Adaptive Total Bandwidth Server: Using Predictive Execution Time

Kiyofumi Tanaka

Adaptive Total Bandwidth Server: Using Predictive Execution Time

Kiyofumi Tanaka

School of Information Science, Japan Advanced Institute of Science and Technology,
Asahidai 1–1, Nomi-city, Ishikawa, 923–1292 Japan
kiyofumi@jaist.ac.jp

Abstract. Along with the growing diversity and complexity of real-time embedded systems, it is becoming common that different types of tasks, periodic tasks and aperiodic tasks, reside in a system. In such systems, it is important that schedulability of periodic tasks is maintained and at the same time response times to aperiodic requests are short enough. Total Bandwidth Server (TBS) is one of convincing task scheduling algorithms for mixed task sets of periodic and aperiodic tasks. This paper proposes a method of using predictive execution times instead of worst-case execution times for deadline calculations in TBS to obtain shorter deadlines and reducing response times of aperiodic execution, while maintaining the schedulability of periodic tasks. From the evaluation by simulation, the proposed method combined with a resource reclaiming technique exhibits better average response times for aperiodic tasks, in case of a heavy load, by up to 39%.

Keywords: Real-time task scheduling, worst-case execution time, predictive execution time, total bandwidth server

1 Introduction

Along with the growing diversity and complexity of embedded systems, it is becoming common that different types of tasks reside in a system. For example, control tasks that are required to completely meet real-time requirements (hard tasks) and user interface tasks that should give a certain level of response times but are not required to completely behave at real-time (soft tasks) would be mixed. To achieve real-time processing required in such a system, real-time scheduling algorithms that involve both hard and soft tasks and guarantee the schedulability of tasks (especially of hard tasks) must be used.

Hard tasks should be periodically invoked and be considered to spend their worst-case execution time (WCET), since the schedulability, that they satisfy their deadline requirements, must be confirmed in advance of system operation. On the other hand, soft tasks can run on aperiodic invocations because of inexact real-time requirements. There is a scheduling algorithm for such hard and soft tasks, Total Bandwidth Server (TBS) [1]. TBS has a merit that CPU utilization can be up to 100% while maintaining schedulability. This study explores algorithms based on TBS.

Complexity of current processors and programs makes estimation of WCET difficult. For example, deep pipelining execution of machine instructions makes estimation of their execution times hard, or in a system with many tasks, whether each memory reference would hit in the cache memory or not is difficult to decide/predict [2]. In addition, the worst-case execution path in a program is almost impossible to trace since it includes many branches and loop structures, and all input patterns give a vast search range [3]. Consequently, WCET is obliged to be pessimistically estimated and leads to having a large gap with actual execution times. Due to this gap, it is difficult to obtain the best schedules by scheduling algorithms that use tasks' execution times.

In this research, it is taken into consideration that soft tasks are not required to completely satisfy the deadline constraints, and, instead of WCET, predictive execution time (PET) is introduced in the TBS-based scheduling algorithm, which aims to shorten response times of soft tasks.

2 Related Works

There are various scheduling algorithms proposed 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 rate monotonic (RM) scheduling [4], which has a merit that higher-priority tasks have lower jitters. As representative examples, Deferrable Server [5], Priority Exchange [5], Sporadic Server [6], and Slack Stealing [7] were proposed. On the other hand, dynamic-priority servers are based on earliest deadline first (EDF) scheduling [4], which provides a strong merit that CPU utilization can reach up to 100% while maintaining schedulability. Dynamic Priority Exchange [1], Dynamic Sporadic server [1], Total Bandwidth Server [1], Earliest Deadline Late Server [1], and Constant Bandwidth Server [8] are examples of dynamic-priority servers. The aim of these algorithms is to make response times of aperiodic requests shorter, while the effectiveness is obtained in exchange for their implementation complexity.

