ECTI Transactions on Computer and Information Technology
Journal homepage: https://ph01.tci-thaijo.org/index.php/ecticit/
Published by the ECTI Association, Thailand, ISSN: 2286-9131

A New Scheduling Algorithm for Shortening Response Time of Static Priority Task

Takaharu Suzuki[1] and Kiyofumi Tanaka[2]

ABSTRACT

In scheduling algorithms based on the Rate Monotonic (**RM**) method which are widely used in development of real-time systems, tasks with shorter periods have higher priorities. In contrast, ones with longer periods are likely to suffer from increased response times and jitter due to their lower priorities. We propose an Execution Right Delegation (**ERD**) method based on RM where a high-priority server for particular (or important) task is introduced to shorten response time and jitter of the task. In the evaluation, it is confirmed that response times and jitter of a particular (important) task are reduced. We also show Response Time Analysis (**RTA**), which assures worst-case response time of the task. This paper shows the algorithm and RTA of ERD and evaluates it by comparing it to a Deadline Monotonic method. The evaluation by simulation shows that ERD can reduce the average worst-case response time by 13.45% at maximum compared to the Deadline Monotonic scheduling. In addition, we confirm that the RTA provides a worst-case response time close to the simulation results.

Article information:

Keywords: Real-time scheduling, Rate Monotonic, Response Time Analysis

Article history:
Received: April 12, 2021
Revised: July 17, 2021
Accepted: September 4, 2021
Published: June 11, 2022 (Online)

DOI: 10.37936/ecti-cit.2022162.245352

1. INTRODUCTION

The main purpose of real-time scheduling algorithms is to achieve optimal scheduling for tasks which have periodic execution times and/or deadlines. It is important for the algorithms to not only meet the deadline of each task but also to shorten its response time and jitter. For periodic task sets, there are two categories in scheduling algorithms; one is Rate Monotonic (**RM**) for static priority and the other is Earliest Deadline First (**EDF**) for dynamic priority [1]. EDF has the advantage of high CPU utilization, while RM causes less runtime overhead and has predictable behavior. In addition, RM is easy to realize on static-priority-based real-time operating systems (RTOSs) such as ITRON [2], by associating tasks' priorities with their periods. As a result, scheduling algorithms based on the RM method are widely used in development of real-time systems.

In RM, tasks with shorter periods have higher priorities. In contrast, ones with longer periods are likely to suffer from increased response times and jitter due to their lower priorities. In this paper, we propose an Execution Right Delegation (**ERD**) method based on RM where a high-priority server for a particular (or important) task[3] is introduced to shorten response time and jitter. We also show Response Time Analysis (**RTA**), which gives the worst case response time (**WCRT**) of ERD.

This paper consists of six sections. Section 2 describes related work in terms of real-time scheduling algorithms and response time analysis. Section 3 proves several theorems and lemmas and shows the algorithm of our ERD method. Section 4 proposes RTA of ERD. An evaluation of the proposed method is shown in Section 5. Finally, Section 6 concludes the paper.

2. RELATED WORK

2.1 Scheduling Algorithms

In the RM model, task $\tau_{i,(1\le i\le n)}$ releases an infinite sequence of Jobs. Once a Job is released, it runs during the defined time C_i. A Job is released ev-

[1,2]The authors are with Japan Advanced Institute of Science and Technology, Tokyo/Ishikawa, Japan, Email: hal-suzuki@jaist.ac.jp and kiyofumi@jaist.ac.jp

[3]We assume the particular task has relatively lower priority due to its longer period. For example, for CAN messages in integrated ECUs, control messages have a shorter period, but notification messages have a longer period. However, some notification messages are urgent and must receive a higher priority (e.g. warning lamp, low fuel).

ery period T_i. Notation of a task is $\tau_i = (C_i, T_i)$. A set of tasks is denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, where the smaller the subscript figure a task has, the shorter the period and the higher priority it has (i.e. $T_1 \leq T_2 \leq ... \leq T_n$). A deadline miss occurs if a Job does not finish by the next task release.

The Deadline Monotonic (**DM**) [3] model includes relative deadline D_i in addition to period T_i and execution time C_i. It is another scheduling algorithm. The shorter the relative deadline of a task is, the higher its priority is. As priority is determined by deadline independent of a period, the response time and jitter of a task with a longer period can be shortened by assigning a short deadline.

There are several fixed-priority server algorithms for shortening response times of particular aperiodic tasks. Deferrable Server (**DS**) [4] and Priority Exchange (**PE**) [4] are representatives. Servers in these algorithms have a period and a capacity which corresponds to execution time. The servers are scheduled along with a set of periodic tasks while their capacity is consumed for execution of aperiodic tasks. In these algorithms, the server period and capacity are decided according to processor utilization of the whole periodic task set, where the importance of a particular periodic task is not considered. On the other hand, the method proposed in this paper obtains and uses the period and the capacity in favor of a particular (important) periodic task, although the server algorithm is basically PE.

Extended Priority Exchange (**EPE**) [5] does not involve a server. Instead, execution time of each task is overestimated. The unused part of the overestimation is exploited as capacity for aperiodic tasks. It can be regarded as padding a bandwidth for aperiodic tasks into the other tasks' execution time. Dual Priority (**DP**) [6] is another approach. In contrast to RM, this algorithm assumes that each task is assigned two priorities. A periodic task has two priorities, low and high, and an aperiodic task has middle priority. After a periodic task is released, it starts to run at the low priority for a while. The aim of DP is to treat aperiodic tasks preferentially by delaying periodic tasks as late as possible while meeting all the deadlines. Zero Slack Scheduling [7] is for a model of mixed-criticality systems. In this model, one system contains tasks with different degrees of importance. A task has criticality as well as a conventional priority. A task runs in a normal mode unless it is urgent. However, it switches to a critical mode at some point to keep schedulability.

2.2 Priority Assignment

The aim of ERD is to shorten response time of an important task regardless of its period. In order to assign priority to the task regardless of the period, Leung and Whitehead generalized that DM priority assignment is optimal for synchronous periodic task sets (without phases) with constrained deadlines [8]. In some different system models (e.g., tasks with phases, tasks with arbitrary deadlines, non-preemptive scheduling), it is known that DM is not optimal [9].

2.3 Response Time Analysis

There is a classical method to judge whether a task set is schedulable or not. Liu et al. [1] proposed a schedulability test by calculating processor utilization by all tasks and comparing it with the least upper bound for the task set. This test provides only a sufficient condition, which leaves the estimate pessimistic. Bini et al. proposed a polynomial-time and less-pessimistic way [10].

Response Time Analysis (**RTA**) [11] provides a necessary and sufficient condition of schedulability for RM, which is defined as follows:

Definition 1 (Response Time Analysis (RTA) [11] of RM). *In fixed-priority scheduling, the longest response time*[4], *R_i, of task τ_i is computed as*[1]*:*

$$R_i \;=\; C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil C_j. \qquad (1)$$

R_i appears on both sides. Thus, R_i is calculated by setting an appropriate value (e.g. C_i) as the initial value to R_i and increasing R_i until both sides become equal.

Several RTA methods for multiprocessor system are proposed. Bertogna et al. proposed RTA for global fixed-priority multiprocessor scheduling [13]. Compared with single processor RTA, multiprocessor RTA provides only a sufficient condition with its pessimism. In the method, interference time, which is a total amount of time that higher priority tasks prevent a certain task from working, is introduced. Guan et al. improved Bertogna's method by limiting carry-in load, a part of interference time, of higher priority tasks (**RTALC**) [14]. Sun et al. revised RTA_LC to reduce its pessimism. The method provides a more accurate result [15]. They applied schedulability test for a multiprocessor partitioned schedule by the RTA [16]. Zhou et al. showed RTA for a model where a task has a non-preemptive point under multiprocessor global scheduling [17]. As mentioned above, there are various RTAs dedicated to specific models or different systems.

