The Journal of Systems and Software 154 (2019) 112–124

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Improved schedulability analysis of the contention-free policy for real-time systems

Hyeongboo Baek[a], Jinkyu Lee[b,*]

[a] Incheon National University (INU), Incheon, South Korea
[b] Sungkyunkwan University (SKKU), Suwon, South Korea

ARTICLE INFO

*Article history:*
Received 21 September 2018
Revised 20 April 2019
Accepted 25 April 2019
Available online 25 April 2019

*Keywords:*
Real-time scheduling
Schedulability analysis
CF policy
Response-time analysis

ABSTRACT

Real-time scheduling is the primary research for designing real-time systems whose correctness is determined by not only logical correctness but also timely execution. Real-time scheduling involves two fundamental issues: scheduling algorithm design and schedulability analysis development, which aim at developing a prioritization policy for real-time tasks and offering their timing guarantees at design time, respectively. Among the numerous scheduling algorithms and schedulability analysis for a multiprocessor platform, the contention-free (CF) policy and response-time analysis (RTA) have received considerable attention owing to their wide applicability and high analytical performance, respectively. Notwithstanding their effectiveness, it has been conjectured that it is not feasible to exploit the two techniques together. In this study, we propose a new schedulability analysis for the CF policy, referred to as pseudo-response time analysis (PRTA), which exploits a new notion of pseudo-response time effectively capturing the time instant at which the schedulability of a task is guaranteed under the CF policy. To demonstrate the effectiveness of PRTA, we apply PRTA to the existing earliest deadline first and rate monotonic scheduling algorithms employing the CF policy, and show that up to 46.4% and 18.3% schedulability performance improvement can be achieved, respectively, compared to those applying the existing schedulability analysis.



## 1. Introduction

Systems are regarded *real-time* when the correctness of the systems depends on not only their functional aspects but also their temporal aspects (Liu and Layland, 1973). For example, in automotive electronics, wheel control systems must correctly execute within a predefined time interval in order to accurately control the vehicle. Real-time scheduling is the primary research for designing such systems, which determines the order of executions of jobs infinitely generated by real-time tasks running on the system in order to satisfy a timing requirement (i.e., deadline). Among the broad spectrum of research in the area of real-time scheduling, scheduling algorithm design and schedulability analysis are the fundamental parts; the former aims at developing a prioritization policy for real-time tasks, and the latter aims to obtain their timing guarantees at design time. A number of studies addressed the former and/or the latter, e.g., Lee et al. (2016) and Lee and Shin (2017).

Over the last few decades, as multiprocessor platforms have been increasingly used in the real-time systems for high-end real-time applications, a number of studies on real-time scheduling theory have been performed so as to effectively exploit such platforms, e.g., Brandenburg and Gul (2016), Melani et al. (2016) and Biondi et al. (2016). Whereas most of them have focused on developing new real-time scheduling algorithms, some studies have proposed polices to prioritize real-time tasks, which can be incorporated into the existing real-time scheduling algorithms to improve their performance. Expanding the latter approaches is quite important, in that it could potentially improve the schedulability performance of not only the existing scheduling algorithms but also those that will be developed in the future. Successful examples of the latter approaches include the zero-laxity (ZL) policy (Baker et al., 2008) and the contention-free (CF) policy (Lee et al., 2011; 2014), which improve the schedulability performance of the existing real-time scheduling algorithms with opposite principles. The ZL policy promotes the priority of a real-time task to the time instant when it should be scheduled to avoid a deadline miss; on the contrary, the CF policy demotes the priority of a real-time task to the time instant when it is guaranteed to complete its execution before its deadline.

Although the ZL and CF policies have received considerable attention owing to their wide applicability and schedulability performance improvement, the CF policy has not been thoroughly

* Corresponding author.
*E-mail address:* jinkyu.lee@skku.edu (J. Lee).

https://doi.org/10.1016/j.jss.2019.04.067
0164-1212/

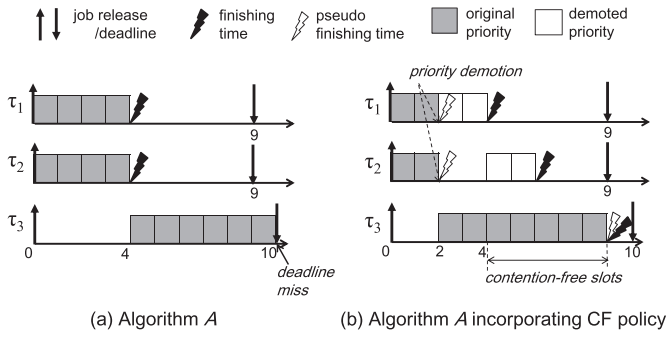



**Fig. 1.** Schedules of base algorithm A and algorithm A incorporating CF policy for $\tau = \{\tau_1 = \tau_2 = \{T_i = 15, C_i = 2, D_i = 9\}, \tau_3 = \{15, 7, 10\}\}$.

investigated with respect to the development of its schedulability analysis. That is, in contrast to the development of several schedulability tests for the ZL policy (Cirinei and Baker, 2007; Davis and Burns, 2011; Lee and Shin, 2013), only one sufficient schedulability test has been proposed for the CF policy, called deadline analysis (DA) (Bertogna et al., 2009). Although response-time analysis (RTA) (Joseph and Pandya, 1986) is known to dominate (i.e., to always provide better analytical capability than) DA, RTA loses such an advantage with respect to a scheduling algorithm incorporating the CF policy, because it does not fully capture the characteristics of real-time tasks' behavior under the CF policy.

Such characteristics stem from the CF policy's principle, which utilizes the notion of a *contention-free slot*, in which the number of jobs with remaining executions is less than or equal to the number of available processors in the time slot. A contention-free slot has an important property that all jobs with remaining execution in the time slot can execute without any contention. The CF policy calculates the lower-bounded number of contention-free slots that exist up to the deadline of each job before its release, and then moves some remaining executions of the jobs that are guaranteed to be schedulable to the contention-free slots by comparing remaining execution times and the remaining lower-bounded number of contention-free slots of each job at each time slot, thereby improving the schedulability of the existing algorithms.

Fig. 1 depicts an example of a task set $\tau$ scheduled by a base algorithm $A$ (Fig. 1(a)) and an algorithm $A$ incorporating the CF policy (Fig. 1(b)), on a two-processor platform. $\tau$ contains three tasks, $\tau_1$, $\tau_2$, and $\tau_3$, with execution times of 4, 4 and 7; deadlines of 9, 9 and 10; and a common period of 15. In the case of Fig. 1(b), the three tasks have minimum numbers of contention-free slots of 2, 2, and 3 until their deadlines, respectively, which are calculated offline by the CF policy. Such an offline calculation is the key technique of the CF policy, whose high-level idea is as follows. At most, $4 + 4 + 7 = 15$ executions can be performed within an interval [0,9) (i.e., the release and deadline of a job of $\tau_1$) and thus, at least two contention-free slots exist in the interval, under the assumption of a two-processor platform (i.e., $9 - \lfloor \frac{15}{2} \rfloor = 2$); this also holds for a job of $\tau_2$. A similar line of reasoning results in $10 - \lfloor \frac{15}{2} \rfloor = 3$ for a job of $\tau_3$. Note that the actual number of contention-free slots can be different from the number calculated by the CF policy (e.g., five actual contention-free slots in [4, 9) of Fig. 1(b)). We will explain how to calculate the number of contention-free slots for a general case in Section 3.1.

Let $J_1$, $J_2$, and $J_3$ be the first jobs of $\tau_1$, $\tau_2$, and $\tau_3$, respectively, and assume that $J_1$ and $J_2$ have higher priorities than $J_3$ by scheduling algorithm $A$. In the case of $J_3$ in Fig. 1(a), it misses its deadline at time instant 10 owing to interference from $J_1$ and $J_2$ in [0, 4). However, $J_3$ in Fig. 1(b) completes its execution at time instant 9, because the priorities of $\tau_1$ and $\tau_2$ are demoted by the CF policy at time instant 2. The underlying principle of such a demotion by the CF policy is that the remaining executions of $J_1$ and $J_2$ at time instant 2 (i.e., 2) will be successfully performed before their deadlines (i.e., 9) because there are at least two contention-free slots up to their deadline (in fact five contention-free slots exist in [4, 9)).

Then, the challenging issue for schedulability analysis is how to capture the characteristics of such online priority demotions conducted by the CF policy so as to effectively obtain offline timing guarantee (i.e., predictability). We can observe that schedulability of each job in Fig. 1(a) is identified at its own finishing time (e.g, time instant 4 for $J_1$ and $J_2$) whereas that in Fig. 1(b) can be done when its own priority is demoted (e.g., time instant 2 for $J_1$ and $J_2$), which is earlier than its own finishing time (e.g., time instants 4 and 6 for $J_1$ and $J_2$ respectively). Then, we can deem that the job is schedulable if such a time instant (a finishing time or a time when the priority demotion occurs) is earlier than its deadline.

In this study, we propose an improved schedulability analysis framework for the CF policy, which effectively considers the time when the priority demotion occurs by the CF policy. To this end, we propose a new notion of *pseudo-response time* at which the priority of a job is demoted (i.e., its schedulability is guaranteed) under the CF policy even before its actual response time, and discuss its properties. Then, we propose a new schedulability analysis exploiting the notion of pseudo-response time, referred to as pseudo-response time analysis (PRTA), which significantly improves the schedulability of DA. PRTA exploits the core idea of RTA to upper-bound the pseudo-response time of a task. We also discuss how to further improve its performance by utilizing a new notion of *pseudo-slack time*. To demonstrate the effectiveness of PRTA, we apply PRTA to the existing earliest deadline first (EDF) and rate monotonic (RM) scheduling algorithms employing the CF policy, and shows that it achieves up to 46.4% and 18.3% performance improvements, respectively, compared to those applying DA.

In summary, this paper makes the following contributions:

- It proposes a new notion of pseudo-response time, which represents a time when priority is demoted by the CF policy, which is exploited as a new criterion to determine the schedulability of a job scheduled by a scheduling algorithm employing the CF policy, and discusses its properties (Section 4.1).
- A new schedulability analysis framework, referred to as PRTA, is proposed, which significantly improves the schedulablity of DA by exploiting the notion of pseudo-response time (Section 4.2).
- A method to further improve the schedulability of PRTA is presented, which utilizes a new notion of pseudo-slack time (Section 4.3).
- It applies PRTA to the existing EDF and RM scheduling algorithms employing the CF policy and demonstrates the effectiveness of PRTA compared with DA (Sections 5 and 6).

## 2. System model

We consider the Liu and Layland's task model (Liu and Layland, 1973) in which a task set $\tau$ contains $n$ sporadic real-time tasks $\tau_i = (T_i, C_i, D_i)$ and is scheduled on $m$ identical processors using a global, preemptive, and work-conserving scheduling algorithm. $T_i$ is a minimum inter-arrival time or period, $C_i$ is the worst-case execution time (WCET) of $\tau_i$, and $D_i$ is a relative deadline. Note that a scheduling algorithm is called global, preemptive and work-conserving if a job can be migrated from one core to another (*global*) and preempted at any time (*preemptive*), and if the scheduled processors are always kept busy when there are released jobs ready to be scheduled (*work-conserving*). Throughout our paper, we assume that $\tau_i$ has an implicit or a constrained deadline satisfying $D_i = T_i$ or $D_i \leq T_i$, respectively. Without loss of generality, we as-

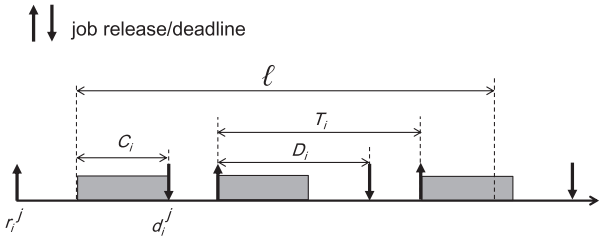



**Fig. 2.** The worst-case scenario in which the workload of a task $\tau_i$ in an interval of length $\ell$ is maximized.

sume a quantum-based time with a quantum length equal to one time unit.

