Improved Schedulability Analysis Using Carry-In Limitation for Non-Preemptive Fixed-Priority Multiprocessor Scheduling

Jinkyu Lee, *Member, IEEE*

Abstract—A time instant is said to be a *critical instant* for a task, if the task's arrival at the instant makes the duration between the task's arrival and completion the longest. Critical instants for a task, once revealed, make it possible to check the task's schedulability by investigating situations associated with the critical instants. This potentially results in efficient and tight schedulability tests, which is important in real-time systems. For example, existing studies have discovered critical instants under preemptive fixed-priority scheduling (P-FP), which limit interference from carry-in jobs, yielding the state-of-the-art schedulability tests on both uniprocessor and multiprocessor platforms. However, studies on schedulability tests associated with critical instants have not matured yet for non-preemptive scheduling, especially on a multiprocessor platform. In this paper, we find necessary conditions for critical instants for non-preemptive global fixed-priority scheduling (NP-FP) on a multiprocessor platform, and develop a new schedulability test that takes advantage of the finding for reducing carry-in jobs' interference. Evaluation results show that the proposed schedulability test finds up to 14.3 percent additional task sets schedulable by NP-FP, which are not deemed schedulable by the state-of-the-art NP-FP schedulability test.

Index Terms—Real-time scheduling, critical instants, non-preemptive scheduling, fixed-priority scheduling, multiprocessor platform

1 Introduction

Due to safety- or mission-critical characteristics, real-time systems usually require timing guarantees of a set of recurring tasks. To this end, many studies for real-time systems have found a time instant at which a request for a task of interest will result in the longest time duration between the release and finishing time of the task, called a *critical instant* [1]. Once critical instants for a task are discovered, we can check the schedulability of the task, only by investigating situations where the critical instants occur, yielding efficient and tight schedulability tests. For example, an exact (i.e., sufficient and necessary) schedulability test has been developed for preemptive fixed-priority scheduling (P-FP) on a uniprocessor platform [2], based on a critical instant found in [1]. When it comes to a multiprocessor platform, existing studies [3], [4] have identified necessary conditions for P-FP's critical instants, which yields a tighter schedulability test than the existing one without utilizing critical instants [5]. However, most attempts to find critical instants have been biased towards preemptive scheduling.

Non-preemptive scheduling not only has its own advantages [6], but also is essential to tasks with extremely large preemption/migration overhead and inherently non-preemptive tasks (e.g., interrupts, transactional operations); however, its underlying theories have not matured yet, especially for multiprocessor systems. While the notion of a critical instant also has potential in improving schedulability guarantee of non-preemptive scheduling, only a few studies have made an endeavor to find critical instants for non-preemptive scheduling [7], [8], especially few studies on a multiprocessor platform.

This paper aims at developing an improved schedulability test for non-preemptive global fixed-priority scheduling (NP-FP). To this end, we first organize a new schedulability test that incorporates an existing carry-in limitation technique to an existing RTA framework for NP-FP, which can be regarded as the state-of-the-art schedulability analysis for NP-FP. Next, we derive necessary conditions for critical instants for NP-FP, which entail two cases to be investigated. Deriving properties for the two cases, we develop a new schedulability test for NP-FP utilizing the properties to limit interference from carry-in jobs,[1] which dominates all existing schedulability tests for NP-FP.

To demonstrate the effectiveness of the proposed schedulability test, we perform extensive simulation. The simulation results show that our schedulability test finds up to 14.3 percent additional NP-FP-schedulable task sets, which are not covered by the state-of-the-art NP-FP schedulability test.

In summary, this paper makes the following contributions.

- We organize the state-of-the-art schedulability test for NP-FP by combining an existing carry-in limitation technique and RTA framework for NP-FP (Section 3);
- We derive necessary conditions for NP-FP's critical instants, which is the first attempt on a multiprocessor platform (Section 4.1);
- We develop a new schedulability test for NP-FP that utilizes the necessary conditions for critical instants (Section 4.2); and
- We demonstrate the effectiveness of the schedulability test through extensive simulation (Section 5).

2 System Model, Assumptions and Notations

Task, Job, and Platform. In this paper, we focus on the sporadic real-time task model [9] in which a task τ_k in a task set τ is represented by T_k (the minimum separation), C_k (the worst-case execution time), and D_k (the relative deadline). In particular, we restrict our attention to implicit and constrained deadline tasks, each of which satisfies $D_k = T_k$ and $D_k \le T_k$, respectively. For convenience' sake, we consider a quantum-based time slot, and let the length of one quantum be one time unit, without loss of generality; therefore, all parameters of T_k, C_k and D_k are positive integer values. Each task invokes a series of jobs; the qth job of τ_k (denoted by J_k^q), once released, should finish its execution within D_k time units. A job is called *active* at t, if the job has remaining execution at t. We assume that each job is independent, and cannot be executed on more than one processor at the same time. We consider a multiprocessor platform consisting of m identical processors.

Scheduling Algorithm. In this paper, we consider global work-conserving scheduling. By *global* scheduling, we mean that a job can be executed in any processor, while partitioned scheduling restricts execution of a job to only one designated processor. By *work-conserving*, we mean that any processor cannot be left idle as long as there exist at least one active job which is not currently-executing. Among many global work-conserving scheduling algorithms, this paper focuses on non-preemptive fixed-priority scheduling. Compared to preemptive fixed-priority scheduling, NP-FP disallows a job to preempt a currently-executing job; thus, it is possible for a lower-priority job to block the execution of a higher-priority job, if the lower-priority job starts its execution before the higher-priority job is released. Conventionally, if a job cannot be executed due to execution of a lower- and higher-priority

- *The author is with the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Suwon-si, Gyeonggi-do 16419, Republic of Korea. E-mail: jinkyu.lee@skku.edu.*

Manuscript received 13 July 2016; revised 3 May 2017; accepted 9 May 2017. Date of publication 11 May 2017; date of current version 14 Sept. 2017.
(Corresponding author: Jinkyu Lee.)
Recommended for acceptance by M. Caccamo.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below
Digital Object Identifier no. 10.1109/TC.2017.2704083

1. The definition of a carry-in job will be shown in Section 2.

job, we call this blocking and interference, respectively. In NP-FP, the priority of a job is determined by the predefined priority of the task invoking the job. Without loss of generality, we assume the smaller the task index, the higher the task priority, i.e., the priority of τ_i is higher than that of τ_j if $i < j$.

Carry-in Job. A job is said to be carry-in in an interval starting at t, if the job is released before t and has remaining execution at t.

Response Time and Critical Instant. The response time of a job J_k^q is the time duration between the release and finishing time of J_k^q, and a task τ_k is said to have the response time R_k ($\leq D_k$) if every job invoked by τ_k finishes its execution within R_k time units after its release. Also, $S_k = D_k - R_k$ (≥ 0) is called the slack of τ_k, meaning that every job of τ_k finishes at least S_k ahead of its deadline. A critical instant for a task τ_k is defined as a time instant at which a request for a job of the task results in the longest response time of the task [1].