[4] If the first jobs of all tasks are released simultaneously at the instant $t = 0$, response time of each task becomes the worst-case for the corresponding task (Critical Instant [1]).
The smaller the subscript a task has, the shorter the period and the higher priority it has. That is, for τ_i and τ_j, if $i < j$, τ_i has higher priority than τ_j.

Coutinho et al. showed another approach for the model of asynchronous periodic and sporadic tasks for a single processor[5][12]. The approaches mentioned above are calculating maximum interference time from higher priority tasks under a pessimistic situation. Coutinho's approach is calculating interference time between the job's release time and the job's response time, in the same way as multiprocessor RTA. This checking is done until hyperperiod ($LCM(T_1, T_2, ..., T_{n-1})$). They reduce pessimism of the algorithm while calculation time easily explodes due to it's requirement of LCM of tasks' periods.

2.4 Reducing Jitter

In a real-time system, especially in closed-loop control systems which operate sampling/sensing and actuating, it is required to reduce not only response time but also jitter [18]. In order to reduce jitter, several methods were studied [19] [20] and evaluated [21]. In this paper, relative finishing jitter (RFJ) and absolute finishing jitter (AFJ) are evaluated by using the following definitions:

$$RFJ_i = \max_k |(f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1})| \quad (2)$$

$$AFJ_i = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k}). \quad (3)$$

where $r_{i,k}$ is the release time of τ_i's kth job and $f_{i,k}$ is the finishing time of τ_i's kth job.

3. EXECUTION RIGHT DELEGATION (ERD)

3.1 System Model and Theorems

The target system model of this study is single processor fixed task priority. ERD assumes that each task has periodic execution time, and that the deadline is equal to its period. A task does not have a phase, which means that the first job is released at $t = 0$. The scheduling rule follows RM except for a particular (target) task to which the proposed virtual server (**VS**) is applied.

First, we describe theorems and lemmas as well as definitions underlying ERD. Based on these, a virtual server VS is introduced into a task set.

Theorem 1 (Schedulability Analysis [11]). *For each task τ_i in a task set Γ, if R_i is equal to or less than T_i, Γ is schedulable.*

Definition 2 (Assigning Priority to a Task). *In the proposed method, in addition to the priority assignment of RM, a particular (e.g. important) task can be assigned priority independent of its period.*

Lemma 1 (Exchanging Priorities between Tasks). *Assume that a schedulable task set Γ includes τ_p and τ_h and that the priority of τ_h is higher by one than that of τ_p. Let Γ' be a task set in which the priorities of τ_p and τ_h are exchanged while the other tasks have the same priority as in Γ. It is schedulable if R_p is equal to or less than T_h.*

Proof. From Theorem 1, schedulability for Γ' is guaranteed by confirming that every task meets its deadline ($R'_i \leq T_i$). (From this point forward, symbols with a prime mark, like R'_i, denote tasks in Γ'.)

First, Γ is divided into three sets, Γ_{high}, $\{\tau_h, \tau_p\}$, and Γ_{low}, so that priorities of tasks in Γ_{high} are higher than that of τ_h and those in Γ_{low} are lower than that of τ_p.

Response time of any task, τ_{high}, in Γ_{high} is not affected by exchanging the priorities of τ_h and τ_p since the priority exchange between tasks with lower priorities than τ_{high} can change scheduling only after completion of τ_{high}'s first job. Similarly, for any task, τ_{low}, in Γ_{low}, its response time is not affected, since the instant when the right of execution is given to τ_{low} is not changed by exchanging the execution order of higher tasks.

The response time of τ_p does not increase since its priority is higher than it was before the exchange. Since $R_p \leq T_h$, $T_h \leq T_p$, and $R'_p < R_p$, $R'_p \leq T_p$, the deadline constraint is met.

Next, consider τ_h in two cases; τ_p or τ_h is blocked by jobs in Γ_{high}, or they are not blocked. For the not-blocked case, let s_h and s_p be start times of Jobs of τ_h and τ_p (before changing priority), respectively. Then the response time of each task is given as:

$$R_h = s_h + C_h, \quad R_p = s_p + C_p.$$

Letting s'_p and s'_h be the start times of Jobs of τ_p and τ_h, respectively. After the priority exchange, their response times, R'_p and R'_h, become:

$$R'_p = s'_p + C_p, \quad R'_h = s'_h + C_h.$$

s_p is equal to R_h ($s_p = R_h$) since neither τ_h nor τ_p is blocked by Jobs in Γ_{high} and Job of τ_p starts immediately when τ_h's Job finishes. This is the case after the priority exchange, that is, $s'_h = R'_p$. In addition, obviously, s'_p is the same as s_h. Then, R'_h becomes equal to R_p as follows:

[5]If release time/offset/phase is introduced to the model, WCRT is unknown as is the case in multiprocessor WCRT.

$$\begin{aligned} R'_h &= s'_h + C_h \\ &= R'_p + C_h \\ &= s'_p + C_p + C_h \\ &= s_h + C_p + C_h \\ &= R_h + C_p \\ &= s_p + C_p \\ &= R_p \end{aligned}$$

From the prerequisite $R_p \leq T_h$, it turns out to be $R'_h \leq T_h$. Thus, the deadline requirement is satisfied. **Fig 1** and **Fig 2** show examples of the case in which tasks in Γ_{high} do not block τ_h or τ_p.

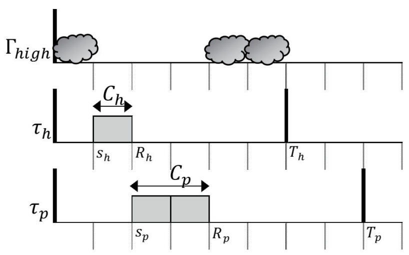

Fig.1: *Not blocked by Γ_{high} (before exchanging priorities).*

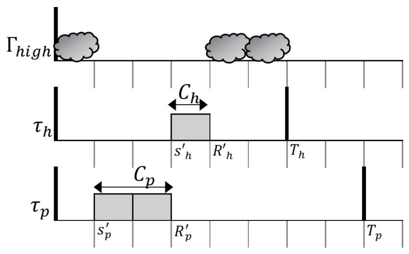

Fig.2: *Not blocked by Γ_{high} (after exchanging priorities).*

Next, consider the other case in which τ_p or τ_h is blocked by new Jobs of tasks in Γ_{high} which are released before T_h. From the prerequisite that τ_h and τ_p finish their jobs by the instant T_h, BT, blocking time of τ_h's and τ_p' s job after s_h, has the following upper bound:

$$BT \leq T_h - s_h - (C_h + C_p) \tag{4}$$

R_p is given as:

$$R_p = s_h + C_h + C_p + BT.$$

After exchanging priorities, R'_h is:

$$R'_h = s'_p + C_p + C_h + BT.$$

With $s'_p = s_h$ and by moving BT to the left side:

$$BT = R'_h - s_h - C_p - C_h.$$

By expanding BT in (4):

$$\begin{aligned} R'_h - s_h - C_p - C_h &\leq T_h - s_h - (C_h + C_p) \\ R'_h &\leq T_h - s_h - (C_h + C_p) + \\ & \quad s_h + C_p + C_h \\ R'_h &\leq T_h. \end{aligned}$$

Thus, the deadline constraint is satisfied. □

Examples of this case are shown in **Fig 3** and **Fig 4**.