A task $\tau_i$ invokes a series of jobs, where the $j$th job $J_i^j$ is released at $r_i^j$ and finished at $f_i^j$. We refer to a job with remaining execution as an active job. $J_i^j$ has an absolute deadline $d_i^j = r_i^j + D_i$, which represents the latest time instant at which we can accept the job to complete its execution. We say that $J_i^j$ is schedulable if $f_i^j \le d_i^j$ holds, and $\tau_i$ is schedulable if every job $J_i^j$ of $\tau_i$ is schedulable. Additionally, a task set $\tau$ is schedulable if every task $\tau_i \in \tau$ is schedulable. The response time $R_i$ of a task $\tau_i$ is defined as $max_{J_i^j \in \tau}(f_i^j - r_i^j)$, and $J_i^*$ is the job that induces $R_i$, *i.e.*, $J_i^*$ is the job exhibiting the largest value of $f_i^j - r_i^j$ among all the jobs of $\tau_i$. By the definition of the response time $R_i$, a task set $\tau$ is schedulable if $R_i \le D_i$ (*i.e.* $f_i^* \le d_i^*$) holds.

We define the interference $I_k(a, b)$ on $\tau_k$ in an interval [a, b) as the cumulative length of all the intervals in which $\tau_k$ is ready to be executed but cannot be scheduled on any processor owing to $m$ higher priority tasks. Additionally, the interference $I_k^i(a, b)$ of $\tau_i$ on $\tau_k$ in an interval [a, b) is defined as the cumulative length of all the intervals in which $\tau_k$ is ready to be executed but cannot be scheduled on any processor while $\tau_i$ executes.

The workload of a task $\tau_i$ in an interval of length $\ell$ is defined as the amount of computation time required for $\tau_i$ in the interval of length $\ell$. Fig. 2 illustrates the scenario in which the workload of a task $\tau_i$ is maximized under any preemptive scheduling. As seen in Fig. 2, the first job of $\tau_i$ begins its execution at the beginning of the interval and completes the execution at $d_i^j$, which executes for $C_i$ without any interference or delay. Then, the following jobs are released and scheduled as soon as possible. Building upon the number of executions of jobs fully executing for $C_i$ and the other jobs executing for a portion of $C_i$, the upper-bounded workload $W_i(\ell)$ is calculated by (Bertogna and Cirinei, 2007)

$$W_i(\ell) = N_i(\ell) \cdot C_i + min\Big(C_i, \ell + D_i - C_i - N_i(\ell) \cdot T_i\Big), \quad (1)$$

where $N_i(\ell)$ is the number of jobs executing for $C_i$ calculated by

$$N_i(\ell) = \left\lfloor \frac{\ell + D_i - C_i}{T_i} \right\rfloor. \quad (2)$$

For the ease of understanding, Table 1 summarizes the notations used in this paper.

## 3. Background

This section briefly introduces the CF policy and the existing RTA framework on a multi-processor platform (Lee et al., 2011; 2014), both of which form the basis for our approach proposed in this study.

### 3.1. CF policy

The principle of the CF policy is to improve schedulability of the existing scheduling algorithms by demoting the priority of a job when its schedulability is guaranteed during its execution, using the advantage of the notion of a contention-free slot. A contention-free slot is a time slot in which active jobs are guaranteed to execute without any contention, which is formally defined as follows.

**Definition 1.** (contention-free slot (from Lee et al., 2014)) : A time slot is contention-free if the number of jobs that are ready to execute in the slot is less than or equal to the number of processors $m$. Additionally, a time slot is contending if the slot is not contention-free.

To manage the active jobs with priorities assigned by algorithm $A$ and demoted by the CF policy, the CF policy uses two separated queues, $Q_H$ and $Q_L$, in which jobs in $Q_H$ have their original priorities assigned by the algorithm $A$, and those in $Q_L$ have priorities demoted by the CF policy. For job $J_i^j$, let $C_i^{(t)}$ be the remaining execution time to finish its execution at time instant $t$. Additionally, let $\Phi_i$ be the lower-bounded number of contention-free slots (calculated offline by the CF policy) that exists in the interval $[r_i^j, d_i^j)$ and $\Phi_i^{(t)}$ be the remaining number of contention-free slots at time instant $t$.

The CF policy conducts the following steps sequentially at each time slot.

If job $J_i^j$ of task $\tau_i$ is released,

1. Put $J_i^j$ with its own priority to $Q_H$, calculate $\Phi_i$ of $J_i^j$ and set $\Phi_i^{(t)}$ to $\Phi_i$.

For job $J_i^j$ in $Q_H$,

**Table 1**
Notations and their description.

| Notation | Description | Notation | Description |
|---|---|---|---|
| $n$ | The number of tasks in $\tau$ | $N_i(\ell)$ | The number of jobs fully executing for $C_i$ in an interval of length $\ell$ |
| $m$ | The number of processors | $Q_H$ | A queue containing jobs with its original priorities assigned by the algorithm $A$ |
| $\tau$ | A task set | $Q_L$ | A queue containing jobs with the priorities demoted by the CF policy |
| $\tau_i$ | A task in $\tau$ | $C_i^{(t)}$ | The remaining execution time to finish it execution at time instant $t$ |
| $T_i$ | A minimum inter-arrival time or period of $\tau_i$ | $\Phi_i$ | The lower-bounded number of contention-free slots that exists in the interval $[r_i^j, d_i^j)$ |
| $C_i$ | WCET of $\tau_i$ | $\Phi_i^{(t)}$ | The remaining number of contention-free slots at time instant $t$ |
| $D_i$ | A relative deadline of $\tau_i$ | $R_k^{ub}$ | An upper-bounded response time of $\tau_k$ |
| $J_i^j$ | The $j$th job invoked by $\tau_i$ | $\tilde{f}_k^j$ | A pseudo-finishing time of $J_k^j$ |
| $r_i^j$ | A release time of $J_i^j$ | $\tilde{R}_k$ | A pseudo-response time of $\tau_k$ |
| $f_i^j$ | A finishing time of $J_i^j$ | $J_k^+$ | The job inducing $\tilde{R}_k$ |
| $d_i^j$ | An absolute deadline of $J_i^j$ | $\tilde{R}_k^{ub}$ | An upper-bounded pseudo-response time |
| $R_i$ | A response time of $\tau_i$ | $\hat{\tau}$ | A modified task set from $\tau$ (detailed in Section 4.2) |
| $J_i^*$ | The job inducing $R_i$ | $\tilde{s}_i$ | A pseudo-slack time |
| $I_k(a, b)$ | The interference on $\tau_k$ in an interval [a, b) | $W_i^s(\ell, \hat{\tau})$ | The worst-case workload of $\tau_i(\hat{\tau})$ |
| $I_k^i(a, b)$ | The interference of $\tau_i$ on $\tau_k$ in an interval [a, b) | $N_i^s(\ell, \hat{\tau})$ | The number of jobs fully executing for $C_i - \Phi_i$ |
| $W_i(\ell)$ | The upper-bounded workload in an interval of length $\ell$ | $E_i^s(D_k, \hat{\tau})$ | The upper-bounded interference of $\tau_i(\hat{\tau})$ to a job $J_k^j$ of $\tau_k(\hat{\tau})$ |

2. If the current time slot is contention-free, reduce $\Phi_i^{(t)}$ by 1.
3. If $C_i^{(t)} \leq \Phi_i^{(t)}$, move $J_i^j$ from $Q_H$ to $Q_L$ and assign the lowest priority to $J_i^j$.

Then,

4. Execute the $m$ highest-priority jobs, i.e., $C_i^{(t)} \leftarrow (C_i^{(t)} - 1)$, and then remove $J_i^j$ from its queue if $C_i^{(t)} = 0$.

Step 3 implies that the job moved to $Q_L$ at time instant $t$ will never miss its deadline, because the number of remaining executions will be successfully performed in the remaining contention-free slots that exists before the deadline of $J_i^j$. Then the remaining issue is how to calculate the lower-bounded number of contention-free slots that exist in the interval $[r_k^j, d_k^j)$ of each job of a task $\tau_k$. $\Phi_k$ is derived as follows; note that two methods were introduced in Lee et al. (2014) to lower-bound contention-free slots, but we use the approach of Lemma 3.4 in Lee et al. (2014) because it outperforms that of Lemma 3.3 in Lee et al. (2014) in most cases.

**Lemma 1.** *(from Lemma 3.4 in Lee et al., 2014) For an active job of a task $\tau_k$ scheduled by A-CF, there are at least $\Phi_k$ contention-free slots between the release and the deadline of the active job, which is computed as follows.*

$$\Phi_k = \max\left(0, D_k - \left\lfloor \frac{C_k + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} W_i(D_k)}{m} \right\rfloor\right). \tag{3}$$

**Proof.** We summarize the proof of Lemma 3.4 in Lee et al. (2014). When we limit our attention to the interval $[r_k^j, d_k^j)$ of an active job $J_k^j$, the upper-bounded workload of a task $\tau_i$ in the interval is $W_i(D_k)$. By Definition 1, at least $m$ executions of active jobs are needed for a time slot to be contending. Thus, there are at least $D_k - \min\left(D_k, \left\lfloor \frac{\sum_{\tau_i \in \tau} W_i(D_k)}{m} \right\rfloor\right)$ contention-free slots in the interval $[r_k^j, d_k^j)$. Then, the workload $W_k(D_k)$ can be reduced to $C_k$ because the number of jobs of a task $\tau_k$ invoked in the interval $[r_k^j, d_k^j)$ is limited to one, unlike the other jobs invoked by $\tau_i \in (\tau \setminus \tau_k)$. Thus, the lemma holds. □

### 3.2. RTA framework on a multi-processor platform

RTA has been a popular schedulability analysis framework owing to its applicability and analytic performance on schedulability. For a given interval $\ell$, RTA focuses on $J_k^j$ of interest of $\tau_k$ and calculates $I_k^i(r_k^j, r_k^j + \ell)$ to derive the upper-bounded response time of task $\tau_k$, which is denoted by $R_k^{ub}$. Because a job cannot execute in a time slot if $m$ other higher-priority jobs execute, $(\sum_{\tau_i \in \tau - \{\tau_k\}} I_k^i(r_k^j, r_k^j + \ell))/m$ represents the length of cumulative intervals in $[r_k^j, r_k^j + \ell)$ such that $J_k^j$ cannot execute owing to the executions of other jobs. Therefore, if the value is no larger than $\ell - C_k$, $J_k^j$ can finish its execution at or before $r_k^j + \ell$. Based on this reasoning, RTA tests the schedulability of $\tau_k$ as follows; we paraphrase Lemma 2 in Lee (2017), but the same is also shown with different descriptions in Theorem 3 in Bertogna and Cirinei (2007), Theorem 1 in Lee (2014), and Eq. (1) in Lee and Shin (2014).

**Lemma 2.** *(from Lemma 2 in Lee, 2017) A task $\tau_k \in \tau$ is schedulable, if every job $J_k^j$ satisfies the following for $C_k \leq \ell \leq D_k$.*

$$C_k + \left\lfloor \frac{1}{m} \sum_{\tau_i \in (\tau \setminus \tau_k)} min\left(I_k^i(r_k^j, r_k^j + \ell), \ell - C_k + 1\right) \right\rfloor \leq \ell. \tag{4}$$

**Proof.** Let us consider a value $X = C_k + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} I_k^i(r_k^j, r_k^j + \ell) \right\rfloor$. $X$ represents the duration between $r_k^j$ and $f_k^j$ for a given $\ell$, i.e., WCET of $J_k^j$ plus interference on $J_k^j$, because a job cannot execute in a time slot if $m$ other higher-priority jobs execute. By the definition of $I_k^i(r_k^j, r_k^j + \ell)$, if $I_k^i(r_k^j, r_k^j + \ell) > \ell - C_k + 1$ for certain tasks $\tau_i$, $J_k^j$ never finishes its execution in $[r_k^j, r_k^j + \ell)$. Thus, if $X$ is strictly larger than $\ell$, the LHS is also strictly larger than $\ell$. By the contra-position, the lemma holds. □

The remaining issue is how to find such a value of $\ell$ and an upper-bound $I_k^i(r_k^j, r_k^j + \ell)$. The existing study proved that the amount of interference of $\tau_i$ on $\tau_k$ in an interval can be upper-bounded by the worst-case workload of $\tau_i$ in the interval (Bertogna and Cirinei, 2007). Thus, it satisfies $I_k^i(r_k^j, r_k^j + \ell) \leq W_i(\ell)$.

