided phases is not required, 100% of processor utilization is allowed, simple schedulability analysis based on processor utilization is applicable, and deadlines tailored for tasks with varying execution times are given.

2.2 Applying Total Bandwidth Server

Total Bandwidth Server (TBS) [14] is a scheduling algorithm that takes charge of aperiodic tasks' execution in a task set consisting of hard periodic tasks and soft or non real-time aperiodic tasks. TBS has characteristics of achieving low cost/overhead implementation, guaranteeing schedulability of hard tasks, and keeping reasonably short response times for aperiodic tasks.

TBS assigns a tentative absolute deadline, d_k, to the k-th aperiodic job that arrives at t = r_k as follows:

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^{WCET}}{U_s}. \quad (1)$$

Here, d_{k-1} is the absolute deadline of the previous ($k-1$-th) job, C_k^{WCET} is the worst-case execution time of the k-th job, and U_s is the server bandwidth (or processor utilization allowed for aperiodic jobs). The max term prevents bandwidths of two consecutive jobs from overlapping each other.

TBS calculates the deadlines depending on the jobs' worst-case execution times. Hence, if a job finishes earlier than its worst-case execution time, it turns out that the deadline was later than necessary. As seen in Eq. (1), this overestimated deadline influences the succeeding jobs' deadline calculations by the max term. To avoid this, Ref. [15] proposed a resource reclaiming technique.

With resource reclaiming, the k-th aperiodic job is given the following deadline, d'_k:

$$d'_k = \bar{r}_k + \frac{C_k^{WCET}}{U_s}. \quad (2)$$

In the above formula, $\bar{r}_k$ is decided as follows:

$$\bar{r}_k = \max(r_k, \bar{d}_{k-1}, f_{k-1}). \quad (3)$$

That is, the maximum value among the release time (r_k), the recalculated deadline of the previous ($k-1$-th) job ($\bar{d}_{k-1}$, described later), and the finishing time of the previous job is regarded as the representative release time. When the $k-1$-th aperiodic job finishes, its deadline is recalculated with the execution time actually spent ($\bar{C}_{k-1}$) as Eq. (4):

$$\bar{d}_{k-1} = \bar{r}_{k-1} + \frac{\bar{C}_{k-1}}{U_s}. \quad (4)$$

This $\bar{d}_{k-1}$ is reflected in Eq. (3), and then the deadline of the succeeding job is obtained by Eq. (2). This technique can take the slack time left by earlier completion of a job into account for the next job's deadline calculation. (On the other hand, the technique described in Section 3.2 predicts earlier completion of a job and utilizes the slack time for its own deadline calculation.) The technique, virtual release advancing, proposed in Section 3.3 makes use of this resource reclaiming.

Periodic tasks are a special case of aperiodic tasks. Therefore,

TBS can be applied to periodic tasks. As described in Section 2.1, Baruah, et al. adopted the use of TBS to reduce the worst jitter by giving an earlier deadline than the period timing [13].

When TBS is applied to a periodic task τ_i with a period T_i and a worst-case execution time C_i^{WCET}, the k-th job is assigned the following absolute deadline:

$$d_{i,k} = \phi_i + k \times T_i + \frac{C_i^{WCET}}{\theta_i} \qquad (k \geq 0). \tag{5}$$

Here, ϕ_i is the phase of τ_i, that is the release time of the first job of τ_i. θ_i is the bandwidth for the task. When θ_i is equal to τ_i's processor utilization factor ($U_i = C_i^{WCET}/T_i$), the deadline coincides with the period timing, that is $d_{i,k} = \phi_i + (k+1) \times T_i$. When the total processor utilization by a task set is less than 100%, the residual bandwidth can be added to θ_i, which leads to an earlier deadline according to Formula 5 since θ_i becomes larger than U_i. An earlier deadline can make EDF schedule this task more preferentially and result in shorter response times or jitters. It is noted that the necessary condition for a task set with n periodic tasks to be schedulable by this method is $\sum_{i=1}^{n} \theta_i \leq 1$. (The sufficient condition is given by schedulability analysis based on processor demands/response times [8].)

3. Proposed Techniques

3.1 Target Task Model and Jitters

A task, τ_i ($i = 1, 2, \ldots$), in a task set has its period, T_i, and worst-case execution time, C_i^{WCET}. Its relative deadline, D_i, is equal to the period, that is $D_i = T_i$. In this task model, an absolute deadline coincides with the period timing.

A relative jitter, RJ_i, and an absolute jitter, AJ_i, of a periodic task τ_i are defined as follows:

$$RJ_i = \max_k |(f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1})| \tag{6}$$

$$AJ_i = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k}). \tag{7}$$

In these definitions, $r_{i,k}$ and $f_{i,k}$ are the release time and the finishing time of τ_i's k-th job, respectively. The relative jitter is the difference between response times of two successive jobs of the same task, and the absolute jitter is the difference between the longest and the shortest response times among all jobs of the same task.

3.2 Applying Adaptive Total Bandwidth Server

It has been proposed that adaptive TBS can reduce response times of aperiodic tasks by giving deadlines to jobs in a stepwise fashion [9], [10]. This section describes how to apply adaptive TBS to periodic tasks.

Suppose τ_i is a target task for which response times and jitters should be reduced and τ_i is served by adaptive TBS. The bandwidth of the adaptive TBS is θ_i which is the same as the one in Section 2.2. For example, when there is one target task, θ_i becomes $1 - (U_p - U_i)$, where U_i is τ_i's processor utilization factor (C_i^{WCET}/T_i) and U_p is the total processor utilization by all periodic tasks. Using θ_i, the k-th job of τ_i is given the stepwise deadlines as follows:

$$d_{i,k}^0 = \phi_i + k \times T_i + \frac{C_i^0}{\theta_i} \tag{8}$$

$$d_{i,k}^j = d_{i,k}^{j-1} + \frac{C_i^j}{\theta_i} \qquad (j > 0). \tag{9}$$

In the above calculations, C_i^j is the supposed (or predicted) execution time spent by the j-th step of τ_i's k-th job. That is, this job is first predicted to finish in C_i^0. When it does not finish in C_i^0, the remaining execution is predicted to finish in C_i^1. This is repeated until the job finally finishes. This technique has the effect of obtaining early deadlines by predicting early finishing. (In the evaluation in this paper, $C_i^j = 1$ is assumed for all js.) For schedulability issue and implementation complexity of the adaptive TBS, see Ref. [10].