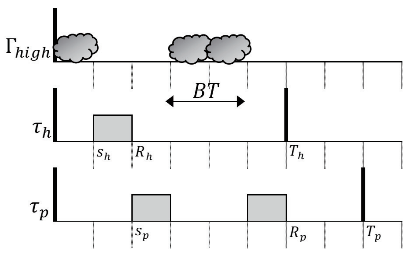

Fig.3: *Blocked by Γ_{hgih} (before exchanging priority).*

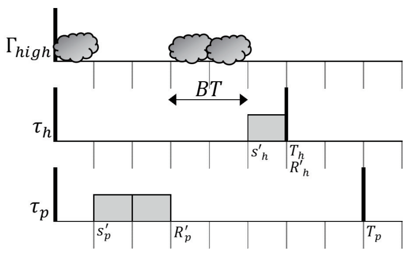

Fig.4: *Blocked by Γ_{hgih} (after exchanging priority).*

Theorem 2 (Changing Priorities of Tasks). *Assume that a schedulable task set Γ includes τ_p and τ_h and that the priority of τ_h is higher than that of τ_p. Let Γ' be a task set in which the priority of τ_p is changed to be higher than τ_h while the other tasks have the same priority as in Γ. It is schedulable if R_p is equal to or less than T_h*[6]

Proof. For periods of tasks, $\tau_h, \tau_{h+1}, \tau_{h+2}, ...,$ and τ_{p-1}, the following condition is satisfied:

$$T_h \leq T_{h+1} \leq T_{h+2} \leq ... \leq T_{p-1}$$

From the prerequisite, $R_p \leq T_h$, and the above condition, the following condition is satisfied.

[6] In Lemma 1, there are no other tasks between τ_p and τ_h. This theorem relaxes the restriction.

$$R_p \leq T_h \leq T_{h+1} \leq T_{h+2} \leq ... \leq T_{p-1}$$

Here, focusing on τ_{p-1} whose priority is higher by one than τ_p, a task set in which priorities of τ_p and τ_{p-1} are exchanged is schedulable from Lemma 1. In the same way, priorities of τ_p and τ_{p-2} are exchangeable. This is repeated until the priority exchange is done between τ_p and τ_h. □

Definition 3 (Idle Time) *Idle Time is a period in which any task's job is not executed. $idle(t)$ gives the total amount of idle time between 0 and the instant t.*

Example 1 (Idle Time). *In Fig*5, $idle(4) = 1$ and $idle(6) = 2$.

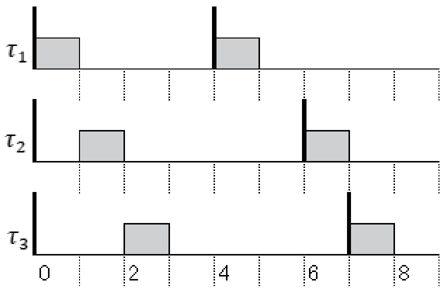

Fig.5: $idle(t)$.

Lemma 2 (Idle Time and Completion of Job). *The first jobs of all tasks in Γ finish by the instant t if* $idle(t) > 0$.

Proof. In fixed priority scheduling, the highest-priority job is necessarily scheduled if there exist released jobs in a ready state. Thus, it turns out that no job exists during an Idle Time. Therefore, the first job of each task has finished execution by t (or by the first Idle Time). □

Lemma 3 (Adding an Idle Task). *If there exists Idle Time between 0 and τ_h's period T_h, adding a new task τ_{idle}, whose execution time is $idle(T_h)$*[7] and whose period is T_h, to Γ leaves the new task set schedulable.

Proof. Γ is divided into two sets, $\Gamma_{high} = \{\tau_1, \tau_2, ..., \tau_{h-1}\}$ and $\Gamma_{low} = \{\tau_h, \tau_{h+1}, \tau_{h+2}, ..., \tau_n\}$. Any task in Γ_{high} has a period equal to or shorter than T_h. On the other hand, any task, except τ_h, in Γ_{low} has a period equal to or longer than T_h. For tasks in Γ_{high}, through the same discussion as in the proof for lemma 1, their feasibility is not affected. For τ_{idle}, the deadline requirement is met since its job's response time clearly becomes equal to or less than T_h. Therefore, it is sufficient to confirm the schedulability for Γ_{low}.

Focusing on the lowest-priority task τ_n in Γ_{low}, the response time R_n of the first job of τ_n and T_h have the following relationship:

$$R_n + idle(T_h) + BT = T_h \tag{5}$$

Here, BT is the sum of time duration when the second or subsequent jobs of tasks in Γ_{high} are scheduled from the instant R_n to T_h. Let R'_n be the response time of τ_n's job after adding $\tau_{idle} = (idle(T_h), T_h)$ to Γ. τ_{idle} releases its job only once from 0 to T_h and interferes with τ_n by $idle(T_h)$.

From lemma 2, before adding τ_{idle} to Γ, all jobs finish the execution by the instant T_h. At the instant T_h after adding τ_{idle}, tasks in Γ_{high} have finished their first job since addition of τ_{idle} does not affect Γ_{high}. Thus, the maximum interference in τ_n by adding τ_{idle} is $idle(T_h)$ mentioned above and the total amount of execution time of the second or subsequent jobs of Γ_{high}'s tasks before T_h [8] is just BT. Therefore, R'_n has the following upper bound.

$$R'_n \quad \leq \quad R_n + idle(T_h) + BT \tag{6}$$

From equations (5) and (6):

$$\begin{aligned} R'_n &\leq R_n + idle(T_h) + BT \\ &= R_n + idle(T_h) + T_h - R_n - idle(T_h) \\ &= T_h. \end{aligned}$$

From $T_h \leq T_n$, $R'_n \leq T_h \leq T_n$ so that τ_n meets its deadline constraint.

For tasks in Γ_{low} after adding τ_{idle}, the response times and periods satisfy the following condition:

$$R'_h \leq R'_{h+1} \leq ... \leq R'_n \leq T_h \leq T_{h+1}, \leq ... \leq T_n$$

Therefore, by theorem 1, all tasks in Γ_{low} meet their deadlines after τ_{idle} is added. □

Example 2 (Addition of τ_{idle}). *Fig*6 *shows an example of adding τ_{idle} to **Fig 5**, where $\tau_h = \tau_2$ and* $T_h = T_2 = 6$.

[7] In this case, a relative time duration, e.g., T_h, is regarded as a time instant ($t = T_h$).

[8] The reason for 'maximum' is simple. Suppose $T_h = T_3 (= 7)$ and $\tau_n = \tau_3$ in **Fig 5**. In that case, $BT = 2$, since the second jobs of τ_1 and τ_2 are scheduled before the instant T_h. After adding τ_{idle}, τ_n is not affected by τ_2's second job, and finally $R'_n = 6 (< 7)$ is given.

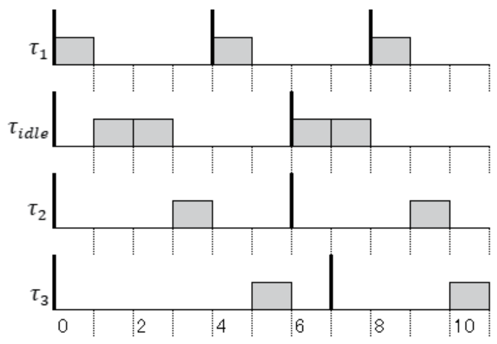

Fig.6: *Addition of τ_{idle}.*

3.2 Execution Right Delegation (ERD)

ERD is a method to shorten response time and jitter of a particular (or important) task, τ_p, in a task set by using a high-priority virtual server, VS, which has capacity of C_s and period of T_s while meeting the deadline constraints of the task set. We assume τ_p has a relatively lower priority due to its longer period. By making the priority of VS high with short T_s, τ_p can be executed at the high priority while consuming C_s. The basic behavior of VS is from PE [4].

Definition 4 (Delegation of Execution Right). *In the ERD method, VS is scheduled based on an RM rule. When VS is given the execution right, a particular periodic task, τ_p, is executed instead of VS. This situation is called Delegation of Execution Right. If Job of τ_p has already finished when the execution right is available, the behavior follows PE rule [4] where jobs of the other tasks are executed while the server capacity is accumulated at the priority level of the running job.*

Next, the following definition gives the algorithm for finding C_s and T_s.