Total Bandwidth Server (TBS) provides good response times while leaving its implementation complexity moderate. It is assumed that hard tasks are invoked periodically and have their deadlines equal to the end of the period, and that soft tasks are invoked irregularly, but do not have their explicit deadline requirements in advance. When a soft task is invoked, tentative deadline is calculated and given to the task as:

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

where k means kth instance of aperiodic tasks, r_k is the arrival time of the kth instance, d_{k-1} is the absolute deadline of the $k-1$th (previous) instance, C_k is WCET of the kth instance, and U_s is the CPU utilization factor by the server which takes charge of execution of aperiodic tasks. The server is considered to occupy the U_s utilization factor and, every time an aperiodic request arrives,

leads to give the instance the bandwidth of U_s. The term, $max(r_k, d_{k-1})$, prevents bandwidths of successive aperiodic instances from overlapping with each other. After an instance of an aperiodic task is given its deadline, all periodic and aperiodic tasks are scheduled by following EDF algorithm. By letting U_p be the CPU utilization factor by all hard periodic tasks, it was proved that a task set is schedulable if and only if $U_p + U_s \leq 1$ [1].

In TBS, overestimated WCET would make the deadline later than necessary by the formula (1). This might delay the execution of the aperiodic instance and cause long response time. The literature [10] showed the method, *resource reclaiming*, where deadline is recalculated by using actually elapsed execution time when the instance finishes, and the new deadline is used for the deadline calculation for 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 deadline d'_k by:

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

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

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

That is, the maximal value among the arrival time, the recalculated deadline of the previous instance and the finishing time of the previous instance is selected as the release time. When the $k-1$th aperiodic instance finishes, the deadline is recalculated by the following formula that includes the actual execution time, $\overline{C}_{k-1}$, of the instance, and is reflected in the formula (3), and then in the formula (2), for the subsequent task.

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

There is another algorithm based on TBS. In the literature [11], Buttazzo, et al. proposed a method for firm (not hard) periodic and soft aperiodic tasks. Since firm deadline allows a task to be missed to some degree, the algorithm achieves shorter response times by skipping periodic executions at times and ensuring larger bandwidth for aperiodic tasks. This method aims to achieve short response times at the sacrifice of completeness of periodic instances. On the other hand, in this paper, a method of shortening response times of aperiodic instances while maintaining schedulability of periodic tasks is proposed.

3 The Adaptive Total Bandwidth Algorithm

As is the case with the total bandwidth server algorithm presented in the literature [1], this paper assumes that task sets consist of periodic tasks with hard deadlines and aperiodic tasks without explicit deadlines, where it is desirable

that aperiodic execution finishes as early as possible. Since aperiodic tasks do not have deadline, it is not necessary to use WCET from a schedulability point of view. Although TBS dynamically gives tentative deadlines to aperiodic instances, missing the deadlines is not serious or catastrophic. Therefore, use of WCET for deadline calculation is not essential. Instead, shorter execution times can be assumed and used for the deadline calculation while maintaining schedulability of the whole task set. When the assumed execution times elapsed but the execution did not finish yet, the deadline only has to be recalculated by using longer execution times, for example, WCET. By this strategy, when aperiodic execution finishes in the assumed time, the corresponding short deadline and EDF algorithm can make the response time shorter.

3.1 Prediction of Execution Time (PET)

Generally, in real-time scheduling and its schedulability analysis, task execution is considered to spend worst-case execution time (WCET). In practice, task execution time is unknown beforehand and therefore WCET must be supposed to spend, especially in hard real-time systems. However, in most cases, actual execution time is shorter than WCET. Since WCET is pessimistically estimated, the difference between the actual execution time and WCET tends to be large.

As the impact of the difference, for example, when SJF (Shortest Job First) algorithm is applied based on WCETs, the average turnaround time would be worse than that of the same algorithm based on actual execution times, although this is an oracle with prior information. In Figure 1, (1) shows that WCETs of task A, task B and task C are 2, 3, and 4, respectively, and the execution order is A, B, and then C based on SJF. When this order is applied to actual executions where actual execution times are 2, 1, and 2 for task A, task B, and task C, respectively, the average turnaround time becomes 3.33 (Figure 1 (2)). On the other hand, if the actual execution times are known in advance and used in the algorithm, the average turnaround time will be 3 under SJF as shown in Figure 1 (3). Like this, decision based on WCETs is not necessarily the best.

