Virtual Release Advancing for Earlier Deadlines

Kiyofumi Tanaka
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
Email: kiyofumi@jaist.ac.jp

***Abstract*—Giving tasks more urgent deadlines leads to shorter response times in EDF-based scheduling. This paper describes a technique that gives earlier deadlines by virtually advancing release times in the Total Bandwidth Server (TBS) context. This technique improves aperiodic tasks' responsiveness while it guarantees the schedulability of the system. Simulations show the technique outperforms TBS and its enhanced one.**

I. Introduction

In real-time systems, not only satisfying deadline requirements but keeping response times and jitters short is important. The author has been trying to achieve this goal with techniques, the adaptive total bandwidth server (Adaptive TBS) [1] and the adaptive EDF [2]. These techniques are based on the total bandwidth server (TBS) [3], but utilize estimated execution times instead of worst-case execution times (WCET) to calculate deadlines adaptively. Estimated execution times shorter than WCETs lead to earlier deadlines following the TBS formula and earlier scheduling based on EDF [4] algorithm.

In this paper, another technique for adaptively obtaining earlier deadlines is proposed. The technique, *virtual release advancing*, introduces virtual release time and uses the virtual release time rather than task's actual release/invocation time in deadline calculation to advance the corresponding deadline. The point resides in an assumption that the task had been invoked at the (earlier) virtual release time. Since TBS calculates an absolute deadline originating a release time, the earlier (virtual) release time leads to a more urgent deadline. This technique advances virtual release times without influencing the past schedules, guaranteeing schedulability of task sets.

II. Related Work

The virtual release advancing technique accompanies TBS (or Adaptive TBS/EDF). This section describes the basics of TBS and its improved method.

TBS is one of the resource reservation methods and is a scheduling algorithm for a mixture of hard periodic and non real-time aperiodic tasks. It provides fair response times for aperiodic tasks while keeping the schedules feasible and its implementation complexity moderate [3]. It is assumed that hard periodic tasks are invoked periodically and have relative deadlines equal to their periods, and that non real-time tasks are requested irregularly without explicit deadlines. When an aperiodic task is invoked, a tentative absolute deadline is calculated and assigned to the job as:

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

where k means the k-th aperiodic job, r_k is the arrival (release) time of the k-th job, d_{k-1} is the absolute deadline for the k-1-th (previous) job, C_k is WCET of the k-th job, and U_s is the bandwidth or the processor utilization factor of the server. The server is able to occupy U_s in the total processor utilization and, every time an aperiodic request arrives, it gives the job the bandwidth equal to U_s. The term, $max(r_k, d_{k-1})$, prevents bandwidths given to successive aperiodic jobs from overlapping with each other. After a job is given the deadline by Formula 1, all periodic and aperiodic jobs are scheduled by EDF. Letting U_p be the total processor utilization by all hard periodic tasks, it was proved that a task set is schedulable if and only if $U_p + U_s \leq 1$ [3].

In TBS following Formula 1, an overestimated WCET (C_k) would cause the deadline to be later than necessary. According to the EDF policy, this might delay the execution order of the job and lead to a long response time. Worse yet, the delayed deadline can influence the following aperiodic jobs by the term, $max(r_k, d_{k-1})$, in Formula 1 (from k-1-th to k-th). The literature [5] proposed a kind of slack reclaiming method, *resource reclaiming*, where the deadline is recalculated by using the actually elapsed execution time when the job finishes, and the new deadline is used for the deadline calculation for the subsequent aperiodic job. By this method, the subsequent jobs benefit from the recalculated deadline of the previous job and their response times would be improved.

In the resource reclaiming, k-th aperiodic job is given the 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 for the previous job ($\overline{d}_{k-1}$, described below), and the finishing time of the previous job (f_{k-1}) is selected as the originating time. When the k-1-th aperiodic job finishes, the deadline is recalculated by the following formula (Formula 4) that includes the actual execution time, $\overline{C}_{k-1}$, of the job, and is reflected in Formula 3, and then in Formula 2, for the subsequent k-th job's deadline.