Definition 5 (Candidates of VS). *Let Γ be a schedulable task set and τ_p in Γ be a particular task whose response time and jitter should be shortened. Then, RTA in Definition 1 is applied to $\tau_1, ...,$ and τ_p. From the relationship between R_p and $T_1, ..., T_{p-1}$, C_s and T_s for VS are obtained as follows:*

$$C_s = \begin{cases} C_p, & \text{if } R_p \le T_{p-1} \quad (7) \\ idle'(T_s), & \text{otherwise} \quad (8) \end{cases}$$

$$T_s = \begin{cases} T_h, & \text{if } R_p \le T_{p-1} \quad (9) \\ t \in \Psi, & \text{otherwise} \quad (10) \end{cases}$$

where

$$\begin{aligned} \Psi &= \{T_1, T_2, ..., T_{p-1}\} \\ T_h &= min(\ \{t \mid t \in \Psi,\ R_p \le t\}\ ) \\ idle'(t) &= t - \sum_{j=1}^{p-1} \lceil \frac{t}{T_j} \rceil Cj \quad (t \in \Psi) \end{aligned}$$

VS derived by the equations (7) and (9) makes it possible for τ_p to be executed at the same priority as a higher priority task. **Fig 7** shows an example. Since $R_p \not\le T_1$ but $R_p \le T_2$, VS $= (C_p, T_2) = (1, 6)$.

Theorem 3 (Schedulability with VS (Exchange of Priority)). *Let Γ be a schedulable task set and Γ' be the new task set which is derived by applying equations (7) and (9) to change the priorities of τ_p and τ_h which has higher priority than τ_p in Γ. Then, Γ' is schedulable.*

Proof. Since $R_p \le T_h$, theorem 2 guarantees that Γ' is schedulable. □

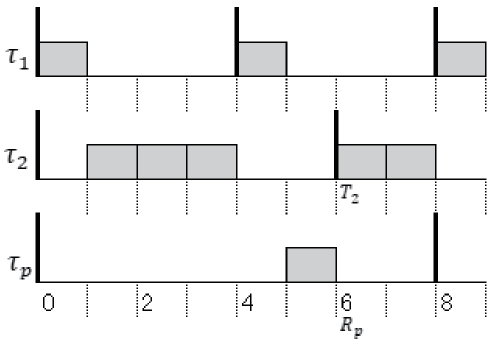

Fig.7: *Example of $R_p \le T_{p-1}$.*

The following gives a scheduling example with VS derived from equations (7) and (9).

Example 3 (ERD method - 1). *With $\Gamma = \{(2,4),(3,12),(3,14)\}$ and $\tau_p = \tau_3$, RTA gives response times of the three tasks as $R_1 = 2, R_2 = 7$, and $R_3 = 12$. Since $R_3 \le T_2 (= 12)$, equations (7) and (9) offer VS $= (3, 12)$.*

*Scheduling results by RM and ERD are shown in **Fig 8** and **Fig 9**, respectively. At the instants $t = 2, 3$, and 6, delegation of execution right can be confirmed. In this example, ERD improves response time of τ_3 from 12 to 7*[9].

[9] Strictly speaking, ERD does not improve response time of τ_3 but τ_3's first job. We discuss this problem in the next section.

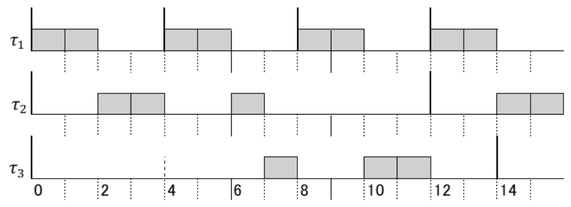

Fig.8: *Schedule by RM for* $\Gamma = \{(2,4),(3,12),(3,14)\}$.

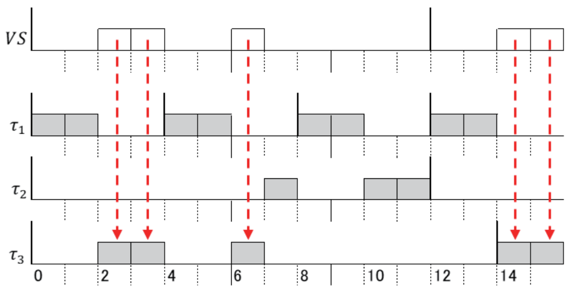

Fig.9: *Schedule by ERD.*

As for VS provided by the equations (8) and (10), more than one candidate can exist. $\Psi = \{T_1, T_2, ...T_{p-1}\}$ is a set of periods which are shorter than T_p, and $idle'(T_i)$ gives the length during which τ_p is executed. (Note that $idle()$ in Definition 3.1and $idle'()$ in Definition 3 are different. While the former gives the sum of all idle times before any instant t, the latter means, in each period in Ψ, the sum of time slots during which any task in $\tau_1, ..., \tau_{p-1}$ is not executed but τ_p is executed due to $R_p > T_{p-1}$.) **Fig 10** shows an example. With $\Psi = \{T_1(=4), T2(=6)\}$, $idle'(4) = 1$ and $idle'(6) = 2$, two candidates for VS are derived: $(1,4)$ and $(2,6)$.

Using VS from (8) and (10), the first job of τ_p is necessarily preempted by the other tasks and the execution is split. For example, in **Fig 10** , when $VS = (1,4)$ is used, the first part of τ_p's job would be executed at the highest priority. In this case, although VS does not shorten response time, it can contribute to reducing jitters in practice if an important part of the job (e.g. polling sensors) is located in the first half of the task program code. In addition, a job does not spend its worst-case execution time in most practical situations. If the actual execution time is less than or equal to the server capacity, the response time is expected to be substantially shortened.

Theorem 4 (Schedulability with VS (Split of Job)). *With a schedulable task set* Γ, τ_p *whose execution is split by VS derived from equations (8) and (10) meets its deadline constraint, and the new task set* Γ' *with VS is schedulable.*

Proof. After introducing VS, schedulability for $\Gamma_{high} = \{\tau_1, \tau_2, ..., \tau_{p-1}\}$, where each element has higher priority than τ_p, and that for $\Gamma_{low} = \{\tau_{p+1}, ..., \tau_n\}$, which includes lower priority tasks than τ_p, are shown along with the feasibility of τ_p.

For the job of τ_p, let τ_{ps} be the part executed by VS and τ_{po} be the remaining part. Note that if the whole job is accommodated by VS, τ_{po} does not exist. Obviously, early completion of a part of a job does not extend the response time. Thus, τ_{po} meets its deadline constraint since it is included in the schedulable task set, Γ.

Next, it is shown that τ_{ps} does not affect the schedulability of Γ_{high}. The existence of VS means a positive idle time $(idle'(T_p))$ exists at the instant T_p. By lemma 3, addition of τ_{idle} leaves Γ_{high} schedulable. Since τ_{idle} can be represented (or replaced) by VS, or τ_{ps}, Γ_{high} is schedulable.

For Γ_{low}, similar to the discussion in the proof for lemma 1, exchange of the execution order among higher priority tasks than Γ_{low} does not change the instant when the right of execution is given to Γ_{low}. Thus, Γ_{low} is schedulable. □

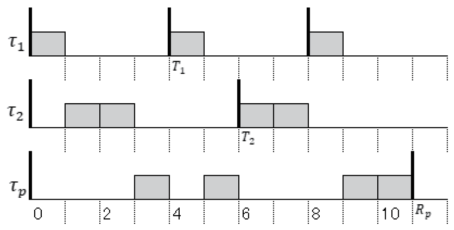

Fig.10: *Example in case of* $R_p > T_{p-1}$.

