Journal of Information Processing Vol.22 No.2 152–159 (Apr. 2014)

[DOI: 10.2197/ipsjjip.22.152]

Regular Paper

Adaptive Real-Time Scheduling for Soft Tasks with Varying Execution Times

Kiyofumi Tanaka[1,a)]

Received: May 2, 2013, Accepted: November 1, 2013

Abstract: As real-time embedded systems get more diverse and more complicated, systems with different types of tasks (e.g., periodic tasks and aperiodic tasks) are prevailing. In such a system, it is important that schedulability of periodic tasks is guaranteed and at the same time response times to aperiodic requests are short enough. Total Bandwidth Server is one of convincing task scheduling algorithms for mixed task sets of periodic and aperiodic tasks. Considering a fact that in most cases tasks' execution times are much shorter than their worst-case execution times, this paper proposes a method of reducing response times of aperiodic execution by using predictive execution times instead of worst-case execution times for deadline calculations in the total bandwidth server to obtain shorter deadlines, while ensuring the integrity of periodic tasks. In the evaluation by simulation, the proposed method combined with a resource reclaiming technique improved average response times for aperiodic tasks, by up to 22% compared with the original total bandwidth server technique, and by up to 48% compared with Constant Bandwidth Server, which is another algorithm appropriate for tasks with varying execution times.

Keywords: Real-time scheduling, total bandwidth server (TBS), worst-case execution time (WCET), predictive execution time (PET)

1. Introduction

Along with the growing diversity and complexity of real-time embedded systems, it is becoming common that different types or criticalities of tasks compose a system and therefore importance of real-time scheduling is increasing [1], [2]. For example, a control task and a user interface task would be mixed in a system. The former is called a hard task and must completely satisfy real-time requirements. The latter is a soft (or non real-time) task, and should finish in a certain level of response times but is not expected to fully behave in real-time [3]. To achieve adequate real-time processing required in such a system, a sophisticated real-time scheduling algorithm that targets both hard and soft (or non real-time) tasks, guarantees the schedulability of hard tasks, and at the same time exhibits short response times for soft (or non real-time) tasks must be used.

The schedulability can be satisfied when all hard tasks' executions meet their deadline requirements. Therefore, hard tasks should be periodically invoked and executed, and be assumed to spend their worst-case execution times (WCETs), since the schedulability must be confirmed before the system starts to operate. On the other hand, soft (or non real-time) tasks can run on aperiodic invocations as long as it does not influence the schedulability of hard tasks, because of inexact (or non) real-time requirements. Total Bandwidth Server (TBS) [4] is a scheduling algorithm for mixed task sets with such hard and soft (or non real-time) tasks. TBS is based on the earliest deadline first (EDF) algorithm [5] and therefore has a characteristic that a processor can be utilized by up to 100% while maintaining the schedulability. This paper explores algorithms based on TBS to further improve the responsiveness of aperiodic tasks.

Due to the large scale and complexity of current processors and application programs, WCETs tend to be difficult to estimate. For example, deep pipelining execution of machine instructions makes precise estimation of the number of their execution cycles hard. In addition, in a system with many tasks, whether each memory reference would hit in the cache memory or not is much difficult to decide or predict [6]. Moreover, the worst-case execution path in a program is almost impossible to find and trace since it includes many branches and loop structures and all input patterns bring a vast search range [7]. Consequently, WCETs are obliged to be pessimistically estimated and lead to having a large gap with actual execution times. This gap becomes an obstacle to obtaining the best schedules under scheduling algorithms that make a decision based on tasks' execution times, for example, Shortest Job First (SJF) and Shortest Remaining Time First (SRTF) [8].

The contribution of this work is an improvement over TBS, where the average response times of aperiodic tasks can be shorter than those in TBS. In this paper, two facts are taken into consideration; one is that soft or non real-time tasks are not required to strictly meet deadline constraints, and the other is that in most cases tasks' execution times are much shorter than their WCETs. From the consideration, a method of shortening response times of soft or non real-time tasks by introducing predictive execution times instead of WCETs into the TBS-based

[1] Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1292, Japan

[a)] kiyofumi@jaist.ac.jp

scheduling algorithm is proposed. In the evaluation with simulation for tasks with varying execution times, the effectiveness of the proposed method is shown by comparing it with the original TBS and Constant Bandwidth Server [9] which is another algorithm appropriate for tasks whose execution times fluctuate.

This paper consists of five sections. Section 2 describes related works for scheduling algorithms, especially for task sets with hard/periodic and soft (or non real-time)/aperiodic tasks. Then, Section 3 proposes a method that is extended from TBS and uses predictive execution times rather than WCETs to improve aperiodic tasks' response times. Evaluation of the proposed method is shown in Section 4. Finally, Section 5 concludes the paper with summary and future works.

2. Related Works

2.1 Scheduling Algorithms for Both Periodic and Aperiodic Tasks

There are various scheduling algorithms for task sets consisting of both periodic and aperiodic tasks. They are categorized to fixed-priority servers and dynamic-priority servers. Fixed-priority servers are based on the rate monotonic (RM) scheduling [5], which has a merit that higher-priority periodic tasks (with shorter-periods) tend to have lower jitters and shorter response times. As representative examples, there are Deferrable Server [10], Priority Exchange [10], Sporadic Server [11], and Slack Stealing [12]. On the other hand, dynamic-priority servers are based on the earliest deadline first (EDF) algorithm, which provides a strong merit that a processor can be utilized by up to 100% while maintaining schedulability. Dynamic Priority Exchange [4], Dynamic Sporadic Server [4], Total Bandwidth Server [4], Earliest Deadline Late Server [4], and Constant Bandwidth Server [9] are examples of dynamic-priority servers. The aim of these algorithms is to make response times of aperiodic requests as short as possible with guaranteed schedulability, while the effectiveness is obtained in exchange for their implementation complexity.