3.3 Virtual Release Advancing

In this section, the technique, virtual release advancing, is proposed to further bring deadlines backward, which is used together with TBS or the adaptive TBS. In TBS, an absolute deadline is calculated by basically originating in a release (or arrival/invocation) time of a job. This means that an earlier deadline would be obtained if the release time was advanced. However, since the release times of periodic tasks are fixed at their period timing, they would not be actually advanced. Instead, with virtual release advancing, an earlier release time is assumed and then the corresponding earlier deadline is calculated.

The basic policy of virtual release advancing is that it moves a release time backward without changing past schedules. That is, even if it is supposed that τ_i's k-th release time, $r_{i,k}$, is virtually moved to a past time, $vr_{i,k}$, and the corresponding deadline is advanced, other jobs with even earlier deadlines have to spend the interval between $vr_{i,k}$ and $r_{i,k}$. Otherwise, τ_i must have been executed in the interval, which leads to changing the past schedule. The point is that a release time can be advanced as long as it does not influence past schedules.

This technique supposes to be used with TBS (or the adaptive TBS). The max term in Eq. (1) indicates that it is useless to advance a release time over the previous deadline, $d_{i,k-1}$, which leads to a fact that this technique cannot be applied to a periodic task since its release time is the same timing as the previous deadline due to $D_i = T_i$. However, the resource reclaiming in Section 2.2 or the adaptive TBS in Section 3.2 can provide an earlier deadline than the next release time. That is, $d_{i,k-1} < r_{i,k}$ ($\overline{d}_{i,k-1} < r_{i,k}$ in the case of the resource reclaiming, or $d_{i,k-1}^j < r_{i,k}$ in the case of the adaptive TBS), which enables the virtual release advancing to utilize the room of $r_{i,k} - d_{i,k-1}$.

3.3.1 Example of Virtual Release Advancing

Figure 1 shows an example of the virtual release advancing. In the figure, three periodic tasks, τ_1, τ_2, and τ_3, have periods, $T_1 = 10$, $T_2 = 9$, and $T_3 = 6$, worst-case execution times, $C_1 = 2$, $C_2 = 2$, and $C_3 = 3$, and phases (or a release time of the first job), $\phi_1 = 0$, $\phi_2 = 1$, and $\phi_3 = 1$. In this example, τ_1 is regarded as a target task (with the highest importance) and its jitter is focused on. The upper figure is the schedule result by the original EDF and the lower one is by TBS plus the virtual release advancing for τ_1. Note that, although the worst-case execution time of τ_1 is two units of time, the actual execution time of its first job is one unit of time and that for the second is two. By the original EDF,

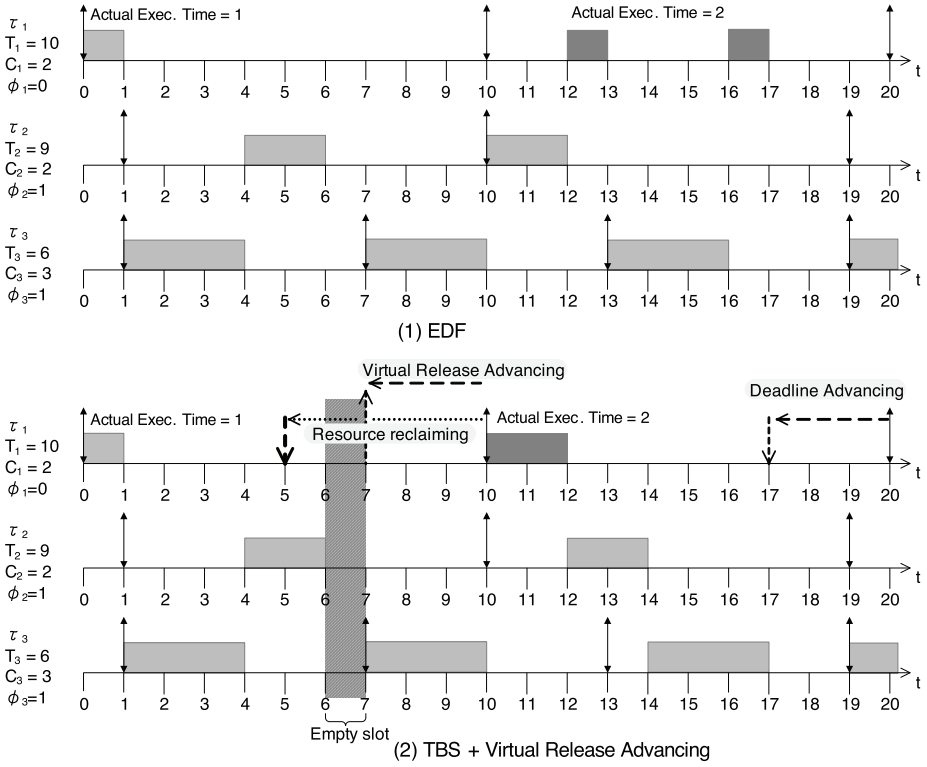

Fig. 1 Example of virtual release advancing (Limit by empty slot).

the response times of the first and the second jobs become one and seven, respectively. Hence it follows that the relative jitter between the two jobs becomes $7 - 1 = 6$.

On the other hand, in the TBS with the virtual release advancing, when the first job of τ_1 finishes, the resource reclaiming mechanism resets the corresponding deadline to $t = 5$ as designated by the downward broken arrow, which enables the release time of the next job to be advanced. In the figure, the virtual release time is set to $t = 7$ as designated by the upward broken arrow. Correspondingly, the deadline becomes $t = 17$. In the time period from $t = 7$ to $t = 10$, τ_3's job was scheduled. The deadline of this job was $t = 13$, and therefore this virtual release advancing never changes the past schedule (since $13 < 17$). As a result, the second job of τ_1 is scheduled prior to the second job of τ_2 with the deadline of $t = 19$, the response time is two, and then the relative jitter becomes $2 - 1 = 1$.