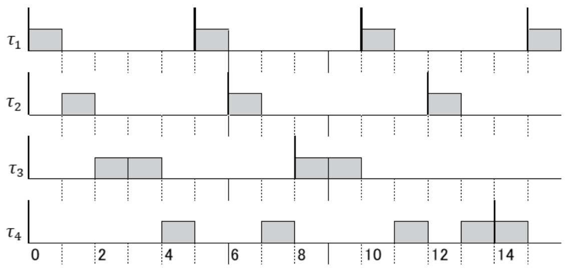

Fig.11: *Schedule by RM for* $\Gamma = \{(1,5),(1,6),(2,8),(4,14)\}$.

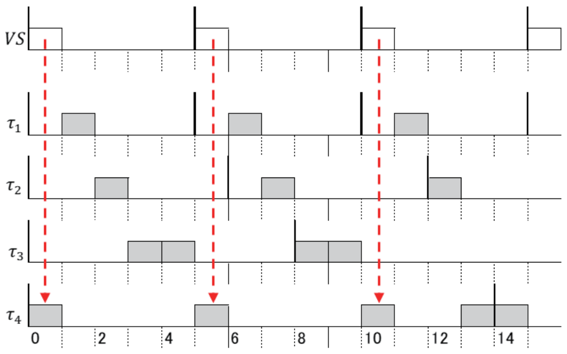

Fig.12: *Schedule by ERD with* VS_1 *for* Γ.

Example 4 (ERD method - 2) *With* $\Gamma = \{(1,5),(1,6),(2,8),(4,14)\}$ *and* $\tau_p = \tau_4$*, RTA gives* $R_1 = 1, R_2 = 2, R_3 = 4,$ *and* $R_4 = 14$ *(**Fig 11**)*

*VS is given by equations (8) and (10) since $R_4 > T_3$. With $\Psi = \{T_1, T_2, T_3\}$, $idle'(T_i)$ is calculated for each period. Since $idle'(T_1) = 1$, $VS_1 = (1, 5)$ is derived. With VS_1, $R'_4 = 14$, which does not shorten response time (**Fig 12**). Similarly, $VS_2 = (1, 6)$ is obtained from $idle'(T_2) = 1$. However, VS_2 still gives $R'_4 = 14$.*

*Then, from $idle'(T_3) = 2$, $VS_3 = (2, 8)$ is derived and found to give $R'_4 = 10$ (**Fig 13**) Finally, the results of VS_1, VS_2, and VS_3 are compared and VS_3 is chosen as VS because of the shortest response time for τ_4.*

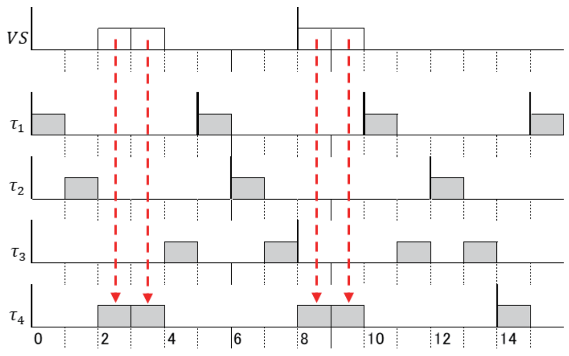

***Fig.13:** Schedule by ERD with VS_3 for Γ.*

It is worth noting that Deadline Monotonic scheduling does not shorten the response time of τ_4 with the above Γ. DM is effective only when the relative deadline of τ_4 can be made less than or equal to the deadline of a higher-priority task. However, for Γ in this example, if the deadline is set to be less than or equal to T_3, τ_3's job misses its deadline.

The following theorem and definition bring more improvement in response time.

Theorem 5 (Recursive Application of VS). *If the response time of τ_p is still equal to or less than T_{h-1} after VS is applied, definition 5 can be applied again and new VS' can be obtained. This is done recursively until the response time becomes longer than T_{h-1}.*

Proof. By theorem 3, $\Gamma' = \{\tau_1, \tau_2, ..., \tau_{h-1}, \mathbf{VS}, \tau_h, ..., \tau_{p-1}, \tau_{p+1}, ...\}$ is a schedulable task set. Thus, definition 5 provides new $\mathbf{VS}'$ and a new schedulable task set Γ'' as a result of applying $\mathbf{VS}'$. □

Definition 6 (Shortening Period of VS). *If VS obtained by equations (7) and (9) results in $R_p \le T_1$, T_s can be C_p (i.e. $\boldsymbol{VS} = \{C_p, C_p\}$)[10] .*

The ERD method can underutilize server's capacity and fail to delegate the execution correctly when T_p and T_s are not synchronized (Section 4.4 discusses the details). This does not happen when definition 6 can be applied.

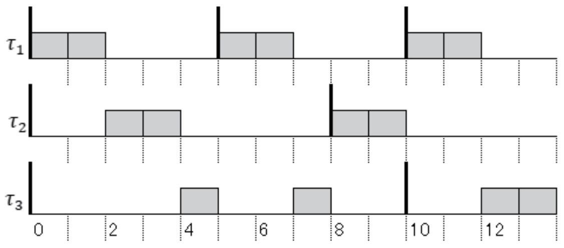

***Fig.14:** Schedule by RM for $\Gamma = \{(2, 5), (2, 8), (2, 10))\}$.*

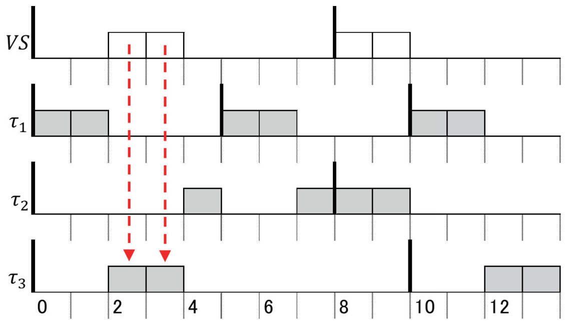

***Fig.15:** Schedule by ERD.*

Example 5 (ERD method - 3).*For $\Gamma = \{(2, 5), (2, 8), (2, 10)\}$ and $\tau_p = \tau_3$, RTA leads to $R_1 = 2, R_2 = 4$, and $R_3 = 8$. From $R_3 \le T_2 (= 8)$, $VS = \{2, 8\}$ is obtained from the equations (7) and (9). **Fig 14** and **Fig 15** show results of scheduling by RM and ERD, respectively.*

*As a result of scheduling by the ERD method, $R'_1 = 2, R'_2 = 8, R'_3 = 4$, and $R'_p \le T1 (= 5)$. In addition to obtaining new server $\boldsymbol{VS}' = (2, 5)$ by theorem 5, it is confirmed that $R_p \le T_1$, which leads to $\boldsymbol{VS}' = (2, 2)$ by definition 6. The result of scheduling with $\boldsymbol{VS}'$ is shown in **Fig 16**.*

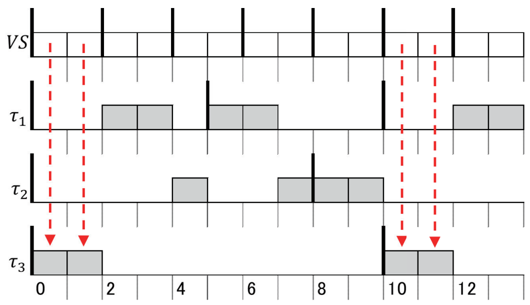

***Fig.16:** Schedule by ERD (applying theorem 5 and definition 6).*

4. RTA OF ERD

This section provides response time analysis (RTA) of ERD.

[10]The apparent CPU utilization of 100% does not spoil the schedulability, since VS only delegates its capacity to a particular task.

4.1 Interference Time

In the previous section, we reviewed several examples in which response time of a specific task τ_p is shortened in the scheduling chart. In other words, it was no more than simulation-based confirmation. In this section, WCRT is provided from a mathematical perspective. In the model of single processor RM, WCRT of a task is equal to the response time of its first released job. On the other hand, as in multiprocessor scheduling, τ_p's WCRT is unknown in ERD. In some cases, τ_p's first job's response time becomes its WCRT like the examples in the previous section. However, **Fig 17** shows the second job's response time is longer than the first job's one.