In the technique proposed in this paper, an aperiodic execution is divided into two parts to obtain earlier deadlines than the original TBS. Similar to this work, there is a model, "imprecise computation," to improve real-time processing by dividing a task into two parts [13]. This assumes that each task consists of two parts, a mandatory subtask "M" and an optional subtask "O," both of which, basically, have the same deadline. When systems experience transient overload situations, this model can mitigate the load by giving up computing the optional subtasks, which means quality or precision is degraded but a minimum of real-time property can be kept. However, this can be done only when the tasks have been originally designed to trade performance with computational requiremetns. On the other hand, the technique in this paper does not assume that a task consists of M and O. All the aperiodic tasks are computed till the end of their codes. Therefore, the technique is a method that provides "precise" computations, not "imprecise" computations.

There is another strategy, slack reclaiming, for improving real-time processing [12], [14], [15][*1]. In the slack reclaiming, when some tasks finish earlier than their WCETs, the slack time (or unused bandwidth) is utilized by other (soft or non real-time) task executions. This means slack time can be utilized only after the early completion of the prior tasks. In contrast, in the technique in this paper, it is predicted that the execution of a target (aperiodic) task would be completed early. Then the corresponding early deadline is given in advance of the execution. This means the future slack by the task's own execution can be utilized for itself.

2.2 Total Bandwidth Server

Resource reservation is a general technique for real-time systems where aperiodic tasks are executed only in some reserved bandwidth and avoid influencing hard tasks' schedulability [3], [16]. Total Bandwidth Server (TBS) is one of the resource reservation methods and is a scheduling algorithm for a mixture of hard and non real-time tasks[*2]. TBS provides fair response times for non real-time tasks while keeping its implementation complexity moderate [4], [17]. It is assumed that hard tasks are invoked periodically and have relative deadlines equal to the length of their period, and that non real-time tasks are requested irregularly but do not have explicit deadline requirements in advance. When a non real-time task is invoked, a tentative absolute deadline is calculated and assigned to the task as:

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

where k means the kth instance of aperiodic tasks, r_k is the arrival (invocation) time of the kth instance, d_{k-1} is the absolute deadline for the $k-1$th (previous) instance, C_k is WCET of the kth instance[*3], and U_s is the processor utilization factor by the server which takes charge of execution of aperiodic tasks. The server is considered to be able to occupy the U_s utilization factor and, every time an aperiodic request arrives, it leads to give the instance the bandwidth equal to U_s. The term, $max(r_k, d_{k-1})$, prevents bandwidths given to successive aperiodic instances from overlapping with each other. After the requested aperiodic task is given the deadline by Eq. (1), all periodic and aperiodic instances are scheduled by following the EDF algorithm. Letting U_p be the processor utilization factor by all hard periodic tasks, it was proved that a task set is schedulable if and only if $U_p + U_s \leq 1$ [4].

In TBS, overestimated WCETs would make the calculated deadlines later than necessary by Eq. (1). According to the EDF scheduling policy, this might delay the execution of the aperiodic instance and cause long response time. To make matters worse, the delayed deadline can influence the following aperiodic instances by the term, $max(r_k, d_{k-1})$, in Eq. (1) (from $k-1$th

[*1] Strictly, Slack Stealing [12] is a method that utilizes slack time obtained by postponing hard tasks' execution, not by expecting their shorter execution than WCETs.

[*2] TBS basically targets hard tasks and non real-time tasks. However, it can handle soft tasks instead of non real-time tasks. Therefore, this paper does not distinguish soft tasks and non real-time tasks.

[*3] In the literature for TBS, it is not clearly described that C_k is WCET. However it is undeniable since, if it is not the case, the schedulability cannot be guaranteed, that is, deadline misses for hard tasks would occur.

to kth). Reference [18] showed the method, *resource reclaiming*, where the deadline is recalculated by using the actually elapsed execution time when the execution of the instance finishes, and the new deadline is used for the deadline calculation for the subsequent aperiodic instances. By this method, the subsequent instances benefit from the earlier deadlines and their response times would be improved.

In the resource reclaiming, kth aperiodic instance is given the deadline d'_k by:

$$d'_k = \bar{r}_k + \frac{C_k}{U_s} \tag{2}$$

$\bar{r}_k$ is the value calculated as:

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

That is, the maximal value among the arrival time, the recalculated deadline for the previous instance ($\bar{d}_{k-1}$, described below), and the finishing time of the previous instance (f_{k-1}) is selected as the release time. ($\bar{d}_{k-1}$ can be earlier than f_{k-1}, since the $k-1$th instance was executed with the old deadline before the deadline recalculation, d'_{k-1}, not with the recalculated deadline, $\bar{d}_{k-1}$.) When the $k-1$th aperiodic instance finishes, the deadline is recalculated by the following formula that includes the actual execution time, $\bar{C}_{k-1}$, of the instance, and is reflected in Eq. (3), and then in Eq. (2), for the subsequent task.

