Design and Timing Guarantee for Non-Preemptive Gang Scheduling

Seongtae Lee*, Nan Guan†, Jinkyu Lee*‡

Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea*
Department of Computer Science, City University of Hong Kong, Hong Kong SAR†
yuns0509@skku.edu, nanguan@cityu.edu.hk, jinkyu.lee@skku.edu

***Abstract*—Due to its efficient and predictable utilization of modern computing units, recent studies have paid attention to gang scheduling in which all threads of a real-time task should be concurrently executed on different processors. However, the studies have been biased to preemptive gang scheduling, although non-preemptive gang scheduling (NPG) is practical for inherently non-preemptive tasks and tasks that incur large preemption overhead. In this paper, focusing on a new type of priority-inversion incurred by NPG, we design a generalized NPG framework, called NPG*, under which each task has an option to allow or disallow the situation that incurs the priority-inversion specialized for NPG. To demonstrate the effectiveness of NPG* in terms of timing guarantees, we target NPG*-FP by employing fixed-priority scheduling (FP) as a prioritization policy, and develop the first NPG*-FP schedulability test and its improved version under a given assignment of the allowance/disallowance option to each task. We then develop the optimal allowance/disallowance assignment algorithm, which finds an assignment (if exists) that makes a target task set schedulable by the proposed schedulability tests. Via simulations, we demonstrate that the assignment algorithm associated with the schedulability tests for NPG*-FP can find a number of additional schedulable task sets, each of which has not been covered by the traditional NPG framework.**

I. Introduction

To support parallel programming models (e.g., [1–3]) that utilize parallel architectures (e.g., [4–6]), scheduling a set of parallel real-time tasks has been widely studied in the real-time systems community (e.g., [7–9]). Among them, gang scheduling determines the execution order of jobs invoked by a set of gang tasks, each of whose multiple threads should be concurrently executed on different processors as shown in Fig. 1, and it has been known to yield efficient and predictable utilization of modern computing units [10–15]. However, studies for gang scheduling have been biased to preemptive gang scheduling (PG) [16–25]. Although non-preemptive gang scheduling (NPG) is not only practical for tasks that incur large preemption overhead but also essential for inherently non-preemptive tasks, only a few studies have paid attention to NPG [26, 27].

Different from PG, NPG incurs a new type of priority-inversion [26] as follows. Consider that a task τ_3 released at $t = -2$ and tasks τ_1, τ_2 and τ_4 released at $t = 0$ are scheduled by NPG on eight processors, as shown in Fig. 1(a). Due to the non-preemptive execution of τ_3 that occupies three processors in $[-2, 2)$, τ_2 that requires six processors cannot start its execution at $t = 0$ regardless of its priority. On the other hand, τ_4 and τ_1 that respectively require three and two processors can start their executions at $t = 0$, which makes τ_4 block τ_2's execution in $[2, 4)$. This is because, if τ_4 did not start its execution at $t = 0$, τ_2 could start its execution at $t = 2$, as shown in Fig. 1(b). Therefore, if the priority of τ_2 is higher than that of τ_4, NPG incurs a new type of priority-inversion, which comes from an integration of different numbers of required processors for each task (that makes τ_4 start execution at $t = 0$ instead of τ_2) and non-preemptive execution (that disallows τ_2 to preempt τ_4 at $t = 2$).

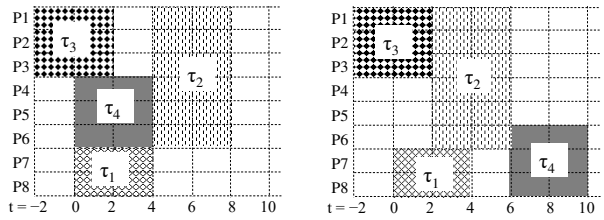


Fig. 1. Four non-preemptive gang tasks τ_1–τ_4 scheduled by the traditional NPG framework and the NPG* framework on eight processors P1–P8

In this paper, we focus on the new type of priority-inversion specialized for NPG, and design a generalized NPG framework called NPG* that controls whether the new priority-inversion is allowed or not (i.e., addressing the scheduling framework *design* issue for NPG). To this end, we add a binary task option that determines whether a higher-priority job allows or disallows a lower-priority job to start its execution when the number of available processors is sufficient for the lower-priority job but not for the higher-priority job. To demonstrate the effectiveness of NPG* in guaranteeing the schedulability of a set of non-preemptive gang tasks, we target NPG*-FP by applying fixed-priority scheduling (FP) as a prioritization policy to NPG*, and address the following questions (i.e., addressing the *timing guarantee* issue for NPG).

Q1. How to develop a schedulability test for NPG*-FP, when an assignment of the allowance/disallowance option for each task is given? How to improve the test by utilizing properties of NPG*-FP?

‡Jinkyu Lee is the corresponding author.

2576-3172/22/$31.00 ©2022 IEEE
DOI 10.1109/RTSS55097.2022.00021

Q2. How to assign the allowance/disallowance option that makes a target task set schedulable (if exists) by the schedulability tests to be developed by addressing Q1?

To answer Q1, we adapt an existing interference-based schedulability analysis framework for preemptive sequential (i.e., single-thread) scheduling in [28, 29] and that for non-preemptive sequential scheduling in [30, 31], for a base structure of the schedulability test to be developed. We then derive a sufficient condition for a job of interest to start its execution no later than the last instant at which the job will miss its deadline unless starting its execution. This is challenging in that the job of interest of a lower-priority task under NPG*-FP cannot start its execution due to not only the lack of available processors, but also the inability to execute a ready (i.e., has been released) job of a higher-priority task with the disallowance option. Considering the challenge, we develop new notions and methods to upper-bound contribution of other jobs to the duration in which the job of interest cannot be executed under NPG*-FP, and establish the first schedulability test for NPG*-FP under a given assignment of the allowance/disallowance option to each task (in Section IV-B). Note that the proposed schedulability test can be also used for traditional NPG-FP, which is equivalent to NPG*-FP by assigning the allowance option to every task. We then derive properties of NPG*-FP that help to calculate tighter upper-bounds, and develop a systematic method to utilize the properties, yielding an improved schedulability test for NPG*-FP (in Section IV-C).

For Q2, we investigate how the allowance/disallowance option affects the schedulability of the proposed schedulability tests. Assuming the allowance/disallowance option is already assigned to every task, we analyze whether the change of the allowance/disallowance option for a single task of interest is beneficial to the schedulability of its higher-priority tasks, the task of interest itself, and its lower-priority tasks. We then develop the optimal allowance/disallowance assignment algorithm based on the analysis, which finds an assignment that makes a target task set schedulable by the proposed schedulability tests as long as such an assignment exists.

The simulation results show that the proposed NPG* framework significantly improves the schedulability performance of the traditional NPG framework. The proposed schedulability test for NPG*-FP with the optimal assignment algorithm finds 161.2%–346.0% and 11.2%–15.6% additional schedulable task sets, respectively compared to the only existing schedulability test for the traditional NPG framework in [26] and the proposed schedulability test for traditional NPG-FP.

This paper makes the following contributions:

- Design of a novel, generalized NPG framework NPG*, which employs a task-level option to allow or disallow the priority-inversion specialized for NPG (Section III),
- Development of the first schedulability tests not only for traditional NPG-FP but also for NPG*-FP (Section IV),
- Development of an assignment algorithm that optimally assigns the allowance/disallowance option to each task (Section V), and
- Demonstration of the effectiveness of the proposed framework in improving schedulability performance (Section VI).

II. System Model

We consider a task set τ consisting of n sporadic gang tasks $\tau_i \in \tau$, each of which is specified by T_i (the minimum separation or the period), C_i (the worst-case execution time), D_i (the relative deadline), and $m_i \geq 1$ (the number of threads that should be executed on different processors in parallel, called task parallelism) [17]. We assume that every task $\tau_i \in \tau$ satisfies $D_i \leq T_i$. Each task τ_i invokes a series of jobs as follows. The release times of two consecutive jobs of τ_i are separated by at least T_i time units; once a job of τ_i is released at t, it should finish its execution until $t+D_i$. For each job of τ_i, m_i threads should be concurrently executed on m_i processors. We consider that every job is non-preemptive; once m_i threads of τ_i's job start to execute at t, their executions last until no later than $t+C_i$ without any preemption. We consider a computing unit that consists of $m \geq 2$ identical processors (cores).

In this paper, we consider fixed-priority scheduling (FP) as a prioritization policy. Let $\tau^{\mathrm{HP}}(\tau_k)$ and $\tau^{\mathrm{LP}}(\tau_k)$ denote a set of tasks in τ whose priority is higher and lower than τ_k, respectively. A sporadic non-preemptive gang task set τ is said to be *schedulable* by a scheduling framework on an m-processor platform, if the following statement is true: every legitimate job sequence generated by τ does not yield any single job deadline miss when the job sequence is scheduled by the framework on the platform. Presenting an equation or inequality, we let LHS and RHS denote the left-hand side and right-hand side, respectively.

A time interval L is not necessarily consecutive, unless we explicitly specify it as a consecutive time interval. We apply the same semantics of the following set operations to time intervals. Let $L' \cup L''$, $L' \cap L''$, and $L' \setminus L''$ respectively imply the union of L' and L'', the intersection of L' and L'', and the time interval that belongs to L' but does not belong to L''; let $L' \subseteq L''$ implies that L' belongs to L''. Also, let $|L|$ denote the length of a time interval L. For example, time interval L' (which is non-consecutive) can be defined as $L' = [-2, 2) \cup [6, 10)$ in which processor 5 is idle in Fig. 1(b), and time interval L'' can be defined as $L'' = [-2, 2)$ in which processor 6 is idle. Then, $L' \cap L'' = [-2, 2)$ holds in which processors 5 and 6 idle, and $L' \setminus L'' = [6, 10)$ holds in which processor 5 idle but processor 6 is not idle. Also, $|L'| = 8$, $|L''| = 4$, $|L' \cap L''| = 4$ and $|L' \setminus L''| = 4$ hold; and $L'' \subseteq L'$ holds.

III. Scheduling Framework Design for Non-Preemptive Gang Scheduling

In this section, we first investigate a property of NPG, which yields a new type of priority-inversion specialized for NPG. We then design a generalized NPG framework by incorporating a task-level option to allow or disallow the priority-inversion.

Since we target non-preemptive gang scheduling (NPG), we need to pay attention to the keywords "non-preemptive" and "gang" in NPG. We first present a property of NPG due to non-preemptiveness as follows.

PR1. A lower-priority job J_i can block the execution of a higher-priority job J_k, if it starts execution before the release time of J_k.

While PR1 is known to hold under non-preemptive sequential (i.e., single-thread) scheduling [32], the property also holds under NPG. For example, consider the job of τ_3 and that of τ_2 on eight processors in Fig. 1(b). Due to the execution of the job of τ_3 that occupies three processors started at $t = -2$, the job of τ_2 that requires six processors cannot start its execution at $t = 0$, at which it is released, even if the job of τ_2 has a higher priority than the job of τ_3.

Second, the following property holds for NPG due to each job's different parallelism (m_i) under gang scheduling.