In this example, it is impossible to further advance the release time. If the release time were reset to $t = 6$, the time slot between $t = 6$ and $t = 7$ must have been occupied by this job's execution since the slot was actually empty. This leads to changing the past schedule.

3.3.2 Definition of Virtual Release Advancing

In this section, how long the release time can be advanced is discussed and defined. There are three factors that become limits of advancing: deadline of the previous job, empty slot, and maximum used deadline.

(1) *Deadline of the previous job:* Moving the release time to the past over the deadline of the previous job is ineffective since, as confirmed in Eq. (1), TBS chooses a larger value between the release time and the previous deadline as an originating point in order to maintain the schedulability. This becomes one of the limiting factors. To implement this, recording the deadline of the previous job is needed, which is already achieved in TBS.

(2) *Empty slot:* An empty slot is a time slot during which no task is executed and becomes the second limit of advancing. As shown in Section 3.3.1 and Fig. 1, the empty slot must not be overtaken by the virtual release time since it would change the past schedule. To implement this limit, the last empty slot needs to be recorded.

(3) *Maxmum used deadline:* Slots that were not empty must have been spent by the tasks' execution. Each slot can be associated with the deadline of a job executed in the slot. The associated deadline is called a "used deadline" for the slot in this paper. Let's consider whether a release time of a job can be advanced over some past slot. If a deadline calculated according to the advanced release time gets earlier than the used deadline of the slot, the slot must have been spent by this job's execution. This means that the past situation would change. Therefore, this condition becomes a limit. To be more exact, a calculated (advanced) deadline must be later than all the used deadlines of slots that have been overtaken. To achieve this limit, it is necessary to record used deadlines for the past slots and then manage a maximum value among them in the process of advancing.

Figure 2 shows an example of a limit by a maximum used deadline. This example assumes the target task for jitter reduction is τ_1. When τ_1's job is released at $t = 5$, a scheduler starts processing the virtual release advancing. Slot 4 (between $t = 4$ and $t = 5$) was already spent by τ_3's second job, and the used deadline for this slot is $t = 8$. (The array dl[] records the used deadlines for past slots.) Assuming that the release time were ad-

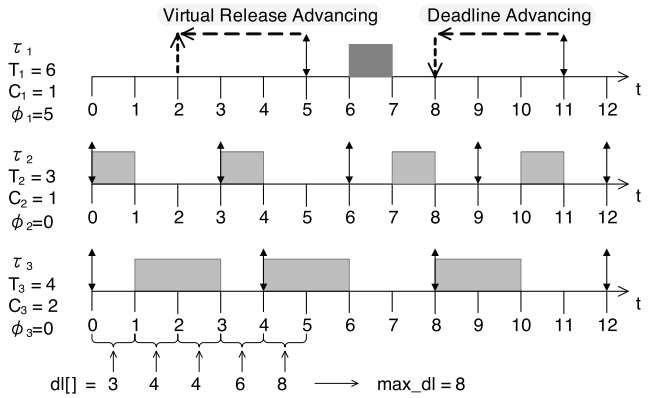

Fig. 2 Example of virtual release advancing (Limit by maxmum used deadline).

vanced by one slot, the corresponding deadline would be $t = 10$. This deadline is no earlier than the used deadline of $t = 8$. Hence, this advancing is allowed and performed. At the same time, the value of 8 is stored in the variable for the maximum used deadline (*max_dl*).

Then, the virtual release time is supposed to be advanced by one slot again and the corresponding deadline would be $t = 9$. The overtaken slot 3 was used by τ_2's job and the used deadline was $t = 6$. This used deadline is smaller than *max_dl* and therefore *max_dl* is not updated and remains 8. This advancement is allowed since the advanced deadline is still later than *max_dl*. Similarly, the next advancement for slot 2 can be achieved and then the virtual release time and the deadline get $t = 2$ and $t = 8$, respectively. If further advancement by one slot was attempted, the resulting deadline would be $t = 7$ and become earlier than *max_dl*, which means this advancement cannot be allowed. Therefore, the net virtual release time is $t = 2$, the deadline is $t = 8$, and the advancing process finishes. In this example, the third job of τ_2 is overtaken, so that the response time of the target task is shortened by one unit of time.

3.3.3 Algorithm of Virtual Release Advancing

Algorithm 1 shows the procedure of the virtual release advancing. This algorithm is for one target periodic task and is applied when the task is released. In the algorithm, a task ID is omitted for brevity. r_k is the actual release time of the k-th job of the target task, and vr_k is the virtual release time of the job. d_{k-1} and d_k are absolute deadlines of the previous and current jobs, respectively, of the same task. C is the worst-case execution time of the task. U_s is the server bandwidth for TBS that takes charge of the target task. If this algorithm is combined with the adaptive TBS, C is replaced by C_i^0, as described in Section 3.2. As for the other variables, *last_empty* records the last empty slot number and *dl* is an array such that each element corresponds to a time slot and holds the used deadline for the slot. In addition, *max_dl* is a variable for keeping the maximum used deadline.

In lines 1 and 2 of the algorithm, *max_dl* and vr_k are initialized with zero and r_k, respectively. Then, the loop execution for the advancing process starts at line 3. In the loop, the deadline calculation for TBS is performed originating in the current virtual release time at line 4. In lines 6 to 8, the limit of the previous job's deadline is checked; if the virtual release time, vr_k, coincides

Algorithm 1 Virtual Release Advancing

```
1:  max_dl ⇐ 0 /* Initializing the maximum used deadline */
2:  vr_k ⇐ r_k   /* Initializing the virtual release time */
3:  while TRUE do
4:      d_k ⇐ vr_k + C/U_s
5:      /* Limit factor by the previous job's deadline (d_{k-1}) */
6:      if vr_k = d_{k-1} then
7:          break
8:      end if
9:      /* Limit factor by empty slot */
10:     if vr_k = last_empty + 1 then
11:         break
12:     end if
13:     if max_dl < dl[vr_k − 1] then
14:         max_dl ⇐ dl[vr_k − 1]
15:     end if
16:     /* Limit factor by the maximum used deadline */
17:     if d_k ≤ max_dl then
18:         break
19:     else
20:         vr_k ⇐ vr_k − 1   /* Advancing by one slot */
21:     end if
22: end while
```

with the previous job's deadline, d_{k-1}, the algorithm finishes.