$$\bar{d}_{k-1} = \bar{r}_{k-1} + \frac{\bar{C}_{k-1}}{U_s} \tag{4}$$

There are other algorithms that derive from TBS. In Ref. [19], Buttazzo, et al. proposed a method for firm (not hard) periodic tasks and soft (or non real-time) aperiodic tasks. Exploiting a fact that a firm deadline allows the corresponding task to be missed to some degree, the algorithm achieves shorter response times for aperiodic tasks by skipping periodic executions at times and ensuring larger bandwidth for the aperiodic tasks. Similarly, Kato, et al., proposed a method where aperiodic instances start execution immediately when the requests arrive, and continue the execution for a certain interval [20]. This would shorten the response times but would cause deadline misses of periodic tasks while controlling the frequency of the deadline misses by feedback control. These two methods aim to achieve short response times at the sacrifice of completeness of periodic task executions. On the other hand, this paper proposes a method of shortening response times of aperiodic instances while ensuring the integrity of hard periodic tasks.

2.3 Constant Bandwidth Server

In almost all task execution models in real-time scheduling theories, tasks' execution times are assumed to be fixed (usually, their WCETs). Constant Bandwidth Server (CBS) [9] is a scheduling algorithm that does not expect fixed execution times. This scheduling algorithm is appropriate for tasks, such as those in multimedia applications, that change their elapsed times from execution to execution.

CBS has its server period and budget to serve aperiodic requests. A common deadline is prepared and managed with the server period and budget, and all aperiodic tasks are given the common deadline, independent of their execution times. The deadline is repeatedly updated according to rules, basically by incrementing by the length of the period. CBS has a merit that resource reclaiming is naturally performed since the budget is decreased by only the amount actually spent for tasks' execution. However the deadline depends on the period, not on tasks' execution times, which might not provide an optimal deadline tailored for each task.

According to Ref. [9], CBS is summarized as follows. A CBS has the server period (T_s), the maximum budget (Q_s), the server bandwidth ($U_s = Q_s/T_s$), the current budget (c_s), and, at each instant, the fixed (common) deadline ($d_{s,k}$). Aperiodic jobs are served in FIFO order. The served (head in FIFO) aperiodic job is given a dynamic deadline equal to the current fixed deadline, and scheduled by EDF together with periodic tasks. Whenever an aperiodic job has been executed, the current budget, c_s, is decreased by the same amount as the elapsed ticks for the execution. When c_s gets zero, c_s is replenished to Q_s, and the next fixed deadline is calculated by $d_{s,k+1} = d_{s,k} + T_s$. When an aperiodic job arrives at $t = r_j$ and the server does not have other pending jobs, if $c_s > (d_{s,k} - r_j)U_s$, it generates the next fixed deadline by $d_{s,k+1} = r_j + T_s$ and c_s is replenished to Q_s. Otherwise (if $c_s \leq (d_{s,k} - r_j)U_s$), it is served with the current deadline of $d_{s,k}$ and the current budget of c_s.

3. The Adaptive Total Bandwidth Server

As is the case with TBS, a method proposed in this chapter, *adaptive total bandwidth server* (*Adaptive TBS*), assumes that task sets consist of periodic tasks with hard deadlines and aperiodic tasks without explicit deadlines, where it is desirable that execution of aperiodic tasks finishes as early as possible. Since aperiodic tasks do not have deadlines, it is not necessary to assume that WCETs are spent for their executions from a schedulability point of view. That is, although TBS dynamically gives tentative deadlines to aperiodic instances, missing the deadlines is not serious or catastrophic. Therefore, use of WCETs for deadline calculation is not essential. Instead, shorter times can be supposed as execution times and used for the deadline calculation while maintaining schedulability of the whole task set. When the supposed execution time elapsed but the execution did not finish yet, the deadline is recalculated by using the longer execution time, that is, WCET, and then the EDF scheduling only has to be performed again. By this strategy, when aperiodic execution finishes in the supposed (short) time, the corresponding short deadline and the EDF algorithm can make the response time shorter.

3.1 Predictive Execution Times (PET)

In practice, actual execution time of a task tends to be much shorter than the corresponding WCET, since the WCET is pessimistically estimated. The proposed adaptive TBS intends to use actual execution times instead of WCETs in the deadline calculation for aperiodic instances. However, since the actual execution times are unknown beforehand, they are predicted. There are various possible ways to obtain the predictive execution times.

(1) Random choice of execution times

This has high possibility of choosing shorter times than actual execution times, and therefore would cause many deadline misses (although the misses are not serious). On the other hand, when the predicted time is much longer than the actual execution time, the calculated deadline would not be early enough.

(2) Measurement in advance
This seems effective in terms of closeness with the actual execution times but has a defect of not being able to follow the change of execution times when a task is executed many times in the system operation.

(3) History-based prediction
This predicts execution times by an execution history of the same task and therefore may follow the fluctuation of execution times.

Although prediction accuracy would affect the effectiveness of the proposed adaptive TBS as confirmed in Section 4, this paper does not go deeply into the elaboration of the prediction method. For the present, the following prediction method which corresponds to the above (3) is used.

$$\begin{aligned} C_{i_k}^{PET} &= \alpha \times C_{i_{k-1}}^{PET} + (1-\alpha) \times C_{i_{k-1}}^{AET} \\ C_{i_0}^{PET} &= C_i^{WCET} \end{aligned} \tag{5}$$

Here, $C_{i_k}^{PET}$ is a predictive execution time (PET) for the kth instance of an aperiodic task J_i. $C_{i_{k-1}}^{AET}$ is the execution time actually spent for the previous $(k-1)$th execution of the same task J_i. The initial value, $C_{i_0}^{PET}$, is equal to WCET of the task, C_i^{WCET}. This formula calculates as the predictive execution time an weighted average of the previous PET and the previous actual execution time with the weighting coefficient α.