Then, RTA works as follows using Equation (4) by substituting $I_k^i(r_k^j, r_k^j + \ell)$ of the LHS into $W_i(\ell)$. Initially, $\ell$ is set to $C_k$ and RTA tests whether the inequality holds. If the inequality holds, the task is deemed schedulable. Otherwise, RTA resets $\ell$ to the previous value of the LHS of the inequality, until the inequality holds or $\ell > D_k$; $\ell > D_k$ represents that $\tau_k$ is deemed unschedulable. If the inequality holds, $\tau_k$ is deemed schedulable, and the value of $\ell$ satisfying the inequality is $R_k^{ub}$, meaning that $R_k^{ub} \leq D_k$ holds.

## 4. Proposed schedulability analysis for CF scheduling

In this section, we propose PRTA, a new schedulability analysis framework for an algorithm A adopting the CF policy (referred to as *A*-CF), which effectively identifies the time when the priority demotion occurs (i.e., schedulability is guaranteed) by the CF policy. To this end, we first define a new notion of pseudo-response time by investigating the characteristics of a task $\tau_i$ under *A*-CF in order to effectively judge the task's schedulability, and discuss its properties. Then, we derive a new schedulablity condition for a task $\tau_i$, which exploits a pseudo-response time of $\tau_i$. We also propose a new notion of pseudo-slack time, which can significantly improve the performance of PRTA.

### 4.1. Pseudo-response time

As we discussed in Section 1, the existing RTA is not capable of identifying the time instant when the priority is demoted under the CF policy. Therefore, we need to define a new criterion to effectively determine the schedulability of tasks scheduled by *A*-CF. To this end, we consider the following two types of $\tau_k$:

1) $\tau_k$ with $\Phi_k = 0$, whose jobs do not migrate to $Q_L$, and
2) $\tau_k$ with $\Phi_k \neq 0$, whose jobs may (or may not) migrate to $Q_L$ during its execution.

For $\tau_k$ of type 1), it is schedulable if $f_k^j$ of every $J_k^j$ is earlier than or equal to $d_k^j$. For $\tau_k$ of type 2), it is schedulable if $f_k^j$ of every $J_k^j$ not migrating to $Q_L$ is earlier than or equal to $d_k^j$. The other jobs migrating to $Q_L$ during its execution are guaranteed to be schedulable, owing to the implication of Step 3 of the CF policy mentioned in Section 3.1. This implies that $\tau_k$ is schedulable if the priority-demotion time or $f_k^j$ of every $J_k^j$ is earlier than or equal to $d_k^j$.

To express the condition of schedulability of a task $\tau_k$ scheduled by *A*-CF, we define new notions of pseudo-finishing time $\tilde{f}_k^j$ of $J_k^j$ and pseudo-response time $\tilde{R}_k$ of $\tau_k$ as follows.

**Definition 2.** (Pseudo-finishing time): The pseudo-finishing time $\tilde{f}_k^j$ of a job $J_k^j$ is a migration time $t'$ if $J_k^j$ migrates to $Q_L$ or a finishing time $f_k^j$ if $J_k^j$ does not.

**Definition 3.** (Pseudo-response time): The pseudo-response time $\tilde{R}_k$ of a task $\tau_k$ is the largest value among all $(\tilde{f}_k^j - r_k^j)$ of all the jobs of $\tau_k$.

Let $J_k^+$ be the job inducing $\tilde{R}_k$. By the definition of pseudo-response time, $\tilde{R}_k = \tilde{f}_k^+ - r_k^+$. Note that $J_k^+$ and $J_k^*$ can be different jobs, because $\tilde{R}_k$ and $R_k$ can be induced by different jobs.

By the definition of the pseudo-finishing time, each job $J_k^j$ of a task $\tau_k$ scheduled by *A*-CF migrates to $Q_L$, or finishes, at $\tilde{f}_k^j$. $J_k^j$ is schedulable if it migrates to $Q_L$ or finishes before $d_k^j$. For a job $J_k^j$ scheduled by *A*-CF, it is schedulable if $\tilde{f}_k^j$ is earlier than or equal to $d_k^j$.

Then, we show that pseudo-response time can be a new criterion to test the schedulability of $\tau_k$ scheduled by *A*-CF as follows. By the definition of pseudo-response time and $J_k^+$, all $J_k^j$ are schedulable if $J_k^+$ is schedulable. Because $\tilde{R}_k = \tilde{f}_k^+ - r_k^+$ and $D_k = d_k^+ - r_k^+$, the relationship $(\tilde{f}_k^+ \le d_k^+) = (\tilde{f}_k^+ - r_k^+ \le d_k^+ - r_k^+) = (\tilde{R}_k \le D_k)$ holds. Based on this reasoning, all jobs are schedulable if $\tilde{R}_k \le D_k$. Therefore, for a task $\tau_k$ scheduled by *A*-CF, it is schedulable if $\tilde{R}_k$ is less than or equal to $D_k$.

### 4.2. PRTA: pseudo-response time analysis

Now, we propose PRTA, which effectively incorporates the notion of pseudo-response time into schedulability analysis. The challenging issue for deriving a better schedulability condition for tasks scheduled by *A*-CF is how to tightly derive an upper-bounded pseudo-response time denoted by $\tilde{R}_k^{ub}$. As shown in Lemma 2, the upper-bounded response time $R_k^{ub}$ is derived by the summation of WCET and the worst-case interference on $\tau_k$ in the interval of $R_k^{ub}$. Following the principle of the derivation of the upper-bounded response-time, we derive $\tilde{R}_k^{ub}$ by using the following two values:

- WCET performed in the interval $[r_k^j, \tilde{f}_k^j)$, and
- the worst-case interference on $J_k^j$ in the interval $[r_k^j, \tilde{f}_k^j)$.

WCET of $J_k^j$ performed in the interval $[r_k^j, \tilde{f}_k^j)$ is bounded by $C_k$ as we consider implicit- or constrained-deadline tasks.

Before we derive the worst-case interference on $J_k^j$ in the interval $[r_k^j, \tilde{f}_k^j)$, we derive the upper-bounded worst-case interference of $J_i^j$ on $J_k^j$ in the interval $[r_k^j, \tilde{f}_k^j)$ based on the observations regarding the behavior of jobs scheduled by *A*-CF. By the definition of the pseudo-finishing time, $J_k^j$ migrates from $Q_H$ to $Q_L$ or finishes its execution in $Q_H$, at $\tilde{f}_k^j$. This implies that $J_k^j$ stays in $Q_H$ before $\tilde{f}_k^j$. Thus, we focus on how job $J_i^j$ scheduled by *A*-CF interferes with $J_k^j$ in $Q_H$, because $J_i^j$ in $Q_L$ cannot interfere with $J_k^j$ in $Q_H$.

To this end, we explicitly show that job $J_i^j$ scheduled by *A*-CF can interfere with any job $J_k^j$ in $Q_H$ during at most $C_i - \Phi_i$ time slots. We first consider the case in which $J_i^j$ stays in $Q_H$ until $d_i^j$, for any job $J_i^j$ scheduled by *A*-CF. In this case, $J_i^j$ faces at least $\Phi_i$ contention-free slots because there are at least $\Phi_i$ contention-free slots in the interval $[r_i^j, d_i^j)$. Because no job interferes with other jobs in contention-free slots, $J_i^j$ interferes with any job in $Q_H$ during at most $C_i - \Phi_i$ time slots. Then, we consider the other case, in which $J_i^j$ migrates from $Q_H$ to $Q_L$ at time $t'$, i.e., $C_i^{(t')} = \Phi_i^{(t')}$. Before $t'$, $J_i^j$ executes in $C_i - C_i^{(t')}$ time slots and faces exactly $\Phi_i - \Phi_i^{(t')}$ contention-free slots. Hence, $C_i - C_i^{(t')} - (\Phi_i - \Phi_i^{(t')}) = C_i - \Phi_i$ executions of $J_k^j$ in $Q_H$ interfere with any job in $Q_H$. After $t'$, $J_i^j$ cannot interfere with any job in $Q_H$, because $J_i^j$ is in $Q_L$. Therefore, a job $J_i^j$ scheduled by *A*-CF can interfere with any job $J_k^j$ in $Q_H$ during at most $C_i - \Phi_i$ time slots. This reasoning indicates that the upper-bounded amount of interference of any job $J_i^j$ on $J_k^j$ in the interval $[r_k^j, \tilde{f}_k^j)$ is at most $C_i - \Phi_i$ time slots, because $J_k^j$ stays in $Q_H$ before $\tilde{f}_k^j$ as, at $\tilde{f}_k^j$, $J_k^j$ migrates to $Q_L$ or finishes its execution in $Q_H$ by the definition of the pseudo-finishing time (Definition 2).

Then, we derive $\tilde{R}_k^{ub}$ based on the RTA framework with a specialized task set. Let us consider a modified task set $\hat{\tau}$ from $\tau$, in which the parameters of the task whose schedulability will be tested (i.e., $\tau_k$) are not changed, and WCET of the other tasks that will be regarded as those interfering with $\tau_k$ (i.e., $\tau_i$) is deducted by $\Phi_i$. We assume that $\hat{\tau}$ is always scheduled by *A* rather than *A*-CF. We will use the term $\hat{\tau}$ when $\hat{\tau}$ is considered (e.g., $\hat{\tau}_k$, $T_k(\hat{\tau})$, $r_k^j(\hat{\tau}_k)$, $W_i(\ell, \hat{\tau})$, etc.). Thus, $\hat{\tau}_k = \{T_k(\hat{\tau}) = T_k, C_k(\hat{\tau}) = C_k, D_k(\hat{\tau}) = D_k\}$ and $\tau_i = \{T_i(\hat{\tau}) = T_i, C_i(\hat{\tau}) = C_i - \Phi_i, D_i(\hat{\tau}) = D_i\}$ (*i.e.*, $k \neq i$) hold.

Next, we derive the upper-bounded response time of $\hat{\tau}_k \in \hat{\tau}$ scheduled by *A* instead of $\tau_k \in \tau$ scheduled by *A*-CF. Although the result may not be the upper-bounded response time of $\tau_k \in \tau$ scheduled by *A*-CF, it can be the safe upper-bound of its pseudo-response time. This is because we already showed that the amount of execution of $J_k$ before the pseudo-finishing time is upper-bounded by $C_k$, and $J_i$ cannot interfere with $J_k$ in $Q_H$ for more than $C_i - \Phi_i$ as long as the CF policy is applied, regardless of when $\tau_i$'s priority is demoted (even if it is not demoted). Note that we do not have to consider $J_k$ in $Q_L$, because the schedulability of $J_k$ in $Q_L$ is guaranteed by the CF policy.

In other words, we show that $R_k^{ub}(\hat{\tau})$ (instead of $\tilde{R}_k^{ub}$) sufficiently upper-bounds $\tilde{R}_k$, meaning that we can judge the schedulability of $\tau_k$ scheduled by *A*-CF using $R_k^{ub}(\hat{\tau})$ as follows.