For task sets consisting of both hard and soft tasks, although WCET must be assumed for execution of hard tasks, execution time shorter than WCET can be assumed for soft tasks since they can miss their deadline to some de-

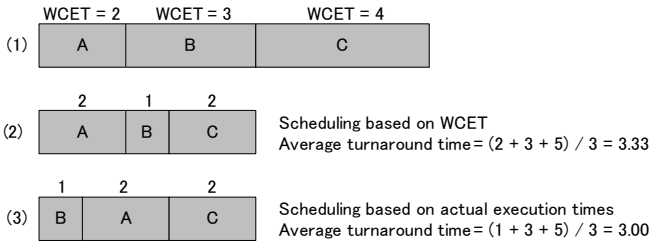

Fig. 1. Scheduling base on Shortest Job First.

gree. Especially in the total bandwidth server environments, deadlines are not given to aperiodic tasks in advance. In run-time, (tentative) deadline is calculated using WCET and is assigned dynamically. If the deadline calculation for aperiodic tasks uses execution time shorter than WCET, earlier deadline can be obtained and therefore shorter response time can be expected. Assumption of shorter execution time can cause deadline misses. However, the deadline misses are not serious since the tasks are for soft real-time processing. After the misses, remaining execution has only to continue.

In this strategy, execution times should be predicted. There are various possible ways to obtain the predictive execution times (PET).

1. Random choice of execution times
2. Measurement in advance
3. Prediction using a history of execution

The above 1 has high possibility of choosing shorter times than actual execution times, and therefore would cause many deadline misses (although the misses are not serious). The next one seems effective but has a defect of not following the change of execution times when a task is executed many times in the system operation. The third one predicts execution times by an execution history of the same task and therefore can follow the fluctuation of execution times. The prediction method is not the most important essence of the proposed adaptive TBS. For the present, this paper uses the following prediction method which corresponds to the above 3.

$$C_{i_{PET_k}} = \alpha \times C_{i_{PET_{k-1}}} + (1-\alpha) \times C_{i_{ET_{k-1}}}, \quad C_{i_{PET_0}} = C_{i_{WCET}} \tag{5}$$

Here, $C_{i_{PET_k}}$ is PET for kth instance of an aperiodic task J_i. $C_{i_{ET_{k-1}}}$ is the execution time actually spent for the previous execution of the same task J_i. The initial value $C_{i_{PET_0}}$ 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 the Adaptive TB Server

In the adaptive TBS, an instance 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 global serial instance numbers, k, according to the request order. Execution of J_k, kth instance of aperiodic tasks, is divided into two parts, J_{PET_k} and J_{REST_k}. J_{PET_k} corresponds to the execution from the beginning of J_k to the predicted finishing time. J_{REST_k} corresponds to the execution from the predicted finishing time. If the execution of J_k finishes at or before the predicted time, J_{REST_k} does not exist. Let the worst case execution time of J_k be C_{WCET_k}, the predictive execution time of J_k be C_{PET_k}, and the execution time of J_{REST_k}

be $C_{REST_k} = C_{WCET_k} - C_{PET_k}$. When the kth aperiodic request arrives at the time $t = r_k$, two instances for the request are assigned deadlines as:

$$d_{PET_k} = max(r_k, d_{k-1}) + \frac{C_{PET_k}}{U_s} \tag{6}$$

$$d_{REST_k} = d_{PET_k} + \frac{C_{REST_k}}{U_s} \tag{7}$$

Deadline assignment in the original TBS was as:

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

From $C_{REST_k} = C_{WCET_k} - C_{PET_k}$ and the formula (6), (7), and (8),

$$\begin{aligned} d_{REST_k} &= max(r_k, d_{k-1}) + \frac{C_{PET_k}}{U_s} + \frac{C_{WCET_k} - C_{PET_k}}{U_s} \\ &= max(r_k, d_{k-1}) + \frac{C_{WCET_k}}{U_s} \quad = \quad d_k \end{aligned}$$

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

3.3 Example of Adaptive Total Bandwidth Server

In this section, an example of Adaptive TBS is shown. In Figure 2, (1) and (2) show scheduling results of the original and adaptive TBS, respectively. There are two periodic tasks, τ_1 and τ_2, and an aperiodic task request. The period of τ_1 is $T_1 = 4$, and its execution time $C_1 = 1$. τ_2 has the period $T_2 = 6$, and its execution time $C_2 = 3$. Therefore, the CPU utilization by the two periodic tasks is $U_p = 0.25 + 0.5 = 0.75$ and the CPU utilization by the aperiodic server is $U_s = 1 - U_p = 0.25$. The aperiodic request occurs at tick 3, and its WCET 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 $d_{WCET} = 3 + 3/0.25 = 15$. Based on EDF algorithm, the aperiodic task starts execution at tick 5, and is suspended at tick 6 by τ_2. Then, after the execution of τ_1, the execution resumes at tick 10 and finishes at tick 11. Consequently, the response time becomes $11 - 3 = 8$. On the other hand, in the adaptive TBS, the two deadlines, $d_{PET} = 3 + 2/0.25 = 11$ and $d_{REST} = 11 + (3 - 2)/0.25 = 15$, are given. Based on EDF, the aperiodic task starts execution at tick 3 and finishes at tick 7, which gives the response time of $7 - 3 = 4$. 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. The adaptive TBS suspends the aperiodic execution at tick 7, resumes the execution at tick 11, and then finishes it at tick 12. Like this, even if the execution time is incorrectly predicted, the response time would be the same as or shorter than that in the original TBS.