PR2. Suppose that a lower-priority job (J_i) invoked by τ_i and a higher-priority job (J_k) invoked by τ_k are released before t but do not start their executions before t. At t, if there are m' available processors such that $m_i \leq m' < m_k$ holds, J_i can start its execution while J_k cannot.

Under PG (preemptive gang scheduling), PR2 not only (i) makes the completion time of J_i earlier due to utilizing the unused processors, but also (ii) does not delay the completion time of J_k assuming no preemption/migration cost (because J_k can preempt J_i when necessary). On the other hand, under NPG, (i) is true while (ii) is not. This is because, once J_i starts its execution, it cannot be preempted by any job. As explained in Section I with Fig. 1(a), due to the execution of the job of τ_4 in $[0, 4)$, the job of τ_2 cannot start its execution before $t = 4$, even if the priority of the job of τ_2 is higher than that of the job of τ_4 and the job of τ_2 is ready to be executed at $t = 0$. However, if we disallow the job of τ_4 to start its execution at $t = 0$ despite the number of available processors sufficient, the job of τ_2 can start its execution at $t = 2$, as shown in Fig. 1(b). This means, different from PR2 under PG, PR2 under NPG causes a new type of priority-inversion due to the execution of a lower-priority job instead of a higher-priority job that is ready to be executed; this is called 2-dimension blocking in [26].

Since PR2 yields a new type of priority-inversion specialized for NPG, we may have the following question regarding PR2: what if we disallow a lower-priority job to start its execution when a higher-priority job is ready but cannot start its execution? The following example shows that such disallowance may and may not improve the schedulability of a task set.

Example 1: Consider the following four tasks executed on eight processors: $\tau_1(T_1{=}25, C_1{=}4, D_1{=}25, m_1{=}2)$, $\tau_2(25, 4, D_2, 6)$, $\tau_3(25, 4, 25, 3)$ and $\tau_4(25, 4, D_4, 3)$. A job of τ_3 is released at $t = -2$, and jobs of other tasks are released at $t = 0$; the job of τ_1 and that of τ_4 have the highest and lowest priority, respectively. If we allow the situation of PR2 for the job of τ_2 (as a higher-priority job), the job of τ_4 (as a lower-priority job) starts its execution at $t = 0$, yielding the schedule in Fig. 1(a). Otherwise, the job of τ_4 is not allowed to start its execution until the job of τ_2 starts its execution, yielding the schedule in Fig. 1(b). In this example, if $D_2 = 6$ and $D_4 = 10$ hold, the disallowance option for τ_2 enables every job deadline to be met while the allowance option yields the job deadline miss of τ_2. On the other hand, if $D_2 = 8$ and $D_4 = 8$ hold, the allowance option for τ_2 enables every job deadline to be met while the disallowance option yields the job deadline miss of τ_4. □

Algorithm 1 NPG* Framework

Selection of jobs to be executed upon job release or completion:

1: **for** every job (denoted by J_k that is invoked by τ_k) in the ready queue from the highest-priority job to the lowest-priority job (sorted by the prioritization policy) **do**
2: **if** the number of available processors is zero **then** return
3: **else if** the number of available processors is no smaller than m_k **then** start to execute J_k
4: **else if** $\phi_k = \texttt{F}$ **then** return
5: **end if**
6: **end for**

According to Example 1, we need to employ the following task option for τ_k: whether a higher-priority, ready job of τ_k allows a lower-priority job of τ_i to start its execution when the number of available processors is smaller than m_k but no smaller than m_i. To this end, we employ an additional binary task option ϕ_k for each $\tau_k \in \tau$ as follows. Suppose that a job of τ_k is ready to be executed because the number of available processors is not sufficient (i.e., less than m_k).

- If $\phi_k = \texttt{T}$, it is *allowed* to start the execution of jobs whose priority is lower than the job of τ_k (which is a mechanism for traditional NPG).
- If $\phi_k = \texttt{F}$, it is *not allowed* to start the execution of jobs whose priority is lower than the job of τ_k.

Using the binary task option ϕ_k, we can formally state the NPG* framework in Algorithm 1. The difference between the traditional NPG framework and the NPG* framework lies in Line 4 in Algorithm 1. That is, while the former finds the next job to be executed until there is no more available processor or there is no more job to be checked in the ready queue, the latter stops finding the next job to be executed if a higher-priority job of τ_k with $\phi_k = \texttt{F}$ cannot start its execution. Therefore, the NPG* framework is a generalization of the traditional NPG framework; NPG is equivalent to NPG* by assigning $\phi_i = \texttt{T}$ for every $\tau_i \in \tau$. While the NPG* framework can be applied to most (if not all) prioritization policies, the rest of this paper focuses on NPG*-FP by applying fixed priority scheduling (FP) to NPG*.

From now on, a job is said to be *pending* at t under NPG*-FP, if the job is ready (i.e., has been released) at t, but cannot execute for whatever reasons (i.e., due to either the lack of available processors, or the existence of a ready job of a higher-priority task τ_h with $\phi_h = \texttt{F}$ that cannot be executed).

IV. Schedulability Analysis for NPG*-FP

In this section, we develop a schedulability test for NPG*-FP for a set of tasks, each of whose ϕ_k is given. To this end, we develop a condition for each job's schedulability, which relies on some values only revealed at run-time (Theorem 1 in Section IV-A). We then derive offline upper-bounds of each of those unknown values individually, developing the first schedulability test for NPG*-FP (Theorem 2 in Section IV-B). Finally, we develop a way to tightly upper-bound a group of individual values, yielding an improved version of the schedulability test for NPG*-FP (Theorem 3 in Section IV-C).

A. Development of Each Job's Schedulability Condition

As a base structure of the schedulability test, we adapt an existing interference-based framework for preemptive sequential (i.e., single-thread) scheduling in [28, 29] and that for non-preemptive sequential scheduling in [30, 31]. It derives a sufficient condition for a job of interest to start its execution no later than the last instant at which the job will miss its deadline unless starting its execution. We focus on a job of τ_k of interest (denoted by J_k) whose release time is r_k, and target the interval $\Delta_k = [r_k, r_k{+}D_k{-}C_k)$ of length $(D_k{-}C_k)$. If the length of the time interval in Δ_k during which J_k is pending is strictly less than $(D_k - C_k)$, the job can start its execution within Δ_k, which implies that the job can finish its execution until its deadline at $r_k + D_k$ due to the job's non-preemptive execution. To calculate the length of the time interval, we should consider an important property of NPG*-FP: a job of a lower-priority task τ_k cannot start its execution, when there exists at least one pending job of a higher-priority task τ_h with $\phi_h = \mathtt{F}$. To express the time interval itself and that associated with/without the important property of NPG*-FP, we define the following notations, where Δ is a consecutive time interval.

- $\tau^{\mathtt{HPF}}(\tau_k)$ is a set of tasks $\{\tau_h \in \tau\}$ whose priority is higher than τ_k and whose ϕ_h is $\mathtt{F}$, i.e., $\tau^{\mathtt{HPF}}(\tau_k) = \{\tau_h \in \tau^{\mathtt{HP}}(\tau_k) | \phi_h = \mathtt{F}\}$.
- $L_k^{\mathtt{P}}(\Delta)$ is the time interval in Δ, where a job of τ_k is pending.
- $L_k^{\mathtt{PH}}(\Delta)$ is the time interval in Δ, where a job of τ_k is pending *and* there exists *at least a task* $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$ such that a job of τ_h is pending, i.e., $L_k^{\mathtt{PH}}(\Delta) = L_k^{\mathtt{P}}(\Delta) \cap \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h^{\mathtt{P}}(\Delta)$.
- $L_k^{\mathtt{PN}}(\Delta)$ is the time interval in Δ, where a job of τ_k is pending *and* there exists *no task* $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$ such that a job of τ_h is pending, i.e., $L_k^{\mathtt{PN}}(\Delta) = L_k^{\mathtt{P}}(\Delta) \setminus L_k^{\mathtt{PH}}(\Delta)$. For the sake of simplicity, we represent $L_k^{\mathtt{PN}}(\Delta)$ as $L_k(\Delta)$ in the rest of the paper.

In short, a job of τ_k is pending at any time instant in $L_k^{\mathtt{PH}}(\Delta)$ due to at least one pending job of a task $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$, while a job of τ_k is pending at any time instant in $L_k(\Delta)$ due to the lack of available processors (i.e., less than m_k) when there is no pending job of a task $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$.

We investigate the following example that shows the situation where a job is pending under NPG*-FP.

Example 2: Consider the four tasks (similar to Example 1) are scheduled by NPG*-FP on eight processors: $\tau_1(T_1{=}25, C_1{=}4, D_1{=}25, m_1{=}2)$, $\tau_2(25, 4, 25, 6)$, $\tau_3(25, 4, 25, 3)$ and $\tau_4(25, 4, 25, 3)$. A job of τ_3 is released at $t = -2$, and jobs of other tasks are released at $t = 0$; τ_1 and τ_4 have the highest and lowest task priority, respectively. If we set $\phi_2 = \mathtt{F}$, the schedule under NPG*-FP is shown in Fig. 1(b). In the figure, the job of τ_4 (with $\Delta_4 = [0, 0{+}25{-}4{=}21)$) is pending in $L_4^{\mathtt{P}}(\Delta_4) = [0, 6)$. In particular, the job is pending in $L_4^{\mathtt{PH}}(\Delta_4) = [0, 2)$, although the number of available processors (i.e., 3) is the same as $m_4 = 3$. This happens due to the salient feature of NPG*; the job of τ_2 with $\phi_2 = \mathtt{F}$ does not allow any lower-priority job to start its execution when the job of τ_2 is pending. On the other hand, the job is pending in $L_4(\Delta_4) = [2, 6)$, because the number of available processors (i.e., 0 in $[2, 4)$ and 2 in $[4, 6)$) is less than the number of its threads to be executed concurrently (i.e., $m_4 = 3$). This type of inability to execute a ready job also happens under the traditional NPG framework.

In the existing interference-based schedulability test framework in [28–31], the schedulability of a job of τ_k is judged by calculating an upper-bound of the duration in which the number of processors unoccupied by other jobs is not sufficient for the execution of the job of interest. However, in $L_k^{\mathtt{PH}}(\Delta_k)$ under NPG*-FP, it is possible for the job of interest to be pending, even though the number of unoccupied processors is sufficient for the job's execution; this happens when there exists a pending job of a higher-priority task $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$. Therefore, in order to utilize the existing interference-based schedulability test framework for NPG*-FP, we need to express the length of $L_k^{\mathtt{PH}}(\Delta_k)$ as the length of $L_h(\Delta_k)$ for $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$, which is addressed by the following lemma.

Lemma 1: The following inequality holds for a job of interest of $\tau_k \in \tau$ released at r_k (i.e., J_k), where $\Delta_k = [r_k, r_k + D_k - C_k)$:

$$|L_k^{\mathtt{PH}}(\Delta_k)| \leq \sum_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} |L_h(\Delta_k)|. \tag{1}$$

Proof: We now prove $L_k^{\mathtt{PH}}(\Delta_k) \subseteq \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h(\Delta_k)$, which is a sufficient condition for the lemma.