To calculate the response time of τ_k, we will use the following notations of task sets for an interval staring at t:

- $\tau^{\mathtt{HN}(k,t)}$: a set of tasks in τ each of which has a priority higher than k and has no carry-in job in an interval starting at t,
- $\tau^{\mathtt{HC}(k,t)}$: a set of tasks in τ each of which has a priority higher than k and has one carry-in job in an interval starting at t, and
- $\tau^{\mathtt{L}(k,t)}$: a set of tasks in τ each of which has a priority lower than k and has one job starting its execution before t as well as continuing its execution after t.

3 Existing Carry-In Limitation Technique into RTA Framework

In this section, we first summarize existing RTA framework for NP-FP. Next, we introduce an existing carry-in limitation technique for NP-FP, and then incorporate the technique into the RTA framework.

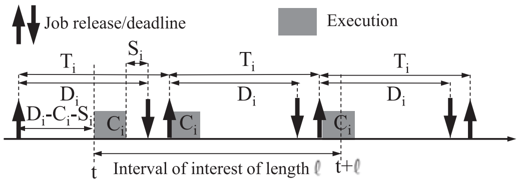
3.1 Existing RTA Framework for NP-FP

For timing guarantees, many schedulability tests have been developed on a multiprocessor platform. Among them, Response Time Analysis (RTA) has been widely used due to its schedulability performance. For example, RTA yields (one of) the best schedulability tests for many scheduling algorithms such as P-EDF (Earliest Deadline First) [5] by Bertogna and Cirinei, P-FP [3], [5] by Guan et al., NP-EDF [10] by Lee and Shin, and NP-FP [11] by Lee and Shin. RTA calculates each task's response time; if every task has its response time no larger than its relative deadline (i.e., $R_i \leq D_i$ for every $\tau_i \in \tau$), the task set is deemed schedulable.

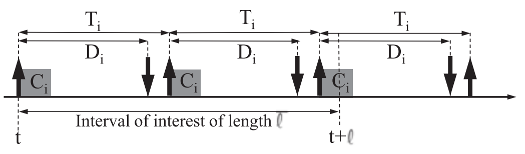
For non-preemptive scheduling, calculation of a job's response time needs to know *when the first unit of the job's execution will be finished* [10], [11]. Provided that it takes ℓ time units, a job of a task τ_k will finish its execution within $\ell + C_k - 1$ time units. To calculate such ℓ, the existing RTA framework calculates the amount of time in $[t, t+\ell)$ such that the job of τ_k cannot execute due to the execution of jobs of τ_i (denoted by $I_{k\leftarrow i}(t, t+\ell)$). If the sum of $I_{k\leftarrow i}(t, t+\ell)$ for every $\tau_i \in \tau \setminus \{\tau_k\}$ is strictly smaller than $m \cdot \ell$, the job of τ_k starts its execution within the interval [10], [11].

Due to the difficulty of the precise calculation of $I_{k\leftarrow i}(t, t+\ell)$, existing studies have calculated its upper-bound under different target scheduling algorithms. For example, under FP (both P-FP and NP-FP) [5], if τ_i has a higher priority than τ_k, we can upper-bound $I_{k\leftarrow i}(t, t+\ell)$ by the amount of workload of jobs of τ_i in $[t, t+\ell)$. In an interval of length ℓ, jobs of τ_i can execute up to $W_i(\ell, D_i - C_i - S_i)$ [5] where

$$W_i(\ell, \alpha) = \min\left(\ell, \left\lfloor \frac{\ell+\alpha}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \ell - \left\lfloor \frac{\ell+\alpha}{T_i} \right\rfloor \cdot T_i\right)\right). \quad (1)$$



(a) Interference with a carry-in job: $W_i(\ell, D_i - C_i - S_i)$ in [5]



(b) Interference without a carry-in job: $W_i(\ell, 0)$ [3]

Fig. 1. Interference of $\tau_i(T_i, C_i, D_i, S_i)$ in $[t, t+\ell)$ with different situations.

Here, ℓ denotes the length of interval of interest, and α is a parameter that adjusts the release pattern (See Figs. 1a and 1b). Fig. 1a depicts $W_i(\ell, D_i - C_i - S_i)$, in which an interval of interest begins when the first job of τ_i in the interval starts its execution; the first job of τ_i executes as late as possible, and other following jobs execute as early as possible.

Different from P-FP, NP-FP allows a job of a lower-priority task τ_i to block a job of a higher-priority task τ_k if the job of τ_i starts its execution before the release time of the job of τ_k. In this case, the amount of blocking from the job of τ_i is at most $C_i - 1$ (and also at most the interval length ℓ), because at least one unit of execution should be done before the release time of the job of τ_k. Considering m is an upper-bound of the number of lower-priority jobs which not only start their executions before the release time of the job of τ_k but also keep their executions after the release time, we can derive an upper-bound of interference/blocking under NP-FP as follows [11]

$$\sum_{\tau_i \in \tau \setminus \{\tau_k\}} I_{k\leftarrow i}(t, t+\ell) \leq \sum_{\tau_i \in \tau | i<k} W_i(\ell, D_i - C_i - S_i) + \sum_{\text{m largest } \tau_i \in \tau | i>k} \min\big(C_i - 1, \ell\big). \quad (2)$$

Using the above inequality, we can calculate the response time under NP-FP as follows.

Lemma 1 (Theorem 1 with Lemma 4 in [11]). *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform. If the following inequality holds, the response time of τ_k is no larger than $\ell + C_k - 1$*

$$\text{The RHS (Right-Hand Side) of Eq. (2)} < m \cdot \ell. \quad (3)$$

Since there is at least one time slot in $[t, t+\ell)$ in which a job of τ_k starts its execution, the job finishes its execution no later than $t + \ell + C_k - 1$, where t denotes the job's release time. Section 4.2 will present how to find ℓ that satisfies Eq. (3), including how to update $\{S_i\}_{\tau_i \in \tau}$.

3.2 RTA with an Existing Carry-In Limitation

While Lemma 1 assumes that all tasks can contribute carry-in jobs to the interference, an existing study upper-bounds the contribution from carry-in jobs [12], which needs to calculate interference from three types of tasks: tasks in $\tau^{\mathtt{HC}(k,t)}$, $\tau^{\mathtt{HN}(k,t)}$ and $\tau^{\mathtt{L}(k,t)}$, defined in Section 2. For an upper-bound of $I_{k\leftarrow i}(t, t+\ell)$ for $\tau_i \in \tau^{\mathtt{HC}(k,t)}$, we use $W_i(\ell, D_i - C_i - S_i)$ in Fig. 1a as explained in Section 3.1. As shown in Fig. 1b, an upper-bound for $\tau_i \in \tau^{\mathtt{HN}(k,t)}$ occurs when the release time of the first job of τ_i is aligned with that of the job of τ_k of interest, and every job executes as early as possible. Therefore, $W_i(\ell, 0)$ is an upper-bound of $I_{k\leftarrow i}(t, t+\ell)$ for $\tau_i \in \tau^{\mathtt{HN}(k,t)}$ [3]. Also, an upper-bound of $I_{k\leftarrow i}(t, t+\ell)$ for $\tau_i \in \tau^{\mathtt{L}(k,t)}$ is $\min(C_i - 1, \ell)$, as explained in Eq. (2).