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

In the literature [6], a technique was proposed to enhance TBS where the deadline initially given by TBS is iteratively fit to a planed finishing time. This is based on the EDF optimality that a task set is scheduled feasibly by EDF if any other

The paper was presented at APRES 2015.

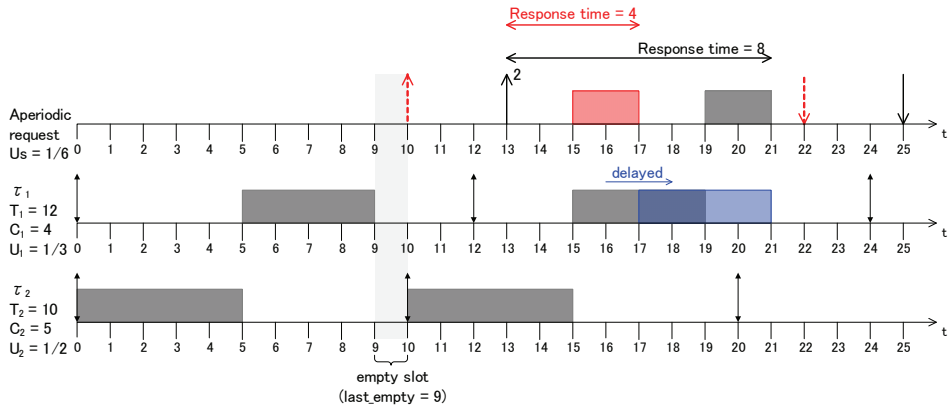

Fig. 1. Example of release advancing with the limit of empty slot.

algorithm leaves the task set feasible [7]. If the algorithm repeats the deadline fitting until the deadline converges (this is called TB*), the optimality for response times is guaranteed for aperiodic tasks which spend their WCETs every time. Since this technique forces considerable complexity for calculating a finishing time repeatedly, it is recommended that the number of iterations is limited to some upper bound. (This is called TB(n) where n is the upper bound.)

III. Virtual Release Advancing

In TBS, a deadline is calculated by basically originating in a release time of an aperiodic request. This means that an earlier release time leads to a more urgent deadline and higher possibility of being scheduled earlier. The virtual release advancing is a technique for obtaining an earlier deadline by virtually and retroactively moving a release time back to the past while not changing the past schedule. This is possible when other jobs with even earlier deadlines were scheduled and executed before the actual release time of the target job. In this case, even if the job had been released at an earlier time, the scheduling results before the actual release time should be the same. This concept can make the corresponding deadline earlier according to the virtually advanced release time without affecting the past schedules.

A. Example of virtual release advancing

In Figure 1, there are two periodic tasks, τ_1 and τ_2. τ_1 has a period of $T_1 = 12$ and execution time of $C_1 = 4$. τ_2 has $T_2 = 10$ and $C_2 = 5$. The utilization by the periodic tasks is $U_p = 4/12 + 5/10 = 5/6$, which leaves the TBS bandwidth of $U_s = 1 - 5/6 = 1/6$. An aperiodic request occurs at $t = 13$ and its execution time is to be two. Serving this request by TBS has finishing time of $t = 21$ and response time of eight. (Gray instances are scheduling results by TBS.)

Suppose that the release time had been at $t = 10$ as indicated by a red, upward dashed arrow. Then, the deadline calculation in TBS would give a deadline at $t = 22$ as a red, downward dashed arrow. According to this deadline, the execution (the red instance) finishes at $t = 17$ and the response time becomes four. Note that the schedule before the real release time should not be affected as a result of the virtual release advancing. Therefore, the updated deadline cannot be earlier than any deadlines of the instances executed before the real release time. In this example, the second instance of τ_2 has a deadline of $t = 20$ and has already started execution before the release time of the aperiodic request. Thus, the updated deadline must be no earlier than $t = 20$. Otherwise, the past schedule would have been different.