**Theorem 1.** *A task $\tau_k$ scheduled by A-CF is schedulable, if every job $J_k^j(\hat{\tau})$ scheduled by A satisfies the following for $C_k(\hat{\tau}) \le \ell \le D_k(\hat{\tau})$:*

$$C_k(\hat{\tau}) + \left\lfloor \frac{1}{m} \sum_{\tau_i(\hat{\tau}) \in (\hat{\tau} \setminus \tau_k(\hat{\tau}))} min\Big(I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell), \ell - C_k(\hat{\tau}) + 1\Big) \right\rfloor \le \ell. \tag{5}$$

**Proof.** $J_k^j$ is in $Q_H$ when it executes or is interfered by $J_i^j$ before $\tilde{f}_k^j$. Every job is scheduled by base algorithm *A* when it is in $Q_H$ (before the priority of the job is demoted). Thus, $J_k^j$ is scheduled by *A* when it executes and is interfered by $J_i^j$ in $Q_H$. In other words, $J_k^j$ is scheduled by *A* before $\tilde{f}_k^j$. Trivially, the amount of execution of $J_k^j$ before $\tilde{f}_k^j$ is upper-bounded by $C_k$. Then, it is same as that of $J_k^j(\hat{\tau})$ scheduled by *A* before $f_k^j(\hat{\tau})$ as a single job scheduled by *A* fully executes before $f_k^j(\hat{\tau})$. We already showed that the amount of interference of $J_i^j$ on $J_k^j$ before $\tilde{f}_k^j$ is upper-bounded by $C_i - \Phi_i$. Then, it is the same as that of $J_k^j(\hat{\tau})$ scheduled by *A* (without the CF policy) before $f_k^j(\hat{\tau})$ as $J_i^j(\hat{\tau})$ executes for at most $C_i - \Phi_i$ by the definition of $\hat{\tau}_i$. Thus, this theorem holds. □

Based on Theorem 1, PRTA judges the schedulability of a task $\tau_k$ scheduled by *A*-CF with $\ell$ and the upper-bounded value of $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$. As $I_k^i(r_k^j, r_k^j + \ell)$ can be upper-bounded by $W_i(\ell)$ (Bertogna and Cirinei, 2007), we derive $W_i(\ell, \hat{\tau})$ by substituting $C_i$ of Equation (1) into $C_i(\hat{\tau}) = C_i - \Phi_i$ and use it to upper-bound $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$.

Fig. 3(a) describes the worst-case scenario for $W_i(\ell, \hat{\tau})$ under any preemptive scheduling algorithm *A*. The configuration of the execution of each job is the same as the worst-case scenario of $W_i(\ell)$, but $J_i^j(\hat{\tau})$ executes for $C_i(\hat{\tau}) = C_i - \Phi_i$.

Then, PRTA works with the same procedure of the RTA framework using Eq. (5) by substituting $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ to the LHS in $W_i(\ell, \hat{\tau})$. If $\ell > D_k(\hat{\tau})$ holds, then $\tau_k(\hat{\tau})$ is deemed unschedulable under *A* meaning $\tau_k$ is deemed unschedulable under *A*-CF. Otherwise, $\tau_k(\hat{\tau})$ is deemed schedulable under *A* meaning $\tau_k$ is deemed

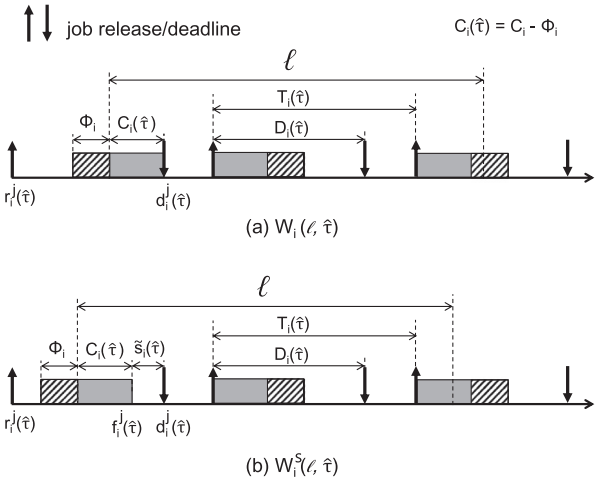



**Fig. 3.** Two worst-case scenarios for $W_i(\ell, \hat{\tau})$ and $W_i^s(\ell, \hat{\tau})$.

schedulable under *A*-CF, and the value of $\ell$ satisfying the inequality is $R_k^{ub}(\hat{\tau}) = \tilde{R}_k^{ub}$; this eventually represents $R_k^{ub}(\hat{\tau}) = \tilde{R}_k^{ub} \le D_k(\hat{\tau}) = D_k$.

### 4.3. Exploiting pseudo-slack time

In this subsection, we show that the performance of PRTA can be further improved using a new notion of pseudo-slack time.

We first show that the amount of the worst-case interference of $\tau_i$ on $\tau_k$ in a given interval of length $\ell$ is reduced with a new notion of pseudo-slack time. Suppose that the schedulability of a given task $\tau_i$ is tested by PRTA, resulting in $\tilde{R}_i^{ub}$, satisfying $\tilde{R}_i^{ub} \le D_i$. This indicates that job $J_i^j$ of task $\tau_i$ under *A*-CF finishes its execution or migrates to $Q_L$ at least $D_k - \tilde{R}_i^{ub}$ time slots ahead of $d_i^j$ because $\tilde{R}_i^{ub}$ represents the worst-case pseudo-finishing time of job $J_i^j$ of task $\tau_i$. Here, we define a new notion of the pseudo-slack time of task $\tau_i$, denoted by $\tilde{s}_i$ as follows.

**Definition 4** (Pseudo slack time) : For task $\tau_i$ scheduled by *A*-CF, the pseudo-slack time $\tilde{s}_i$ of task $\tau_i$ is defined as the difference between the upper-bounded pseudo-response time $\tilde{R}_i^{ub}$ and the relative deadline $D_i$ of the task, which exists only if $\tau_i$ is deemed schedulable under the CF policy.

In the previous subsection, we showed that $D_i = D_i(\hat{\tau})$ and $\tilde{R}_i^{ub} = R_i^{ub}(\hat{\tau})$. Thus, we can calculate the pseudo-slack time $\tilde{s}_i$ of each task scheduled by *A*-CF by calculating the difference between $R_i^{ub}(\hat{\tau})$ and $D_i(\hat{\tau})$ under scheduling algorithm *A*.

Using this difference, we can reduce the worst-case workload $W_i(\ell, \hat{\tau})$ and use it to upper-bound the interference of $\tau_i$ on $\tau_k$. Let $W_i^s(\ell, \hat{\tau})$ be the worst-case workload of a task $\tau_i(\hat{\tau})$ considering the pseudo-slack time of $\tau_i$. Fig. 3(b) illustrates the worst-case scenario for $W_i^s(\ell, \hat{\tau})$ in the given interval of $\ell$. As seen in Fig. 3(b), the leftmost job $J_i^j(\hat{\tau})$ begins its execution at the beginning of the interval $\ell$, executes for $C_i - \Phi_i$ without any interference or delay, and finishes at $f_i^j(\hat{\tau}) = d_i^j(\hat{\tau}) - \tilde{s}_i(\hat{\tau})$. Thereafter, the following jobs are released and scheduled as soon as possible. Compared with $W_i(\ell, \hat{\tau})$, $\ell$ of $W_i^s(\ell, \hat{\tau})$ moves to the left by the amount of $\tilde{s}_i(\hat{\tau})$, such that the execution of the rightmost job can be reduced by at most the amount of $\tilde{s}_i(\hat{\tau})$. Then, $W_i^s(\ell, \hat{\tau})$ is calculated as follows.

$$W_i^s(\ell, \hat{\tau}) = N_i^s(\ell, \hat{\tau}) \cdot (C_i - \Phi_i) + min\Big(C_i - \Phi_i, \ell + D_i - (C_i - \Phi_i) - \tilde{s}_i - N_i^s(\ell, \hat{\tau}) \cdot T_i\Big), \tag{6}$$

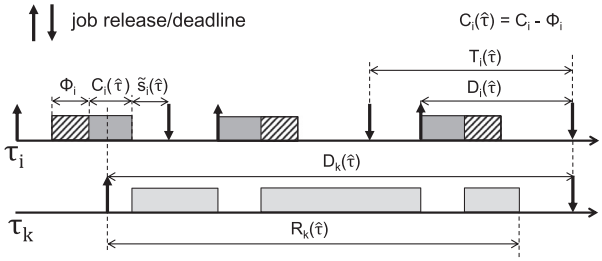



**Fig. 4.** Worst-case scenario for $E_i^s(D_k(\hat{\tau}), \hat{\tau})$.

where $N_i^s(\ell, \hat{\tau})$ is the number of jobs fully executing for $(C_i - \Phi_i)$, calculated by

$$N_i^s(\ell, \hat{\tau}) = \left\lfloor \frac{\ell + D_i - (C_i - \Phi_i) - \tilde{s}_i}{T_i} \right\rfloor. \tag{7}$$

PRTA then exploits the pseudo-slack value $\tilde{s}_i$ as follows.

1. Initially, $\tilde{s}_i$ of every task is set to zero, and $\tilde{R}_i^{ub}$ of every task is calculated.
2. We reset every $\tilde{s}_i$ of every schedulable task to $D_i - \tilde{R}_i^{ub}$ if $\tilde{R}_i^{ub} \le D_i$ holds.
3. We repeat calculating $\tilde{R}_i^{ub}$ of every task, until all tasks are deemed schedulable (schedulable task set) or there is no slack value update (unschedulable task set).

## 5. PRTA for EDF-CF and RM-CF

In this section, we apply our PRTA approach to the two scheduling algorithms EDF-CF and RM-CF. Although Eq. (6) can safely upper-bound the worst-case interference of $\tau_i$ on $\tau_k$ scheduled by *A*-CF, there is a room to reduce it when it comes to EDF-CF. Let $E_i^s(D_k, \hat{\tau})$ be the upper-bounded interference of $\tau_i(\hat{\tau})$ to a job $J_k^j(\hat{\tau})$ of $\tau_k(\hat{\tau})$ scheduled by EDF-CF in the interval of $D_k(\hat{\tau})$, which considers the pseudo-slack time. As shown in Fig. 4, $E_i^s(D_k, \hat{\tau})$ is maximized when the rightmost job of $\tau_i(\hat{\tau})$ in the interval of $D_k(\hat{\tau})$ is aligned with the deadline of $J_k^j(\hat{\tau})$ because a job $J_i^j(\hat{\tau})$ with a later absolute deadline cannot interfere with $J_k^j(\hat{\tau})$. With this reasoning, we derive $E_i^s(D_k, \hat{\tau})$ as follows.

$$E_i^s(D_k, (\hat{\tau})) = \left\lfloor \frac{D_k}{T_i} \right\rfloor (C_i - \Phi_i) + \min\left(C_i - \Phi_i, D_k - \left\lfloor \frac{D_k}{T_i} \right\rfloor T_i - \tilde{s}_i\right). \tag{8}$$

We then derive a tighter schedulability analysis condition for EDF-CF as follows.

**Theorem 2.** *Task $\tau_k$ scheduled by EDF-CF is schedulable, if every job $J_k^j(\hat{\tau})$ scheduled by A satisfies the following for $C_k(\hat{\tau}) \le \ell \le D_k(\hat{\tau})$:*

$$C_k(\hat{\tau}) + \left\lfloor \frac{1}{m} \sum_{\tau_i(\hat{\tau}) \in (\hat{\tau} \setminus \tau_k(\hat{\tau}))} min(W_i^s(\ell, (\hat{\tau})), E_i^s(D_k, (\hat{\tau})), \ell - C_k(\hat{\tau}) + 1) \right\rfloor \le \ell. \tag{9}$$

**Proof.** As the amount of $W_i^s(\ell, \hat{\tau})$ varies according to the length of $\ell$, we use the smaller value between $W_i^s(\ell, \hat{\tau})$ and $E_i^s(D_k, \hat{\tau})$ to upper-bound the worst-case interference of a job $J_i^j$ on a job $J_k^j$. Then, this theorem holds by Theorem 1. □

We now discuss some characteristics of PRTA for EDF-CF, including the time complexity and dominance relation between PRTA and the existing schedulability analysis. First, the time complexity of PRTA is derived as follows. $W_i^s(\ell, \hat{\tau})$ and $E_i^s(D_k, \hat{\tau})$ are computed in a constant time, and these are computed $n$ times because we

consider all $\tau_i$ in Eq. (9) (i.e., $O(n)$). Then, the LHS of Eq. (9) is computed recursively until Eq. (9) holds for each task, and $\ell$ cannot be larger than $D_k$ (i.e., $O(D_{max})$ where $D_{max}$ is the largest value among all the tasks' deadlines). Such computation is conducted for every task (i.e., $O(n)$). Thereafter, exploiting the pseudo-slack value of each task is conducted until the values of all the tasks are converged, and each value cannot be larger than each task's relative deadline (i.e., $O(D_{max})$). Thus, the time complexity of PRTA is $O(n^2 {D_{max}}^2)$, which is the same as that of the existing RTA.