To express the interference from tasks in $\tau^{\mathtt{HN}(k,t)}$ and $\tau^{\mathtt{HC}(k,t)}$ together, we basically add $W_i(\ell,0)$ for every higher-priority task assuming each task does not have its carry-in job. Then, if τ_i is selected to have its carry-in job (we will explain how to select tasks with carry-in jobs later in Section 4.2), we add difference between the two upper-bounds for $\tau^{\mathtt{HC}(k,t)}$ and $\tau^{\mathtt{HN}(k,t)}$, which is denoted as $\mathsf{DIFF}_i(\ell)$ and calculated by (similar to [13])

$$\mathsf{DIFF}_i(\ell) = W_i(\ell, D_i - C_i - S_i) - W_i(\ell,0). \tag{4}$$

Finally, the total interference can be upper-bounded as follows (the core idea is from [12])

$$\sum_{\tau_i\in\tau\setminus\{\tau_k\}} I_{k\leftarrow i}(t,t+\ell) \le \sum_{\tau_i\in\tau|i<k} W_i(\ell,0) + \sum_{\tau_i\in\tau^{\mathtt{HC}(k,t)}} \mathsf{DIFF}_i(\ell) + \sum_{\tau_i\in\tau^{\mathtt{L}(k,t)}\cup\{\tau_k\}} \min\big(C_i-1,\ell\big). \tag{5}$$

Note that $|\tau^{\mathtt{HC}(k,t)}|\le m-1$ and $|\tau^{\mathtt{HC}(k,t)}\cup\tau^{\mathtt{L}(k,t)}\cup\{\tau_k\}|\le m$ hold [12]. Then, we can derive a new response time analysis for NP-FP as follows.

Lemma 2 (Carry-in limitation of [12] into RTA framework in [10], [11]). *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform. If the following inequality holds, the response time of τ_k is no larger than $\ell + C_k - 1$*

$$\text{The RHS (Right-Hand Side) of Eq.(5)} < m\cdot\ell. \tag{6}$$

Different from the second term of the RHS of Eq. (2), the third term in the RHS of Eq. (5) includes τ_k itself, which may result in over-estimation of interference in Lemma 2. However, there is no evidence that Eq. (5) still holds in spite of removing τ_k in the third term in the RHS of Eq. (5); in fact, the removal makes the lemma wrong. Addressing this issue, the next section will develop a new, tight schedulability test for NP-FP.

4 Improved RTA for NP-FP

In this section, we first derive necessary conditions for NP-FP's critical instants. We then develop a new schedulability analysis for NP-FP, which effectively reduces interference from carry-in jobs using the necessary conditions.

4.1 Necessary Conditions for NP-FP's Critical Instants

As Guan *el al.* derived P-FP's critical instants [3], the first step to derive necessary conditions for NP-FP's critical instants is to define a *level-k busy interval for NP-FP*. Different from preemptive scheduling, it is possible for a lower-priority job to block a higher-priority job under non-preemptive scheduling. Therefore, we need to tailor the definition of a level-k busy interval for NP-FP, so as to accommodate the effect of lower-priority jobs' blocking as follows.

Definition 1. *An interval $[t_a,t_b)$ is said to be level-k busy under NP-FP, if m processors in the interval are occupied by jobs, each of which either (i) has a higher priority than k, or (ii) has a priority lower than or equal to k and starts its execution before t_a. Likewise, the interval is said to be level-k idle under NP-FP, if there exists a time slot $[t,t+1)$ within the interval such that at least one processor is idle or occupied by a job which has a priority lower than or equal to k and starts its execution no earlier than t_a.*

Then, if any job of τ_k is released within a level-k busy interval, the job cannot be executed in the interval.

Using the definition, we now derive necessary conditions for NP-FP's critical instants, starting from the following property for NP-FP.

- Under NP-FP, the release/execution patterns of jobs of τ_k *may affect* execution of jobs whose priorities are higher than τ_k.

The property indicates that if we shift all the jobs of τ_k released before or at t by 1, we cannot guarantee $[t-1,t)$ that was level-k busy (*likewise* idle) before the shift is still level-k busy (*likewise* idle) after the shift. This means that the existence of a level-k busy interval $[t-1,t)$ does not always enable J_k^q (released at t) to yield a critical instant, which is different from P-FP.

Therefore, we need to handle the following two cases differently depending on possibility to shift the release time without changing the level-k busy state of $[t-1,t)$.

$\mathtt{X}_1$. Shifting J_k^q's release time from t to $t-1$ cannot change whether $[t-1,t)$ is level-k busy or not; and
$\mathtt{X}_2$. Other than $\mathtt{X}_1$.

For $\mathtt{X}_1$, we can apply the same approach as P-FP, which is, there are at most $m-1$ carry-in jobs in an interval starting at t. That is, a critical instant for τ_k occurs with J_k^q whose release time is t, only if $[t-1,t)$ is level-k idle. For $\mathtt{X}_2$, we need to handle this situation differently. From $\mathtt{X}_1$ and $\mathtt{X}_2$, we derive the following necessary conditions for NP-FP's *critical instants*.

Theorem 1. *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform. A critical instant for a task $\tau_k\in\tau$ occurs, only if its job J_k^q whose release time is t is associated with one of the following situations.*

$\mathtt{S}_\mathtt{A}$. *$[t-1,t)$ is level-k idle; or*
$\mathtt{S}_\mathtt{B}$. *$[t-1,t)$ is level-k busy and J_k^{q-1}'s release time is $t-T_k$.*

Proof. We first investigate $\mathtt{X}_1$. Since we can shift J_k^q's release time without changing the level-k busy state of $[t-1,t)$, we should exclude the situation where $[t-1,t)$ is level-k busy from a critical instant in that we can increase J_k^q's response time by moving its release time from t to $t-1$. Therefore, $\mathtt{S}_\mathtt{A}$ includes all situations for critical instants under $\mathtt{X}_1$.

For $\mathtt{X}_2$, we focus on the situation where $[t-1,t)$ is level-k busy (because the other situation belongs to $\mathtt{S}_\mathtt{A}$). If J_k^{q-1}'s release time is earlier than $t-T_k$, we can shift J_k^q's release time from t to $t-1$ without changing the level-k busy state of $[t-1,t)$; this implies that the current situation belongs to $\mathtt{X}_1$. Therefore, $\mathtt{S}_\mathtt{B}$ includes all situations for critical instants under $\mathtt{X}_2$ which do not belong to $\mathtt{S}_\mathtt{A}$. □

To utilize NP-FP's critical instants for a tighter schedulability test, we need to consider both conditions $\mathtt{S}_\mathtt{A}$ and $\mathtt{S}_\mathtt{B}$, which will be discussed in the next section.