3.2 Definition of Adaptive TBS

As in Ref. [4], it is assumed that a periodic task have a constant period, hard deadline equal to the end of its period, and constant worst-case execution time, and that an aperiodic task does not have deadline in advance and its arrival time is unknown.

In the adaptive TBS, execution of an aperiodic task is divided into two sub instances. They are regarded as different instances, and then the original TBS is naturally applied.

In the following descriptions, aperiodic tasks are not distinguished and they are supposed to have a global serial instance number, k, according to the request order. Execution of kth aperiodic instance, J_k, is divided into two parts, J_k^{PET} and J_k^{REST}. J_k^{PET} corresponds to the execution from the beginning of J_k to the predicted finishing time. J_k^{REST} corresponds to the execution from the predicted finishing time. If the execution of J_k finishes at or before the predicted time, J_k^{REST} does not exist. Let the worst-case execution time of J_k be C_k^{WCET}, the predictive execution time of J_k be C_k^{PET}, and the execution time of J_k^{REST} be $C_k^{REST} = C_k^{WCET} - C_k^{PET}$. (Practically, $0 \le C_k^{REST} \le C_k^{WCET} - C_k^{PET}$. The worst-case scenario, $C_k^{REST} = C_k^{WCET} - C_k^{PET}$ is assumed for schedulability.) When the kth aperiodic request arrives at the time $t = r_k$, two instances for the request are assigned deadlines as:

$$d_k^{PET} = \max(r_k, d_{k-1}) + \frac{C_k^{PET}}{U_s} \tag{6}$$

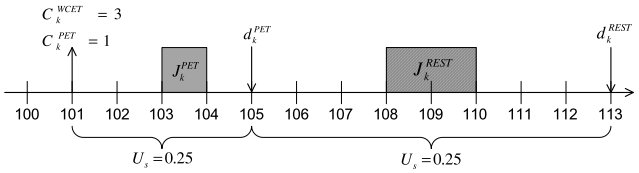

Fig. 1 Deadline assignment in Adaptive TBS.

$$d_k^{REST} = d_k^{PET} + \frac{C_k^{REST}}{U_s} \tag{7}$$

Deadline assignment in the original TBS was as:

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k^{WCET}}{U_s} \tag{8}$$

From $C_k^{REST} = C_k^{WCET} - C_k^{PET}$ and Eqs. (6), (7), and (8),

$$\begin{aligned} d_k^{REST} &= \max(r_k, d_{k-1}) + \frac{C_k^{PET}}{U_s} + \frac{C_k^{WCET} - C_k^{PET}}{U_s} \\ &= \max(r_k, d_{k-1}) + \frac{C_k^{WCET}}{U_s} \\ &= d_k \end{aligned}$$

Therefore, two deadlines can be calculated by Eqs. (6) and (8) at the arrival time. The use of Eq. (8) is more suitable than Eq. (7) since the second term in the right expression is calculated with two constants and therefore has only to be calculated once in advance of the system operation.

Figure 1 is an example of assigning deadlines to an aperiodic request occurring at tick 101. This aperiodic task is supposed to have the worst-case execution time of $C_k^{WCET} = 3$ and the predictive execution time of $C_k^{PET} = 1$. The processor utilization reserved for the aperiodic server is 0.25. The deadline for J_k^{PET} becomes $d_k^{PET} = 101 + 1/0.25 = 105$, and that for the remaining execution (J_k^{REST}) becomes $d_k^{REST} = 105 + (3-1)/0.25 = 113$ (= d_k^{WCET}). In this example, if the task finishes within the predictive execution time, the response time is $104-101 = 3$. Otherwise, the remaining part should be executed, and the response time would be $110 - 101 = 9$. (This example implies that other tasks with deadlines earlier than d_k^{WCET} are executed from tick 104 to 108.)

3.3 Example of Adaptive TBS

In this section, an example of scheduling by the adaptive TBS is shown. In **Fig. 2**, (1) and (2) show scheduling results of the original TBS and the adaptive TBS, respectively. There are two periodic tasks, τ_1 and τ_2, and an aperiodic request arriving at tick 51. The period of τ_1 is $T_1 = 4$, and its execution time is $C_1 = 1$. τ_2 has the period $T_2 = 6$, and the execution time is $C_2 = 3$[*4]. Therefore, the processor utilization by the two periodic tasks is $U_p = 0.25 + 0.5 = 0.75$ and the utilization by the aperiodic server leads to $U_s = 1 - U_p = 0.25$.

WCET of the aperiodic request is supposed to be 3, while the predictive execution time and the actual execution time are 2. In the original TBS, the deadline of the aperiodic task is $d_k^{WCET} = 51 + 3/0.25 = 63$. According to the EDF algorithm, the aperiodic task starts execution at tick 55, is suspended at tick

[*4] In this example, it is assumed that the execution times of τ_1 and τ_2 are fixed.