If the check is passed, the second limit factor (by empty slot) is considered in lines 10 to 12; if the virtual release time is next to an empty slot, further advancing cannot be done. If so, the algorithm finishes.

Then, the limit by the maximum used deadline is checked. As preparation, in lines 13 to 15, *max_dl* is compared with the deadline of the previous slot, and if *max_dl* is smaller, it is updated to maintain the maximum value. In lines 17 to 18, the current calculated deadline is compared with the maximum used deadline. If the deadline is equal to or earlier than *max_dl*, the advancing of the deadline cannot be done and the algorithm finishes.

When all the above limit conditions are passed, the virtual release time is advanced by one slot at line 20, and then the loop execution continues.

3.3.4 Schedulability of Virtual Release Advancing

With the virtual release advancing technique, although the target job was supposed to be released at the virtual release time, the virtually advanced release time never influences past schedules. In other words, even if the job had been actually released at the virtual release time, TBS would have resulted in the same schedule before the actual release time. Especially with the limit checking for the previous job's deadline, this technique completely follows the deadline calculation rule in TBS. Therefore, the schedulability feature of TBS is maintained; if and only if the total processor utilization is equal to or less than 100%, the task set is schedulable.

3.3.5 Implementation Complexity and Runtime Overhead of Virtual Release Advancing

In Algorithm 1, division (C/U_s) is performed at line 4 of every loop iteration. To mitigate execution overheads of the algorithm, this division should be done beforehand (statically) and the static result can be used at runtime.

In terms of storage overheads, *last_empty* and *max_dl* are sin-

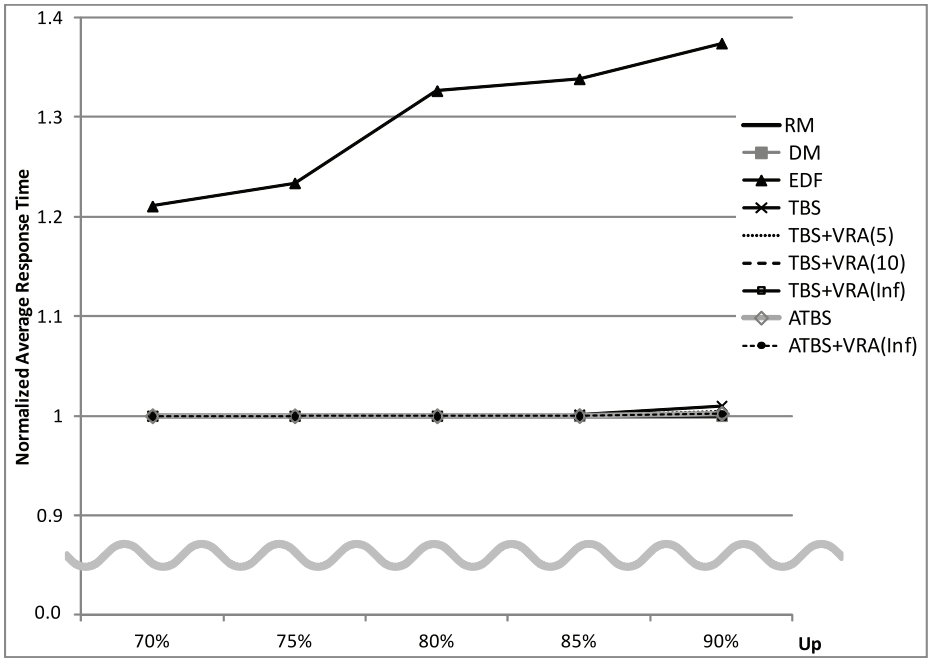

Fig. 3 Average response times (Shortest-period task).

gle data and do not become a problem. On the other hand, the size of the *dl* array must be considered. It is not realistic to provide an array that covers all time slots for the operating time. Therefore, the size should be reduced by limiting the number of slots to be advanced; that is, the number of loop iterations in the algorithm. The effects from the limitation on response times/jitters are evaluated in Section 4.

4. Evaluation

4.1 Evaluation Methodology

Effects on response times and jitters by the proposed techniques are evaluated by simulations with modeled task sets. In the evaluation, the proposed techniques are compared with RM, EDF, Deadline Monotonic (DM) [16], and the application of TBS [13] as described in Section 2. Although the aim of the application of TBS in Ref. [13] is to minimize jitters over a task set by distributing residual bandwidth to all tasks, the residual bandwidth is given only to the target (important) tasks in this evaluation.

DM [16] is one of the fixed-priority algorithms that weakens the condition in RM, "deadline is equal to its period." The scheduling rule is that shorter relative deadlines have higher priorities. By exploiting this relaxed condition, the same idea as Eq. (5) can be applied to DM; a relative deadline shorter than its period can be assigned to an important task by utilizing extra bandwidth. However, θ_i (in Eq. (5)) is derived based on the upper bound of processor utilization for RM and DM, $U_{lub} = n(2^{1/n} - 1)$ (n is the number of tasks), while that for EDF and TBS is based on 100% [8].

Constant Bandwidth Server (CBS) [17] is one of the dynamic priority servers for aperiodic tasks and can cope with tasks which take varying execution times. When an aperiodic job finishes early, CBS reduces the response time of the next job by utilizing the budget (slack) left by the former job and its (early) deadline. To benefit from this mechanism, the next job has to be released before the deadline of the former (or the server's period timing). This means that CBS has no direct effect on periodic tasks the jobs of which are periodically released. Therefore, this paper excludes CBS from the methods to be compared.