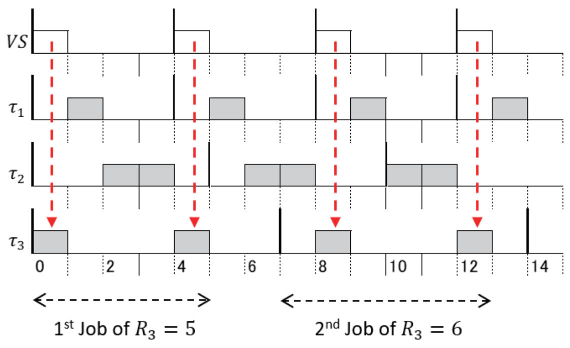

Fig.17: *Second Job has longer response time.*

Then, it is required to build RTA of ERD. Let R_i^{RM} be a WCRT of τ_i in RM scheduling, and R_i^{ERD} be a WCRT of τ_i in ERD scheduling.

Definition 7 (τ_{vs} and τ_h) *τ_{vs} is a task whose period is the same as VS and whose priority is treated as being lower than VS. τ_h is a task whose priority is higher than τ_p but excludes τ_{vs}.*

τ_2 in **Fig 9**, τ_1 in **Fig 12**, and τ_3 in **Fig 13** are τ_{vs}.

Lemma 4 (Upper bound of WCRT in ERD). *For τ_p, $R_p^{ERD} \leq R_p^{RM}$.*

Proof. In the ERD method, τ_p can be executed if VS whose priority is higher than τ_p has execution right. As a result, τ_p's execution is advanced. Otherwise, τ_p is scheduled according to the same rule as the RM method. Therefore, the response time can be no longer than that in the RM method. □

Theorem 6 (WCRT of High Priority Task). *A task subset Γ_{hp} consists of tasks whose priority is higher than τ_p. For any $\tau_i \in \Gamma_{hp}(i < p)$, a response time of τ_i's first job exhibits R_i^{ERD}.*

Proof. For τ_i, VS can be regarded as a task with execution time C_s and period T_s. Let VS be appended as a periodic task to Γ_{hp}, and make a task set $\Gamma_{hp'}$. After scheduling $\Gamma_{hp'}$ with RM rule, WCRT of τ_i can be retrieved from Definition 2.3 □

Definition 8(Interference Time). *Interference time I_i^{ERD} is the maximum time during which some higher-priority task τ_i affects response time of τ_p under ERD scheduling. I_i^{RM} is the case under RM scheduling and calculated as $I_i^{RM} = \lceil \frac{R_p^{RM}}{T_i} \rceil C_i$.*

Lemma 5 (Upper bound of Interference Time of τ_h). *For $\tau_h \in \Gamma_{hp}(h < p, h \neq vs)$, $I_h^{ERD} \leq I_h^{RM}$.*
Proof. From a contribution of VS, $I_h^{ERD} \leq \lceil \frac{R_p^{ERD}}{T_h} \rceil C_h$. From Lemma 4, $\lceil \frac{R_p^{ERD}}{T_h} \rceil C_h \leq \lceil \frac{R_p^{RM}}{T_h} \rceil C_h = I_h^{RM}$. Thus, $I_h^{ERD} \leq I_h^{RM}$. □

Now that we have obtained an upper bound of interference time of τ_h, with regards to the estimate of interference time of τ_{vs}, we start a method based on Bertogna's way [13]. We focus on the interference time while a task τ_{vs} with higher priority than τ_p operates in an interval L. L is divided into 3 sub intervals (**Fig 18**): L_{body} is the interval consisting of τ_{vs}'s successive periods which are totally included in L. L_{carry_in}, or L_{cin} is the interval between the beginning of L and the beginning of L_{body}. L_{carry_out}, or L_{co} is the interval between the end of L_{body} and the end of L. Similar to the single processor RTA in Definition 1, ERD RTA repeats the calculations so that the right and left sides get equal. Let the initial value of L be the response time of τ_p in the RM method.

$$L = R_p^{RM} \tag{11}$$

Now we consider values for L_{cin}, L_{body}, and L_{co} that maximize an interference time of τ_{vs}. There are two cases in which the interference time is maximized: L starts at the time when the job of τ_{vs} starts executing, and the job finishes at the end of L. The same discussion can be made for both cases. We choose the former case here.

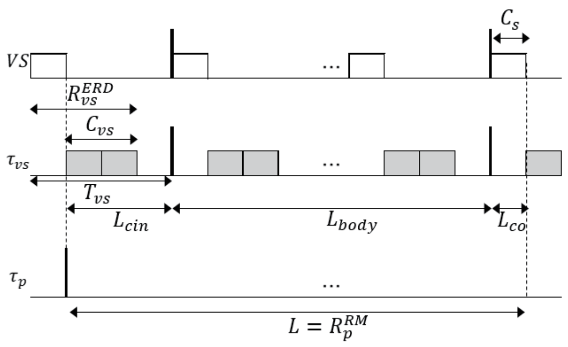

Fig.18: *Body, Carried-in and Carried-out jobs.*

In the interval L_{cin}, in order to maximize interference time of τ_{vs}, the whole job is required to be executed. On the other hand, in order to maximize the length of L_{body} plus L_{co}, L_{cin} must be minimized.

Such an L_{cin} is calculated as:

$$L_{cin} = T_{vs} - R_{vs}^{ERD} + C_{vs}, \quad (12)$$

where R_{vs}^{ERD} is derived from Theorem 6. With L_{cin}, L_{body} and L_{co} are:

$$\begin{aligned} nbody &= \lfloor \frac{L - L_{cin}}{T_{vs}} \rfloor \\ L_{body} &= nbody \times T_{vs} \\ L_{co} &= L - (L_{cin} + L_{body}). \end{aligned}$$

Thus, the execution times, cin and $body$, of τ_{vs}'s jobs in the intervals L_{cin} and L_{body}, respectively, are calculated as follows:

$$\begin{aligned} cin &= C_{vs} \\ body &= nbody \times C_{vs} \end{aligned}$$

Lemma 6 (Carried-out job of τ_{vs}) *Upper bound of the execution time, co, of τ_{vs}'s job in the interval L_{co} is calculated as follows:*

$$co = min(max(L_{co} - C_s, 0), C_{vs})$$

Proof. From Definition 7, τ_{vs}'s priority is evidently lower than VS and its period is the same as VS. That is, at the release instant of τ_{vs}'s job, VS is ready to execute. If there is no higher priority task's job than VS, τ_p's job is executed. In the case that a higher priority task's job is released, that job is executed. In any case, τ_{vs}'s job must wait for C_s, since τ_p's job does not finish before the end of L. □

Lemma 7 (Interference Time of ERD). *Interference Time I_i^{ERD} of τ_i during an interval L is calculated as follows.*

$$I_i^{ERD} = \begin{cases} cin + body + co & \text{if } \tau_i = \tau_{vs} \quad (13) \\ I_i^{RM} & \text{otherwise} \quad (14) \end{cases}$$

*Proof.*By Lemmas 5, 6 and the discussion in this section. □

4.2 Reducing Carry-in

The discussion in the previous subsection supposes that, in an interval L, the interference time to τ_p is the maximum one. Thus, the response time of τ_p is to be pessimistic. In regard to τ_{vs}'s carry-in job, we consider whether it is possible to be shorten it or not. First, focus on the interval L' which is τ_p's period immediately before L (**Fig 19**).