By definition, $L_k^{\mathtt{PH}}(\Delta_k) = L_k^{\mathtt{P}}(\Delta_k) \cap \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h^{\mathtt{P}}(\Delta_k)$ holds; therefore, $L_k^{\mathtt{PH}}(\Delta_k) \subseteq \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h^{\mathtt{P}}(\Delta_k)$ holds. Applying this relation, it suffices to prove $\bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h^{\mathtt{P}}(\Delta_k) = \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h(\Delta_k)$. Let τ_{h_x} denote the x^{th} highest-priority task in $\tau^{\mathtt{HPF}}(\tau_k)$. We prove it by mathematical induction.

(Base case) Since τ_{h_1} is the highest-priority task in $\tau^{\mathtt{HPF}}(\tau_k)$, $L_{h_1}^{\mathtt{PH}}(\Delta_k) = \emptyset$ holds, which implies $L_{h_1}^{\mathtt{P}}(\Delta_k) = L_{h_1}(\Delta_k)$.

(Inductive case) Suppose that $\bigcup_{1 \leq x \leq n-1} L_{h_x}^{\mathtt{P}}(\Delta_k) = \bigcup_{1 \leq x \leq n-1} L_{h_x}(\Delta_k)$ holds for any $n \geq 2$. The following holds.

$$\begin{aligned}
&\bigcup_{1 \leq x \leq n} L_{h_x}^{\mathtt{P}}(\Delta_k) \\
&= \bigcup_{1 \leq x \leq n-1} L_{h_x}^{\mathtt{P}}(\Delta_k) \cup L_{h_n}^{\mathtt{P}}(\Delta_k) \\
&= \bigcup_{1 \leq x \leq n-1} L_{h_x}^{\mathtt{P}}(\Delta_k) \cup L_{h_n}^{\mathtt{PH}}(\Delta_k) \cup L_{h_n}(\Delta_k) \\
&\qquad\qquad \text{(by the definition of } L_{h_n}^{\mathtt{P}}(\Delta_k))
\end{aligned}$$

$= \bigcup_{1 \le x \le n-1} L^{\mathtt{P}}_{h_x}(\Delta_k) \cup \{L^{\mathtt{P}}_{h_n}(\Delta_k) \cap \bigcup_{1 \le x \le n-1} L^{\mathtt{P}}_{h_x}(\Delta_k)\} \cup L_{h_n}(\Delta_k)$ (by the definition of $L^{\mathtt{PH}}_{h_n}(\Delta_k)$)
$= \bigcup_{1 \le x \le n-1} L^{\mathtt{P}}_{h_x}(\Delta_k) \cup L_{h_n}(\Delta_k)$ (by $L \cup \{L' \cap L\} = L$)
$= \bigcup_{1 \le x \le n-1} L_{h_x}(\Delta_k) \cup L_{h_n}(\Delta_k)$ (by the supposition)
$= \bigcup_{1 \le x \le n} L_{h_x}(\Delta_k)$

By the base and inductive case, $\bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L^{\mathtt{P}}_h(\Delta_k) = \bigcup_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} L_h(\Delta_k)$ holds, which proves the lemma. ■

Using the lemma, we can test J_k's schedulability by using terms for $L_x(\Delta_k)$ only (without using terms for $L^{\mathtt{PH}}_x(\Delta_k)$ and $L^{\mathtt{P}}_x(\Delta_k)$) as follows.

Lemma 2: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. If Eq. (2) holds, a job of interest of τ_k released at r_k (i.e., J_k) cannot miss its deadline, where $\Delta_k = [r_k, r_k + D_k - C_k)$.

$$\big|L_k(\Delta_k)\big| + \sum_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} \big|L_h(\Delta_k)\big| < D_k - C_k \tag{2}$$

Proof: By the definition of $L^{\mathtt{P}}_k(\Delta_k)$, $L_k(\Delta_k)$ and $L^{\mathtt{PH}}_k(\Delta_k)$, the following conditions hold: $L_k(\Delta_k) \cup L^{\mathtt{PH}}_k(\Delta_k) = L^{\mathtt{P}}_k(\Delta_k)$ and $L_k(\Delta_k) \cap L^{\mathtt{PH}}_k(\Delta_k) = \emptyset$, which yields the relation of $|L^{\mathtt{P}}_k(\Delta_k)| = |L_k(\Delta_k)| + |L^{\mathtt{PH}}_k(\Delta_k)|$. Once we apply Eq. (1) to the relation, $|L^{\mathtt{P}}_k(\Delta_k)| \le$ the LHS of Eq. (2) holds. Therefore, if Eq. (2) holds, $|L^{\mathtt{P}}_k(\Delta_k)| < D_k - C_k$ holds.

Considering the definition of $L^{\mathtt{P}}_k(\Delta_k)$ and the length of Δ_k (that is $(D_k - C_k)$), $|L^{\mathtt{P}}_k(\Delta_k)| < D_k - C_k$ is a sufficient condition for J_k to start its execution no later than the end of Δ_k (i.e., $r_k + D_k - C_k$), meaning that J_k finishes its execution no later than its deadline $r_k + D_k$ by non-preemptive execution. ■

To express the contribution of jobs of a single task τ_i ($\neq \tau_k$) to $|L_k(\Delta_k)|$ and $|L_h(\Delta_k)|$ for $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$ in Eq. (2), we define the following notation, where Δ is a consecutive time interval.

- $L_{k \leftarrow i}(\Delta)$ is the time interval in $L_k(\Delta)$, where a job of τ_i is executed. By definition, $L_{k \leftarrow i}(\Delta) \subseteq L_k(\Delta)$ holds.

Then, we can upper-bound $|L_k(\Delta_k)|$ and $|L_h(\Delta_k)|$ for $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$ in Eq. (2), respectively using the terms $|L_{k \leftarrow i}(\Delta_k)|$ and $|L_{h \leftarrow i}(\Delta_k)|$ for $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$, as follows.

Lemma 3: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. Then, Eq. (3) holds for a job of interest of $\tau_k \in \tau$ released at r_k (i.e., J_k), where $\Delta_k = [r_k, r_k + D_k - C_k)$.

$$\big|L_k(\Delta_k)\big| \le \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \frac{\big|L_{k \leftarrow i}(\Delta_k)\big| \cdot \min(m_i, m - m_k + 1)}{m - m_k + 1} \tag{3}$$

Also, if there is no execution of J_k in Δ_k, Eq. (4) holds for every τ_h ($\neq \tau_k$) $\in \tau$.

$$\big|L_h(\Delta_k)\big| \le \sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{\big|L_{h \leftarrow i}(\Delta_k)\big| \cdot \min(m_i, m - m_h + 1)}{m - m_h + 1} \tag{4}$$

Proof: Suppose that Eq. (3) does not hold; we show the contradiction of the supposition.

We define the amount of execution as follows: if a job J_k of τ_k is executed in an interval L on m_k processors, its amount of execution in L is $|L| \cdot m_k$. By the definition of $L_k(\Delta_k)$, there should be at least $(m - m_k + 1)$ processors that perform other jobs' execution at any time instant in $L_k(\Delta_k)$; otherwise, there should be at least m_k available processors, which contradicts that J_k is pending without any task $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$ such that a job of τ_h is pending. Therefore, at each time instant in $L_k(\Delta_k)$, we can select arbitrary $(m - m_k + 1)$ busy processors and call them "counted processors"; then, the amount of execution of all jobs in $L_k(\Delta_k)$ on the counted processors is exactly $|L_k(\Delta_k)| \cdot (m - m_k + 1)$.

On the other hand, the amount of execution of jobs of a task $\tau_i \in \tau \setminus \{\tau_k\}$ in $L_k(\Delta_k)$ on the counted processors is upper-bounded by $|L_{k \leftarrow i}(\Delta_k)| \cdot \min(m_i, m - m_k + 1)$. Hence, the amount of execution of jobs of all tasks $\tau_i \in \tau \setminus \{\tau_k\}$ in $L_k(\Delta_k)$ on the counted processors is upper-bounded by $\sum_{\tau_i \in \tau \setminus \{\tau_k\}} |L_{k \leftarrow i}(\Delta_k)| \cdot \min(m_i, m - m_k + 1)$. The supposition implies that the upper-bound is strictly smaller than the exact value, which contradicts.

Once we replace τ_k with τ_h in Eq. (3), we can prove Eq. (4) under the condition that there is no execution of jobs of τ_k in Δ_k, by removing the term regarding τ_k in the summation of the RHS of Eq. (4). ■

By applying Lemma 3 to Lemma 2, we have the following condition for each job's schedulability condition.

Theorem 1: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. If Eq. (5) holds, a job of interest of τ_k released at r_k (i.e., J_k) can start its execution no later than $r_k + D_k - C_k$ (and therefore cannot miss its deadline), where $\Delta_k = [r_k, r_k + D_k - C_k)$.

$$\sum_{\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)} \left(\sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{\big|L_{h \leftarrow i}(\Delta_k)\big| \cdot \min(m_i, m - m_h + 1)}{m - m_h + 1} \right) + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \frac{\big|L_{k \leftarrow i}(\Delta_k)\big| \cdot \min(m_i, m - m_k + 1)}{m - m_k + 1} < D_k - C_k \tag{5}$$

Proof: By applying Eqs. (3) and (4) to Eq. (2), we have Eq. (5). ■

We would like to summarize the main reason why we succeed to derive Theorem 1. First, we separate the time interval where J_k is pending (i.e., $L^{\mathtt{P}}_k(\Delta_k)$), into the interval where there is a pending job of a higher-priority task with $\phi_h = \mathtt{F}$ (i.e., $L^{\mathtt{PH}}_k(\Delta_k)$) and the interval where there is no pending job of a higher-priority task with $\phi_h = \mathtt{F}$ (i.e., $L_k(\Delta_k)$). Second, considering the latter (i.e., $L_k(\Delta_k)$) can be utilized by the existing interference-based schedulability test framework while the former (i.e., $L^{\mathtt{PH}}_k(\Delta_k)$) cannot, we develop a way to express the term of $L^{\mathtt{PH}}_k(\Delta_k)$ as the terms of $L_h(\Delta_k)$ for $\tau_h \in \tau^{\mathtt{HPF}}(\tau_k)$.

Since Eq. (5) requires the values of $|L_{h \leftarrow i}(\Delta_k)|$ and $|L_{k \leftarrow i}(\Delta_k)|$ that are only available at run-time, we need to

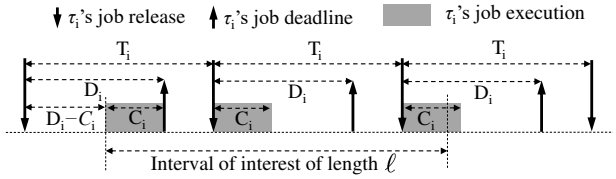


Fig. 2. The job release and execution scenario for $W_i(\ell)$ [29]

derive an upper-bound of the values in order to develop an offline schedulability test that requires the task parameters only. To this end, we utilize two conditions for $|L_{x\leftarrow i}(\Delta_k)|$ where τ_x is τ_k or $\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)$: a job of τ_x is pending in Δ_k, but a job of τ_i executes in Δ_k. While Section IV-B upper-bounds individual terms (i.e., $|L_{x\leftarrow i}(\Delta_k)|$), Section IV-C will derive different types of upper-bounds for a group of terms.