For each total processor utilization (U_p) from 70% to 90% at intervals of 5%, thirty task sets consisting of periodic tasks are prepared. The utilization is based on the tasks' worst-case execution times. In this section, the average values of the results of the thirty simulations are shown. For each task, its period is decided by uniform distribution between 1 and 100 ticks. Its worst-case execution time is decided by uniform distribution between 1/10 and 1/3 of the period. Actual execution times of jobs of the target task is decided by uniform distribution between 1/3 and 1/1 of the worst-case execution time. (Jobs from the target task have different execution times.) As a result of using the above conditions, most task sets include three to five tasks. (Only a few sets included more tasks.) The observation period is 100,000 ticks. When $U_p = 85\%$ and $U_p = 90\%$, a few deadline misses were observed for RM and DM. This fact is not considered in particular in this paper.

4.2 Response Times of a Target Task

First, a task with the shortest period is regarded as the target task. Average response times of the target task are shown in **Fig. 3**. In the figure, the horizontal axis indicates U_p, and the vertical axis indicates average response times normalized to the results of RM. In the legend, "+VRA" means that the virtual release advancing is used together. The number in parentheses is the limit on the number of times the loop iteration is executed in the algorithm. "Inf" means the limit is infinity. "ATBS" is the adaptive TBS.

In the figure, all techniques except EDF overlap with each other. RM and DM have good reason to achieve the best results since they always give the highest priority to the target task

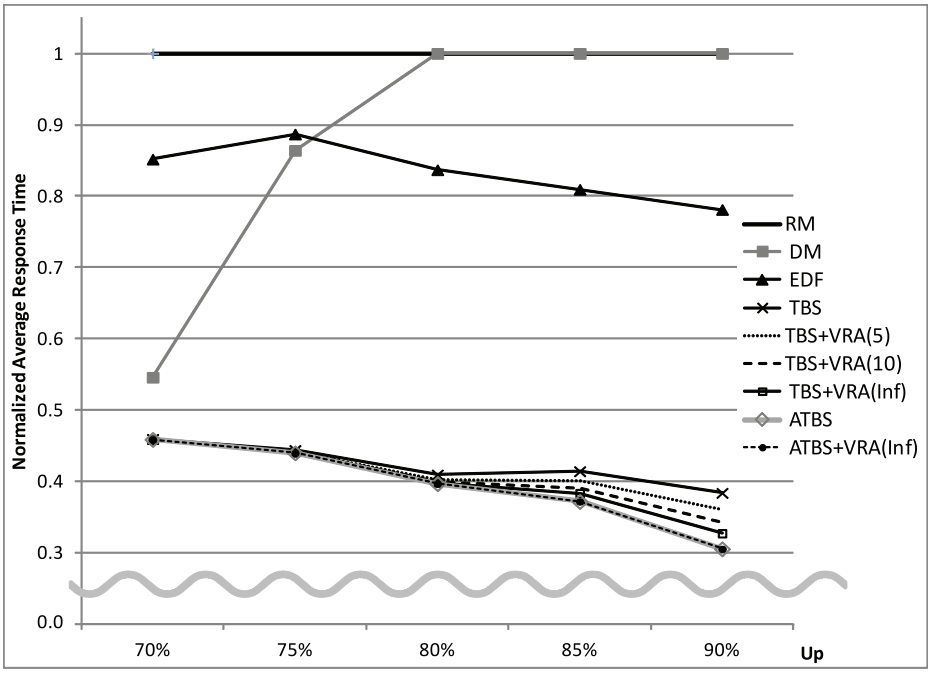

Fig. 4 Average response times (Longest-period task).

that has the shortest period. However, TBS and ATBS (with and without the virtual release advancing) can achieve the same performance although they are based on dynamic priorities. This is mainly because the residual bandwidth is well utilized.

Figure 4 shows average response times when a task with the longest period is regarded as the target. The results of RM are worst since it gives the lowest priority to the target with the longest period. DM can exhibit shorter response times by utilizing the extra bandwidth when U_p is low, while it is the same as RM when U_p is 80% or higher since high U_p implies no residual bandwidth exploited. EDF is better than RM in this case where the priority of the target task is not fixed at the lowest priority or it dynamically changes.

In the figure, the application of TBS shows large improvement in average response times. Adding the virtual release advancing, the response times are further improved depending on the limit of loop iterations. Among all techniques, adaptive TBS exhibits the best results. When U_p is 90%, the adaptive TBS improves the average response times by 20.5% compared to TBS. Consequently, it can be said that stepwise deadline updating for a target task is effective in reducing response times.

In addition, adaptive TBS is not required to come with the virtual release advancing. (The results of ATBS and ATBS+VRA completely overlap with each other in the figure.) This is because the amount of advancing tends to zero in most cases since the initial deadline based on C_i^0 is so early that it is rare to pass the condition for the maximum used deadline. (The only chance to apply the virtual release advancing is when the target job is released, since the later deadline updates cannot pass the first condition of deadline of the (its own) previous job.)

4.3 Jitters of a Target Task

Figures 5 and **6** show relative jitters and absolute jitters, respectively, normalized to the results of RM. The target task is one with the longest period. (The case of the shortest period is omitted since the differences between methods were not confirmed just as the case for response times in Section 4.2.) From these figures, relative jitters and absolute jitters have a similar trend.

The relationship among RM, DM, and EDF is similar to the results of response times. In addition, TBS and the virtual release advancing significantly improve jitters similarly to Fig. 4. However, there are two differences compared to the case of average response times. The first difference is that the amount of improvement by the virtual release advancing for TBS is larger in jitters than in response times. This is discussed as follows. Jitters are directly influenced by reduction in the longest response time as shown in Eqs. (6) and (7). The task's execution tends to exhibit the longest response time when it's execution time is close to its WCET. When the execution time is long, it has a high possibility of being preempted by later-requested jobs. Advancing deadlines by the virtual release advancing has the effect of not only overtaking previously-requested jobs but reducing the possibility of preemption by later-requested ones. Therefore, long response times can be expected to be shortened and jitters are mitigated. On the ohter hand, when the execution time is short, it experiences less preemption even in the case without the virtual release advancing. Therefore, the effects of the advancing technique are limited and the response time is not much shortened. Since the average response time is influenced by all the jobs' response times, not only by the longest response time, its effects seem lower then that of jitters.