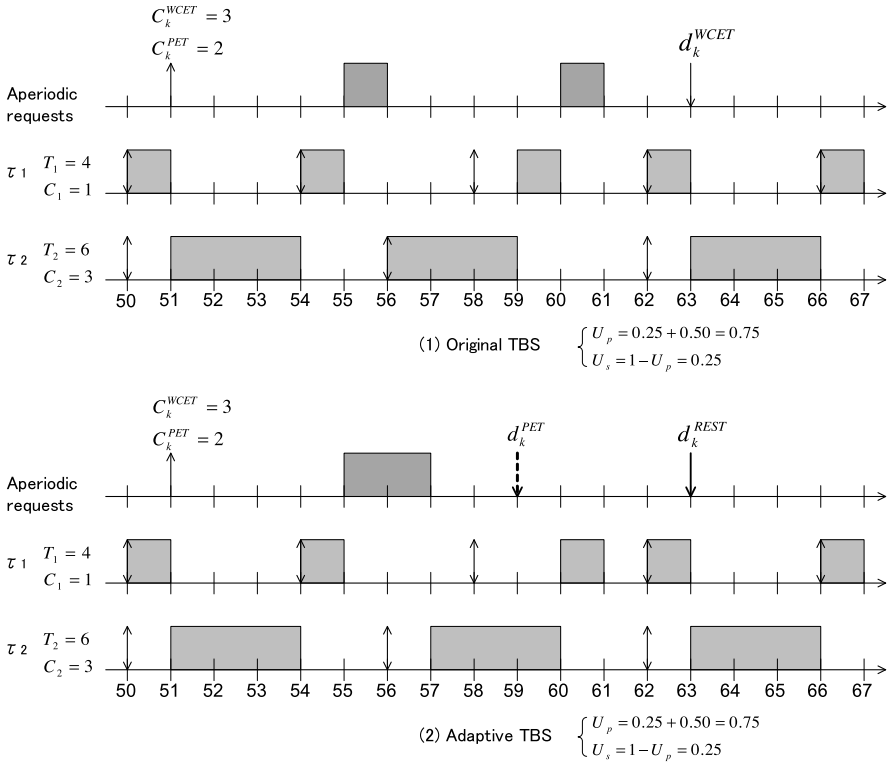

Fig. 2 Example of schedules by the original and adaptive TBS.

56, resumes at tick 60, and finishes at tick 61. Consequently, the response time becomes $61 - 51 = 10$ ticks. On the other hand, in the adaptive TBS, the two deadlines, $d_k^{PET} = 51 + 2/0.25 = 59$ and $d_k^{REST} = 59 + (3-2)/0.25 = 63$, are given. Based on EDF, the aperiodic task starts execution at tick 55 and finishes at tick 57, which gives the response time of $57 - 51 = 6$ ticks. In this example, the adaptive TBS shortens the response time by 4 ticks compared with the original TBS.

Suppose that the same task set is scheduled except that the actual execution time of the aperiodic task is 3 ticks. In the original TBS, the aperiodic execution would finish at tick 62. On the other hand, the adaptive TBS suspends the aperiodic execution at tick 57, resumes the execution at tick 61, and then finishes it at tick 62. Like this scenario, even if the execution time is incorrectly predicted, the response time would be the same as or shorter than that in the original TBS.

3.4 Schedulability of Adaptive TBS

After an aperiodic request is divided into two sub instances, the adaptive TBS behaves just as the original TBS, where the two (different) sub instances can be considered to arrive at the same time. Obviously, from Eqs. (6) and (7), the utilization by the two sub instances between $max(r_k, d_{k-1})$ and $d_k^{REST}(= d_k)$ is the same as that in the original TBS as follows.

$$U_{J_k^{PET}} = \frac{C_k^{PET}}{d_k^{PET} - \max(r_k, d_{k-1})} = U_s$$

$$U_{J_k^{REST}} = \frac{C_k^{REST}}{d_k^{REST} - d_k^{PET}} = U_s$$

Therefore, schedulability of the adaptive TBS leads to be the same as that of the original TBS presented in Ref. [17].

3.5 Implementation Complexity

In the proposed algorithm, aperiodic task execution is divided into two sub instances. However, operating systems should manage a task with a single information set, task control block (TCB). This is realized by re-setting up the deadline and re-inserting the task in the ready queue when PET elapses and the task has not finished, which is the only difference from the original TBS. To find that the execution reaches PET, the scheduler should be executed every tick timing. This is achieved by calling the scheduler when timer/tick interrupts occur, which is a natural procedure that operating systems usually follow. In addition, as described in Section 3.2, the value of the second term in the right side in Eq. (8) should be statically computed and used when necessary to reduce the recalculation overheads.

3.6 Affinity with Resource Reclaiming

In the TBS, when deadline is calculated for the kth aperiodic request, d_{k-1} is needed. In the Adaptive TBS, since the previous ($k-1$th) aperiodic execution is divided into two sub instances, the deadline for the second sub instance, that is d_{k-1}^{REST}, is used for the calculation. However, when the execution of the $k-1$th request finishes in its PET (C_{k-1}^{PET}), the second sub instance is not executed. In this case, instead of d_{k-1}^{REST}, d_{k-1}^{PET} can be used to calculate the deadline for the kth task. This is the first resource reclaiming method for the adaptive TBS.