B. Developement of Schedulability Analysis for NPG-FP*

We now explain how to upper-bound $|L_{k\leftarrow i}(\Delta_k)|$ with two cases $\phi_k = \mathtt{T}$ and $\phi_k = \mathtt{F}$.

For τ_k that satisfies $\phi_k = \mathtt{T}$, we consider three cases for the relationship between τ_i ($\neq \tau_k$) and τ_k; (i) $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$, (ii) $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i \geq m_k$, and (iii) $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i < m_k$, by adapting the techniques in [28–31].

First, if $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$ holds, we can upper-bound $|L_{k\leftarrow i}(\Delta_k)|$ by $W_i(D_k - C_k)$, where $W_i(\ell)$ is the maximum duration of execution of jobs of τ_i in a consecutive interval of length ℓ [29]. As shown in Fig. 2, the maximum duration occurs when the first job of τ_i is executed at the latest and the following jobs of τ_i (that are periodically released) are executed as early as possible; also, the interval of interest begins at which the first job's execution starts. Then, the number of jobs of τ_i whose next job's release time is within the interval of interest (e.g., the first and second jobs in Fig. 2) is calculated by $N_i(\ell) = \left\lfloor \frac{\ell + D_i - C_i}{T_i} \right\rfloor$, and the jobs are executed for C_i each. Also, if there exists a job in the interval of interest, which does not belong to the $N_i(\ell)$ jobs (e.g., the third job in Fig. 2), the job is executed for $\min\left(C_i, \ell + D_i - C_i - N_i(\ell)\cdot T_i\right)$ in the interval. Considering the interval length ℓ, $W_i(\ell)$ can be calculated as follows [29].

$$W_i(\ell) = \min\left(\ell, N_i(\ell)\cdot C_i + \min\left(C_i, \ell + D_i - C_i - N_i(\ell)\cdot T_i\right)\right) \tag{6}$$

Second, if $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i \geq m_k$ holds, we can upper-bound $|L_{k\leftarrow i}(\Delta_k)|$ by $\min(D_k - C_k, C_i)$, which is no larger than $W_i(D_k - C_k)$. Since m_i is no smaller than m_k, the situation of PR2 in Section III cannot occur for a pair of a lower-priority job of τ_i and a higher-priority job of τ_k. Therefore, the only situation where any job of τ_i can prevent J_k from executing is PR1 in Section III, which is, the job of τ_i starts its execution before r_k. In this case, the duration of the execution of jobs of τ_i in Δ_k is upper-bounded by a single execution of a job of τ_i and the interval length, yielding $\min(D_k - C_k, C_i)$ [30, 31].

Third, if $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i < m_k$ holds, $\min(D_k - C_k, C_i)$ is not a safe upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$. This is because, if the number of available processors at t is smaller than m_k but no smaller than m_i, a ready job of τ_i can start its execution while the ready job of τ_k cannot; this results in a situation where multiple jobs of τ_i start their executions while the ready job of τ_k cannot. PR2 in Section III explains this priority-inversion, which is also shown in Example 1 associated with Fig. 1(a). Therefore, we upper-bound $|L_{k\leftarrow i}(\Delta_k)|$ by $W_i(D_k - C_k)$, which is an upper-bound of the duration of the execution of jobs of τ_i in an interval of length $(D_k - C_k)$.

Let $E_{k\leftarrow i}$ denote an upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$. Then, we summarize the case for $\phi_k = \mathtt{T}$ as follows.

For $\phi_k = \mathtt{T}$,

$$E_{k\leftarrow i} = \begin{cases} W_i(D_k - C_k), & \text{if } \tau_i \in \tau^{\mathrm{HP}}(\tau_k), \\ \min(D_k - C_k, C_i), & \text{if } \tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i \geq m_k, \\ W_i(D_k - C_k), & \text{if } \tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i < m_k. \end{cases} \tag{7}$$

On the other hand, for τ_k that satisfies $\phi_k = \mathtt{F}$, we consider two cases for the relationship between τ_i ($\neq \tau_k$) and τ_k: (i) $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$, and (ii) $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)$.

First, if $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$ holds, we can upper-bound $|L_{k\leftarrow i}(\Delta_k)|$ by $W_i(D_k - C_k)$, which is the same as the case for $\phi_k = \mathtt{T}$.

Second, if $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)$ holds, any job of τ_i cannot start its execution when a job of τ_k with $\phi_k = \mathtt{F}$ is pending due to the mechanism of the NPG* framework. Therefore, we need to consider PR1 only, but not PR2; this is the main feature of NPG*. This implies that we can use $\min(D_k - C_k, C_i)$ as an upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$ not only for the case for $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i \geq m_k$ but also for the case for $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i < m_k$, which is different from the cases for $\phi_k = \mathtt{T}$.

Then, we summarize the case for $\phi_k = \mathtt{F}$ as follows.

For $\phi_k = \mathtt{F}$,

$$E_{k\leftarrow i} = \begin{cases} W_i(D_k - C_k), & \text{if } \tau_i \in \tau^{\mathrm{HP}}(\tau_k), \\ \min(D_k - C_k, C_i), & \text{if } \tau_i \in \tau^{\mathrm{LP}}(\tau_k). \end{cases} \tag{8}$$

Finally, the following lemma records the upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$.

Lemma 4: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. Then, Eq. (9) holds for a job of interest of $\tau_k \in \tau$ released at r_k (i.e., J_k), and $\tau_i \in \tau \setminus \{\tau_k\}$.

$$\left|L_{k\leftarrow i}(\Delta_k)\right| \leq E_{k\leftarrow i}, \tag{9}$$

where $\Delta_k = [r_k, r_k + D_k - C_k)$, and $E_{k\leftarrow i}$ for $\phi_k = \mathtt{T}$ and $\phi_k = \mathtt{F}$ is shown in Eqs. (7) and (8), respectively.

Proof: By the definition of $L_{k\leftarrow i}(\Delta_k)$, a job of τ_i is executed at any time instant in $L_{k\leftarrow i}(\Delta_k)$. By the calculation of $W_i(D_k - C_k)$, jobs of τ_i cannot be executed for more than $W_i(D_k - C_k)$ in any consecutive interval of length $(D_k - C_k)$. Therefore, considering $L_{k\leftarrow i}(\Delta_k)$ is a subset of Δ_k, Eq. (9) holds for all the cases where $E_{k\leftarrow i} = W_i(D_k - C_k)$, which are $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$ and $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i < m_k$ for $\phi_k = \mathtt{T}$ in Eq. (7), and $\tau_i \in \tau^{\mathrm{HP}}(\tau_k)$ for $\phi_k = \mathtt{F}$ in Eq. (8).

Then, the remaining cases to be proved are $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)|m_i \geq m_k$ for $\phi_k = \mathtt{T}$ in Eq. (7), and $\tau_i \in \tau^{\mathrm{LP}}(\tau_k)$

for $\phi_k = \texttt{F}$ in Eq. (8). In the former case, $m_i \geq m_k$ makes it impossible for a job of τ_i (J_i) to start its execution at t when J_i and J_k are active at t; this implies, PR2 in Section III does not occur. Therefore, PR1 in Section III (i.e., blocking) is the only situation where a job of τ_i is executing while J_k is not. By considering the interval length of $L_{k\leftarrow i}(\Delta_k)$ is at most $(D_k - C_k)$ (because $L_{k\leftarrow i}(\Delta_k)$ is a subset of Δ_k) and only one job of τ_i can block J_k, $E_{k\leftarrow i} = \min(D_k - C_k, C_i)$ holds.

In the latter case, since J_k disallows a lower-priority job to be executed when it is pending (by $\phi_k = \texttt{F}$) by NPG*, PR1 in Section III (i.e., blocking) is the only situation where a job of τ_i is executing while J_k is not. The remaining proof is the same as the former case. ■

The next step is to upper-bound $|L_{h\leftarrow i}(\Delta_k)|$ for $\tau_h \in \tau^{\texttt{HPF}}(\tau_k)$ in Eq. (5). In any interval that belongs to $L_{h\leftarrow i}(\Delta_k)$, (i) a job of τ_i is executed, (ii) a job of τ_h is pending, and (iii) there is no pending job of $\tau_g \in \tau^{\texttt{HPF}}(\tau_h)$. Since it is complex to upper-bound $|L_{h\leftarrow i}(\Delta_k)|$ by considering conditions (ii) and (iii), we now use condition (i) only; however, conditions (ii) and (iii) will be used in the next subsection. As we explained, $W_i(\ell)$ is the maximum duration of execution of jobs of τ_i in an interval of length ℓ; therefore, $|L_{h\leftarrow i}(\Delta_k)| \leq W_i(D_k - C_k)$ holds regardless of ϕ_k. Let $E_{h\leftarrow i}(\tau_k)$ for $\tau_h \in \tau^{\texttt{HPF}}(\tau_k)$ and $\tau_k \neq \tau_i \neq \tau_h$ denote an upper-bound of $|L_{h\leftarrow i}(\Delta_k)|$, which can be calculated as follows.

$$\text{For both } \phi_k = \texttt{T} \text{ and } \phi_k = \texttt{F},$$
$$E_{h\leftarrow i}(\tau_k) = W_i(D_k - C_k). \tag{10}$$

We record this derivation in the following lemma.

Lemma 5: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. Then, Eq. (11) holds for a job of interest of $\tau_k \in \tau$ released at r_k (i.e., J_k), and every combination of $\tau_h \in \tau^{\texttt{HPF}}(\tau_k)$ and $\tau_i \in \tau \setminus \{\tau_h, \tau_k\}$.

$$\left|L_{h\leftarrow i}(\Delta_k)\right| \leq E_{h\leftarrow i}(\tau_k), \tag{11}$$

where $\Delta_k = [r_k, r_k + D_k - C_k)$, and $E_{h\leftarrow i}(\tau_k)$ is shown in Eq. (10).

Proof: By the definition of $L_{h\leftarrow i}(\Delta_k)$, a job of τ_i is executed in $L_{h\leftarrow i}(\Delta_k)$. By the calculation of $W_i(D_k - C_k)$, jobs of τ_i cannot be executed for more than $W_i(D_k - C_k)$ in any consecutive interval of length $(D_k - C_k)$. Therefore, the lemma holds. ■

Applying Lemmas 4 and 5 to Theorem 1, we develop a schedulability test for NPG*-FP.

Theorem 2: A task set τ is schedulable by NPG*-FP on an m-processor platform, if the following inequality holds for every $\tau_k \in \tau$.

$$\sum_{\tau_h \in \tau^{\texttt{HPF}}(\tau_k)} \left(\sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{E_{h\leftarrow i}(\tau_k) \cdot \min(m_i, m - m_h + 1)}{m - m_h + 1} \right) + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \frac{E_{k\leftarrow i} \cdot \min(m_i, m - m_k + 1)}{m - m_k + 1} < D_k - C_k \tag{12}$$

Proof: Suppose that there exists a job of τ_k released at r_k (denoted by J_k) that misses its deadline, even though Eq. (12) holds. We show the contradiction of the supposition.