The second difference with the case of response times is that TBS plus the virtual release advancing is more effective than the adaptive TBS (with or without the virtual release advancing). This is discussed as follows.

Average response times can be reduced by shortening the response time of each job. Adaptive TBS achieves this by giving each job an appropriate deadline according to its actual execu-

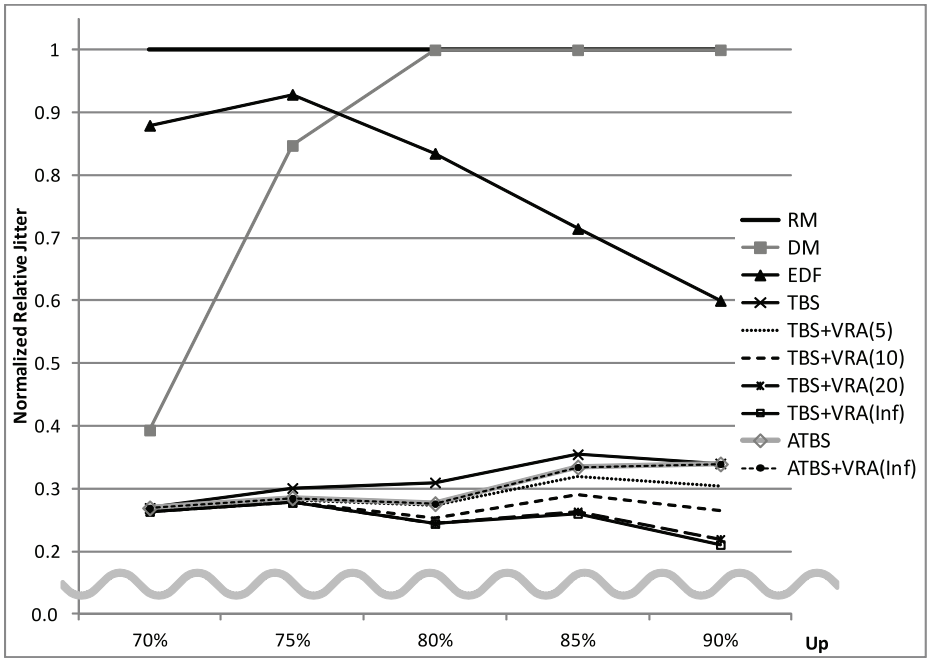

Fig. 5 Relative jitters (Longest-period task).

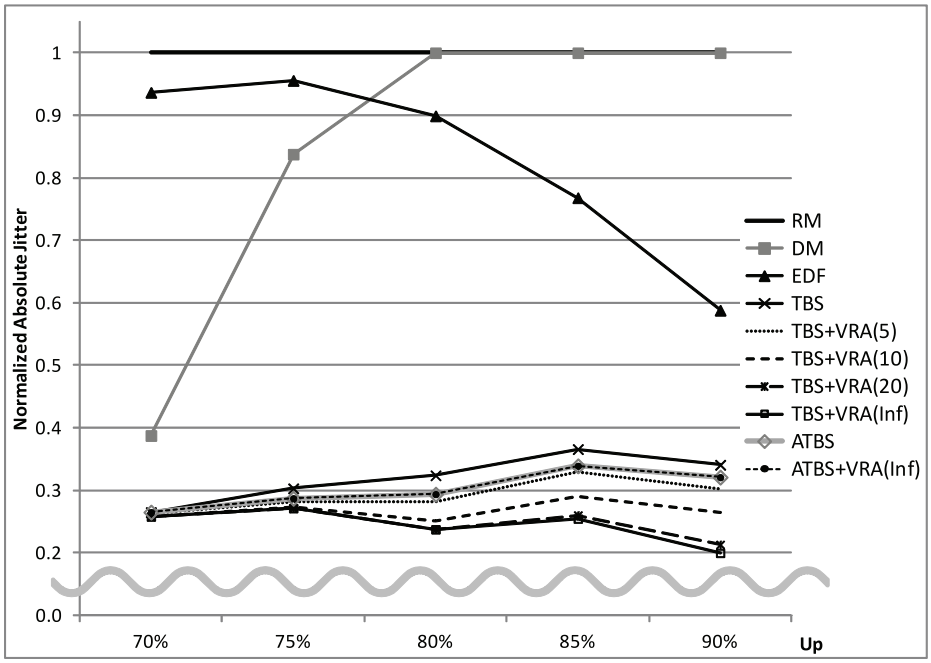

Fig. 6 Absolute jitters (Longest-period task).

tion time. On the other hand, to improve jitters, it is important to shorten the longest response time of the target task as described above. If a job spent its worst-case execution time, the adaptive TBS, having experienced stepwise deadline calculations repeatedly, would finally give the same deadline as that by TBS. Eventually, the adaptive TBS does not have superiority over TBS when a job's execution time is as long as the worst-case execution time. However, a job spending its worst-case execution time does not always exhibit the longest response time. Thus, adaptive TBS, which can reduce response times on average, shows slightly better jitters than TBS.

Combined with TBS, the virtual release advancing reduces jitters. This is because it possesses certain properties of shortening response times by advancing release times/deadlines even when it is applied to a job spending its worst-case execution time. It is confirmed that the effects depend on the number of iterations executed in the algorithm and that execution of twenty iterations (TBS+VRA(20)) approaches the results of that without the limitation on the iteration count (TBS+VRA(Inf)). Therefore, the effects can be obtained with a practical number of *dl* elements in Section 3.3.3. From the results, TBS combined with the virtual release advancing with the loop count of 20 reduced relative jitters by 35.4% for U_p = 90% compared to TBS without the virtual release advancing.

Similar to the cases of average response times, virtual release advancing could not yield gains when used in the adaptive TBS.