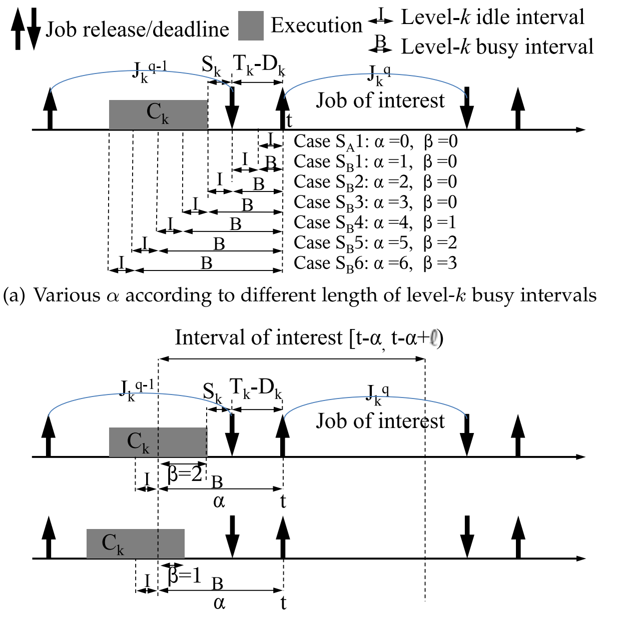
4.2 Improved RTA with Carry-In Limitation

Let us consider an interval starting at $t-\alpha$ ($\alpha\ge 0$) such that $[t-\alpha,t)$ is level-k busy and $[t-\alpha-1,t-\alpha)$ is level-k idle; by the definition of $\mathtt{S}_\mathtt{A}$ and $\mathtt{S}_\mathtt{B}$, $\alpha=0$ and $\alpha>0$ hold for $\mathtt{S}_\mathtt{A}$ and $\mathtt{S}_\mathtt{B}$, respectively. If we calculate response times of a task for all possible α and take the maximum, we can derive an upper-bound of the task's response time.[2] Here we have two challenges: (i) how to limit the number of cases for α (which seems infinitely many), and (ii) how to calculate a tight upper-bound of interference/blocking by limiting the number of carry-in jobs.

To handle (i), we focus on a time instant in which the execution of J_k^{q-1} starts (denoted by t_0). By the definition of a level-k busy interval, $[t_0,t_0+1)$ cannot be level-k busy. This is because, if $[t_0,t_0+1)$ is level-k busy, the execution of J_k^{q-1} whose priority is k cannot start at t_0. This derives a property that reduces search space of α from infinite cases to only a small number of cases, recorded as follows.

Lemma 3. *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform, and t denotes J_k^q's release time. If $[t-\alpha,t)$ is level-k*

2. The generic technique was developed in [14], but the notion of a busy interval and underlying principles are different in that [14] was designed for P-EDF.



(a) Various α according to different length of level-k busy intervals

(b) Various β for given α

Fig. 2. Interval of interest $[t-\alpha, t-\alpha+\ell)$ for deriving a new schedulability test.

busy and $[t-\alpha-1, t-\alpha)$ is level-k idle, $t-\alpha$ is no earlier than t_0+1, where t_0 is the time instant at which J_k^{q-1} starts its execution, and $\alpha \geq 0$.

Proof. Suppose that $t-\alpha$ is t_0 or earlier. Then, by definition, J_k^{q-1} cannot execute in $[t_0, t)$, which contradicts the definition of t_0. □

Thanks to Lemma 3, we need to investigate only several cases of α instead of infinitely many cases. For example, the only possible choices for α in Fig. 2a are 0 for $\mathtt{S_A}$, and 1, 2, 3, 4, 5 and 6 for $\mathtt{S_B}$.

Now, we address (ii)—how to calculate a tight upper-bound of interference/blocking for given α. To this end, we need to know the contribution of J_k^{q-1} to make $[t-\alpha, t)$ level-k busy. Let β denote the amount of execution of J_k^{q-1} in $[t-\alpha, t)$, as shown in Fig. 2b; by Lemma 3, β cannot be larger than $C_k - 1$, yielding $0 \leq \beta \leq C_k - 1$. Along with the definition of $\mathtt{S_A}$ and $\mathtt{S_B}$, $\alpha = \beta = 0$ holds for $\mathtt{S_A}$, and $\alpha > 0$ and $0 \leq \beta \leq C_k - 1$ hold for $\mathtt{S_B}$. Then, for given α and β, the following lemma derives some properties to be used for calculating response times.

Lemma 4. *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform, and t denotes J_k^q's release time. Also, suppose that $[t-\alpha-1, t-\alpha)$ is level-k idle, $[t-\alpha, t)$ is level-k busy, and β denotes the amount of execution of J_k^{q-1} in $[t-\alpha, t)$, where $\alpha \geq 0$ and $\beta \geq 0$. Then, P_1–P_5 hold as follows.*

- P_1. $|\tau^{\mathtt{HC}(k,t-\alpha)}| \leq m-1$.
- P_2. *The amount of execution of a carry-in job of $\tau_i \in \tau^{\mathtt{HC}(k,t-\alpha)}$ in an interval starting at $t-\alpha$ is at most $C_i - 1$.*
- P_3. $|\tau^{\mathtt{HC}(k,t-\alpha)} \cup \tau^{\mathtt{L}(k,t-\alpha)}| \leq m$, *if* $\beta = 0$,
  $|\tau^{\mathtt{HC}(k,t-\alpha)} \cup \tau^{\mathtt{L}(k,t-\alpha)}| \leq m-1$, *if* $\beta > 0$.
- P_4. *The amount of execution of a carry-in job of $\tau_i \in \tau^{\mathtt{L}(k,t-\alpha)}$ in an interval starting at $t-\alpha$ is at most $C_i - 1$.*
- P_5. $0 \leq \beta \leq C_k - 1$.

Proof. Proof of P_1: To make $[t-\alpha-1, t-\alpha)$ level-k idle (defined in Definition 1), (a) at least one processor is idle in $[t-\alpha-1, t-\alpha)$ or (b) $[t-\alpha-1, t-\alpha)$ should be occupied by at least one job which satisfies the following two conditions: (b1) it has a priority lower than or equal to k and (b2) it starts its execution at $t-\alpha-1$. Suppose $|\tau^{\mathtt{HC}(k,t-\alpha)}| \geq m$. Since each task belonging to $\tau^{\mathtt{HC}(k,t-\alpha)}$ has its active job at $t-\alpha-1$, we have at least m active jobs at $t-\alpha-1$. Therefore, (a) is impossible, and it is impossible for a job satisfying both (b1) and (b2) to occupy $[t-\alpha-1, t-\alpha)$. By contradiction, P_1 holds.

Proof of P_2: By the definition of the level-k idle interval, there exists at least one job (called J) in $[t-\alpha-1, t-\alpha)$, which satisfies (b1) and (b2). This implies that all active jobs whose priority is higher than J are executed in $[t-\alpha-1, t-\alpha)$; otherwise, J cannot be executed in $[t-\alpha-1, t-\alpha)$. Therefore, by the definition of a carry-in job, all jobs whose invoking tasks belong to $\tau^{\mathtt{HC}(k,t-\alpha)}$ are executed in $[t-\alpha-1, t-\alpha)$. Therefore, P_2 holds.

Proof of P_5: Since Lemma 3 proved that $t-\alpha$ cannot be earlier than t_0+1, at least one time unit of J_k^{q-1}'s execution is excluded from $[t-\alpha, t)$, which proved P_5.