By Lemmas 4 and 5, the LHS of Eq. (5) is upper-bounded by the LHS of Eq. (12). This means, the supposition (i.e., Eq. (12) holds) implies that Eq. (5) holds. Therefore, Theorem 1 guarantees there is no deadline miss of J_k, which contradicts the supposition. ■

Considering NPG-FP is the same as NPG*-FP by assigning $\phi_i = \texttt{T}$ for every $\tau_i \in \tau$, Theorem 2 is not only the first schedulability test for NPG*-FP, but also that for NPG-FP under our system model. Note that there exists a schedulability test for NPG-FP, but it works only when there are a finite number of jobs whose release patterns are known a priori [27].

Time-complexity of Theorem 2. Let n' and n denote the number of tasks $\tau_i \in \tau$ that satisfy $\phi_i = \texttt{F}$ and the number of tasks in τ, respectively. For given $\tau_k \in \tau$, we need $O(n)$ computations to calculate the summation of the fraction that includes $E_{k\leftarrow i}$. For given $\tau_k \in \tau$ and $\tau_h \in \tau^{\texttt{HPF}}(\tau_k)$, we also need $O(n)$ computations to calculate the inner summation of the fraction that includes $E_{h\leftarrow i}(\tau_k)$. Since we have at most $(1 + n')$ tasks in $\{\tau_k\} \cup \tau^{\texttt{HPF}}(\tau_k)$, the calculation of the LHS of Eq. (12) for given τ_k takes $O(n \cdot n')$ time-complexity. Since Eq. (12) should be checked for every $\tau_k \in \tau$ (i.e., n tasks), the time-complexity of Theorem 2 is $O(n^2 \cdot n')$.

C. Improvement of Schedulability Analysis for NPG*-FP

Although we successfully developed a schedulability test for NPG*-FP in Theorem 2, we may have room for further improvement of the schedulability test. If we focus on a single task $\tau_i \in \tau \setminus \{\tau_k\}$ when we test the schedulability of τ_k in the theorem, τ_i can contribute to the LHS of Eq. (12) not only by $E_{h\leftarrow i}(\tau_k)$ for different tasks $\tau_h \in \tau^{\texttt{HPF}}(\tau_k)$ but also by $E_{k\leftarrow i}$. This means, it may be possible for jobs of τ_i to make excessive contributions to those upper-bounds beyond the jobs' execution capability, to be explained in the following example.

Example 3: Consider the four tasks (similar to Example 2) are scheduled by NPG*-FP on eight processors: $\tau_1(T_1{=}25, C_1{=}4, D_1{=}25, m_1{=}2)$, $\tau_2(25, 4, 25, 6)$, $\tau_3(25, 4, 25, 3)$ and $\tau_4(25, 4, 25, 3)$, with $\phi_2 = \phi_3 = \texttt{F}$. τ_1 ($= \tau_i$) can contribute to the LHS of Eq. (12) for τ_4 ($= \tau_k$) by $E_{4\leftarrow 1}$, $E_{3\leftarrow 1}(\tau_4)$ and $E_{2\leftarrow 1}(\tau_4)$. Since τ_1 cannot be executed for more than $W_1(D_4 - C_4) = 8$ in the interval of length $D_4 - C_4$, the sum of their contributions should not exceed $W_1(D_4 - C_4)$, which cannot be captured by the LHS of Eq. (12).

If we carefully deduct those excessive contributions, we can improve the schedulability performance, which is a matter of this subsection. To this end, we derive properties of NPG*-FP that can address such pessimism in calculating the upper-bounds in the theorem. As a first step, we derive the disjoint property of the intervals shown in Lemma 2 as follows.

Lemma 6: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. For every $\tau_h, \tau_{h2} \in \tau^{\texttt{HPF}}(\tau_k)$ where $\tau_h \neq \tau_{h2}$, Eqs. (13) and (14) hold for

$\Delta_k = [r_k, r_k + D_k - C_k)$, an interval for the job of interest of τ_k released at r_k.

$$L_k(\Delta_k) \cap L_h(\Delta_k) = \emptyset \tag{13}$$

$$L_{h_2}(\Delta_k) \cap L_h(\Delta_k) = \emptyset \tag{14}$$

Proof: Suppose that Eq. (13) does not hold, meaning that there exists a non-empty interval $L = L_k(\Delta_k) \cap L_h(\Delta_k)$. By the definition of $L_h(\Delta_k)$ and the fact that L belongs to $L_h(\Delta_k)$, there exists a job of τ_h that is pending in L. By the definition of $L_k(\Delta_k)$ and the fact that L belongs to $L_k(\Delta_k)$, there exists no job of $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ that is pending in L. This contracts the existence of the interval L, which proves Eq. (13).

Without loss of generality, we assume τ_{h_2} has a lower priority than τ_h. Then, the proof of Eq. (14) is the same as the proof of Eq. (13), if we replace τ_k with τ_{h2}. ■

Lemma 6 indicates that all intervals in Eq. (2) (i.e., $L_k(\Delta_k)$ and $L_h(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$) are disjoint. Since $L_{k\leftarrow i}(\Delta_k) \subseteq L_k(\Delta_k)$ and $L_{h\leftarrow i}(\Delta_k) \subseteq L_h(\Delta_k)$ hold by definition, the lemma also indicates that all intervals that related to given τ_i in Eq. (5) (i.e., $L_{h\leftarrow i}(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and $L_{k\leftarrow i}(\Delta_k)$) are disjoint. Also, by definition, in any interval within $L_{h\leftarrow i}(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and that within $L_{k\leftarrow i}(\Delta_k)$, there exists a job of τ_i that is executing.

We focus on the situation where there is no execution of J_k in Δ_k (whose beginning is the release time of J_k). Then, in any interval that related to given τ_i in Eq. (5), J_k is pending while a job of τ_i is executing. Considering the disjoint property for the intervals that related to given τ_i in Eq. (5), we can upper-bound the length of the intervals, by the maximum duration in which jobs of τ_i are executed while J_k is not in Δ_k, recoreded as follows.

Lemma 7: Suppose that a task set τ is scheduled by NPG*-FP on an m-processor platform. Let r_k denote the release time of J_k invoked by τ_k. If there is no execution of J_k in $\Delta_k = [r_k, r_k + D_k - C_k)$, Eq. (15) holds for every $\tau_i \in \tau \setminus \{\tau_k\}$.

$$\left|L_{k\leftarrow i}(\Delta_k)\right| + \sum_{\tau_h \in \tau^{\text{HPF}}(\tau_k)} \left|L_{h\leftarrow i}(\Delta_k)\right| \leq \quad E_{k\leftarrow i} \tag{15}$$

Proof: First, we prove the following: if J_k does not start its execution in Δ_k, the duration of the execution of jobs of τ_i in Δ_k is upper-bounded by $E_{k\leftarrow i}$. Recall the process of deriving $E_{k\leftarrow i}$ as an upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$ in Section IV-B. In the process of deriving an upper-bound of $|L_{k\leftarrow i}(\Delta_k)|$, we use the conditions of $L_{k\leftarrow i}(\Delta_k)$: (i) a job of τ_i is executed, and (ii) a job of interest of τ_k (i.e., J_k) is pending. Therefore, $E_{k\leftarrow i}$ upper-bounds the duration of the execution of jobs of τ_i in Δ_k when J_k is not executed in Δ_k.

Based on this property, we can now prove the lemma. Suppose that Eq. (15) is violated even though there is no execution of J_k in Δ_k. Since $L_{h\leftarrow i}(\Delta_k)$ belongs to $L_h(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and $L_{k\leftarrow i}(\Delta_k)$ belongs to $L_k(\Delta_k)$, all $L_{h\leftarrow i}(\Delta_k)$ for $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and $L_{k\leftarrow i}(\Delta_k)$ are disjoint by Lemma 6. Also, by definition, in any interval within $L_{h\leftarrow i}(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and that within $L_{k\leftarrow i}(\Delta_k)$, there exists a job of τ_i that is executing. Therefore, the summation of the interval length of $L_{h\leftarrow i}(\Delta_k)$ for every $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and that of $L_{k\leftarrow i}(\Delta_k)$ is upper-bounded by the duration of the execution of jobs of τ_i in Δ_k when there is no execution of J_k in Δ_k. Therefore, the supposition contradicts the above proof, which proves the lemma. ■

Applying Lemma 7 to Theorem 1, we can develop a schedulability test for NPG*-FP.

Lemma 8: A task set τ is schedulable by NPG*-FP on an m-processor platform, if the following statement is true for every $\tau_k \in \tau$: Eq. (16) holds for every combination of $E'_{k\leftarrow i} \geq 0$ for $\tau_i \in \tau \setminus \{\tau_k\}$ and $E'_{h\leftarrow i}(\tau_k) \geq 0$ for a pair of $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and $\tau_i \in \tau \setminus \{\tau_h, \tau_k\}$ that satisfy Eq. (17).

$$\sum_{\tau_h \in \tau^{\text{HPF}}(\tau_k)} \left(\sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{E'_{h\leftarrow i}(\tau_k) \cdot \min(m_i, m - m_h + 1)}{m - m_h + 1} \right) + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \frac{E'_{k\leftarrow i} \cdot \min(m_i, m - m_k + 1)}{m - m_k + 1} \quad < \quad D_k - C_k \tag{16}$$

$$E'_{k\leftarrow i} + \sum_{\tau_h \in \tau^{\text{HPF}}(\tau_k)} E'_{h\leftarrow i}(\tau_k) \;\leq\; E_{k\leftarrow i} \tag{17}$$

Proof: Suppose that there exists a job of τ_k released at r_k (denoted by J_k) that misses its deadline, even though Eq. (16) holds for every combination of $E'_{k\leftarrow i}$ for $\tau_i \in \tau \setminus \{\tau_k\}$ and $E'_{h\leftarrow i}(\tau_k)$ for a pair of $\tau_h \in \tau^{\text{HPF}}(\tau_k)$ and $\tau_i \in \tau \setminus \{\tau_h, \tau_k\}$ that satisfy Eq. (17).

By Eq. (15), Eq. (16) subject to Eq. (17) is a sufficient condition for Eq. (5). Therefore, Theorem 1 guarantees that there is no deadline miss of J_k, which contracts the supposition. ■

If we compare the LHS of Eq. (16) subject to Eq. (17) with the LHS of Eq. (12), the former is always smaller than or equal to the latter. This is because, Eq. (17) disallows $E'_{h\leftarrow i}(\tau_k)$ to have a larger value than $E_{h\leftarrow i}(\tau_k)$, as $E_{h\leftarrow i}(\tau_k) = W_i(D_k - C_k)$ holds by Eq. (10) that is no smaller than $E_{k\leftarrow i}$ in Eqs. (7) and (8). Eq. (17) also disallows $E'_{k\leftarrow i}$ to have a larger value than $E_{k\leftarrow i}$, as the RHS of Eq. (17) is $E_{k\leftarrow i}$. Therefore, Lemma 8 is an improved version of the schedulability test in Theorem 2. However, since Lemma 8 entails many assignments to be investigated, we would like to find a single maximum assignment of $E'_{k\leftarrow i} = E^*_{k\leftarrow i}$ and $\{E'_{h\leftarrow i}(\tau_k) = E^*_{h\leftarrow i}(\tau_k)\}_{\tau_h \in \tau^{\text{HPF}}(\tau_k)}$ for given τ_i, which maximizes the LHS of Eq. (16) subject to Eq. (17).