Then, we derive a dominance relation between PRTA for EDF-CF and two the existing schedulability analysis with the following theorem.

**Theorem 3.** *PRTA for EDF-CF strictly dominates not only RTA for EDF but also DA for EDF-CF.*

**Proof.** By comparing Eq. (4) of RTA and Eq. (5) of PRTA, we can easily see that PRTA finds $\ell$ no larger than $\ell$ of RTA from Equation (5), because $C_i(\hat{\tau})$ is not smaller than $C_i$, resulting in $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell) \leq I_k^i(r_k^j, r_k^j + \ell)$. Hence, PRTA for EDF-CF dominates RTA for EDF. Additionally, PRTA considers the interference of $\tau_i$ (actually $\tau_i(\hat{\tau})$) on $\tau_k$ within the worst-case pseudo-response time, whereas DA considers it within the deadline. Thus, PRTA for EDF-CF dominates DA for EDF-CF. □

As RM is fixed-priority scheduling, a priority is assigned to a task rather than each job, and only the tasks with priories higher than that of a task $\tau_k$ can interfere with a job $J_k^j$ of task $\tau_k$. Thus, we derive the following theorem.

**Theorem 4.** *Task $\tau_k$ scheduled by RM-CF is schedulable, if every job $J_k^j(\hat{\tau})$ satisfies the following for $C_k(\hat{\tau}) \leq \ell \leq D_k(\hat{\tau})$:*

$$C_k(\hat{\tau}) + \left\lfloor \frac{1}{m} \sum_{\tau_i(\hat{\tau}) \in hep(\tau_k(\hat{\tau}))} min(W_i^s(\ell, \hat{\tau}), \ell - C_k(\hat{\tau}) + 1) \right\rfloor \leq \ell. \tag{10}$$

*where $hep(\tau_k(\hat{\tau}))$ is a set of tasks with priorities higher than $\tau_k(\hat{\tau})$.*

## 6. Evaluation

In this section, we evaluate the performance of our approach compared to the existing techniques. For performance metrics, we measure how many randomly generated task sets are deemed schedulable by each schedulability analysis.

### 6.1. Experiment environment

In order to evaluate the performance of the considered techniques, we randomly generate task sets using a task set generation method used in many of the existing studies (Bertogna et al., 2009; Baker, 2005; Andersson et al., 2008). We consider two types of task sets, in which tasks have implicit and constrained deadlines respectively, and four different numbers of processors, i.e., $m \in \{2, 4, 8, 16\}$. The utilization ($C_i/T_i$) of each task is determined by a bimodal or exponential distribution with their input parameters selected in {0.1, 0.3, 0.5, 0.7, 0.9} (Lee et al., 2014). A value for $C_i/T_i$ is uniformly selected in [0, 0.5) and [0.5, 1) with probability $p$ and $1 - p$, respectively, for a given bimodal parameter $p$, and the value is selected according to the exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$ for a given exponential parameter $1/\lambda$. For each task, $T_i$ is uniformly chosen in [1,1000], $C_i$ is chosen with the bimodal or exponential parameter, and $D_i$ is uniformly chosen in $[C_i, T_i]$. Based on the parameters, the task sets are generated as follows:

1. Initially, a set of $m + 1$ tasks is generated.
2. Next, the generated task set is tested to ascertain whether it passes the necessary feasibility condition in Baker and Cirinei (2006).
3. If it fails the test, the task set is discarded and return to Step 1. Otherwise, this set is included for evaluation. Then, we use this set as a basis for the next new task set; we create a new task set by adding a new task into the existing task set and return to Step 2.

The idea behind the necessary feasibility condition in Step 2 implies that, if $U_{sys}$ of a task set is greater than $m$, the task set cannot be schedulable with any scheduling algorithm. We generate 10,000 task sets for each bimodal or exponential distribution with their individual input parameters (e.g., bimodal distribution with 0.1), each individual value of $m$ (e.g., $m = 2$), and each type of deadline (e.g., implicit deadline). Because we consider ten different distribution settings (e.g., bimodal or exponential distribution with five different input values), four different values of $m$ (e.g., 2, 4, 8, or 16), and two types of deadline (e.g., implicit or constrained), we generate $10{,}000 \cdot 10 \cdot 4 \cdot 2 = 800{,}000$ task sets in total.

Then, we measure the performance of PRTA for EDF-CF compared to the existing techniques by investigating how many task sets are deemed schedulable by each technique. We consider the following schedulability analysis techniques.

- DA{EDF} : DA test for EDF proposed in Bertogna et al. (2009, 2005),
- RTA{EDF} : RTA test for EDF with slack reclamation proposed in Bertogna and Cirinei (2007),
- DA{EDF-CF} : DA test for EDF incorporating the CF policy proposed in Lee et al. (2014), and
- PRTA{EDF-CF} : PRTA test for EDF incorporating the CF policy with slack reclamation proposed in Section 5.

### 6.2. Example of a task set: ACSW in satellite system

In this subsection, we justify the practicability of task sets synthetically generated by the previous subsection. To this end, we show an actual real-time system and the task parameters of its real-time tasks.

A satellite system is a compelling example of a huge-size real-time system. Among various satellite systems, we discuss a reconnaissance satellite system equipped with a reconnaissance antenna that transmits and receives radio frequency signals to obtain an image of the target terrain even in cloudy weather or at night. Antenna controller software (ACSW) (Baek et al., 2018a) controls a reconnaissance antenna in a satellite system, in which tasks are scheduled by RM on a space-specific RTOS (real-time operating system) called RTEMS (real-time executive for multi-processor systems) (RTEMS). ACSW conducts five tasks named tHigh, tMilbus, tOne, tTwo and tSync, respectively, whose main roles are described as follows.

- tHigh takes each macro command (MCMD) in an MCMD queue, which is received from the ground station and set out proper information in each job of the other tasks.
- tMilbus receives an MCMD via MIL-STD-1553B protocol (Excalibur) from the ground station and verifies integrity of each received MCMD by utilizing various verification mechanisms such as CRC (cyclic redundancy check) before it is inserted into a MCMD queue.
- tOne conducts all jobs required for internal mode transitions of a satellite system, such as turning on/off relevant equipment and transmitting internal telemetries via SpaceWire protocol (European).
- tTwo executes a number of work such as FDIR (fault detection, isolation and recovery), formatting network packets containing

**Table 2**
Task parameter (millisecond base) of ACSW.

| | $T_i$ | $D_i$ | WCET | BCET | ACET |
|---|---|---|---|---|---|
| tHigh | 62.5 | 50 | 2.98 | 0.08 | 0.14 |
| tMilbus | 125 | 100 | 0.54 | 0.11 | 0.21 |
| tOne | 250 | 200 | 30.08 | 0.05 | 0.29 |
| tTwo | 500 | 400 | 231.72 | 37.7 | 147.5 |

information of operation results and current status of the system to be transmitted to the ground station.
- tSync conducts operation preparation whenever it has surplus computing resources.

Task parameters of ACSW are described in Table 2; $T_i$ and $D_i$ are determined by a system designer, and BCET (the best-case execution time), WCET, ACET (the average-case execution time) were measured in various actual operation scenarios (e.g., standard mode and high resolution mode) under the target system that is equipped with 256 MB SDRAM, a multi-processor platform based on FT Leon3 CPU architecture (80 Mhz clock speed), and SPARC BSP (Cobham) supporting MIL-STD-1553B (for external communication) and SpaceWire (for internal communication) protocols. tSync does not have a period and a relative deadline since it executes without any deadline whenever the other tasks are not active.

Each task's period is determined by the cycle of each task's main role: checking the MCMD queue to take an MCMD in every 62.5 ms, receiving an MCMD from the ground station in every 125 ms, turning on/off individual equipment regarding mode transitions in every 250 ms, and transmitting system status to the ground station in every 500 ms.

As the task set generation method considered in the previous section produces a number of task sets assuming random task parameters, it covers various real-time embedded systems in which tasks conduct different roles in various operation scenarios.

### 6.3. Schedulability evaluation

For the discussion of the considered schedulability analysis tests' performance, we first identify how many task sets are deemed schedulable by each schedulability analysis technique and compare them to each other for different numbers of processors (i.e., $m$ from 2 to 16), as shown in Table 3(a) and (b). Then, we investigate how the schedulability performance of each schedulability analysis technique is influenced by the various average numbers of tasks in a task set (denoted by $\overline{n}$) and average task utilization of tasks in a task set (denoted by $\overline{C_i/T_i}$). Fig. 5 presents this (for implicit deadline task sets for $m = 2$ and 16) by considering bimodal distributions with 0.9 and exponential distributions with 0.1 and 0.9; among ten task utilization distributions (bimodal and exponential utilization distributions with five input parameters), we consider three that produce well-separated $\overline{n}$ (the smallest, largest, and medium) and $\overline{C_i/T_i}$ (the largest, smallest, and medium). Each subfigure in Fig. 5 plots the number of task sets deemed schedulable by each schedulability test against the varying task set utilization ($U_{sys} \triangleq \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) on a given task utilization distribution. TOT represents the number of generated task sets with the corresponding utilization distribution.

We first interpret the following observations of implicit deadline task sets, which are obtained from Table 3(a) and Fig. 5.

*O*1. The number of task sets deemed schedulable by RTA{EDF} is substantially higher than that of DA{EDF} for all values of $m$, and it is also higher than that of DA{EDF-CF} for $m = 2$, 4, and 8 (from Table 3(a)).
*O*2. The performance of DA{EDF-CF} is higher than that of RTA{EDF} for $m = 16$ (from Table 3(a)).