Proof of P_3: While τ_k cannot have its carry-in job for an interval starting at t (since J_k^q is released at t), τ_k can have its carry-in job for an interval starting at $t-\alpha$ due to the execution of J_k^{q-1} in the interval. Considering P_5 and the definition of β, we can consider two cases $\beta = 0$ and $1 \leq \beta \leq C_i - 1$. The former and the latter imply there exists no and one carry-in job of τ_k in an interval starting at $t-\alpha$, respectively. Considering tasks in $\tau^{\mathtt{HC}(k,t-\alpha)}$ or $\tau^{\mathtt{L}(k,t-\alpha)}$ should execute in $[t-\alpha-1, t-\alpha)$, if $\beta = 0$, it holds $|\tau^{\mathtt{HC}(k,t-\alpha)} \cup \tau^{\mathtt{L}(k,t-\alpha)}| \leq m$, which corresponds to the first case of P_3. If $1 \leq \beta \leq C_i - 1$, a carry-in job of τ_k should also execute in $[t-\alpha-1, t-\alpha)$. Therefore, the number of tasks in $\tau^{\mathtt{HC}(k,t-\alpha)}$ or $\tau^{\mathtt{L}(k,t-\alpha)}$ cannot exceed $m-1$.

Proof of P_4: A lower-priority job can block a higher-priority job only if the lower-priority job starts its execution before the release time of the higher-priority job. This means that any lower-priority carry-in job of tasks belonging to $\tau^{\mathtt{L}(k,t)}$ should perform its execution in $[t-\alpha-1, t-\alpha)$, yielding P_4. □

Using Lemma 4, we can calculate the response time for a given pair of α and β, as stated in the following lemma.

Lemma 5. *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform, and t denotes J_k^q's release time. Also, suppose that $[t-\alpha-1, t-\alpha)$ is level-k idle, $[t-\alpha, t)$ is level-k busy, and β denotes the amount of execution of J_k^{q-1} in $[t-\alpha, t)$, where $\alpha \geq 0$ and $\beta \geq 0$. If the following inequality holds, the response time of J_k^q is no larger than $R_k = \ell - \alpha + C_k - 1$*

$$\beta + \sum_{\tau_i \in \tau | i < k} W_i(\ell, 0) + \sum_{\tau_i \in \tau^{\mathtt{HC}(k,t-\alpha)}} \mathsf{DIFF}_i(\ell) + \sum_{\tau_i \in \tau^{\mathtt{L}(k,t-\alpha)}} \min\big(C_i - 1, \ell\big) < m \cdot \ell. \tag{7}$$

Note that $\tau^{\mathtt{HC}(k,t-\alpha)}$, $\tau^{\mathtt{L}(k,t-\alpha)}$ and β in Eq. (7) satisfy P_1, P_3, and P_5 in Lemma 4. We also note that $\mathsf{DIFF}_i(\ell)$ can be $W_i(\ell+1, D_i - C_i - S_i) - 1 - W_i(\ell, 0)$ once we apply P_2, which is slightly smaller than Eq. (4).

Proof. By definition, the Left-Hand Side (LHS) of Eq. (7) except the term of β is an upper-bound of the sum of contributions of all tasks except τ_k to either make $[t-\alpha, t)$ level-k busy or interfere with (or block) J_k^q in $[t, t-\alpha+\ell)$. Therefore, if Eq. (7) holds, it means that either (i) J_k^q starts its execution in $[t, t-\alpha+\ell)$ or (ii) $[t-\alpha, t)$ is level-k idle. Since J_k^q's release time is t, Case (i) implies that the response time is no larger than $\ell - \alpha + C_k - 1$. Also, Case (ii) contradicts the level-k busy state of $[t-\alpha, t)$. This prove the lemma. □

One may wonder how to calculate the maximum of the RHS of Eq. (7), which entails the right choice of $\tau^{\mathtt{HC}(k,t)}$ and $\tau^{\mathtt{L}(k,t)}$ without compromising P_1 and P_3 in Lemma 4. To this end, we construct an empty set $\mathcal{S} = \emptyset$, and add $\mathsf{DIFF}_i(\ell)$ for all tasks with a higher priority than k and $\min(C_i - 1, \ell)$ for all tasks with a lower priority than k. If we focus on P_3 only, the LHS of Eq. (7) is maximized when we choose tasks each of whose value in $\mathcal{S}$ is one of the m largest values (or choose all tasks if the number of elements in $\mathcal{S}$ is smaller than

m) and include the tasks in either $\tau^{\mathtt{HC}(k,t)}$ or $\tau^{\mathtt{L}(k,t)}$ depending on their priority (higher than k for the former and lower than k for the latter). If we consider P_1 along with P_3, the above result holds only when the number of tasks in $\tau^{\mathtt{HC}(k,t)}$ is not larger than $m-1$ (i.e., complying P_1). If the number of tasks in $\tau^{\mathtt{HC}(k,t)}$ is m, we reconstruct $\tau^{\mathtt{HC}(k,t)}$ by choosing tasks each of whose value in $\mathcal{S}$ is one of the $m-1$ largest values, and $\tau^{\mathtt{L}(k,t)}$ by choosing a task whose value in $\mathcal{S}$ is the largest among values from tasks each of whose priority is lower than k in $\mathcal{S}$. This takes linear time, since we can choose the m largest values in a given set in linear time [14].

Now, we explain how to find ℓ that satisfies Eq. (7) in Lemma 5 (*likewise* Eq. (3) in Lemma 1, and Eq. (6) in Lemma 2), including how to update S_i; note that the original idea for this iteration technique is given in [5]. Starting from $\ell^0 = 1$, we test Eq. (7). If the inequality holds, the response time of τ_k is $\ell^0 + C_k - 1$. Otherwise, we set ℓ^1 as follows [11]:

$$\ell^{x+1} = 1 + \left\lfloor \frac{1}{m} \cdot \Big(\text{The LHS of Eq.(7) with } \ell^x \Big) \right\rfloor. \qquad (8)$$

We repeat this process for ℓ^0, ℓ^1, ℓ^2 and so on, until it finds ℓ^x satisfying the inequality (schedulable task) or $\ell^x > D_k - C_k + 1$ (unschedulable task).

To derive a tighter schedulability test for a given ℓ, slack value S_i can be used as follows. First, the response time of every task is calculated with $S_i = 0$ for every $\tau_i \in \tau$. For each task τ_i whose response time R_i is strictly smaller than the relative deadline D_i, we update its slack value as $S_i = D_i - R_i$. Then, we repeat the same procedure until every task's response time is no larger than its relative deadline (schedulable task set), or there is no update for any slack value (unschedulable task set).

As of now, the LHS of Eq. (7) can calculate the response time of a task using an upper-bound of interference/blocking for a given pair of α and β. The remaining issue is how to calculate the LHS of Eq. (7) with every possible pair of α and β, in an efficient manner. To this end, we investigate two cases $\beta = 0$ and $\beta > 0$, deriving their properties that can significantly reduce the number of candidates to be calculated for $\mathtt{S_A}$ and $\mathtt{S_B}$.

First, we derive the following property regarding the relationship between the response time under $\mathtt{S_A}$ (i.e., $\beta = \alpha = 0$) and that under $\mathtt{S_B}$ with $\beta = 0$.