Algorithm 2 represents how to calculate $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}$ for given τ_k and τ_i ($\neq \tau_k$). We distribute the budget of $E_{k\leftarrow i}$ (i.e., the RHS of Eq. (17)) to $E^*_{k\leftarrow i}$ or one of $\{E^*_{h\leftarrow i}(\tau_k)\}$ (i.e., terms in the LHS of Eq. (17)), such that the LHS of Eq. (16) is maximized. In Lines 1–2, $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}$ are set to 0. Since an increment of $E^*_{x\leftarrow i}$ or one of $\{E^*_{x\leftarrow i}(\tau_k)\}$ by 1 results in an increment of $\min(m_i, m - m_x + 1)/(m - m_x + 1)$ in the LHS of Eq. (16), we find τ_x that

Algorithm 2 Calculation of $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}$

For given τ_k and τ_i ($\neq \tau_k$);

1: $E^*_{k\leftarrow i} \leftarrow 0$
2: $E^*_{h\leftarrow i}(\tau_k) \leftarrow 0$ for every task τ_h in $\tau^{\mathrm{HPF}}(\tau_k)$
3: Let τ_x denote a task with the largest value of $\min(m_i, m-m_x+1)/(m-m_x+1)$ in $\{\tau_k\} \cup \tau^{\mathrm{HPF}}(\tau_k)$
4: **if** $\tau_x = \tau_k$ **then** $E^*_{k\leftarrow i} \leftarrow E_{k\leftarrow i}$;
5: **else** $E^*_{x\leftarrow i}(\tau_k) \leftarrow E_{k\leftarrow i}$;
6: **end if**

has the largest value of $\min(m_i, m-m_x+1)/(m-m_x+1)$ in $\{\tau_k\} \cup \tau^{\mathrm{HPF}}(\tau_k)$ (Line 3). If the selected task τ_x is τ_k, we assign $E_{k\leftarrow i}$ to $E^*_{k\leftarrow i}$ (Line 4). Otherwise (i.e., if the selected τ_x belongs to $\tau^{\mathrm{HPF}}(\tau_k)$), we assign $E_{k\leftarrow i}$ to $E^*_{x\leftarrow i}(\tau_k)$ (Line 5); this holds by $E_{k\leftarrow i} \le E_{h\leftarrow i}(\tau_k)$ (because of the RHS of Eqs. (7) and (8), which are no larger than the RHS of Eq. (10)).

Then, we can prove an important property of Algorithm 2 regarding the maximum assignment.

Lemma 9: $E'_{k\leftarrow i} = E^*_{k\leftarrow i}$ and $\{E'_{h\leftarrow i}(\tau_k) = E^*_{h\leftarrow i}(\tau_k)\}$ assigned by Algorithm 2 for given τ_k and τ_i ($\neq \tau_k$) maximize Eq. (18) (i.e., the terms that include a given $\tau_i \in \tau \setminus \{\tau_k\}$ in the LHS of Eq. (16)) subject to Eq. (17).

$$\sum_{\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)} \frac{E'_{h\leftarrow i}(\tau_k) \cdot \min(m_i, m-m_h+1)}{m-m_h+1} + \frac{E'_{k\leftarrow i} \cdot \min(m_i, m-m_k+1)}{m-m_k+1} \quad (18)$$

Proof: Suppose that there exist other assignments $E'_{k\leftarrow i} = E''_{k\leftarrow i}$ and $\{E'_{h\leftarrow i}(\tau_k) = E''_{h\leftarrow i}(\tau_k)\}$ which yield a larger value of Eq. (18) than that by $E'_{k\leftarrow i} = E^*_{k\leftarrow i}$ and $\{E'_{h\leftarrow i}(\tau_k) = E^*_{h\leftarrow i}(\tau_k)\}$. We consider two cases for τ_x assigned by Line 3: (i) $E''_{k\leftarrow i} \neq E^*_{k\leftarrow i}$ (if $\tau_x = \tau_k$ by Line 4), and (ii) $E''_{x\leftarrow i}(\tau_k) \neq E^*_{x\leftarrow i}(\tau_k)$ (otherwise by Line 5).

We will prove the case of $\tau_x = \tau_k$; the proof of the case for $\tau_x = \tau_h \in \tau^{\mathrm{HPF}}(\tau_k)$ is similar to that for $\tau_x = \tau_k$. First, we consider $E''_{k\leftarrow i} > E^*_{k\leftarrow i}$. Then, $E''_{k\leftarrow i} > E_{k\leftarrow i}$ holds by Line 4 in the algorithm, implying Eq. (17) is violated.

Second, we consider $E''_{k\leftarrow i} < E^*_{k\leftarrow i}$ holds. We focus on the case where the LHS of Eq. (17) is the same as the RHS; otherwise, we increase any arbitrary $E''_{k\leftarrow i}$ or one of $\{E''_{h\leftarrow i}(\tau_k)\}$ to further increase Eq. (18) without compromising Eq. (17). Then, it is possible to increase $E''_{k\leftarrow i}$ by $\left(E^*_{k\leftarrow i} - E''_{k\leftarrow i}\right)$ at the expense of decreasing $E''_{h\leftarrow i}(\tau_k)$ (for some τ_h) by $\left(E^*_{k\leftarrow i} - E''_{k\leftarrow i}\right)$; this results in increasing Eq. (18) by $\left(E^*_{k\leftarrow i} - E''_{k\leftarrow i}\right) \cdot \left(\frac{\min(m_i, m-m_k+1)}{m-m_k+1} - \frac{\min(m_i, m-m_h+1)}{m-m_h+1}\right) \ge 0$, for the assignment of $E''_{k\leftarrow i}$ and $\{E''_{h\leftarrow i}(\tau_k)\}$. Then, $E''_{k\leftarrow i}$ and $\{E''_{h\leftarrow i}(\tau_k)\}$ become the same as $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}$, respectively after this increment/decrement; this contracts the supposition (i.e., the existence of the assignments which yield a larger value of Eq. (18) than that assigned by Algorithm 2). ■

Then, we can present an improved schedulability test for NPG*-FP, by replacing $E_{k\leftarrow i}$ and $E_{h\leftarrow i}(\tau_k)$ in Eq. (12) with $E^*_{k\leftarrow i}$ and $E^*_{h\leftarrow i}(\tau_k)$ assigned by Algorithm 2, respectively, which is recorded in the following theorem.

Theorem 3: A task set τ is schedulable by NPG*-FP on an m-processor platform, if the following inequality holds for every $\tau_k \in \tau$:

$$\sum_{\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)} \left(\sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{E^*_{h\leftarrow i}(\tau_k) \cdot \min(m_i, m-m_h+1)}{m-m_h+1} \right) + \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \frac{E^*_{k\leftarrow i} \cdot \min(m_i, m-m_k+1)}{m-m_k+1} < D_k - C_k, \quad (19)$$

where $E^*_{k\leftarrow i}$ for every $\tau_i \in \tau \setminus \{\tau_k\}$ and $E^*_{h\leftarrow i}(\tau_k)$ for every pair of $\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)$ and $\tau_i \in \tau \setminus \{\tau_h, \tau_k\}$ are assigned by Algorithm 2.

Proof: By Lemma 9, the LHS of Eq. (19) is the maximum of the LHS of Eq. (16) subject to Eq. (17). Therefore, the theorem holds by Lemma 8. ■

Time-complexity of Theorem 3. Algorithm 2 needs $O(n')$ computations to find τ_x with the largest value of $\min(m_i, m-m_h+1)/(m-m_h+1)$, for a given pair of τ_k and τ_i ($\neq \tau_k$), where n' is the number of tasks $\tau_i \in \tau$ that satisfy $\phi_i = \mathtt{F}$. Therefore, to calculate $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}_{\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)}$ for every pair of τ_k and τ_i ($\neq \tau_k$), we need $O(n^2 \cdot n')$, where n is the number of tasks in τ.

Once we have $E^*_{k\leftarrow i}$ and $\{E^*_{h\leftarrow i}(\tau_k)\}_{\tau_h \in \tau^{\mathrm{HPF}}(\tau_k)}$ for every pair of τ_k and τ_i ($\neq \tau_k$) by Algorithm 2, the time-complexity of the remaining calculation for Theorem 3 is the same as the total time-complexity of Theorem 2, which is $O(n^2 \cdot n')$. Therefore, the total time-complexity of Theorem 3 is $O(n^2 \cdot n')$.

V. Optimal Assignment of $\{\phi_j\}$ for NPG*-FP

In Section IV, we developed schedulability tests for NPG*-FP with given assignments of $\{\phi_j\}_{\tau_j \in \tau}$. Since ϕ_j is a task option controllable by the scheduler, we can find an assignment of $\{\phi_j\}_{\tau_j \in \tau}$ associated with the proposed schedulability tests, which makes τ (that is unschedulable by other assignments) schedulable. In this section, we first derive the properties of the proposed schedulability tests, and then develop an algorithm that optimally assigns $\{\phi_j\}_{\tau_j \in \tau}$ based on the properties.

We investigate how a change of ϕ_h for a single task $\tau_h \in \tau$ affects the schedulability of its higher-priority tasks, its lower-priority tasks, and τ_h itself, summarized as follows.

Lemma 10: Suppose τ is scheduled by NPG*-FP on an m-processor platform, and ϕ_j for every $\tau_j \in \tau \setminus \{\tau_h\}$ is set (to either T or F, independently). Then, S1–S3 hold for $\tau_k \in \tau$, when we change a single ϕ_h for $\tau_h \in \tau$.

S1. If $\tau_h \in \tau^{\mathrm{LP}}(\tau_k)$ holds, the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{T}$ is the same as the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{F}$.

S2. If $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$ holds, the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{T}$ is no larger than the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{F}$.

Algorithm 3 Optimal Assignment of $\{\phi_j\}_{\tau_j \in \tau}$

1: $\phi_j \leftarrow \mathtt{T}$ for every task $\tau_j \in \tau$
2: **for** every task τ_j in τ from the highest-priority task to the lowest-priority task **do**
3: **if** Eq. (19) does not hold for $\tau_k = \tau_j$ **then** $\phi_j \leftarrow \mathtt{F}$
4: **if** Eq. (19) does not hold for $\tau_k = \tau_j$ **then** return unschedulable
5: **end if**
6: **end if**
7: **end for**
8: Return schedulable

S3. If $\tau_k = \tau_h$ holds, the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{T}$ is no smaller than the LHS of Eq. (12) for τ_k when $\phi_h = \mathtt{F}$.