B. Definition of virtual release advancing

How long to the past the release time can go back is defined. There are three limits of advancing: previous deadline, empty slot, and previously-used maximum deadline.

1) Previous deadline: Moving a release time backward over the deadline of the previous aperiodic job is of no effect since TBS chooses a larger value between the release time and the previous deadline in Formula 1.

2) Empty slot: An empty slot is a time slot during which no task is executed. In Figure 1, the slot 9 (which is duration between 9 and 10) is an empty slot. If release time is moved to the past over an empty slot, the empty slot must have been spent by the job. This leads to changing the past schedules. Therefore, an empty slot becomes the second limit. To recognize this limitation, the system needs to record the last empty slot number.

3) Previously-used maximum deadline: Slots that were not empty must have been spent by tasks' execution. Each slot can be associated with a deadline of a job executed during it. Let's consider whether a release time of a job can be advanced over some specific past slot. If a deadline calculated with the advanced release time gets earlier than the associated deadline of the slot, the slot must have been spent for the job. 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 any associated deadlines of the slots that are overtaken. To achieve this limit, it is necessary to record associated deadlines for the past slots and manage a maximum value among them in the process of advancing.

Figure 2 shows an example of the previously-used maximum deadline. The release time of an aperiodic job with execution time of one is 5 and the deadline given by TBS is 11. The slot 4 was occupied by τ_2's second instance and its associated deadline is 8 (as the "dl[]" value in the figure). This value is recorded as the maximum deadline ("max_dl"). Since

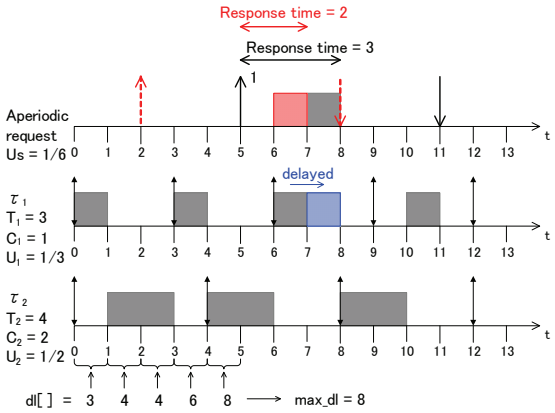

Fig. 2. Example of release advancing with the limit of previously-used maximum deadline.

11 is greater than 8, the release time can be advanced by one. This is repeated until the virtual release time and the deadline reach 2 and 8, respectively. Further advancing would generate the deadline earlier than the maximum deadline. Therefore, this advancing is not allowed and the advancing process finishes. As a result, the response time is shortened by one compared to the schedule without the virtual release advancing.

C. Algorithm for virtual release advancing

Algorithm 1 shows the algorithm of the virtual release advancing. In the algorithm, r_k and vr_k are the real and virtual release times of the k-th aperiodic job, respectively, d_k and C_k are the deadline and execution time of the job, d_{k-1} is the deadline of the k-1-th (previous) aperiodic job, and U_s is the bandwidth of TBS. For other variables, $last\_empty$ is the slot number of the last empty slot, and dl is the array of elements each of which ($dl[i]$) contains an associated deadline of the i-th slot. In the process of advancing to the past, "max_dl" holds the maximum value among traced dl elements.

The lines 1 and 2 initialize max_dl and vr_k, respectively. The lines 5 to 9 correspond to the limit of the previous deadline. When the virtual release time is earlier than or equal to d_{k-1}, the virtual release time is set to d_{k-1}, the corresponding deadline is calculated, and the algorithm finishes. If the first condition is passed, the deadline is calculated in the line 10. Then, in the lines 12 to 14, the second limit is checked where the virtual release time is compared to the end of the last empty slot. If the condition is met, the advancing algorithm finishes.

Next, in the lines 15 to 17, max_dl is compared to the dl element of the previous slot of the current virtual release time. If the former is smaller, it is updated. Then, in the lines 19 to 20, the third limit is checked. That is, if max_dl is larger than or equal to the current deadline calculated, the advancing algorithm closes. After all the three limits are passed, the virtual release time is advanced to the past by one slot in the lines 21 to 24, and then the control goes to the next iteration.