Lemma 6. *We now compare the response time of J_k^q calculated by Lemma 5 with $\beta = 0$ and $\alpha = 0$ (denoted by R'_k), and that with $\beta = 0$ and any valid $\alpha > 0$ (denoted by $R''_k(\alpha)$). Then, $R''_k(\alpha) = R'_k - \alpha$ holds.*

Proof. Since the LHS of Eq. (7) is independent of α, $\beta = 0$ with different α yields the same ℓ that satisfies Eq. (7). Since $R_k = \ell - \alpha + C_k - 1$, the response time of J_k^q with $\beta = 0$ and $\alpha > 0$ is $X - \alpha$ if that of J_k^q with $\beta = \alpha = 0$ is X. Note that if $X - \alpha$ is no larger than 0, this means that $[t-\alpha, t)$ is not level-k busy, which yields a contradiction. This proves the lemma. □

Since we need to take the maximum of all the response times under $\mathtt{S_A}$ and $\mathtt{S_B}$, the lemma indicates that we do not need to calculate the response time under $\mathtt{S_B}$ with $\beta = 0$ and $\alpha > 0$, as long as we apply Lemma 5 for the case of $\mathtt{S_A}$ (i.e., $\beta = \alpha = 0$).

Second, the following lemma significantly reduces the number of candidates for pairs of α and β when β is positive for $\mathtt{S_B}$.

Lemma 7. *Suppose that Lemma 5 with a given positive β and $\alpha = \beta + (T_k - D_k + S_k)$ calculates that the response time of J_k^q is X $(\le D_k)$. Then, Lemma 5 with the given β and any valid α calculates that the response time of J_k^q is no larger than X.*

Proof. We will prove two properties for given α and β: (i) $\beta + (T_k - D_k + S_k)$ is the smallest valid value for α, and (ii) the smaller the α, the larger the response time.

Case (i): By the definition of slack value, J_k^{q-1} cannot be executed in $[t - (T_k - D_k + S_k), t)$ as shown in Fig. 2b. Therefore, the smallest valid α for a given β is $\alpha = \beta + (T_k - D_k + S_k)$ as shown in the first situation in Fig. 2b.

Case (ii): Since the LHS of Eq. (7) does not depend on α, $R_k = \ell - \alpha + C_k - 1$ increases as α decreases.

By (i) and (ii), the lemma holds. □

Thanks to the above two lemmas, it is sufficient to check only one α for each β, which will be used for developing an efficient RTA.

So far, we developed RTA for NP-FP under $\mathtt{S_A}$ and $\mathtt{S_B}$ in Lemma 5, respectively, and derived properties that reduce search space of candidates in Lemmas 6 and 7. Now, we assemble all the theories into the following theorem.

Theorem 2. *Suppose that a task set τ is scheduled by NP-FP on an m-processor platform. Then, the response time of each task $\tau_k \in \tau$ is upper-bounded by $R_k = \max\big(R_k(0), \max_{\beta=1}^{C_k-1} R_k(\beta)\big)$, where*

- *$R_k(0)$ denotes an upper-bound of the response time of J_k^q with $\beta = \alpha = 0$, which is calculated by Lemma 5; and*
- *$R_k(\beta > 0)$ denotes an upper-bound of the response time of J_k^q with given $\beta > 0$ and $\alpha = \beta + (T_k - D_k + S_k)$, which is calculated by Lemma 5.*

Proof. By Theorem 1, we can calculate the response time of each task τ_k by taking the maximum of the response times from Lemma 5. The only remaining issue is how to calculate an upper-bound of the LHS of Eq. (7) with all possible combinations of α and β. By Lemma 4, we can limit the choices for β (i.e., no larger than $C_k - 1$). For $\beta = 0$, Lemma 6 proves that the response time with $\beta = 0$ and $\alpha > 0$ is less than that with $\beta = \alpha = 0$. For $\beta = 1, 2, \ldots, C_k - 1$, Lemma 7 proves that the response time for given β and any valid α is upper-bounded by that for the same β and $\alpha = \beta + (T_k - D_k + S_k)$. Therefore, the theorem holds. □

For completeness, here we summarize the detailed calculation of Theorem 2. Basically, we calculate each of $R_k(0)$ and $R_k(\beta > 0)$ for given β independently. For calculation of each response time, we need to address three issues: (i) how to iterate ℓ, (ii) how to reclaim the slack value, and (iii) how to find the maximum of the corresponding sum of interference/blocking (e.g., the LHS of Eq. (7)) among many choices of $\tau^{\mathtt{HC}(k,\cdot)}$ and $\tau^{\mathtt{L}(k,\cdot)}$, all of which share the same techniques explained right after Lemma 5. Note that one may wonder how to calculate $\alpha = \beta + (T_k - D_k + S_k)$ for $R_k(\beta)$ since S_k is derived from $R_k(\beta)$ itself. However, S_k is calculated as a part of (ii). That is, for given β and initial slack values $\{S_k\}_{\tau_i \in \tau} = 0$, we calculate $\{R_k(\beta)\}_{\tau_k \in \tau}$ using $\{S_k\}_{\tau_k \in \tau}$. Then, we update the slack value of every task τ_k if the task satisfies $D_k - R_k(\beta) > 0$. We will repeat this process until every task τ_k satisfies $R_k(\beta) \le D_k$ (schedulable task set for given β) or there is no update for any slack value (unschedulable task set for given β).

Time-Complexity. Now, we analyze time-complexity of Theorem 2 using the big-O notation with n and other task parameters, where n denotes the number of tasks in each task set τ. We first consider the case without exploiting slack reclamation, i.e., S_i is set to 0 for every $\tau_i \in \tau$. If we calculate $R_k(0)$ or $R_k(\beta)$ for a given β in Theorem 2, it needs $O(n \cdot D_k)$ operations. Since we need to calculate $R_k(0)$ and $R_k(\beta)$ for $\beta = 1, 2, \ldots, C_k - 1$, the time-complexity of calculating the response time of a single task τ_k is $O(n \cdot D_k \cdot C_k)$ and that of calculating the response time of every task in a task set τ is $O(n^2 \cdot \max_{\tau_i \in \tau} D_i \cdot \max_{\tau_i \in \tau} C_i)$.

We now consider the case with slack reclamation for Theorem 2. Then, the time-complexity is $O(n^3 \cdot \max_{\tau_i \in \tau} D_i^2 \cdot \max_{\tau_i \in \tau} C_i)$ since the entire process for calculating the response time of every task in τ

TABLE 1
The Number of Task Sets Proven Schedulable by `GYG`, `DBM`, `GYD2`, `LeSh` and `New`

	$T_{max} = 10$						$T_{max} = 1,000$					
	The number of schedulable task sets					Ratio	The number of schedulable task sets					Ratio
m	`GYG`	`DBM`	`GYD2`	`LeSh`	`New`	`New`/`LeSh`	`GYG`	`DBM`	`GYD2`	`LeSh`	`New`	`New`/`LeSh`
2	90,373	363,239	394,511	548,863	625,976	**114.0%**	145,165	125,178	184,476	260,416	274,459	**105.4%**
4	19,730	245,284	283,874	388,574	444,006	**114.3%**	73,543	68,224	112,382	157,312	163,028	**103.6%**
8	3,020	178,473	221,092	279,918	309,664	**110.6%**	41,262	37,698	68,663	94,645	96,931	**102.4%**

can be repeated up to $\sum_{\tau_i \in \tau} D_i = O(n \cdot \max_{\tau_i \in \tau} D_i)$ times. Note that the time-complexity of Lemmas 1 and 2 with slack reclamation is $O(n^3 \cdot \max_{\tau_i \in \tau} D_i^2)$. Due to the calculation of $R_k(\beta)$ for multiple β, the time-complexity of Theorem 2 has additional term of $\max_{\tau_i \in \tau} C_i$, compared to that of Lemmas 1 and 2. Since our main interest is offline schedulability guarantee, the time-complexity of Theorem 2 is reasonable.