**Table 3**
The number of task sets deemed schedulable by individual techniques.

| $m$ | DA{EDF} | RTA{EDF} | DA{EDF-CF} | PRTA{EDF-CF} | PRTA{EDF-CF} / RTA{EDF} | PRTA{EDF-CF} / DA{EDF-CF} |
|---|---|---|---|---|---|---|
| 2 | 20999 | 47033 | 36929 | 50106 | 106.5% | 135.6% |
| 4 | 11528 | 32779 | 28227 | 38657 | 117.9% | 136.9% |
| 8 | 6261 | 23807 | 23637 | 32120 | 134.9% | 135.8% |
| 16 | 3351 | 17706 | 21521 | 28094 | 158.6% | 130.5% |

(a) EDF, implicit deadline

| $m$ | DA{EDF} | RTA{EDF} | DA{EDF-CF} | PRTA{EDF-CF} | PRTA{EDF-CF} / RTA{EDF} | PRTA{EDF-CF} / DA{EDF-CF} |
|---|---|---|---|---|---|---|
| 2 | 9872 | 34251 | 27659 | 40519 | 118.3% | 146.4% |
| 4 | 4604 | 19783 | 20301 | 27678 | 139.9% | 136.3% |
| 8 | 2114 | 11996 | 16844 | 21249 | 177.1% | 126.1% |
| 16 | 936 | 7592 | 15357 | 18038 | 237.5% | 117.4% |

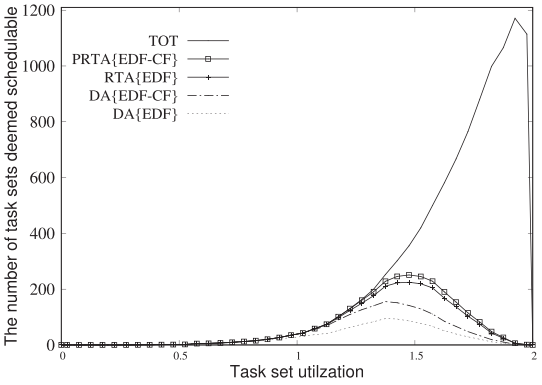
(b) EDF, constrained deadline

| $m$ | DA{RM} | RTA{RM} | DA{RM-CF} | PRTA{RM-CF} | PRTA{RM-CF} / RTA{EDF} | PRTA{RM-CF} / DA{EDF-CF} |
|---|---|---|---|---|---|---|
| 2 | 47581 | 58183 | 50063 | 59258 | 101.8% | 118.3% |
| 4 | 35425 | 42232 | 40233 | 45181 | 106.9% | 112.2% |
| 8 | 27074 | 31953 | 34601 | 37511 | 117.3% | 108.4% |
| 16 | 20636 | 24292 | 30728 | 32599 | 134.1% | 106.0% |

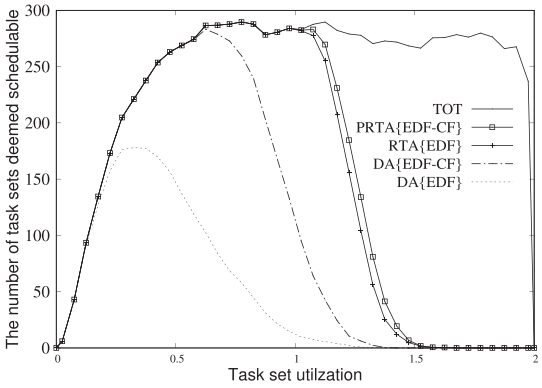
(c) RM, implicit deadline

| $m$ | DA{RM} | RTA{RM} | DA{RM-CF} | PRTA{RM-CF} | PRTA{RM-CF} / RTA{EDF} | PRTA{RM-CF} / DA{EDF-CF} |
|---|---|---|---|---|---|---|
| 2 | 38393 | 44981 | 45660 | 50440 | 112.1% | 110.4% |
| 4 | 24386 | 28058 | 32946 | 35245 | 125.6% | 106.9% |
| 8 | 16681 | 18837 | 26241 | 27431 | 145.6% | 104.5% |
| 16 | 11772 | 13129 | 22361 | 23003 | 175.2% | 102.8% |

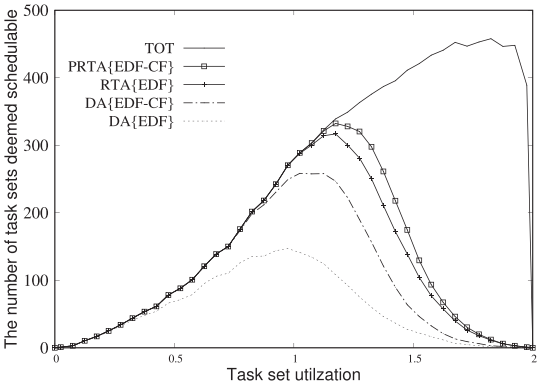
(d) RM, constrained deadline



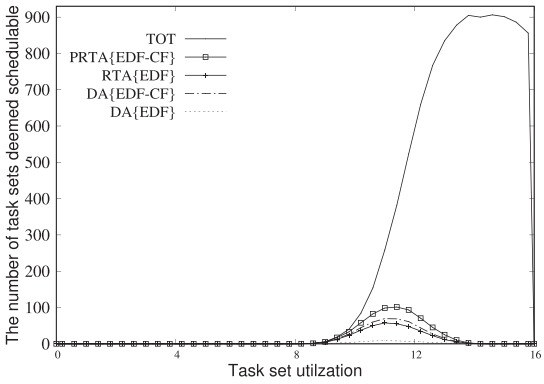

(a) Bimodal distribution with 0.9 for $m = 2$ ($\overline{n} = 3.1$, $\overline{C_i/T_i} = 0.56$)



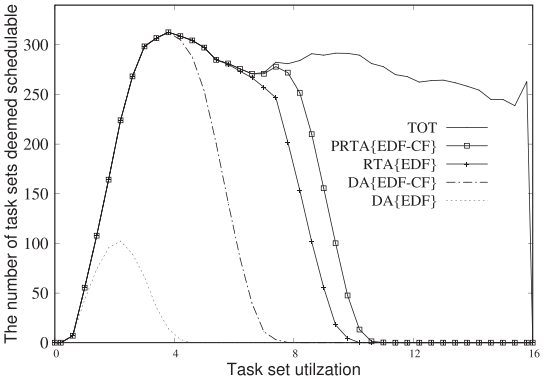

(b) Exponential distribution with 0.1 for $m = 2$ ($\overline{n} = 11.6$, $\overline{C_i/T_i} = 0.1$)



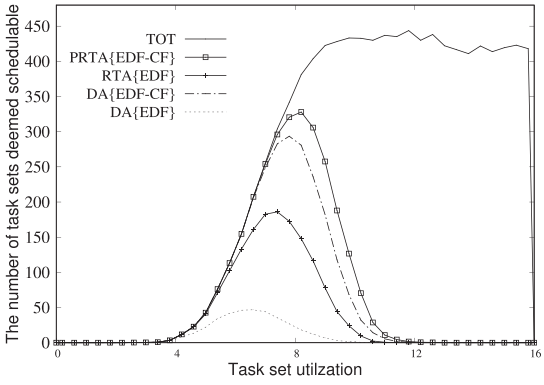

(c) Exponential distribution with 0.9 for $m = 2$ ($\overline{n} = 4.3$, $\overline{C_i/T_i} = 0.34$)




(d) Bimodal distribution with 0.9 for $m = 16$ ($\overline{n} = 19.8$, $\overline{C_i/T_i} = 0.69$)




(e) Exponential distribution with 0.1 for $m = 16$ ($\overline{n} = 83.2$, $\overline{C_i/T_i} = 0.1$)




(f) Exponential distribution with 0.9 for $m = 16$ ($\overline{n} = 28.0$, $\overline{C_i/T_i} = 0.4$)

**Fig. 5.** Schedulability tests for implicit deadline task sets.

*O*3. PRTA{EDF-CF} outperforms both RTA{EDF} and DA{EDF-CF} for all values of $m$ (from Table 3(a)).
*O*4. For $m = 16$, RTA{EDF} performs better than DA{EDF-CF} when an exponential utilization distribution with 0.1 is considered, whereas it does not when the others are considered (from Figures 3(d), (e) and (f)).
*O*5. RTA{EDF} makes a higher number of task sets schedulable than DA{EDF-CF} for $m = 2$ with any utilization distribution (from Fig. 3(a)–(c)).
*O*6. PRTA{EDF-CF} outperforms both RTA{EDF} and DA{EDF-CF} for $m = 2$ and 16 with any utilization distribution (from Fig. 3).

*O*1 and *O*2 stem from the advantage of the response-time-based approach (the aspect of the schedulability analysis) and the CF policy (the aspect of scheduling algorithm). *O*1 indicates that response-time-based approach such as RTA{EDF} finds a much higher number of schedulable task sets, because it includes only the interference of higher-priority tasks on job $J_k$ of task $\tau_k$ in the interval $[r_k^*, f_k^*)$ whereas DA{EDF} and DA{EDF-CF} consider the worst-case interference on $J_k$ in the interval $[r_k^*, d_k^*)$. *O*2 holds because as the number of processors $m$ increases, the number of contention-free slots for $J_k$ that exist in $[r_k^j, d_k^j)$ also increases; thus DA{EDF-CF} takes advantage of the increased lower-bounded number of contention-free slots $\Phi_k$.

*O*3 demonstrates that, by exploiting the merits of both the response-time-based approach and the CF policy, PRTA{EDF-CF} outperforms both RTA{EDF} and DA{EDF-CF} for all values of $m$. As $m$ increases from 2 to 16, there are more contention-free slots, and therefore, PRTA{EDF-CF} improves compared to the performance of RTA{EDF} by successively higher percentage (e.g., 58.6% for $m = 16$) by exploiting the merit of contention-free slots. Additionally, for all values of $m$, the PRTA{EDF-CF} improves the performance of DA{EDF-CF} by a similar percentage (e.g., 35.6% for $m = 2$) owing to the advantage of the response-time-based approach.

*O*4 and *O*5 are interpreted using the following factors: (*F*1) DA performs well for a small value of $\overline{n}$, because the amount of pessimistic upper-bounded interference from higher-priority tasks is proportional to the number of tasks in a task set; (*F*2) the CF policy performs well for a small value of $\overline{C_i/T_i}$,because each task possibly has many contention-free slots; and (*F*3) the CF policy performs well for a large value of $m$, for a similar reason to that of *F*2. In the case of *O*4 (for $m = 16$), DA{EDF-CF} outperforms RTA{EDF} for a bimodal utilization distribution with 0.9 and an exponential distribution with 0.9 (in Fig. 5(d) and (f)), owing to the combination of *F*1 and *F*2. However, when exponential utilization distribution with 0.1 is considered (Fig. 5(e)), RTA{EDF} performs better than DA{EDF-CF} from the inverse of *F*1 (owing to the large value of $\overline{n}$, i.e., 83.2). Although $\overline{C_i/T_i}$ is quite small (i.e., 0.1) in the utilization distribution, this factor does not fully overcome the disadvantage of the large value of $\overline{n}$ to DA{EDF-CF}. However, the phenomenon *O*4 does not occur for $m = 2$ as *O*5 indicates, because the inverse of *F*3 (i.e., the smaller value of $m$) is a significantly stronger factor for the performance gap between RTA{EDF} and DA{EDF-CF}. As shown in Theorem 3, PRTA dominates RTA and DA, *A*-CF dominates the base algorithm *A*, and PRTA{EDF-CF} dominates RTA{EDF} and DA{EDF-CF} for all values of $\overline{n}$ and $\overline{C_i/T_i}$, as indicated by *O*6.

Table 3(b) shows the evaluation results for task sets in which a task has a constrained deadline. The similar trend that occurs between RTA{EDF} and PRTA{EDF-CF} in Table 3(a) is also shown in Table 3(b); as $m$ increases from 2 to 16, PRTA{EDF-CF} improves the performance of RTA{EDF} at a progressively higher rate. However, the performance improvement of RTA{EDF} by PRTA{EDF-CF} is much higher (e.g., 137.5% for $m = 16$) than the case of implicit deadlines, because each $J_i$ does not execute in the interval $[d_i^j, r_i^{j+1})$, and thus the schedules of jobs produces a higher number of contention-free slots. Owing to this fact, the performance improvement of DA{EDF-CF} by PRTA{EDF-CF} decreases as the value of $m$ increases from 2 to 16 (e.g., 46.4% for $m = 2$ and 17.4% for $m = 16$). DA{EDF-CF} also exploits a large number of contention-free slots in the case of many processors (e.g. $m = 16$), while there is less room for performance improvement achieved by PRTA{EDF-CF} using the advantage of response-time-based approach.

Then, we evaluate the performance of the schedulability analysis methods for RM and RM-CF. We also consider the four schedulability analysis techniques obtained by changing the target scheduling algorithm from EDF to RM, and from EDF-CF to RM-CF, which are denoted as DA{RM}, RTA{RM}, DA{RM-CF}, and PRTA{RM-CF}.

Table 3(c) and (d) present the evaluation results for task sets of implicit and constraint deadlines, respectively. Table 3(c) and (d) show similar trends to those shown in Table 3(a) and (b), respectively. However, the degrees of performance improvement of RTA{RM} and DA{RM-CF} by PRTA{RM-CF} are smaller than those shown in Table 3(a) and (b). This is because as all techniques exclude the interference of lower-priority tasks on task $\tau_k$, they find a higher number of task sets deemed schedulable on average compared to the case of EDF scheduling. Thus, there is less room to improve the schedulability. In spite of this fact, PRTA{RM-CF} still improves the performance of RTA{RM} and DA{RM-CF} by a high rate for $m = 16$ (e.g., 34.1% for implicit- and 75.2% for constrained-deadline tasks) and $m = 2$ (e.g., 18.3% for implicit- and 10.4% for constrained-deadline tasks), respectively.