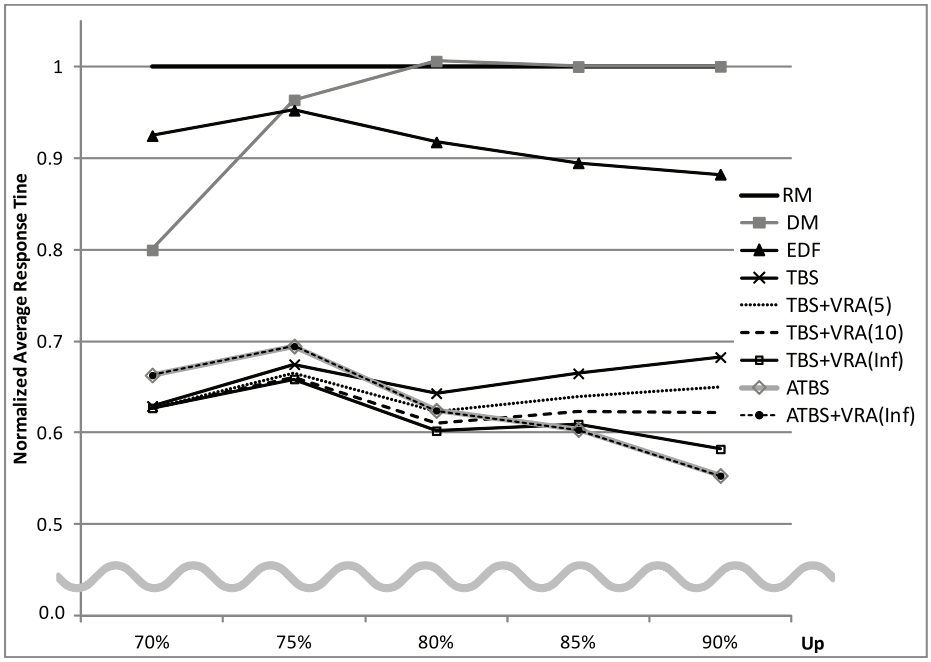

Fig. 7 Average response times (Two longest-period tasks).

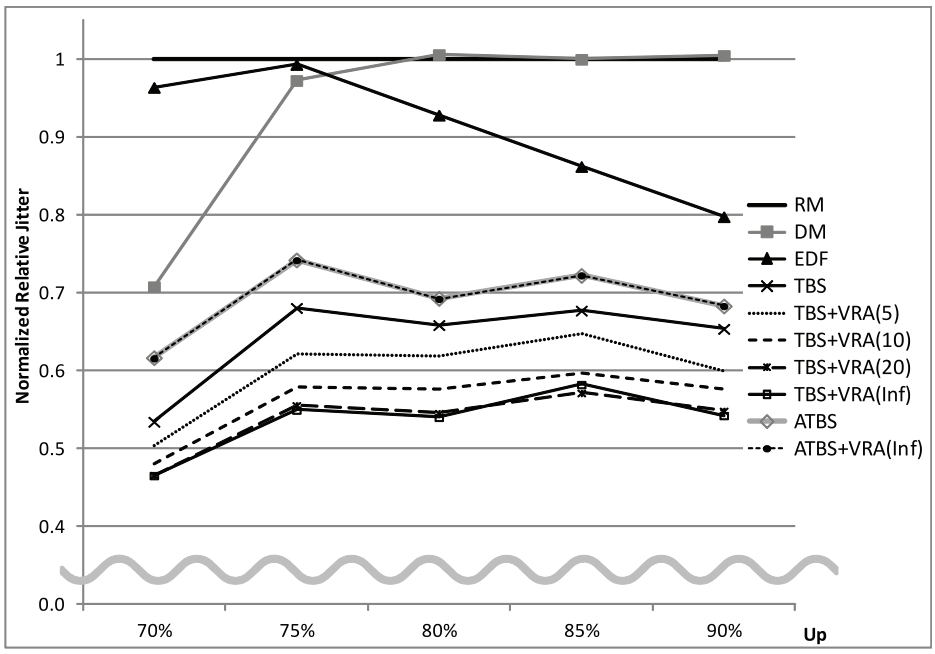

Fig. 8 Relative jitters (Two longest-period tasks).

The reason is the same as in Section 4.2.

4.4 Response Times and Jitters of Two Target Tasks

In Sections 4.2 and 4.3, a single task in the task sets was the target of reducing response times/jitters. In this section, two tasks that have the longest and second longest periods are the targets. Since most task sets consist of three to five tasks, two target tasks turn out to occupy 40% to 67% of all tasks. The residual bandwidth is split in half and then distributed to the two tasks, except in RM and EDF which do not utilize the residual bandwidth. The average values of the results for the two tasks are shown in **Figs. 7** and **8**, which correspond to average response times and relative jitters, respectively. (The results of absolute jitters are omitted due to limitations of space.)

As for average response times, although the trends similar to Fig. 4 can be seen, the improvement gets lower since the residual bandwidth utilized by each target task is halved. In addition, adaptive TBS does not achieve better results than TBS when U_p is not high. This is because two target jobs preempt each other repeatedly through multiple times of deadline recalculations and therefore jobs, especially with long execution times, tend to experience long response times. On the other hand, in TBS, a deadline for each job is calculated only once and the priority order between the two jobs does not change, which does not bring about the above preemption.

Similarly, in Fig. 8, the improvement for relative jitters is lower

than in Fig. 5. Moreover, jitters in the adaptive TBS are always worse than in TBS since the preemption mentioned above prolongs the response time of a job spending a long execution time, which has a higher impact on jitters than on "average" response times. On the other hand, it can be confirmed that TBS with the virtual release advancing can substantially alleviate jitters even when the number of target tasks is increased.

4.5 Additional Overheads

The maximum number of iterations executed in Algorithm 1 was obtained from the simulation corresponding to Sections 4.2 and 4.3. The maximum number throughout the simulations was 39. In Algorithm 1, each iteration consists of five addition operations, four comparisons, three assignments, and two references to the array element. Therefore, the estimated number of steps (or cycles) of the additional operations is $39 \times (5 + 4 + 3 + 2) = 546$.