D. Schedulability

The virtual release advancing technique does not change the past schedules at all. It only assumes that a job had been released at the virtual release time, vr_k. This technique

Algorithm 1 Virtual Release Advancing

```
1:  max_dl ⇐ 0 /* Initialization */
2:  vr_k ⇐ r_k
3:  while TRUE do
4:    /* Reaching the limit of d_{k-1} */
5:    if vr_k ≤ d_{k-1} then
6:      vr_k ⇐ d_{k-1}
7:      d_k ⇐ vr_k + C_k/U_s
8:      break
9:    end if
10:   d_k ⇐ vr_k + C_k/U_s
11:   /* Reaching the limit of empty slot */
12:   if vr_k = last_empty + 1 then
13:     break
14:   end if
15:   if max_dl < dl[vr_k − 1] then
16:     max_dl ⇐ dl[vr_k − 1]
17:   end if
18:   /* Reaching the limit of prev-used max deadline */
19:   if d_k ≤ max_dl then
20:     break
21:   else
22:     /* Release advancing by one time slot */
23:     vr_k ⇐ vr_k − 1
24:   end if
25: end while
```

generates the same schedule as that in TBS where the request actually happens at vr_k. Obviously, the schedule with the virtual release advancing is feasible since TBS can provide a feasible schedule for the same (virtual) situation.

Note that the virtual release advancing never leads to the result where the calculated deadline gets earlier than the time of $r_k + C_k^{WCET}$. Such excessive advancing implies a possibility of a deadline miss of the job. The advancing algorithm does not affect the schedule in the time interval between vr_k and r_k. Therefore, if it were assumed that this kind of miss happened, the original TBS would not be able to feasibly schedule the same situation with vr_k. This is a contradiction to the schedulability of TBS. Therefore, the virtual release advancing is safe.

E. Implementation complexity and runtime overhead

In Algorithm 1, division of C_k/U_s is performed in the line 7 and 10. In practice, from a runtime overhead point of view, the division should be done only once before entering this algorithm. This is possible since the value does not change in the algorithm. In addition, while the memory storages and management overheads for $last\_empty$ and max_dl are negligible, the size of the array dl and how long past the advancing is performed to are subject to discussion. Therefore, the number of iterations in Algorithm 1 should be bounded if the arising overheads would be relatively large. This is discussed through simulations in Section IV.

IV. Evaluation

In this section, the proposed technique is compared with TBS and TB*/TB(n) through simulations. In the simulations, task sets generated by using probabilistic distributions are

used. Each task set consists 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). All the aperiodic servers are supposed to 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 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. In this task execution model, actual execution times of aperiodic jobs can be shorter than their WCETs. This situation does not guarantee the optimality of TB*. Each job 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 each U_p from 60% to 90%, 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.

The average response times in TBS, TBS with the virtual release advancing, and TB*/TB(n=2 or 3) are shown in Figure 3 in which the horizontal axis indicates U_p and the vertical axis indicates the average response time of aperiodic task executions normalized to the results of TBS. VRA* means that the virtual release advancing is performed without upper bounds for the loop count. VRA(n) has the upper bound of n.

In the figure, TB* shows the shortest average response times for all U_p. However, when the number of iterations is limited, TB(n) gets worse as U_p increases. For U_p=90%, TB(2) and TB(3) generate longer average response times than TBS. This is because the resource reclaiming technique described in Section II is not included in TB* and TB(n). When a task execution does not spend WCET, TB* is not necessarily optimal. VRA* and VRA(n) give similar average response times shorter than TBS. Compared to TBS, VRA* and VRA(80) reduce the average response time by 15.0% and 13.5%, respectively, for U_p=90%.