Sustainability. When we derive Theorem 2, we implicitly assume that every job's actual execution time is equal to the worst-case execution time of the task invoking the job, which the case yields the longest response time. However, one may wonder whether the theorem remains valid even if the actual execution time is arbitrarily less than the worst-case execution time.

If we focus on interference calculation in the LHS of Eq. (7), each term assumes the case that incurs the maximum interference, meaning that the interference will decrease if the actual execution time is less than the worst-case execution time. Also, the properties in Lemmas 4, 6 and 7 easily hold with the actual execution time which is less than the worst-case one. Therefore, Theorem 2 is sustainable with respect to execution times.

5 Evaluation

In this section, we evaluate the schedulability performance of our proposed NP-FP schedulability test, which is Theorem 2 in this paper (denoted by `New`). To this end, we perform simulations, and compare the number of task sets deemed schedulable by `New`, with that by `GYG`, `LeSh`, `DBM`, and `GYD2`, which denote existing NP-FP schedulability tests in [13], [11], and [15], and Lemma 2 in this paper (i.e., applying the carry-in limitation technique of [12] into RTA framework in [11]), respectively. `LeSh` corresponds to Lemma 1 in this paper, which is the state-of-the-art schedulability test *without carry-in limitation* for NP-FP, while `GYD2` is regarded as the state-of-the-art schedulability test *with carry-in limitation* for NP-FP. `DBM` can be a NP-FP schedulability test since [15], [16] developed a schedulability test for FP with deferred preemption that is a generalization of NP-FP.[3]

We generate task sets using an existing popular set generation technique used in many studies, e.g., [17], [18]. We have three input parameters to accommodate various characteristics of real-time applications: (a) the number of processors m, (b) task utilization (C_i/T_i) distribution, and (c) the maximum period. For (a), we consider three options: 2, 4, and 8 processors. For (b), we consider ten options: bimodal distribution with parameters 0.1, 0.3, 0.5, 0.7 and 0.9, and exponential distribution with the same parameters, which are described in [18]. Finally, we consider two different parameters for the maximum period (T_{max}): 10 and 1,000.

For each task τ_i, T_i is uniformly chosen in $[1, T_{max}]$ where T_{max} is given by (c), C_i is determined based on (b), and D_i is set to T_i (i.e., implicit-deadline task). Note that we only consider positive integer values for T_i, C_i and D_i as explained in Section 2; if the generated value of each task parameter is not an integer value, we set the task parameter to the closest integer value.

If we consider (a), (b), and (c) all together, we have 60 $(3 \cdot 10 \cdot 2)$ options in total. For each option, we repeat the following procedure and generate 100,000 task sets.

(1) We generate $m+1$ tasks since m tasks are trivially schedulable on an m-processor platform.
(2) We check the exact feasibility condition of preemptive scheduling, i.e., $\sum_{\tau_i \in \tau} C_i/T_i \leq m$.
(3) If the current task set fails to pass 2), we abandon the task set, and go to 1). Otherwise, we include this task set for evaluation, add one more task to the current task set, and go to 2).

Then, for each combination of m and T_{max}, we finally have 1,000,000 task sets in total.

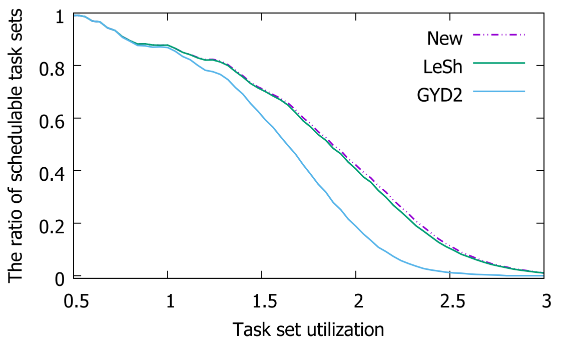
We count the number of task sets deemed schedulable by `GYG`, `DBM`, `GYD2`, `LeSh`, and `New` among 1,000,000 task sets for each pair of m and T_{max} when task priority is determined by Rate Monotonic (RM) [1] (the smaller the period, the higher the priority), and present the number in Table 1. Note that the feasibility condition presented in 2) is exact for preemptive scheduling; since many task sets we tested are inherently unschedulable by every non-preemptive scheduling (but we do not know the exact feasibility condition for non-preemptive scheduling), the number of task sets proven schedulable by each schedulability test seems relatively low.

We can easily observe that `GYG` exhibits poorer schedulability performance than other schedulability tests (i.e., `DBM`, `GYD2`, `LeSh` and `New`), which indicates that the RTA framework is more effective than any other schedulability test framework for NP-FP. When its comes to schedulability tests with the RTA framework, `New` and `DBM` are the best and worst in terms of the number of schedulable task sets.

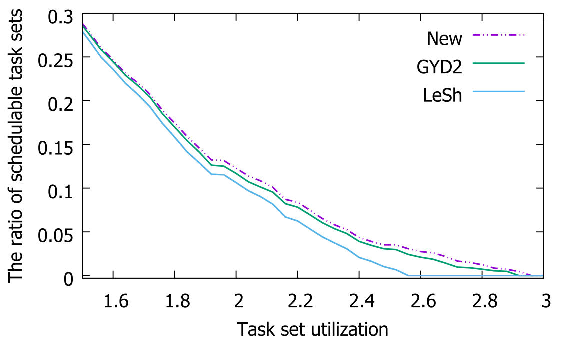
To investigate the schedulability performance of the three best schedulability tests (i.e., `GYD2`, `LeSh`, and `New`) with different settings, we select exponential distribution with parameters 0.1 and 0.9, which yield a small and large average number of tasks in each task set, respectively. Figs. 3a and 3b present the former and latter when $m = 4$; here, the x- and y-axes are task set utilization and the ratio of schedulable task sets, respectively. If we focus on task sets with exponential distribution 0.9 where the average number of tasks in each task set is 7.6, `LeSh` is better than `GYD2` as shown in Fig. 3a. This is because, `GYD2` gets small benefit from carry-in limitation when the number of tasks in each task set is small, while its schedulability performance is degraded by including τ_k itself into interference. On the other hand, for task sets with exponential distribution 0.1 (where the average number of tasks in each task set is 22.2), the interference reduction from carry-in limitation for `GYD2` suppresses its disadvantage, yielding better schedulability of `GYD2` over `LeSh`. While there is no dominance relationship between `GYD2` and `LeSh`, `New` dominates both `GYD2` and `LeSh`; also, `GYD2` dominates `DBM`, thereby yielding a dominance relationship of `New` over `DBM`.