Also, S1–S3 hold by replacing the LHS of Eq. (12) with that of Eq. (19).

Proof: First, we prove S1–S3 for the LHS of Eq. (12). In the LHS of Eq. (12), the only parts that can be affected by $\{\phi_j\}_{\tau_j \in \tau}$ are $E_{k \leftarrow i}$, $E_{h \leftarrow i}(\tau_k)$, and the target task set $\tau^{\mathrm{HPF}}(\tau_k)$ in the summation.

If we focus on $E_{k \leftarrow i}$, they are affected by ϕ_k. This is because, the value of $E_{k \leftarrow i}$ depends on ϕ_k: either Eq. (7) for $\phi_k = \mathtt{T}$ or Eq. (8) for $\phi_k = \mathtt{F}$. The only difference between Eq. (7) for $\phi_k = \mathtt{T}$ and Eq. (8) for $\phi_k = \mathtt{F}$ is the case for $\tau_i \in \tau^{\mathrm{LP}}(\tau_k) | m_i < m_k$. Since $W_i(D_k - C_k)$ for $\phi_k = \mathtt{T}$ with $\tau_i \in \tau^{\mathrm{LP}}(\tau_k) | m_i < m_k$ is no smaller than $\min(D_k - C_k, C_i)$ for $\phi_k = \mathtt{F}$ with $\tau_i \in \tau^{\mathrm{LP}}(\tau_k) | m_i < m_k$, $\phi_k = \mathtt{F}$ yields the same or a smaller value of $E_{k \leftarrow i}$ than $\phi_k = \mathtt{T}$. On the other hand, $E_{h \leftarrow i}(\tau_k)$ is not affected by ϕ_k as shown in Eq. (10). Therefore, S3 holds.

Different from ϕ_k, both $E_{k \leftarrow i}$ and $E_{h \leftarrow i}(\tau_k)$ are not affected by ϕ_i and ϕ_h by their definitions, which proves S1. On the other hand, if a higher-priority task τ_h assigns $\phi_h = \mathtt{F}$, a lower-priority task τ_k has a new term of $\sum_{\tau_i \in \tau \setminus \{\tau_h, \tau_k\}} \frac{E_{h \leftarrow i}(\tau_k) \cdot \min(m_i, m - m_h + 1)}{m - m_h + 1}$ in the LHS of Eq. (12). This is why S2 holds although $E_{k \leftarrow i}$ and $E_{h \leftarrow i}(\tau_k)$ are not affected by ϕ_i.

Second, we prove S1–S3 for the LHS of Eq. (19). Since $E^*_{k \leftarrow i}$ and $E^*_{h \leftarrow i}(\tau_k)$ in the LHS of Eq. (19) are calculated from Algorithm 2 based on Lemma 8, we need to show that every upper-bound in Lemma 8 accords with S1–S3. We can observe that the upper-bound shown in Eq. (17) in Lemma 8 is $E_{k \leftarrow i}$ only, whose impact on the schedulability is already addressed in the proof of S1–S3 for the LHS of Eq. (12). Hence, S1–S3 for the LHS of Eq. (19) also hold. ■

Using Lemma 10, we present how to assign $\{\phi_j\}_{\tau_j \in \tau}$ in Algorithm 3. After setting ϕ_j to $\mathtt{T}$ for all $\tau_j \in \tau$ as default (Line 1), we determine ϕ_j for every $\tau_j \in \tau$ from the highest-priority task to the lowest-priority task (Lines 2–7) because the schedulability of a higher-priority task τ_j is not affected by its lower-priority tasks' ϕ_h (explained in S1).

For each task τ_j whose ϕ_j is selected to be assigned, we check whether $\phi_j = \mathtt{T}$ makes Eq. (19) for $\tau_k = \tau_j$ true. If so, the final assignment of ϕ_j is $\mathtt{T}$ (by no change of $\phi_j = \mathtt{T}$ due to no "else" statement for the "if" statement in Line 3); this is because, the assignment of $\phi_j = \mathtt{T}$ is beneficial to its lower-priority tasks' schedulability compared to that of $\phi_j = \mathtt{F}$ (explained in S2). Otherwise, the only way for τ_j to be schedulable without changing ϕ_h of its higher-priority tasks $\tau_h \in \tau^{\mathrm{HP}}(\tau_j)$ (which is already determined) is to assign $\phi_j = \mathtt{F}$ according to S3 (Line 3). If τ_j cannot be deemed schedulable even after assigning $\phi_j = \mathtt{F}$, there is no way for τ_j to be schedulable according to S3; in this case, we return unschedulable (Line 4). Finally, if ϕ_j of every task $\tau_j \in \tau$ is assigned without returning unschedulable, we return schedulable (Line 8). We state and prove the optimality of Algorithm 3 in the following theorem.

Theorem 4: If Algorithm 3 returns unschedulable for a task set τ, there is no assignment of $\{\phi_j\}_{\tau_j \in \tau}$ such that Theorem 3 deems τ schedulable by NPG*-FP on an m-processor platform.

Proof: Suppose that Algorithm 3 returns unschedulable, but there exists an assignment of $\{\phi_j\}_{\tau_j \in \tau}$ such that Theorem 3 deems τ schedulable by NPG*-FP on an m-processor platform; we denote the assignment as $\{\phi'_j\}_{\tau_j \in \tau}$. By the supposition, there should be a task τ_k that makes Algorithm 3 return unschedulable in Line 4. This means, Eq. (19) for τ_k does not hold for both $\phi_k = \mathtt{T}$ and $\phi_k = \mathtt{F}$, under the assignment of ϕ_h for every $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$ by Algorithm 3; we denote the assignment as $\{\phi^*_h\}_{\tau_h \in \tau^{\mathrm{HP}}(\tau_k)}$.

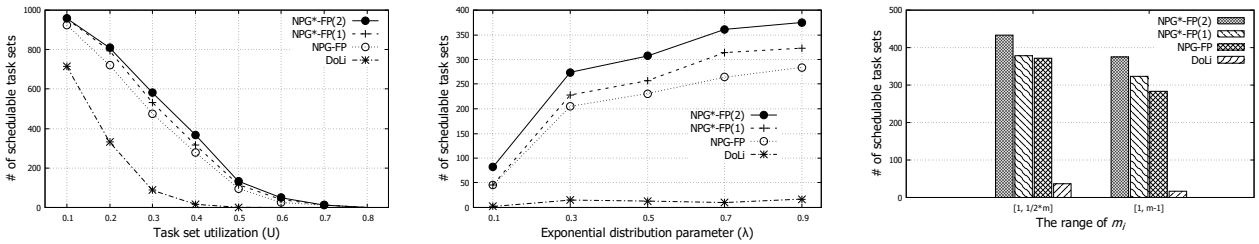
We investigate every $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$ that satisfies $\phi'_h \neq \phi^*_h$ from the highest-priority task to the lowest-priority task. (Case 1) If there is no such τ_h, ϕ'_h is equal to ϕ^*_h for every $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$. This contradicts the unschedulability of τ_k when $\{\phi'_h\} = \{\phi^*_h\}$ for every $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$ is assigned by Algorithm 3, because $\{\phi'_g\}$ for every $\tau_g \in \tau^{\mathrm{LP}}(\tau_k)$ does not affect the schedulability of τ_k by S1.

(Case 2) If there is at least one τ_h that satisfies $\phi^*_h = \mathtt{F}$ and $\phi'_h = \mathtt{T}$, it contradicts the mechanism of Algorithm 3. This is because, Algorithm 3 checks whether τ_h is schedulable with $\phi_h = \mathtt{T}$ (in Line 3) before that with $\phi_h = \mathtt{F}$ (in Line 4), and any assignment of $\{\phi_g\}$ for $\tau_g \in \tau^{\mathrm{LP}}(\tau_h)$ does not affect the schedulability of τ_h (by S1).

(Case 3) Therefore, the remaining case is that there are at least one task τ_h that satisfies $\phi^*_h = \mathtt{T}$ and $\phi'_h = \mathtt{F}$. This also contradicts the schedulability and unschedulability of τ_k under the assignment of $\{\phi'_j\}_{\tau_j \in \tau}$ and $\{\phi^*_j\}_{\tau_j \in \tau}$, respectively. This is because, such $\phi_h = \mathtt{F}$ non-decreases the LHS of Eq. (19) for τ_k, compared to $\phi_h = \mathtt{T}$ (as explained in S2). Therefore, τ_k under the assignment of $\{\phi^*_h\}_{\tau_h \in \tau^{\mathrm{HP}}(\tau_k)}$ cannot return unschedulable if τ_k under the assignment of $\{\phi'_j\}_{\tau_j \in \tau}$ is schedulable.

In summary, $\tau_h \in \tau^{\mathrm{HP}}(\tau_k)$ that satisfies $\phi'_h \neq \phi^*_h$ cannot exist, which implies non-existence of τ_k that makes Algorithm 3 return unschedulable. This contradicts the supposition, which proves the lemma. ■

Note that the optimality of Algorithm 3 also holds if we apply Theorem 2 to Algorithm 3 by replacing Eq. (19) with Eq. (12). Also, the time-complexity of Algorithm 3 is the same as that of its underlying schedulability test (i.e., $O(n^2 \cdot n')$ for



(a) Different ranges of U, when $m_i \in [1, m)$ and $u_i \sim \texttt{exp}(0.9)$

(b) Different distributions of u_i, when $m_i \in [1, m)$ and $U \in [0.3, 0.4)$

(c) Different ranges of m_i, when $u_i \sim \texttt{exp}(0.9)$ and $U \in [0.3, 0.4)$

Fig. 3. Schedulability performance of DoLi, NPG-FP, NPG*-FP(1) and NPG*-FP(2) under different settings when $m = 8$

Theorems 2 and 3), because each task τ_k checks its underlying schedulability test at most twice in Algorithm 3.

VI. Evaluation

In this section, we evaluate the effectiveness of the proposed NPG*-FP framework in improving schedulability performance, via simulations.

We randomly generate a number of task sets, based on recent studies for gang scheduling [21, 23]. We consider four options for the number of processors m (8, 16, 32 and 64), with three parameters: (i) the distribution of the task utilization $u_i \stackrel{\texttt{def.}}{=} C_i/T_i$ (the exponential distribution with $\lambda = 0.1$, 0.3, 0.5, 0.7 and 0.9, denoted by $\texttt{exp}(\lambda)$), (ii) the range of the task parallelism m_i ($[1, \frac{1}{2}m]$ and $[1, m)$), and (iii) the range of the task set utilization $U \stackrel{\texttt{def.}}{=} \sum_{\tau_i \in \tau} \frac{u_i \cdot m_i}{m}$ ($[0.0, 0.1)$, $[0.1, 0.2)$, $[0.2, 0.3)$, ... , $[0.9, 1.0)$). For given (i) and (ii), a task is generated as follows. The period T_i is uniformly selected in $[10ms, 1000ms]$; the relative deadline D_i is set to T_i; C_i is set to $u_i \cdot T_i$, where u_i is generated according to the given (i); and m_i is uniformly distributed in the given (ii). For every combination of (i), (ii) and (iii), we generate 1,000 task sets, yielding 5·2·10·1,000=100,000 task sets in total for each m.