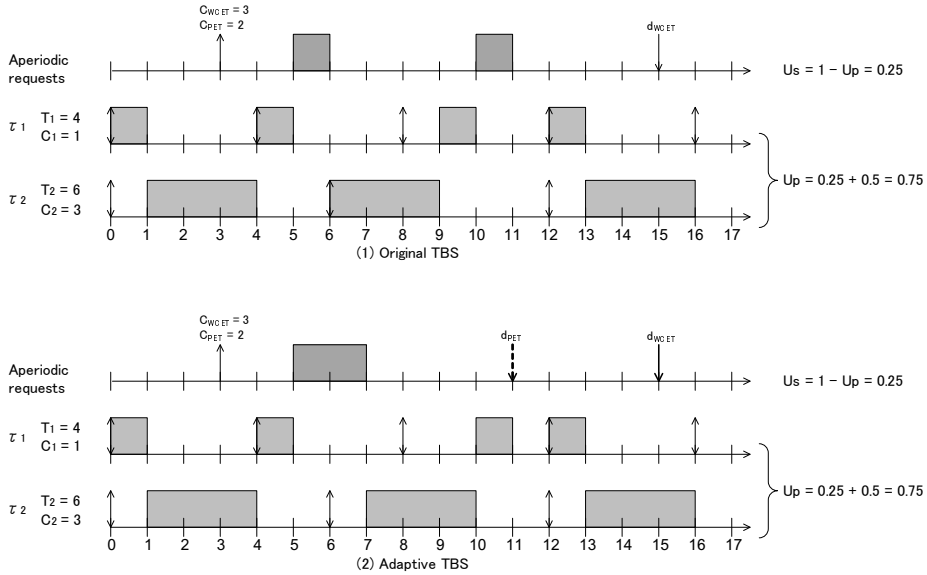

Fig. 2. Example of original and adaptive TBS.

3.4 Adaptive Total Bandwidth Schedulability

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

$$U_{J_{PET_k}} = \frac{C_{PET_k}}{d_{PET_k} - max(r_k, d_{k-1})} = U_s, \quad U_{J_{REST_k}} = \frac{C_{REST_k}}{d_{REST_k} - d_{PET_k}} = U_s$$

Therefore, schedulability of the adaptive TBS leads to be the same as that of the original TBS presented in the literature [9].

3.5 Implementation Complexity

In the proposed algorithm, task execution is divided into two instances. However, operating systems should manage a task with a single information set, task control block. This is realized by re-setting up deadline and re-inserting the task in a ready queue when PET elapses and the task has not finished, which is the only difference from the original TBS. To find that 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 the section 3.2, the value of the second term in the right side of the formula (8) should be statically computed and used when necessary to reduce the recalculation overheads.

3.6 Affinity with Resource Reclaiming

In the proposed adaptive TBS, when deadline is calculated for kth aperiodic request, d_{k-1} is needed. Since the previous ($k-1$th) aperiodic request is divided into two instances, the deadline for the second instance, that is $d_{REST_{k-1}}$, is used for the calculation. However, when the execution of the $k-1$th request finishes in its PET ($C_{PET_{k-1}}$), the second instance is not executed. In this case, instead of $d_{REST_{k-1}}$, $d_{PET_{k-1}}$ can be used to calculate the deadline for the kth task. This can be applied when the execution of the first instance of the $k-1$th aperiodic task finishes before the kth aperiodic task arrives. This is one of resource reclaiming methods.