Suppose the processor's clock frequency is 100 MHz and the tick length is 1 millisecond. Then, the maximum additional overhead from 546 cycles per tick leads to $546/100{,}000 = 0.55\%$. Considering the maximum number of iterations executed (39), this amount can be expected to be further reduced to a half by limiting the iteration count to 20 without large degradation in jitters, as seen in Figs. 3 and 4.

4.6 Discussion

From the results of Section 4.2 to Section 4.4, the following points can be deduced in terms of the number of target tasks, the purposes of scheduling (reducing response times or jitters), processor utilization, and the scheduling algorithms.

- When the purpose is to shorten response times and a single task is the target, adaptive TBS is the best choice, where it does not need to be accompanied by the virtual release advancing.
- When the purpose is to shorten response times and two or more tasks (to around half the total number of tasks) are the targets, the choice of an appropriate scheduling scheme depends on the processor utilization. TBS with virtual release advancing is a potential candidate if the utilization is low, while adaptive TBS (without the virtual release advancing) would be promising during high utilization.
- To mitigate jitters, TBS with virtual release advancing is a convincing technique regardless of the number of the target tasks or processor utilization.

In addition, it turns out that virtual release advancing needs a practical iteration count to be as effective as the case with infinite iterations, for example, twenty times as shown in the evaluation.

The evaluation in this section targets tasks with longest (or shortest) periods. Even when tasks with medium periods were targets, similar trends could be confirmed except that the improvements are slightly lowered compared to the cases for longest periods. The results are left out due to limitations of space.

5. Concluding Remarks

In this paper, application of adaptive TBS and virtual release advancing are proposed to shorten response times and mitigate jitters of particular, important periodic tasks. Adaptive TBS is a technique that gives deadlines in a stepwise/adaptive fashion according to the actual execution times, and virtual release advancing is a technique to virtually move release times to the past without influencing past schedules, leading to earlier deadlines. In the evaluation, compared to the existing technique of applying TBS to periodic tasks, the application of adaptive TBS shortened average response times of important tasks by up to 20.5% and virtual release advancing reduced relative jitters by up to 35.4%.

With the growing diversity and mixed-criticality, various kinds of tasks would coexist in a real-time embedded system. From the results obtained in this paper, it can be said that it is effective to select appropriate scheduling strategies according to each task's performance to be improved.

The evaluation in this paper used task sets with randomly-generated parameter values. To reflect actual behavior of applications and systems, evaluation with real program codes and scheduling overheads should be performed. In addition, actual systems can experience transition in processor utilization as a result of creation and deletion of tasks. Therefore, how to dynamically change the scheduling mechanisms according to the utilization should be investigated.

References

[1] Åström, K.J. and Wittenmark, B.: *Computer-Controlled Systems: Theory and Design*, 3rd ed., Prentice Hall (1997).

[2] Martí, P., Fuertes, J.M., Fohler, G. and Ramamritham, K.: Jitter Compensation for Real-Tiime Control Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.39–48 (2001).

[3] Buttazzo, G.C.: Rate Monotonic vs. EDF: Judgment Day, *Journal of Real-Time Systems*, Vol.29, No.1, pp.5–26 (2005).

[4] de Niz, D. Lakshmanan, K. and Rajkumar, R.: *On the Scheduling of Mixed-Criticality Real-Time Task Sets*, *Proc. IEEE Real-Time Systems Symposium*, pp.291–300 (2009).

[5] Baruah, S., Li, H. and Stougie, L.: Towards the Design of Certifiable Mixed-Criticality Systems, *Proc. IEEE Real-Time and Embedded Technology and Application Symposium*, pp.13–22 (2010).

[6] Lluesma, M., Cervin, A., Balbastre, P., Ripoll, I. and Crespo, A.: Jitter Evaluation of Real-Time Control Systems, *Proc. IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications*, pp.257–260 (2006).

[7] Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the Association for Computing Machinery*, Vol.20, No.1, pp.46–61 (1973).

[8] Buttazzo, G.C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd Edition, Springer (2011).

[9] Tanaka, K.: Adaptive Total Bandwidth Server: Using Predictive Execution Time, *Proc. Intl. Embedded Systems Symposium*, pp.250–261 (2013).

[10] Tanaka, K.: A Method of Shortening Average Response Times by Adaptive Scheduling–Effects of Estimating Execution Times, *IPSJ Journal*, Vol.55, No.8, pp.1856–1865 (2014). (in Japanese).

[11] Buttazzo, G. and Cervin, A.: Comparative Assessment and Evaluation of Jitter Control Methods, *Proc. Intl. Conf. on Real-Time and Network Systems*, pp.137–144 (2007).

[12] Balbastre, P., Ripoll, I., Vidal, J. and Crespo, A.: A Task Model to Reduce Control Delays, *Journal of Real-Time Systems*, Vol.27, No.3, pp.215–236 (2004).

[13] Baruah, S., Buttazzo, G., Gorinsky, S. and Lipari, G.: Scheduling Periodic Task Systems to Minimize Output Jitter, *Proc. IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications*, pp.62–69 (1999).

[14] Spuri, M. and Buttazzo, G.C.: Efficient Aperiodic Service under Earliest Deadline First Scheduling, *Proc. IEEE Real-Time Systems Symposium*, pp.2–11 (1994).

[15] Spuri, M., Buttazzo, G. and Sensini, F.: Robust Aperiodic Scheduling under Dynamic Priority Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.210–219 (1995).

[16] Leung, J. and Whitehead, J.: On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks, *Performance Evaluation*,

Vol.2, No.4, pp.237–250 (1982).
[17] Abeni, L. and Buttazzo, G.: Integrating Multimedia Applications in Hard Real-Time Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.4–13 (1998).

Kiyofumi Tanaka received his B.S., M.S., and Ph.D. degrees from the University of Tokyo in 1995, 1997, and 2000, respectively. His research interests are computer architecture, operating systems, and real-time embedded systems. He is a member of IEEE, ACM, and IEICE.