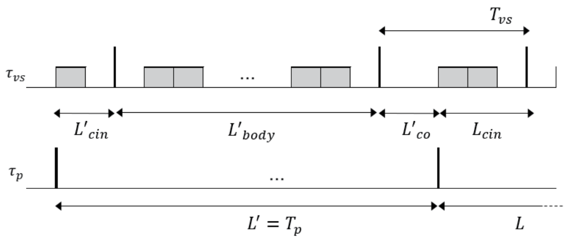

Fig.19: *Body, Carried-in and Carried-out length in L'.*

Each sub interval of L' is calculated as follows:

$$\begin{aligned} L' &= T_p \\ L'_{co} &= T_{vs} - L_{cin} \\ nbody' &= \lfloor \frac{L' - L'_{co}}{T_{vs}} \rfloor \\ L'_{body} &= nbody' \times T_{vs} \\ L'_{cin} &= L' - (L'_{co} + L'_{body}) \end{aligned}$$

Thus, $co', body'$, and cin' are as follows:

$$\begin{aligned} co' &= 0 \\ body' &= nbody' \times C_{vs} \\ cin' &= min(max(L'_{cin} - T_{vs} + R_{vs}^{ERD}, 0), C_{vs}) \end{aligned}$$

Let $empty(l)$ be an idle/empty time in the interval l. Intuitively, l minus the sum of C_p and the interference time of all tasks of higher priority than τ_p is idle/empty time. If there is idle/empty time, it means there is room to advance the carry-in job of τ_{vs} in L_{cin}. Then, we calculate empty time in the interval L' as follows:

$$\begin{aligned} empty(L') = {} & max(L' - (cin' + body' + co') - \\ & C_p - \sum_{j=1, j \neq vs}^{p-1} I_j^{ERD}, 0) \end{aligned}$$

$empty(L')$ is the amount of time by which the carry-in job is advanced. That is, a part of the carry-in job in L_{cin} is moved to L'_{co}. Then, we update the execution time of the carry-in job as $ncin$.

$$ncin = cin - empty(L')$$

Compared to the formula (12), L_{cin} of interval L can be reduced as follows:

$$L_{cin} = T_{vs} - R_{vs}^{ERD} + ncin$$

We can replace cin and L_{cin} in the formulas in the previous subsection by $ncin$ and the reduced L_{cin}, respectively, and therefore can alleviate the pessimism of the interference time.

4.3 Updating L

With the updated I_i^{ERD}, the initial value of R_p^{ERD} is calculated as follows:

$$R_p^{ERD} = \sum_{i<p} I_i^{ERD} + C_p$$

Recalling (11), L can be updated by R_p^{ERD}. In addition, the formula (14) in Lemma 7 can be updated since the response time of τ_p is now shortened compared with RM (see Definition 8). Consequently, the formula of Lemma 7 is updated as follows:

$$I_i^{ERD} = \begin{cases} cin + body + co & \text{if } \tau_i = \tau_{vs} \quad (15) \\ \lceil \frac{R_p^{ERD}}{T_i} \rceil C_i & \text{otherwise} \quad (16) \end{cases}$$

Theorem 7 (RTA of ERD). *In ERD, τ_p's WCRT is calculated as :*

$$R_p^{ERD} = \sum_{i<p} I_i^{ERD} + C_p, \qquad (17)$$

where the initial value of R_p^{ERD} is R_p^{RM}. Calculation is repeated until R_p^{ERD} is unchanged.

Proof. By the discussion in this section. □

5. EVALUATION

5.1 Task Sets and Simulation Environment

In this section, we evaluate the proposed method by showing how much the analyzed WCRT is shortened compared with RM. For comparison, we show the results of ERD RTA and DM. In addition, the longest response times in the task scheduling simulation with ERD (ERD simulation) are shown for reference[11]. We also provide simulation results for AFJ[12]. For evaluation, 30,000 task sets are synthetically generated. Each task set is schedulable (with no deadline misses) under RM and has processor utilization of 72% to 96%. When generating task sets, tasks' periods are decided by random numbers based on the uniform distribution between 10 and 100. Their execution times are decided by random numbers based on the exponential distribution where the upper bound, U_{mt}, is 25% or 10% of the corresponding periods. On average, each task set has 5.06 tasks and 9.38 tasks in case of $U_{mt} = 25\%$ and $U_{mt} = 10\%$, respectively.

For the evaluation, the target task τ_p whose response time should be shortened is a task with the lowest priority (longest period). To raise the priority of τ_p, we applied the following policy. For DM, τ_p was given the shortest deadline possible while the schedulability was guaranteed. For ERD, after several candidates of VS were derived, the best server for the results of τ_p was selected.

The simulation period is 10,000 ticks. The same task sets are used for ERD RTA and DM as well as ERD simulation. We use Python 3.6.5 for the simulation. For task set generation, we use schedcat [22] modified for our experiment environment. For simulation, the source code including the ERD scheduling algorithm is available in the author's repository [23].

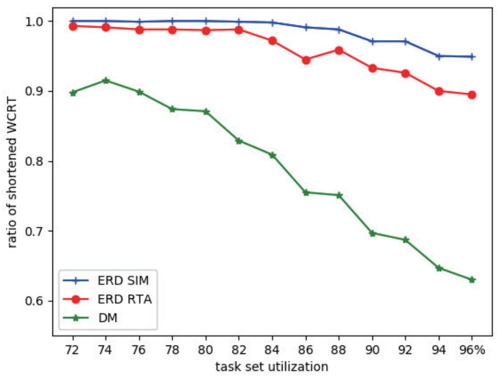

Fig.20: *WCRT Evaluation : Ratio of task sets with shortened WCRT ($U_{mt} = 25\%$).*

5.2 Results

The ERD method shortens the response time by using a high-priority virtual server, while the DM method can shorten the response time when the priority of τ_p can be raised by regulating its relative deadline. **Fig 20** shows the ratio of task sets which can reduce WCRT when U_{mt} is 25%. The horizontal axis indicates the processor utilization of the task sets, and the vertical axis indicates the ratio of the task sets that can have WCRT reduced. In the ERD simulation, the longest response times can be reduced for almost all task sets whose processor utilization is up to 84%. Even when the utilization is 96%, 95% of the task sets succeeded to reduce τ_p's response time. In the DM method, it is confirmed that fewer task sets than in ERD RTA can have shortened WCRT for τ_p.

[11] In ERD simulation, the obtained longest response times might not be WCRTs, since the simulation period is limited.
[12] We exclude RFJ since there was not big difference between AFJ and RFJ.

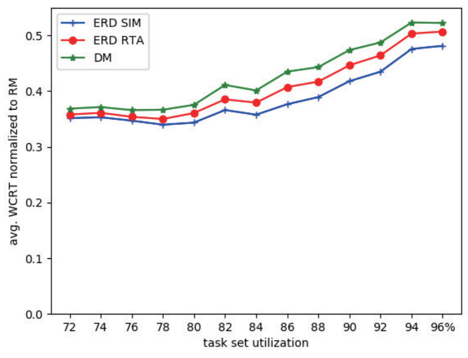

Fig.21: *WCRT Evaluation : Average WCRT normalized to RM ($U_{mt} = 25\%$).*

Fig 21 shows the average WCRT normalized to RM RTA. While ERD RTA exhibits shorter WCRT than DM, it is longer than ERD simulation due to its pessimism, which implies there is room for further improving the analysis. When the utilization is 86%, it is confrmed that ERD can reduce the average WCRT by 13.45% at maximum compared to the DM.