A greedier method [18] described in Section 2 can be easily applied to the adaptive TBS. When the execution of an aperiodic request finishes, whether or not the execution is for the first or

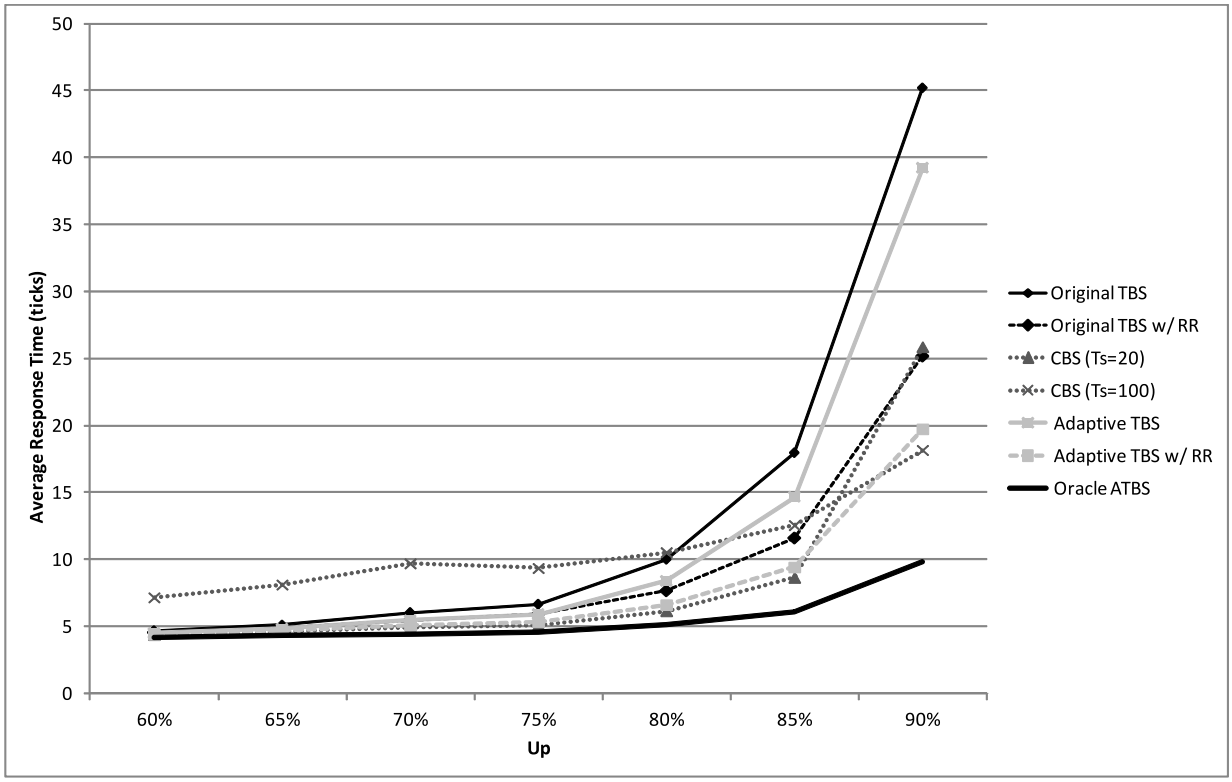

Fig. 3 Average response times.

second sub instance after the division, the deadline is recalculated by Eq. (4), then the updated deadline is applied to Eq. (3), and consequently the following aperiodic tasks can be given earlier deadlines by Eq. (2). This can be the second resource reclaiming method for the adaptive TBS, and naturally includes the first method above. For the evaluation in this paper, the second resource reclaiming method described above is used with the adaptive TBS.

4. Evaluation

In this Section, the original TBS, CBS described in Section 2, and the proposed adaptive TBS are compared by simulation for task sets with varying execution times.

4.1 Evaluation Methodology

The original TBS (Original TBS), the original TBS with the resource reclaiming technique (Original TBS w/ RR), CBS, the adaptive TBS (Adaptive TBS), the adaptive TBS with the resource reclaiming (Adaptive TBS w/ RR), and the ideal adaptive TBS where execution times of aperiodic tasks are known[*5] (Oracle ATBS) are compared in terms of average response times.

In the simulations, task sets generated by using probabilistic distributions are used similar to related works [4], [9], [10], [11], [12], [15], [16], [17], [18], [19], [20]. Task sets consist of periodic tasks with the total processor utilization (U_p) of 60% to 90% at intervals of 5% and aperiodic tasks with the total utilization of about 2% in the observation period (100,000 ticks). Both the aperiodic servers for CBS and the adaptive TBS have the utilization $U_s = 1 - U_p$. For periodic tasks, their periods are decided by exponential distributions where the average value is 100 ticks. Their WCETs and actual execution times are equal and obtained by exponential distributions with the average of 10 ticks. As for aperiodic tasks, a task set includes four different aperiodic tasks. Each aperiodic task in a set is invoked multiple times and the arrival times are decided by Poisson distribution with 1.25 per 1,000 ticks on average. The WCETs are decided by exponential distributions with the average of 8 ticks. Each task instance has its actual execution time decided by exponential distributions with the average of 4 ticks, under the condition that the upper bound is the corresponding WCET. For all aperiodic task sets, the average ratio of actual execution times to the corresponding WCETs was about 0.33.

For each U_p from 60% to 90%[*6], all combinations of ten periodic task sets and ten aperiodic task sets (total 100 task sets) are simulated and the average value of aperiodic response times is shown. For the adaptive TBS, the weighting coefficient for PET calculation (α in Section 3.1) is 0.5. For CBS, two cases, with the server period of 20 ticks and with that of 100 ticks, are simulated. The maximum budget becomes $Q_s = \lfloor T_s \times U_s \rfloor$.

4.2 Results

The results are shown in **Fig. 3** in which the horizontal axis indicates the processor utilization by periodic tasks (U_p), and the vertical axis indicates the average response time of aperiodic task executions. Under U_p of 65%, the average response times are almost the same for all the methods except CBS with $T_s = 100$.