A greedier method, resource reclaiming technique [10] described in the section 2, can be easily applied to the proposed adaptive TBS. When the execution of an aperiodic instance finishes, whether or not the execution is for the first or second instance after the task is divided, the deadline is recalculated by the formula (4), then the deadline is applied to the formula (3), and finally the following aperiodic instances can be given earlier deadlines by the formula (2).

In this paper, the former is called "simple resource reclaiming" and the latter "greedier resource reclaiming". In the evaluation section, these two resource reclaiming methods are combined with the proposed adaptive TBS.

4 Evaluation

4.1 Evaluation methodology

In this section, simulation results of the proposed TBS are shown. In the evaluation, six methods, Original TBS, the original TBS with resource reclaiming described in the section 2 (Original TBS-95), the adaptive TBS without any resource reclaiming (ATBS w/o RR), the adaptive TBS with simple resource reclaiming described in the section 3.6 (ATBS w/ RR), the adaptive TBS with greedier resource reclaiming (ATBS-95), and the ideal TBS where execution times of instances are known (Oracle), are compared.

In the simulation, task sets consist of periodic tasks with the total CPU utilization (U_p) from 60% to 90% at intervals of 5% and aperiodic tasks with the total utilization from 0.5% to 2% in the observation period (100,000 ticks). The aperiodic server has 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 WCET and actual execution times are equal and obtained by exponential distributions with the average of 10 ticks. For aperiodic tasks, a task set is supposed to contain 1 to 4 different tasks. Each task in a set is invoked multiple times and the arrival times are decided by Poisson distributions 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 WCET was about 0.33.

For each U_p, all combinations of ten periodic task sets and ten aperiodic task sets (total 100 task sets) are simulated and the average value is shown. For the proposed methods, the weighting coefficient for PET calculation (α in the section 3.1) is 0.5.

4.2 Results

Figure 3 is the results where each task set contains only one aperiodic task. The utilization by the aperiodic task's execution is about 0.5%. In the figure, the horizontal axis indicates the CPU utilization by periodic tasks (U_p), and the vertical axis indicates the average response time of aperiodic task executions. Under 65%, the response times are almost the same for all the six methods. This is because the server utilization, $U_s = 1 - U_p$, is large enough to quickly serve the aperiodic requests. Over 70%, the differences gradually appear. When U_p is 90%, the response time of the original TBS is about 32 ticks, and that of the original TBS-95 is 28.5 ticks. On the other hand, ATBS w/o RR, ATBS w/ RR, and ATBS-95 exhibit the response times of 20.5, 18, and 17.5 ticks, respectively. In this evaluation, it is found that a method with deadline assignment based on PET (ATBS w/o RR) improves the response time by 36% compared to the original WCET-based method, and that a method with greedier resource reclaiming (ATBS-95) outperforms the original ATBS-95 by 39%. Consequently, the PET-based method exhibits better ability when it is applied with resource reclaiming.

Figure 4 is the results where four aperiodic tasks are included in the task sets. The utilization by the aperiodic tasks is about 2%. The trend is similar to Figure 3 except that the improvement by resource reclaiming is larger. This is because the higher aperiodic utilization leads to the situation where occurrences of aperiodic requests overlap with each other and the resource reclaiming is applied more frequently. The improvement of ATBS w/o RR to Original TBS is 13%, while ATBS-95 achieves more improvement over the original TBS-95, which is 22%.

The use of PET is discussed. The ratio of aperiodic executions that finished in PET was 56%. Table 1 is the average of the shortened deadline length for aperiodic instances that finished in their PET in the simulation of Figure 4. ("Shortened length" means how shorter the deadline is than that based on WCET.) The difference between ATBS w/o RR, ATBS w/ RR, and ATBS-95 does not exist, and therefore the table shows the ratio collectively. 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 2nd term of the right expression in the formula (1) would be larger and then the shortened length would be longer. Consequently, ATBS methods using PET provide larger improvements when the 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 actuallxy affected the deadline calculation of the succeeding tasks (that is, ratios of the cases that d_{k-1} is the maximum in the