Since we would like to investigate the schedulability improvement by the necessary conditions of critical instants we derived, the main point observed from Table 1 should be the schedulability

3. In [15], there are two types of schedulability tests: without and with the limited carry-in technique. Since the former corresponds to `LeSh`, we denote the latter as `DBM` so as to figure out the impact of [15] associated with the limited carry-in technique.



(a) Task sets with exponential distribution 0.9 where the average number of tasks in each task set is 7.6



(b) Task sets with exponential distribution 0.1 where the average number of tasks in each task set is 22.2

Fig. 3. The ratio of schedulable task sets by New, LeSh, and GYD2 when $m = 4$.

performance difference between New and the state-of-the-art existing schedulability tests (i.e., LeSh and GYD2). Therefore, in addition to the number of task sets deemed schedulable by LeSh and New, we also present their ratio; here we do not present ratio between GYD2 and New since the number of schedulable task sets by LeSh is larger than that by GYD2. As shown in the table, New can find up to 14.3 percent additional schedulable task sets, which are not covered by the corresponding schedulability test LeSh. In particular, while the improvement stands out with $T_{max} = 10$ due to a high distribution of $C_i = 1$ (resulting in $\beta = 0$), the improvement is reasonable (up to 5.4 percent) even for a more general case (i.e., $T_{max} = 1,000$).

6 Related Work

When it comes to finding critical instants and developing schedulability tests for P-FP, a seminal paper by Liu and Layland [1] found a critical instant and developed a sufficient schedulability test on a uniprocessor platform. Later, Audsley et al. derived a response time analysis using the critical instant, yielding an exact schedulability test [2]. The RTA framework was extended to a multiprocessor platform, obliviously to critical instants by Bertogna and Cirinei [5]. Then, Guan et al., and David and Burns [3], [4] found necessary conditions for critical instants on a multiprocessor platform, which significantly improved the state-of-the-art schedulability test for P-FP on the platform.

While P-FP's schedulability tests progressed along with its critical instants, the same cannot be said to NP-FP on a multiprocessor platform. For a uniprocessor platform, a schedulability test for NP-FP was developed [7], [8]. When it comes to multiprocessors, a few studies developed schedulability tests on a multiprocessor platform [11], [12], [13], [19]. Although some of the studies by Guan et al. [12], [13] utilized the notion of a busy interval, no study identified critical instants under NP-FP on a multiprocessor platform, yielding a room for further improvement. Also, there are some attempts to develop schedulability analysis of NP-EDF [10], [11], [20].

Here, we should clarify a couple of studies that aim at reducing interference by limiting carry-in jobs' interference without deriving critical instants. Studies in [16] by Davis et al. and [21] by Marinho et al. implicitly use a claim that necessary conditions for critical instants for P-FP *always* hold for FP with limited preemption and FP with deferred preemption, respectively, both of which are a generalization of NP-FP.[4] However, the claim is wrong—the lemma holds only when $\mathtt{S_A}$ holds. The study in [16] was later corrected in [15] (denoted by DBM in this paper) by accounting for the push-through blocking effect (i.e., by adding $C_k - 1$ amount of interference from the previous job of τ_k itself); however, adding additional interference yields pessimistic calculation of interference, and therefore DBM does not perform well for NP-FP, even compared to GYD2 and LeSh that does not limit carry-in jobs' interference as shown in Section 5; this is inevitable in that DBM targets FP with a more general preemption policy, instead of being specialized for NP-FP. In summary, while some studies for a generalization of NP-FP tried to limit carry-in jobs' interference, they does not derive necessary conditions for critical instants, yielding less schedulability improvement for NP-FP.

4. For limited preemption policies, there are two types of approaches: the eager and lazy preemption policies. Since non-preemptive scheduling is a special case of both eager and lazy preemption policies, there is no issue to choose either the eager or lazy preemption policy in this paper.

7 Conclusion

In this paper, we sought to answer (i) how to find critical instants for NP-FP and (ii) how to improve a schedulability test for NP-FP using critical instants. To this end, we derived necessary conditions for NP-FP's critical instants on a multiprocessor platform. Then, we developed a tighter schedulability test for NP-FP by utilizing the necessary conditions. Our simulation results showed that the test can find up to 14.3 percent additional NP-FP-schedulable task sets, which are not covered by the corresponding existing schedulability test.

We have two directions for our future work. First, we want to find a relationship between task priority assignment and schedulability improvement achieved by critical instants. We may find other task priority assignment whose schedulability improvement is more significant than RM. Second, we would like to apply the technique that finds NP-FP's critical instants to FP with more general preemption policies. For example, it would be interesting to find critical instants of a task set consisting of preemptive and non-preemptive tasks [10], [11], which makes sense in many practical systems.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2017R1A2B2002458), and the Ministry of Education (NRF-2016R1D1A1B03930580).

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Proc. IEEE Workshop Real-Time Operating Syst. Softw.*, May 1991, pp. 133–137.
[3] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proc. IEEE Real-Time Syst. Symp.*, 2009, pp. 387–397.

[4] R. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Syst.*, vol. 47, pp. 1–40, 2011.

[5] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. IEEE Real-Time Syst. Symp.*, 2007, pp. 149–160.

[6] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems: A survey," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 3–15, Feb. 2013.

[7] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," INRIA, Le Chesnay, France, Tech. Rep. RR-2966, 1996.

[8] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, pp. 239–272, 2007.

[9] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Laboratory Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1983.

[10] J. Lee and K. G. Shin, "Controlling preemption for better schedulability in multi-core systems," in *Proc. IEEE Real-Time Syst. Symp.*, 2012, pp. 29–38.

[11] J. Lee and K. G. Shin, "Improvement of real-time multi-core schedulability with forced non-preemption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1233–1243, May 2014.

[12] N. Guan, W. Yi, Q. Deng, Z. Gu, and G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling," *J. Syst. Archit.*, vol. 57, no. 5, pp. 536–546, 2011.

[13] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, "New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms," in *Proc. IEEE Real-Time Syst. Symp.*, 2008, pp. 137–146.

[14] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *Proc. IEEE Real-Time Syst. Symp.*, 2007, pp. 119–128.

[15] R. I. Davis, A. Burns, J. Marinho, V. Nelis, S. M. Petters, and M. Bertogna, "Global and partitioned multiprocessor fixed priority scheduling with deferred preemption," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, pp. 47:1–47:28, 2015.

[16] R. I. Davis, A. Burns, J. Marinho, V. Nelis, S. M. Petters, and M. Bertogna, "Global fixed priority scheduling with deferred preemption," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2013, pp. 1–11.

[17] T. P. Baker, "Comparison of empirical success rates of global versus partitioned fixed-priority and EDF scheduling for hard real time," Dept. Comput. Sci., Florida State Univ., Tallahasee, FL, USA, Tech. Rep. TR-050601, 2005.

[18] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2011, pp. 235–244.

[19] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Syst.*, vol. 32, no. 1, pp. 9–20, 2006.

[20] H. Leontyev and J. H. Anderson, "A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling," in *Proc. IEEE Real-Time Syst. Symp.*, 2008, pp. 375–384.

[21] J. Marinho, V. Nelis, S. M. Petters, M. Bertogna, and R. I. Davis, "Limited pre-emptive global fixed task priority," in *Proc. IEEE Real-Time Syst. Symp.*, 2013, pp. 182–191.