[*5] In other words, PET is perfectly predicted and therefore task execution finishes only by the first sub instance.

[*6] Since tasks' execution times are decided by using probability distributions, U_p is, for example, not exactly 60%, but about 60% (e.g., 60.2%).

This is because the server utilization, $U_s = 1 - U_p$, is large enough to quickly serve aperiodic requests. Over 70%, the differences gradually appear. When U_p is 90%, the differences are largest; the average response time of Original TBS is 45.2 ticks, that of Original TBS w/ RR is 25.2 ticks, that of CBS with $T_s = 20$ is 25.9 ticks, and that of CBS with $T_s = 100$ is 18.2 ticks. On the other hand, Adaptive TBS, Adaptive TBS w/ RR, and Oracle ATBS exhibit the average response times of 39.2, 19.7, and 9.8 ticks, respectively.

4.2.1 Adaptive TBS vs. Original TBS

In this evaluation, the method with deadline assignment based on PET (Adaptive TBS) improved the average response time by 13% compared to Original TBS that is based on WCET, and the proposed method with the resource reclaiming technique (Adaptive TBS w/ RR) outperformed Original TBS w/ RR by 22%. Consequently, the PET-based method exhibits better ability when it is applied with the resource reclaiming technique.

The use of PET is discussed. The ratio of aperiodic executions that finished in PET was 57%. **Table 1** shows the average of the shortened deadline length for aperiodic instances that finished in their PET in the simulation of Fig. 3. "Shortened deadline length" means how shorter in ticks the deadline is than in the case based on WCET. The difference between Adaptive TBS and Adaptive TBS w/ RR does not exist, and therefore the table shows the values collectively. It is confirmed that the larger U_p is, the longer the shortened length is. This is because larger U_p corresponds to smaller $U_s (= 1 - U_p)$, therefore the second term of the right expression in Eq. (1) would be larger and then the shortened length would be longer. Consequently, the adaptive TBS methods using PET provide larger improvements when the processor utilization by periodic tasks is high, in other words, when the capacity of the aperiodic server is small.

Next, effects of resource reclaiming are discussed. **Table 2** shows ratios of resource reclaiming that actually affected the deadline calculation for the succeeding tasks (that is, ratios of the cases where d_{k-1} is the maximum in Eq. (1) before resource reclaiming). In addition, the table includes average shortened deadline lengths in parentheses by the resource reclaiming. From the table, it is confirmed that, when U_p is larger, more and longer resource reclaiming effects are obtained.

Table 1 Average of shortened deadline length brought by PETs (ticks).

U_p (%)	Average of shortened deadline
60	19.7
65	22.8
70	26.5
75	31.8
80	40.0
85	54.1
90	84.6

Table 2 Affected resource reclaiming ratio (%) and average shortened deadline lengths.

U_p (%)	Original TBS w/ RR	Adaptive TBS w/ RR
60	13.7 (20.1)	12.5 (19.7)
65	16.0 (23.2)	14.8 (22.8)
70	18.8 (28.2)	17.6 (27.7)
75	22.7 (36.8)	21.4 (36.3)
80	28.8 (52.7)	27.4 (51.6)
85	38.3 (85.5)	36.6 (83.3)
90	57.9 (299.1)	56.0 (297.4)

4.2.2 Adaptive TBS vs. CBS

As for CBS with the short server period ($T_s = 20$ ticks), when U_p is not heavy (from 60% to 85%), the budget is large enough and therefore the average response times become short. However, when U_p is heavy (90%), the budget is small and then aperiodic instances might not finish in a period. In this case, the deadline is postponed to the end of the next period. Then the EDF algorithm would give the instances low priority and the response time would be longer. When U_p is 90%, CBS with $T_s = 20$ exhibited the average response time of 25.9 ticks, which is longer than CBS with $T_s = 100$ and Adaptive TBS w/ RR.

On the other hand, with the long server period ($T_s = 100$ ticks), since CBS gives aperiodic instances deadlines equal to the period even if the execution times are short, the response times would not be short especially when U_p is not heavy. Actually, at a maximum in these results, the average response time of Adaptive TBS w/ RR is shorter by 48% than that of CBS with $T_s = 100$ when $U_p = 70\%$. On the other hand, when U_p is large (90%) and the budget is small, the average response time of CBS with $T_s = 100$ is shorter than that of Adaptive TBS w/ RR by about 1.5 ticks. The adaptive TBS cannot give deadlines short enough to aperiodic instances when their PETs longer than the actual execution times are estimated. In such a case, CBS might make response times shorter if the execution times fit the budget.

As the results imply, an appropriate server period of CBS varies depending on the utilization. Short periods would be appropriate for low U_p and long ones for high U_p. Although periodic task sets are fixed in actual applications, their execution times tend to vary as those of aperiodic tasks. Since U_p is decided based on tasks' WCETs, the actual utilization would be smaller than U_p. This means optimal Ts would be unknown and difficult to decide beforehand. In addition, in this evaluation, arrivals and execution times of the four aperiodic tasks followed the same probability distribution. If the execution times much differ from task to task, it would be more difficult to find the appropriate server period and budget for all the tasks.