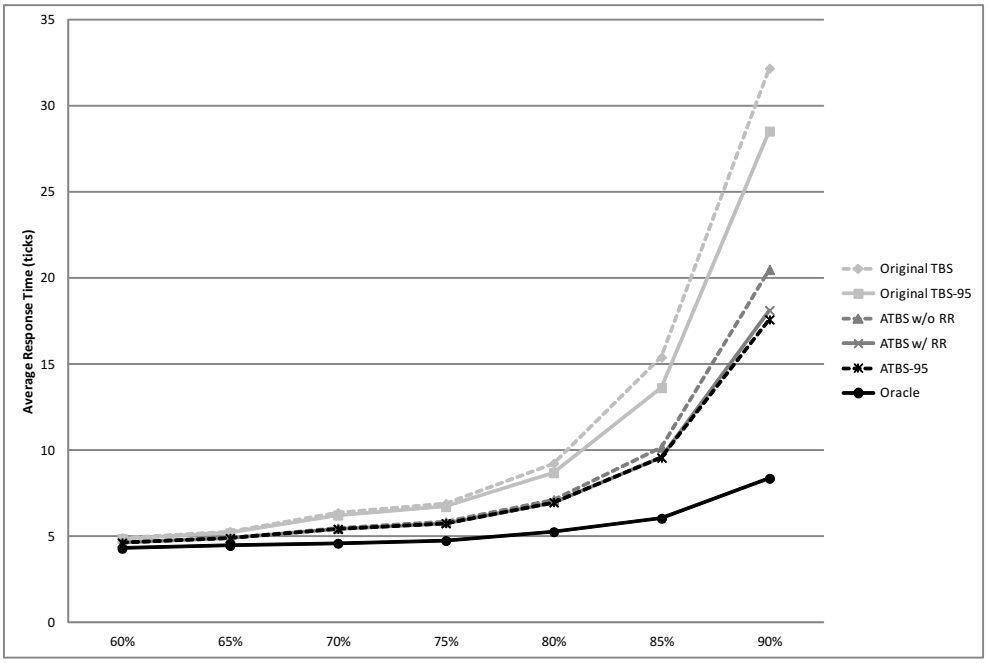

Fig. 3. Average response time (One aperiodic task).

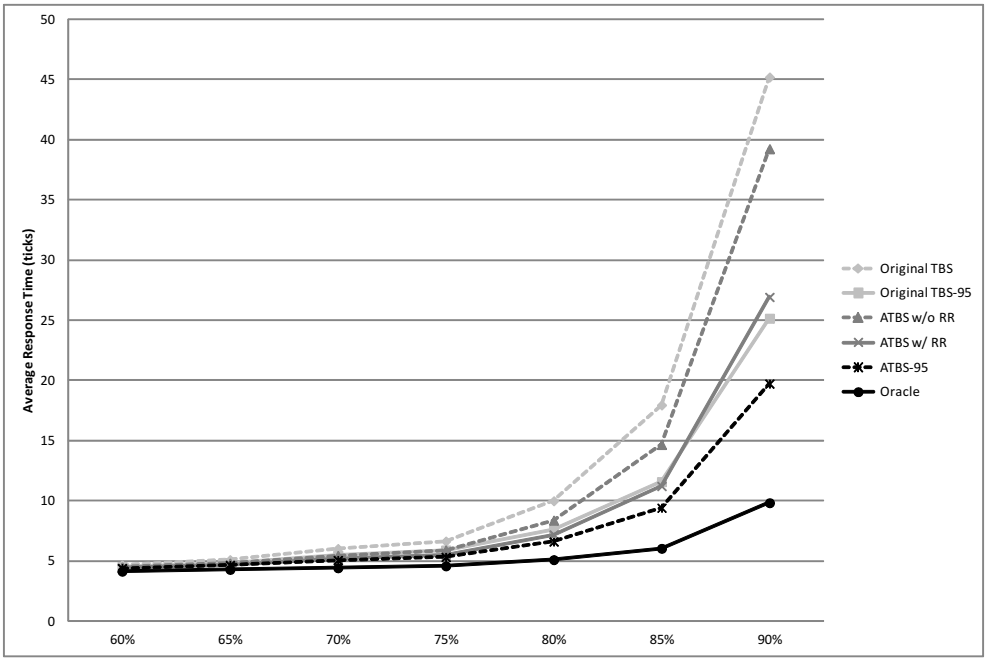

Fig. 4. Average response time (Four aperiodic tasks).

Table 1. Shortened deadline length (Four aperiodic tasks).

U_p (%)	60	65	70	75	80	85	90
Shortened length	19.7	22.8	26.5	31.8	40.0	54.1	84.6

formula (1) before resource reclaiming). In addition, it shows average shortened deadline lengths in parentheses. From the table, when U_p is larger, more and longer resource reclaiming is performed. For ATBS, greedier resource reclaiming (ATBS-95) provides more frequent and longer resource reclaiming than the simple resource reclaiming (ATBS w/ RR).