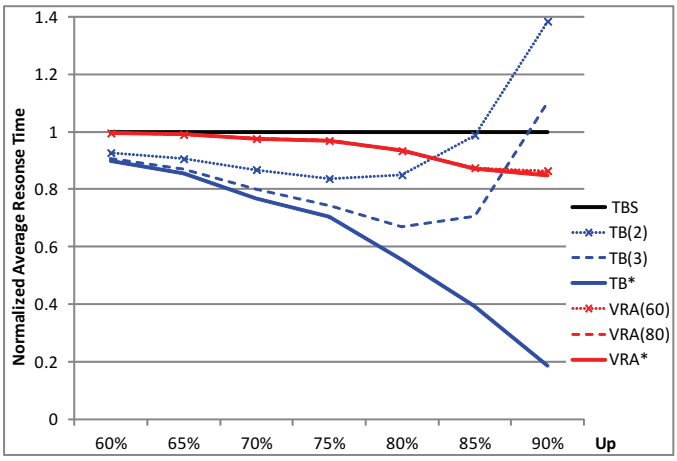

Fig. 3. Normalized average response times of aperiodic task execution.

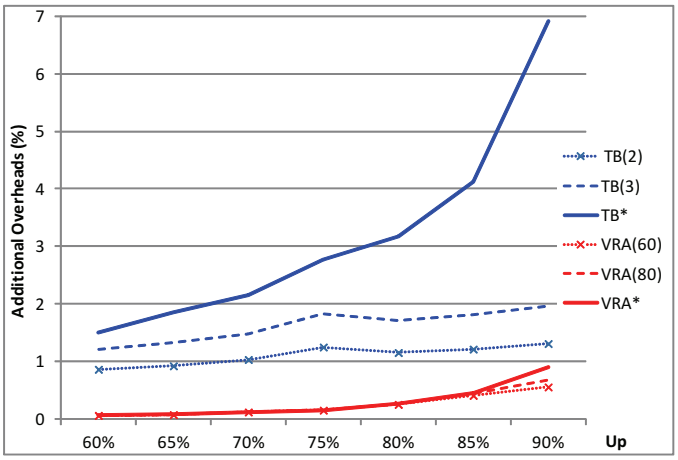

Fig. 4. Percentage of additional overheads.

Additional execution overheads for VRA*/VRA(n) and TB*/TB(n) are estimated from the information obtained in the simulations. (The number of iterations executed, etc.) According to Cortex-A9 [8], [9], arithmetic, logical, and control operations in the algorithms are assigned latencies and the percentages of the overheads in addition to TBS are calculated. It is assumed that a processor's clock frequency is 100MHz and the tick length is 1 msecond. Figure 4 shows the average values of the maximum overheads in a tick. It is clear that TB* generates much more overheads than the others. TB(2) and TB(3) mitigate the overheads but they are still larger than the VRAs. As a whole, the virtual release advancing exhibits shorter response times with smaller overheads when U_p is high.

V. Conclusion

In this paper, the virtual release advancing is proposed and evaluated. The technique adaptively advances deadlines and shortens response times of target tasks, while maintaining schedulability of hard periodic tasks. Simulations showed that, for tasks with varying execution times, the technique could outperform TB(n) when the system utilization is high.

References

[1] K. Tanaka, "Adaptive Total Bandwidth Server: Using Predictive Execution Time," Proc. of International Embedded Systems Symposium (IESS), Springer, pp.250–261, 2013.

[2] K. Tanaka, "Adaptive EDF: Using Predictive Execution Time," ACM SIGBED Reivew, Vol.10, No.4, pp.41-44, Dec, 2013.

[3] M. Spuri and G. C. Buttazzo, "Efficient Aperiodic Service under Earliest Deadline First Scheduling," Proc. of Real-Time Systems Symposium, pp. 2–11, December 1994.

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

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

[6] G .C. Buttazzo and F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Environments," IEEE Computers, Vol.48, No.10, pp.1035–1052 (1999).

[7] M. L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes," Information Processing, Vol.74, pp.807–813 (1974).

[8] "Cortex-A9 Technical Reference Manual," ARM.

[9] "Cortex-A9 Floating-Point Unit Technical Reference Manual," ARM.