## 7. Discussion

In this section, we discuss various factors affecting analytic capability of PRTA. PRTA judges schedulability of each task $\tau_k$ by comparing its pseudo-response time $\tilde{R}_k$ and its relative deadline $D_k$. Since $\tilde{R}_k$ is calculated by the summation of WCET $C_k(\hat{\tau})$ and the worst-case interference $I_k(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ on $\hat{\tau}_k$ during execution (as Theorem 1 indicates), how to effectively estimate the former and latter determines the analytic capability of PRTA, whose relevant factors will be discussed in the following subsections.

### 7.1. Shared computing resources

As mentioned in Section 2, we consider the Liu and Layland's task model assuming the fixed worst-case execution $C_i$ of each task $\tau_i$, which implicitly includes the worst-case time induced by inter-core interference on shared computing resources such as shared cache, memory bus, and main memory on a multi-processor platform. Since mutual exclusion is mandatory to utilize such shared computing resources, one job never enters its critical section at the instance when another concurrent job enters its own critical section. This mechanism is referred to as the resource-locking protocol that has been extensively discussed in a number of existing studies. The representative resource-locking protocols for multiprocessor systems include the priority inheritance protocol (PIP) (Sha et al., 1990; Easwaran and Adersson, 2009), priority ceiling protocol (PCP) (Chen and Lin), flexible multiprocessor locking protocol (FMLP) (Block et al., 2007), $O(m)$ locking protocol (OMLP) (Brandenburg and Anderson, 2010) and real-time nested locking protocol (RNLP) (Ward and Anderson, 2012). Considering resource-locking protocol potentially improves analytic capability of PRTA since it reduces response time of $\tau_k$ by relieving a pessimistic assumption of underlying the worst-case execution time of the Liu and Layland's task model.

To discuss how analytical capability of our PRTA can be improved by removing pessimistic assumption regarding accessing shared computing resources in WCET of the Liu and Layland's task model, we consider the priority inheritance protocol (PIP) (Sha et al., 1990; Easwaran and Adersson, 2009) for RM scheduling incorporating the CF policy. In PIP, $p$ different kinds of resources are considered, and jobs can issue requests for exclusive access to the shared resources $\delta_1, \cdots, \delta_p$. A task holding a resource can be preempted by the processor scheduler but the task still holds the resource until it completes the resource usage. We consider *non-nested* shared resources, indicating that a job does not request for a shared resource while it holds another one. A job $J_i^j$ of a task $\tau_i$ could request resource $\delta_x$ $(1 \le x \le p)$ multiple times during its execution. $C_{i,x}$ denotes the worst-case (i.e., longest) resource usage time among all requests for a resource $\delta_x$ by jobs of $\tau_i$. Further, the set of all resources accessed by jobs of $\tau_i$ is denoted by $\delta S_i \subseteq \{\delta_1, \cdots, \delta_p\}$.

Under PIP, when a job $J_k^j$ of a task $\tau_k$ is holding (i.e., using) a shared resource $\delta_x$ and another higher priority job $J_i^j$ of a task $\tau_i$ requests the same shared resource $\delta_x$, the priority inheritance operates such that the priority of $J_k^j$ is promoted to $i$; we assume that smaller $i$ indicates a higher priority in the RM scheduling. We call such temporally promoted priority as the effective priority compared to the base priority initially assigned by RM scheduler offline. $J_i^j$ may also experience interference from other lower priority

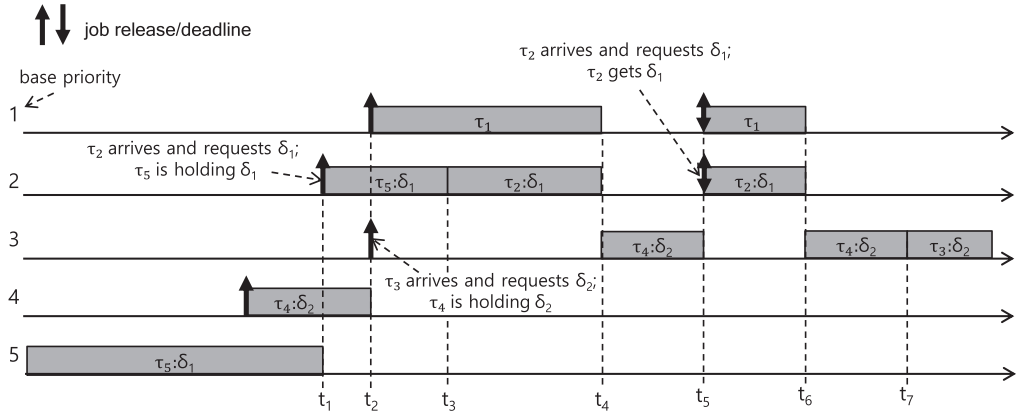



**Fig. 6.** Priority inheritance protocol for $m = 2$.

jobs under PIP because the effective priority of a lower priority job can be higher than $i$ due to priority inheritance.

Fig. 6 illustrates an example schedule of $\tau = \{\tau_1, \cdots, \tau_5\}$ under PIP for $m = 2$, where task indexes 1–5 indicate their base-priorities. In the example, $\tau_2$ and $\tau_5$ share $\delta_1$, $\tau_3$ and $\tau_4$ share $\delta_2$, and $\tau_1$ does not use any resource. Initially, $\tau_5$ and $\tau_4$ hold $\delta_1$ and $\delta_2$, respectively, and perform their executions. At $t_1$, $\tau_2$ arrives and requests $\delta_1$, but $\tau_2$ cannot get $\delta_1$ because $\tau_5$ is holding $\delta_1$. Instead, the effective priority of $\tau_5$ is promoted to 2. At $t_2$, $\tau_1$ arrives and $\tau_4$ is preempted by $\tau_1$ because $\tau_4$'s effective (even base) priority is lower than $\tau_1$. At the same time, $\tau_3$ arrives and requests $\delta_2$, but $\tau_3$ cannot get $\delta_2$ because $\tau_4$ is holding $\delta_2$. Thus, $\tau_4$'s priority is promoted to 3 at $t_2$. At $t_3$, $\tau_5$ completes its execution with $\delta_1$, and $\tau_2$ begins its execution with $\delta_1$. At $t_4$, $\tau_1$ and $\tau_2$ finish its execution simultaneously, and $\tau_4$ resumes its execution with $\delta_2$. At $t_5$, $\tau_1$ and $\tau_2$ arrive at the same time, and $\tau_4$ is preempted by them. At $t_6$, $\tau_4$ resumes its execution with $\delta_2$, and $\tau_4$ completes its execution and $\tau_3$ beings its execution with $\delta_2$ at $t_7$.

To calculate the response time under PIP, we need to define the new worst-case execution $C_k^\delta$, which includes CPU execution time only and does not include the worst-case waiting time to get required resources. This is different from $C_i$ of the Liu and Layland model, which assumes that it always includes the worst-case waiting time for it. Then, the response time under PIP is determined by three factors: (i) the worst-case execution $C_k^\delta$ without the worst-case waiting time for $\delta S_k$, (ii) the maximum amount of time that a job $J_k^j$ of $\tau_k$ waits to get its required resources $\delta_x \in \delta S_k$, and (iii) the maximum amount of time that $J_k^j$'s execution that is hindered by other jobs not requiring any resource $\delta_x \in \delta S_k$, whose effective or base priorities are higher than $J_k^j$'s one. (i) is supposed to be given, while we need to consider the following two terms to calculate (ii).

- $DB_k$ denotes the upper-bounded total amount of time that a job $J_k^j$ of $\tau_k$ waits to get any resource $\delta_x \in \delta S_k$ while $\delta_x$ is used by lower base-priority (its effective priority may be promoted to $k$) jobs, and
- $Ihp_k^{(dsr)}(\ell)$ (direct shared resource) denotes the upper-bounded total amount of time that $J_k^j$ waits to get any resource $\delta_x$ while $\delta_x$ used by higher base-priority jobs in an interval of length $\ell$.

Also, (iii) is determined by the following three terms.

- $Ihp_k^{(osr)}(\ell)$ (other shared resources) denotes the upper-bounded total amount of time that higher base-priority jobs $J_i^j$ execute holding a resource not in $\delta S_k$ (i.e., $\delta S_k \cap \delta S_i = \emptyset$) in an interval of length $\ell$,
- $Ihp_k^{(nsr)}(\ell)$ (no shared resource) denotes the upper-bounded total amount of time of other higher base-priority jobs $J_i^j$ executing without any resource (i.e., $\delta S_i = \emptyset$) in an interval of length $\ell$, and
- $Ilp_k(\ell)$ denotes the upper-bounded total amount of time of lower base-priority jobs in an interval of length $\ell$, when having an effective-priority greater than $J_k^j$.

$J_k^j$ cannot execute when $J_k^j$ requests any resource $\delta_x \in \delta S_k$ while another job $J_k^j$ is holding the same resource. On the other hand, $m$ jobs are needed to hinder $J_k^j$'s execution if such jobs are holding resources not in $\delta S_k$ or do not request any resource during their execution. Thus, task $\tau_k$ scheduled by RM scheduling with PIP can be schedulable, if every job $J_k^j$ of $\tau_k$ satisfies the following for $C_k^\delta \le \ell \le D_k$ (Easwaran and Adersson, 2009):

$$C_k^\delta + DB_k + Ihp_k^{(dsr)}(\ell) + \left\lfloor \frac{Ihp_k^{(osr)}(\ell) + Ihp_k^{(nsr)}(\ell) + Ilp_k(\ell)}{m} \right\rfloor \le \ell. \quad (11)$$

Then, RTA finds such $\ell$, using the procedure explained in Section 3.2.

Now, we discuss how to incorporate the CF policy into RM scheduling and develop PRTA with PIP. To this end, we need to address the following three questions.

Q1. How to calculate the lower-bounded number of contention-free slots that exists in the interval $[r_i^j, d_i^j)$,
Q2. How to effectively utilize such lower-bounded number of contention-free slots during scheduling to improve schedulability, and
Q3. How to develop PRTA to guarantee schedulability of RM scheduling under PIP.

We first address Q1 as follows. By the definition of terms that we mentioned, $J_k^j$ cannot execute in an interval of length $DB_k + Ihp_k^{(dsr)}(\ell)$. In addition, at least $m$ executions of jobs are needed for a time to be contending as explained in Lemma 1. Therefore, the lower-bounded number of contention-free slots that exists in the interval $[r_i^j, d_i^j)$ under PIP (denoted by $\Phi_k^\delta$) can be calculated by

$$\Phi_k^\delta = \max\left(0, D_k - DB_k - Ihp_k^{(dsr)}(D_k) - \left\lfloor \frac{C_k^\delta + Ihp_k^{(osr)}(D_k) + Ihp_k^{(nsr)}(D_k) + Ilp_k(D_k)}{m} \right\rfloor\right). \quad (12)$$

To address Q2, we need to carefully consider two scenarios where the priority of a job $J_k^j$ is demoted by the CF policy when $J_k^j$ is holding a certain resource and is not. Since the priority demotion of a job $J_k^j$ causes a complicated influence to PIP, we can relieve such complication as we demote $J_k^j$'s base priority only when

remaining contention-free slots $\Phi_k^\delta$ (under PIP) for $J_k^j$ is sufficiently larger than the summation of $J_k^j$'s remaining execution and the total amount of time holding all resources in $\delta S_k$. Thereafter, we promote $J_k^j$'s priority again when it requests any resource in $\delta S_k$.