In this comparison, although effectiveness of CBS and the adaptive TBS seems to depend on the cases of T_s and U_p, it would be likely that Adaptive TBS w/ RR gives more convincing performance than the others throughout all U_p. Furthermore, as Oracle ATBS shows, the adaptive TBS with the perfect prediction of PETs always outperforms CBS. This means that better PETs would improve the response times. The average prediction accuracy in the simulations is shown in **Table 3**, where percentages of predicted PETs matching the actual execution times as well as percentages of underestimated and overestimated PETs are given. With the adaptive TBS, when PET is underestimated, (not serious) deadline misses might occur and the remaining part is executed with the deadline based on WCET, which means that

Table 3 Average prediction accuracy for PETs (%).

Relation	Ratio (%)
Matched	13.86
Underestimated	43.16
Overestimated	42.98
(Over- but shorter than WCET)	(42.16)

short response time is difficult to achieve. This is the case in 43% of aperiodic executions from the table. On the other hand, when PET is overestimated, enough urgent deadline is not given, and then response time short enough cannot be expected. (However, overestimated PET shorter than WCET can bring some effect. The bottom line in the table shows this ratio.) This table indicates that the prediction accuracy is not high enough and there are frequent overestimations and underestimations. Therefore, appropriate deadlines are not obtained in most cases. Better prediction methods should be investigated to improve the effects.

5. Concluding Remarks

Total Bandwidth Server is task scheduling algorithm for task sets consisting of periodic tasks with hard deadlines and aperiodic tasks without deadlines. In this paper, for improving the TBS, a method that uses predictive execution times (PET) instead of worst-case execution times for deadline calculation of aperiodic instances is proposed. The use of PETs is allowed since aperiodic tasks do not have explicit deadlines. The aim of the method is to shorten response times of aperiodic tasks, while the schedulability of periodic tasks is not influenced. The method can be used with the resource reclaiming technique to further reduce response times.

The evaluation by simulations confirmed that the use of PETs could shorten response times of aperiodic executions by 13% in the case without resource reclaiming, and by 22% in the case with resource reclaiming. In addition, it was found that CBS that is tolerant of variation in tasks' execution times has difficulty in fixing the server period appropriate for any processor utilization. The adaptive TBS has a possibility of always outperforming CBS if the prediction accuracy for PETs improves. Therefore, better prediction methods for PETs need to be explored, than that simply based on the weighted average of the previous execution time and the previous PET, described in this paper. Unlike estimation of WCETs, prediction of PETs does not involve schedulability issues. Therefore, occasional over/underestimation is not serious. This means that various history-based or statistical techniques that are more sophisticated than that in this paper can be candidates for the prediction methods.

The evaluation in this paper used task sets that were generated based on probability distributions. To reflect actual situations where task execution times fluctuate, actual program codes, such as multimedia tasks mainly targeted in CBS [9], should be used. In the future, evaluation with actual program codes and scheduling overheads should be performed.

References

[1] 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).

[2] 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).

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

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

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

[6] Lundqvist, T. and Stenström, P.: Timing Anomalies in Dynamically Scheduled Microprocessors, *Proc. IEEE Real-Time Systems Symposium*, pp.12–21 (1999).

[7] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenström, P.: The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools, *ACM Trans. Embedded Computing Systems*, Vol.7, No.3, pp.1–53 (2008).

[8] Silberschatz, A., Galvin, P.B. and Gagne, G.: *Operating System Concepts*, 8th edition, John Wiley & Sons, Inc. (2009).

[9] Abeni, L. and Buttazzo, G.: Integrating Multimedia Applications in Hard Real-Time Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.4–13 (1998).

[10] Lehoczky, J.P., Sha, L. and Strosnider, J.K.: Enhanced Aperiodic Responsiveness in Hard Real-Time Environments, *Proc. IEEE Real-Time Systems Symposium*, pp.261–270 (1987).

[11] Sprunt, B., Sha, L. and Lehoczky, J.: Aperiodic Task Scheduling for Hard-Real-Time Systems, *J. Real-Time Systems*, Vol.1, No.1, pp.27–60 (1989).

[12] Lehoczky, J.P. and Ramos-Thue, S.: An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.110–123 (1992).

[13] Liu, J.W., Kwei-Jay Lin, W.-K.S. and shi Yu, A.C.: Algorithms for Scheduling Imprecise Computations, *IEEE Computer*, Vol.24, No.5, pp.58–68 (1991).

[14] Caccamo, M., Buttazzo, G. and Sha, L.: Capacity Sharing for Overrun Control, *Proc. IEEE Real-Time Systems Symposium*, pp.295–304 (2000).

[15] Lin, C. and Brandt, S.A.: Improving Soft Real-Time Performance Through Better Slack Reclaiming, *Proc. IEEE Real-Time Systems Symposium*, pp.410–421 (2005).

[16] Abeni, L. and Buttazzo, G.: Resource Reservation in Dynamic Real-Time Systems, *J. Real-Time Systems*, Vol.27, No.2, pp.123–167 (2004).

[17] Spuri, M. and Buttazzo, G.: Scheduling Aperiodic Tasks in Dynamic Priority Systems, *J. Real-Time Systems*, Vol.10, No.2, pp.179–210 (1996).

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

[19] Buttazzo, G.C. and Caccamo, M.: Minimizing Aperiodic Response Times in a Firm Real-Time Environment, *IEEE Trans. Software Engineering*, Vol.25, No.1, pp.22–32 (1999).

[20] Kato, S. and Yamasaki, N.: Feedback-Controlled Server for Scheduling Aperiodic Tasks, *World Academy of Science, Engineering and Technology*, No.9, pp.164–169 (2007).

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.