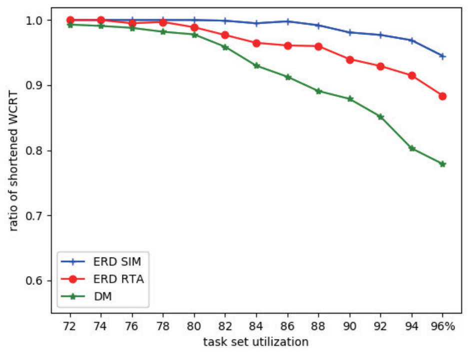

Fig.22: *WCRT Evaluation : Ratio of task sets with shortened WCRT ($U_{mt} = 10\%$).*

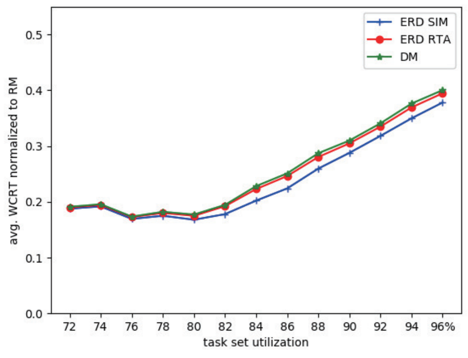

Fig.23: *WCRT Evaluation : Average WCRT normalized to RM ($U_{mt} = 10\%$).*

Fig 22 and **Fig 23** are the results when $U_{mt} = 10\%$. Similar trends to **Fig 20** and **Fig 21** can be seen. **Fig 22** exhibits better results compared to **Fig 20**. This shows that if U_{mt} is lower, VS could easily obtain high priority in ERD case and τ_p could increase its priority in DM case even when it has the lowest priority.

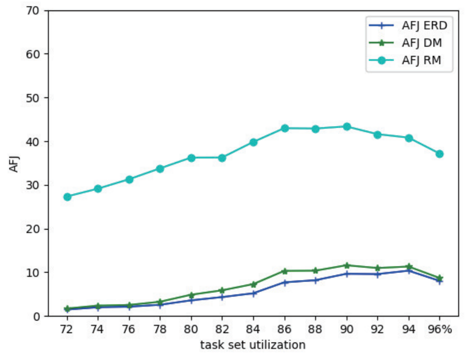

Fig.24: *Jitter Evaluation : comparison of average AFJ ($U_{mt} = 25\%$).*

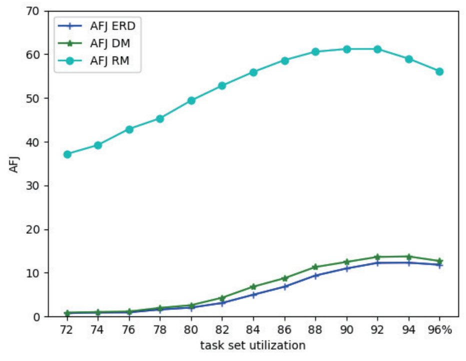

Fig.25: *Jitter Evaluation : comparison of average AFJ ($U_{mt} = 10\%$) .*

Fig 24 shows simulation results of the average AFJ when $U_{mt} = 25\%$. The vertical axis indicates absolute value of the average AFJ. In RM, AFJ is around 30 to 35 when the utilization is in the 72% to 78% range. Over the 80% range, AFJ is around 40. The AFJ of ERD and DM is less than 5 when utilization is below the 80% range and up to 10 or 11 when utilization is more than 80%. **Fig 25** shows the results when U_{mt} is 10%. In this case, the AFJ of RM is higher than that for U_{mt} which is 25%. The AFJ of ERD and DM is almost the same as U_{mt} is 10%.

For jitter, it can be said that ERD does not have a big advantage over DM when the utilization is between 72% and 78%. These methods show good results compared with RM.

The algorithm we proposed in this paper handles only one target (important) task, τ_p. Theoretically, the capacity of VS can be used for any task whose priority is higher than τ_p. However, this approach is not sufficient to shorten multiple target tasks' WCRT due to the limited capacity of VS on a single processor. Since multi processor systems have more capacity than a single processor system, building a multi processor version of ERD which serves multiple target tasks is a goal of future work.

6. CONCLUSIONS

Execution Right Delegation (ERD) is a real-time scheduling technique that improves real-time performance of tasks which have long periods but are of high importance, while RM necessarily gives low priority to tasks with long periods. In this paper, after reviewing ERD, we showed a response time analysis (RTA) method of ERD and evaluated it in comparison with DM and the simulated behavior of ERD. The results showed that ERD RTA can give shorter worst-case response times (WCRTs) than DM but is still a pessimistic analysis compared with the simulation results. There is room to improve the analysis for further reduction of WCRT.

ACKNOWLEDGMENT

This work is supported by JSPS KAKENHI Grant Number JP 19K11873.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46-61, 1973.

[2] `http://www.tron.org/`

[3] N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," *IFAC Proceedings*, vol. 24, no.2, pp. 127-132, 1991.

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

[5] B. Sprunt, J. Lehoczky and L. Sha, "Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm," *Proceedings. Real-Time Systems Symposium*, pp. 251-258, 1988.

[6] R. Davis and A. Wellings, "Dual priority scheduling," *Proceedings 16th IEEE Real-Time Systems Symposium*, pp. 100-109, 1995.

[7] D. de Niz, K. Lakshmanan and R. Rajkumar, "On the Scheduling of Mixed-Criticality Real-Time Task Sets," *2009 30th IEEE Real-Time Systems Symposium*, pp. 291-300, 2009.

[8] J. Y-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, no. 4, pp. 237-250, 1982.

[9] R. I. Davis, L. Cucu-Grosjean, M. Bertogna and A. Burns, "A review of priority assignment in real-time systems," *Journal of systems architecture*, vol. 65, pp. 64-82, 2016.

[10] E. Bini, G. Buttazzo and G. Buttazzo, "A hyperbolic bound for the rate monotonic algorithm," *Proceedings 13th Euromicro Conference on Real-Time Systems*, pp. 59-66, 2001.

[11] M. Joseph and P. Paritosh, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390-395, 1986.

[12] M. Coutinho, J. Rufino and C. Almeida, "Response Time Analysis of Asynchronous Periodic and Sporadic Tasks Sheduled by a Fixed Priority Preemptive Algorithm," *2008 Euromicro Conference on Real-Time Systems*, pp. 156-167, 2008.

[13] M. Bertogna and M. Cirinei, "Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms," *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 149-160, 2007.

[14] N. Guan, M. Stigge, W. Yi and G. Yu, "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling," *2009 30th IEEE Real-Time Systems Symposium*, pp. 387-397, 2009.

[15] Youcheng Sun, G. Lipari, N. Guan and W. Yi, "Improving the response time analysis of global fixed-priority multiprocessor scheduling," *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1-9, 2014.

[16] Y. Sun and D. N. Marco, "Pessimism in multicore global schedulability analysis," *Journal of Systems Architecture*, vol. 97, pp. 142-152, 2019.

[17] Q. Zhou, G. Li, J. Li, C. Deng and L. Yuan, "Response Time Analysis for Tasks with Fixed Preemption Points under Global Scheduling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5, pp. 1-23, 2019.

[18] P. Marti, J. M. Fuertes, G. Fohler and K. Ramamritham, "Jitter compensation for real-time control systems," *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pp. 39-48, 2001.

[19] G. Buttazzo and C. Anton, "Comparative assessment and evaluation of jitter control methods," *15th International Conference on Real- Time and Network Systems*, 2007.

[20] K. Tanaka, "Real-time scheduling for reducing jitters of periodic tasks," *Journal of Information Processing*, vol. 23, no. 5, pp. 542-552, 2015.

[21] G. C. Buttazzo, “Rate monotonic vs. EDF: Judgment day. Real-Time Systems,”*Springer Science + Business Media Inc. Manufactured in The Netherlands*, vol. 29, no. 1, pp. 5-26, 2005.

[22] `https://github.com/brandenburg/schedcat`

[23] `https://github.com/takahalsuzuki/erdsingle`

Takaharu Suzuki received his B.S. degree from Nihon University in 2003 and his M.S. degree from Japan Advanced Institute of Science and Technology in 2015. His research interests are real-time scheduling, operating systems, and automotive software. He is a member of the IPSJ.

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 the IEEE, ACM, IEICE, and IPSJ.