Even though such policy may utilize smaller amount of contention-free slots compared to $\Phi_k^\delta$, it makes easier to address Q3. Since the priority of a job is temporally demoted only when it does not hold any resource, it changes the value of $Ihp_k^{(nsr)}(\ell)$ only by the definition of $Ihp_k^{(nsr)}(\ell)$. The amount of interference of $\tau_i$ contributing to $Ihp_k^{(nsr)}(\ell)$ (in Eq. (11)) can be deducted at least as much as $\max(0, \Phi_i^\delta - \sum_{\delta_x \in \delta S_i} C_{i,x})$. So far, we briefly discussed how to apply PIP to PRTA.

Although the locking protocol such as PIP is an effective approach to relieve a pessimistic assumption regarding shared computing resource in the Liu and Layland's task model, the derived worst-case waiting time for resources under such locking protocol may be reduced, if we consider more specific CPU architecture, which is, modern multicore systems use hardware memory controller for arbitrating concurrent memory accesses from multiple cores. If multiple active jobs access data in the different banks in DRAM simultaneously, they could utilize their required resources without any delay. However, the key difficulty in exploiting such advantage of concurrent accesses on a resource for hard real-time systems is that the exact locations of the allocated memory over the banks are unpredictable. Some studies such as Predator (Akesson et al., 2007) and AMC (Paolieri et al., 2009) DRAM controller treat multiple banks as a single one to improve predictability, but such approaches do not take advantage of concurrent accesses to DRAM to reduce the worst-case waiting time. Other approaches employ private banking schemes so that each core only can access its designated bank by modifying hardware design (Reineke et al., 2011; Wu et al., 2013). While private banking schemes require a hardware modification, a software approach such as PALLOC (Yun et al., 2014) just modifies kernel codes so that a system designer can partition DRAM banks in a flexible manner to improve predictability. Using PALLOC, partitioned tasks in a certain core is allowed to access to a designated bank, and it makes possible multiple access to DRAM with predictability. However, it forces a partitioning scheduling and, non-designated banks cannot be used even if it is not used by any task. To effectively exploit the approaches mentioned above, we may further improve the analytical capability of PRTA by considering memory access mechanism.

### 7.2. Preemption and migration costs

The fixed worst-case execution time under the Liu and Layland's task model also assumes to include preemption and migration costs. Since a migration from one processor to another conditionally occurs when a preemption happens, we upper-bound the potential number of migrations by measuring the number of preemptions under the target scheduling algorithm. According to the experiment results in a previous study (Lee et al., 2011; 2014), the average actual number of preemptions incurred by each implicit-deadline (i.e., $D_i = T_i$) task set scheduled by EDF-CF during 100,000 time units is 1,071.1 for $m = 2$ and 2,321.1 for $m = 8$, meaning that possibility of the occurrence of a preemption at each time unit is from 1.0% to 2.3% depending on the number of processors. Although PRTA does not provide a mechanism to calculate (or upper-bound) the number of preemptions that occurs during each job's execution, such experimental results imply that it is acceptable not to take preemption cost into account (or it is acceptable to have similar WCET with the CF policy to that without the CF policy) because the CF policy does not cause a large amount of additional time delay due to preemptions (and migrations).

### 7.3. Worst-case interference

When it comes to the worst-case interference $I_k(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ on $\hat{\tau}_k$, overestimation can intervene in upper-bounding both $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ and $I_k(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$. Since upper-bounding $I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ is based on the worst-case scenario in Fig. 3, the analytic capability of PRTA can be improved if we find the more optimistic worst-case scenario. Also, PRTA upper-bounds $I_k(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$ with the summation of $min(I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell), \ell - C_k + 1)$ of all tasks in $\hat{\tau} \setminus \tau_k(\hat{\tau})$ divided by the number of processors $m$. As the proof of Lemma 2 indicates, the underlying idea on such calculation is that a job cannot execute in a time slot if $m$ other higher-priority jobs execute and execution of each higher-priority job is assumed to contribute to $min(I_k^i(r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell), \ell - C_k + 1)$ as much as possible. However, such an idea is quite pessimistic since a certain portion of a higher-priority job's execution can be performed in parallel with a job of $\tau_k(\hat{\tau})$. Therefore, the performance of PRTA can also be enhanced if we reduce upper-bounded execution of higher-priority jobs in $[r_k^j(\hat{\tau}), r_k^j(\hat{\tau}) + \ell)$.

## 8. Conclusion

In this study, we aimed at developing a tighter schedulability analysis for *A*-CF by inspecting the limitations of the RTA framework. To this end, we proposed a new schedulability analysis for *A*-CF, referred to as PRTA, by investigating the properties of the priority demotion conducted by the CF policy. Building upon the investigation, we defined the notion of pseudo-response time as a tighter criterion than response time for the CF policy, which represents the worst-case time when the schedulability of a task is guaranteed; we observed that the pseudo-response time of a task can be derived by either the time when priority demotion occurs or the finishing time. We then derived a tighter schedulability condition of a task scheduled by *A*-CF. We also showed that performance of PRTA can be further improved by exploiting a new notion of pseudo-slack time. To demonstrate the effectiveness of PRTA, we applied PRTA to the existing EDF and RM scheduling algorithms employing the CF policy, and showed that up to 46.4% and 18.3% schedulability performance improvement can be achieved, respectively, compared to those applying the DA test.

For future work, we are planning to develop an extended PRTA, which is applicable to a combination of the CF and ZL policies. We also would like to extend PRTA from the single-level to multi-level CF policy (Baek et al., 2018b). In order to improve analytic capability of the proposed PRTA, we need to address the issues discussed in Section 7 to relieve pessimism of the overestimated response time on PRTA.

## Acknowledgement

This research was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2019R1A2B5B02001794).

## References

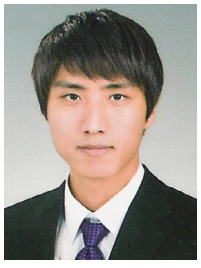
Akesson, B., Goossens, K., Ringhofer, M., 2007. Predator: a predictable sdram memory controller. In: Proceedings of IEEE/ACM international conference on Hardware/software codesign and system synthesis, pp. 251–256.

Andersson, B., Bletsas, K., Baruah, S., 2008. Scheduling arbitrary-deadline sporadic task systems on multiprocessor. In: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 197–206.

Baek, H., Lee, H., Lee, H., Lee, J., Kim, S., 2018. Improved schedulability analysis for fault-tolerant space-borne sar system. In: Proceedings of Conference on Korea Institute of Military Science and Technology (KIIT), pp. 1231–1232.

Baek, H., Lee, J., Shin, I., 2018. Multi-level contention-free policy for real-time multiprocessor scheduling. J. Syst. Software 137, 36–49.

Baker, T., 2005. An analysis of EDF schedulability on a multiprocessor. IEEE Trans. Parallel Distrib. Syst. 16 (8), 760–768.
Baker, T.P., Cirinei, M., 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 178–190.
Baker, T.P., Cirinei, M., Bertogna, M., 2008. EDZL scheduling analysis. Real-Time Syst. 40, 264–289.
Bertogna, M., Cirinei, M., 2007. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 149–160.
Bertogna, M., Cirinei, M., Lipari, G., 2005. Improved schedulability analysis of EDF on multiprocessor platforms. In: Proceedings of Euromicro Conference on Real–Time Systems (ECRTS), pp. 209–218.
Bertogna, M., Cirinei, M., Lipari, G., 2009. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. IEEE Trans. Parallel Distrib.Syst. 20, 553–566.
Biondi, A., Buttazzo, G.C., Bertogna, M., 2016. Schedulability analysis of hierarchical real-time systems under shared resources. IEEE Trans. Comput. 65 (5), 1593–1605.
Block, A., Leontyev, H., Brandenburg, B.B., Anderson, J.H., 2007. A flexible real–time locking protocol for multiprocessors. In: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp. 1–10.
Brandenburg, B.B., Anderson, J.H., 2010. Optimality results for multiprocessor real-time locking. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 49–60.
Brandenburg, B.B., Gul, M., 2016. Global scheduling not required: simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 1–15.
Chen, M. I., Lin, K. J.,. Dynamic priority ceilings: a concurrency control protocol for real-time system. Real-Time Syst. 2(4), 325–346.
Cirinei, M., Baker, T.P., 2007. EDZL scheduling analysis. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 9–18.
Cobham, Gaisler, VxWorks 7 SPARC architectural port and BSP. https://www.gaisler.com.
Davis, R.I., Burns, A., 2011. FPZL schedulability analysis. In: RTAS, pp. 245–256.
Easwaran, A., Adersson, B., 2009. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 377–386.
European, Space agency, SpaceWire. http://spacewire.esa.int.
Excalibur, Systems, MIL-STD-1553b. https://www.mil-1553.com.
Joseph, M., Pandya, P., 1986. Finding response times in a real-time system. Comput. J. 29 (5), 390–395.
Lee, J., 2014. Time-reversibility of schedulability tests. In: Proceedings of IEEE Real–Time Systems Symposium (RTSS), pp. 294–303.
Lee, J., 2017. Time-reversibility for real-time scheduling on multiprocessor systems. IEEE Trans. Parallel Distrib. Syst. 28 (1), 230–243.
Lee, J., Chwa, H.S., Lee, J., Shin, I., 2016. Thread-level priority assignment in global multiprocessor scheduling for dag tasks. J. Syst. Software 113, 246–256.
Lee, J., Easwaran, A., Shin, I., 2011. Maximizing contention-free executions in multiprocessor scheduling. In: Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 235–244.
Lee, J., Easwaran, A., Shin, I., 2014. Contention-free executions for real-time multiprocessor scheduling. ACM Trans. Embedded Comput. Syst. 13 (69), 1–69.
Lee, J., Shin, K., 2014. Improvement of real-time multi-core schedulability with forced non-preemption. IEEE Trans. Parallel Distrib. Syst. 25 (5), 1233–1243.
Lee, J., Shin, K.G., 2013. Schedulability analysis for a mode transition in real–time multi-core systems. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 11–20.
Lee, J., Shin, K.G., 2017. Development and use of a new control task model for cyber-physical systems: a real-time scheduling perspective. J. Syst. Software 126, 45–56.
Liu, C., Layland, J., 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. J. ACM 20 (1), 46–61.
Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A., Buttazzo, G., 2016. Schedulability analysis of conditional parallel task graphs in multicore systems. IEEE Trans. Comput. 66 (2), 339–353.
Paolieri, M., Quinones, E., Cazorla, F.J., Valero, M., 2009. An analyzable memory controller for hard real-time cmps. In: IEEE Embedded Systems Letters, pp. 86–90.
Reineke, J., Liu, I., Patel, H.D., Kim, S., Lee, E.A., 2011. Pret dram controller: bank privatization for predictability and temporal isolation. In: Proceedings of tIEEE/ACM international conference on Hardware/software codesign and system synthesis, pp. 99–108.
RTEMS, Community. RTEMS real-time operating system https://www.rtems.org.
Sha, L., Rajkumar, R., Lehoczky, J.P., 1990. Priority inheritance protocols: an approach to real-time synchronization. IEEE Trans. Comput. 1175–1185.
Ward, B.C., Anderson, J.H., 2012. Supporting nested locking in multiprocessor real–time systems. In: Proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pp. 223–232.
Wu, Z.P., Krish, Y., Pellizzoni, R., 2013. Worst case analysis of dram latency in multi-requestor systems. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 372–383.
Yun, H., Mancuso, R., Wu, Z.P., Pellizzoni, R., 2014. Palloc: dram bank-aware memory allocator for performance isolation on multicore platforms. In: Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS), pp. 155–166.



**Hyeongboo Baek** is an assistant professor in Department of Computer Science and Engineering, Incheon National University (INU), South Korea, He received the BS degree in Computer Science and Engineering from Konkuk University, South Korea in 2010 and the MS and PhD degrees in Computer Science from KAIST, South Korea in 2012 and 2016, respectively. His research interests include cyber-physical systems, real-time embedded systems and system security. He won the best paper award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

**Jinkyu Lee** is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), South Korea, where he joined in 2014. He received the BS, MS, and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2004, 2006, and 2011, respectively. He has been a research fellow/visiting scholar in the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.