Table 2. Affected resource reclaiming ratio (%) (Four aperiodic tasks).

U_p (%)	TBS–95	ATBS w/ RR	ATBS–95
60	13.7 (20.1)	8.6 (18.1)	12.5 (19.7)
65	16.0 (23.2)	10.4 (20.8)	14.8 (22.8)
70	18.8 (28.2)	12.6 (25.3)	17.6 (27.7)
75	22.7 (36.8)	15.7 (32.6)	21.4 (36.3)
80	28.8 (52.7)	21.1 (44.5)	27.4 (51.6)
85	38.3 (85.5)	29.1 (68.9)	36.6 (83.3)
90	57.9 (299.1)	48.6 (252.9)	56.0 (297.4)

5 Conclusion

In this paper, for Total Bandwidth Server which is task scheduling algorithm for task sets consisting of periodic tasks with hard deadlines and aperiodic tasks without deadlines, the method that uses predictive execution times (PET) instead of worst-case execution times for deadline calculation of aperiodic instances is proposed. The use of PET 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 resource reclaiming techniques to further reduce response times.

From the evaluation by simulation, it was confirmed that the use of PET can shorten response times of aperiodic executions and that resource reclaiming can provide further improvements.

Currently, obtaining PET is simply based on the weighted average of the previous execution time and the previous PET. Better calculation methods of PET need to be explored. In addition, in this paper, an aperiodic task execution is divided into two instances. There is a choice that it is divided into three or

more instances and stepped deadlines are assigned to them. This choice is worth evaluating.

The evaluation in this paper used task sets that were generated based on probability distribution. To reflect actual situations where task execution times fluctuate, evaluation with actual program codes is desired. In addition, in the current evaluation, the greedier resource reclaiming exhibits better improvement than the simple resource reclaiming. However, considering the calculation overheads of reclaiming, the effects might be degraded. In such cases, the practicality of the simple resource reclaiming might emerge. In the future, evaluation with actual program codes and scheduling overheads should be performed.

References

1. Spuri, M., Buttazzo, G. C.: Efficient Aperiodic Service under Earliest Deadline First Scheduling. In: IEEE Real-Time Systems Symposium, pp.2–11, IEEE Computer Society, San Juan (1994)
2. Lundqvist, T., Stenström, P.: Timing Anomalies in Dynamically Scheduled Microprocessors. In: IEEE Real-Time Systems Symposium, pp.12–21, IEEE Computer Society, Phoenix (1999)
3. 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., Stenström, P.: The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools. ACM Trans. on Embedded Computing Systems, Vol.7, No.3, pp.1–53 (2008)
4. Liu, C. L., 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)
5. Lehoczky, J. P., Sha, L., Strosnider, J. K.: Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. In: IEEE Real-Time Systems Symposium, pp.261–270, IEEE Computer Society, San Jose (1987)
6. Sprunt, B., Sha, L., Lehoczky, J.: Aperiodic Task Scheduling for Hard-Real-Time Systems. Journal of Real-Time Systems, Vol.1, No.1, pp.27–60 (1989)
7. Lehoczky, J. P., Ramos-Thue, S.: An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems. In: IEEE Real-Time Systems Symposium, pp.110–123, IEEE Computer Society, Vienna (1992)
8. Abeni, L., Buttazzo, G.: Integrating Multimedia Applications in Hard Real-Time Systems. In: IEEE Real-Time Systems Symposium, pp.4–13, IEEE Computer Society, Madrid (1998)
9. Spuri, M., Buttazzo, G.: Scheduling Aperiodic Tasks in Dynamic Priority Systems. Journal of Real-Time Systems, Vol.10, No.2, pp.179–210 (1996)
10. Spuri, M., Buttazzo, G., Sensini, F.: Robust Aperiodic Scheduling under Dynamic Priority Systems. In: IEEE Real-Time Systems Symposium, pp.210–219, IEEE Computer Society, Pisa (1995)
11. Buttazzo, G. C., Caccamo, M.: Minimizing Aperiodic Response Times in a Firm Real-Time Environment. IEEE Trans. on Software Engineering, Vol.25, No.1, pp.22–32 (1999)