We compare the following schedulability tests for NPG and those for NPG*:[1]

- DoLi (from the authors' name): an existing schedulability test for NPG-EDF [26], which is the only known existing schedulability test for traditional NPG under our system model,
- NPG-FP: Theorem 2 for NPG-FP (i.e., NPG*-FP by assigning $\phi_i = \texttt{T}$ for every $\tau_i \in \tau$), and[2]
- NPG*-FP(1) and NPG*-FP(2): Theorems 2 and 3 for NPG*-FP with the assignment of $\{\phi_j\}$ by Algorithm 3, respectively.

For each m, we count the number of schedulable task sets by the four schedulability tests, which are presented in Table I. We have the following observations. First, NPG*-FP(2) covers 161.2%–346.0% and 11.2%–15.6% more schedulable task sets than DoLi and NPG-FP, respectively. In terms of schedulability performance, the former implies that our schedulability test for NPG* significantly outperforms the existing schedulability test for traditional NPG, and the latter implies that NPG* fairly enhances traditional NPG under the same prioritization policy. Second, NPG*-FP(2) finds 6.8%–8.5% additional schedulable task sets that are not proven schedulable by NPG*-FP(1), which demonstrates the effectiveness of the techniques in Section IV-C. Third, as m gets larger, the number of schedulable task sets by DoLi significantly decreases while that by our schedulability tests does not. This indicates that the schedulability performance of our schedulability tests is more robust to larger m, than the existing one.

TABLE I
The number of schedulable task sets, and its normalized value by NPG-FP (shown in the parentheses)

m	DoLi	NPG-FP	NPG*-FP(1)	NPG*-FP(2)
8	10655 (43.0%)	24806 (100.0%)	26070 (105.1%)	27836 (112.2%)
16	8070 (35.5%)	22718 (100.0%)	24190 (106.5%)	25994 (114.4%)
32	6233 (29.1%)	21446 (100.0%)	22863 (106.6%)	24813 (115.7%)
64	5423 (25.9%)	20929 (100.0%)	22337 (106.7%)	24187 (115.6%)

While NPG*-FP(2) and NPG*-FP(1) respectively dominate NPG*-FP(1) and NPG-FP, our schedulability tests cannot dominate DoLi. However, almost all task sets deemed schedulable by DoLi are also deemed schedulable by our tests; for example, out of 100,000 task sets with $m = 64$, there are 5,423 task sets schedulable by DoLi, out of which *only ten* task sets are not proven schedulable by NPG*-FP(2). Note that the huge performance gap between DoLi and NPG-FP does not necessarily mean FP outperforms EDF for NPG scheduling; DoLi is an instance of EDF schedulability analysis, and the performance comparison between FP and EDF is a long-term question that researchers will try to answer in the future by developing tighter schedulability analysis.

We then investigate how the schedulability performance varies according to a change of (i) the range of U, (ii) the distribution of u_i, and (iii) the range of m_i. To show the representative results, Fig. 3 focuses on $m = 8$, and presents the number of task sets schedulable by the four schedulability tests under different settings for one of (i), (ii) and (iii) while fixing the settings for the other two of (i), (ii) and (iii) by $U \in [0.3, 0.4)$, $u_i \sim \texttt{exp}(0.9)$ and $m_i \in [1, m)$, respectively.

[1]We apply DM (Deadline Monotonic) as a task priority assignment policy to both NPG-FP and NPG*-FP. It is beyond the scope of this paper to find the optimal task priority assignment for FP.

[2]Theorems 2 and 3 are equivalent for NPG-FP.

First, we confirm that similar observations to the first and second ones for Table I hold in most ranges of U, as shown in Fig. 3(a) that presents different (i).

Second, Fig. 3(b) for different (ii) shows that, as λ in $\texttt{exp}(\lambda)$ increases from 0.1 to 0.9, the number of schedulable task sets by our schedulability tests increases. This holds mainly because, a smaller λ in $\texttt{exp}(\lambda)$ tends to assign a smaller u_i for each task, yielding a larger number of tasks in each task set for a given U. In our schedulability tests, as the number of tasks in each task set gets larger, the more pessimism is accumulated to the upper-bounds calculated in the tests, yielding less schedulability performance; this has been reported in the existing interference-based schedulability tests (e.g., [28–31]).

Third, Fig. 3(c) for different (iii) shows that the schedulability performance of all the schedulability tests under $m_i \in [1, \frac{1}{2}m]$ is better than that under $m_i \in [1, m)$; this is because, a task with a larger m_i makes it more difficult to yield its schedulability. On the other hand, we observe that the schedulability improvement of $\texttt{NPG}^*\texttt{-FP(2)}$ over $\texttt{NPG-FP}$ under $m_i \in [1, m)$ (i.e., 32.0%) is higher than that under $m_i \in [1, \frac{1}{2}m]$ (i.e., 16.7%). This is because, although the priority-inversion specialized for NPG increases with a larger range of m_i, our NPG* framework can effectively eliminate such blocking by properly assigning $\phi_j = \texttt{F}$.

VII. Related Work

There have been studies on gang scheduling for real-time systems, most of which deal with preemptive gang scheduling (PG). Among the studies on PG, a group of them have targeted the most fundamental prioritization policies, which are EDF (Earliest Deadline First) and FP (Fixed-Priority scheduling), and developed schedulability tests for PG-EDF [17, 21, 23, 33] and for PG-FP [18, 19, 33]; note that flaws in [17] were reported in [22]. Also, an optimal scheduling algorithm for PG has been explored in [16, 20]. A new concept of stationary scheduling, which is a generalization of partitioned scheduling, has been developed for PG [34], and it has been addressed how to schedule DAG-structured tasks, in which each node is expressed as a gang task [35]. While those studies have focused on a set of single-criticality hard real-time tasks, they have been extended to a set of mixed-criticality hard real-time tasks [24] and that of single-criticality soft real-time tasks [25].

On the other hand, only a few studies have focused on non-preemptive gang scheduling (NPG). The study in [26] has developed the first schedulability test for NPG-EDF (traditional NPG with EDF), and introduced a new type of priority-inversion incurred by NPG (i.e., PR2 in Section III). A schedulability test of NPG-FP has been recently developed [27], but its applicability is limited to a set of a finite number of jobs whose release patterns are known in advance. Different from the previous studies, our study proposes a new, generalized design of NPG, called the NPG* framework. We then developed schedulability tests for NPG*-FP under a given setting of $\{\phi_j\}$, addressed an assignment problem that finds an assignment of $\{\phi_j\}$ (if exists) that makes a target task set schedulable by the schedulability tests, and compared the schedulability performance of NPG*-FP with the $\{\phi_j\}$ assignment algorithm, with that of the traditional NPG framework. All of the proposed design, development and comparison are the first attempt in the area of scheduling a set of non-preemptive gang tasks.

VIII. Conclusion

In this paper, we proposed a new generalized NPG framework, NPG*, under which each task has an option to allow or disallow the priority-inversion specialized for NPG. To demonstrate the effectiveness of the NPG* framework, we developed a schedulability test for NPG*-FP, its improved version, and an algorithm that optimally assigns the task-level option. The simulation results showed that the NPG* framework associated with the proposed schedulability tests and the optimal assignment algorithm finds a number of additional schedulable task sets, each of which has not been proven schedulable by the traditional NPG framework. In the future, we would like to further improve the schedulability performance of the NPG* framework. One direction is to derive tighter upper-bounds of the terms shown in the proposed schedulability tests. It is also a promising direction to develop a schedulability test for NPG* associated with other prioritization policies such as EDF.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2021R1A2B5B02001758, NRF-2021K2A9A1A01101570, NRF-2022R1A4A3018824) and Hong Kong Research Grant Council (GRF 11208522 and GRF 15206221).

References

[1] OpenMP. http://openmp.org.

[2] M. Frigo, C. E. Leiserson, and K. H. Randall, "The implementation of the Cilk-5 multithreaded language," *ACM SIGPLAN Notices*, vol. 33, no. 5, pp. 212–223, 1998.

[3] J. Reinders, *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*. O'Reilly Media Inc., 2007.

[4] D. Kirk, "NVIDIA Cuda software and GPU parallel computing architecture," in *Proceedings of the international symposium on memory management*, 2007, pp. 103–104.

[5] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE micro*, vol. 31, no. 5, pp. 7–17, 2011.

[6] TILE-Gx. https://www.pcmag.com/encyclopedia/term/tile-gx.

[7] Y. Cho, D. H. Kim, D. Park, S. S. Lee, and C.-G. Lee, "Conditionally optimal task parallelization for global edf on multi-core systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 194–206.

[8] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 128–140.

[9] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2020, pp. 226–238.

[10] D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *Journal of Parallel and distributed Computing*, vol. 16, no. 4, pp. 306–318, 1992.

[11] M. A. Jette, "Performance characteristics of gang scheduling in multiprogrammed environments," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 1997.
[12] B. B. Zhou and R. P. Brent, "Gang scheduling with a queue for large jobs," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2001.
[13] K. D. Ryu, N. Pachapurkar, and L. L. Fong, "Adaptive memory paging for efficient gang scheduling of parallel applications," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2004.
[14] S. Singh, "Performance optimization in gang scheduling in cloud computing," *International Organization of Scientific Research-Journal of Computer Engineering*, vol. 2, no. 4, pp. 49–52, 2012.
[15] W. Ali and H. Yun, "RT-Gang: Real-time gang scheduling framework for safety-critical systems," in *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2019, pp. 143–155.
[16] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Information processing letters*, vol. 106, no. 5, pp. 180–187, 2008.
[17] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, December 2009, pp. 459–468.
[18] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," *CoRR, abs/1006.2617 URL: http://arxiv.org/abs/1006.2617*, 2010.
[19] V. Berten, P. Courbin, and J. Goossens, "Gang fixed priority scheduling of periodic moldable realtime tasks," in *In 5th Junior Researcher Workshop on Real-Time Computing*, 2011, pp. 9–12.
[20] J. Goossens and P. Richard, "Optimal scheduling of periodic gang tasks," *Leibniz transactions on embedded systems*, pp. 04:1–04:18, 2016.
[21] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, December 2017, pp. 128–138.
[22] R. Richard, J. Goossens, and S. Kato, "Comments on "gang EDF schedulability analysis"," *arXiv preprint arXiv:1705.05798*, 2014.
[23] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," *Real-Time Systems*, vol. 55, no. 4.
[24] A. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, December 2019, pp. 469–480.
[25] Z. Dong, K. Yang, N. Fisher, and C. Liu, "Tardiness bounds for sporadic gang tasks under preemptive global edf scheduling," *IEEE Transactions on Parallel and Distributed Systems*, 2021, Early Access.
[26] Z. Dong and C. Liu, "Work-in-progress: Non-preemptive scheduling of sporadic gang tasks on multiprocessors," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, December 2019, pp. 512–515.
[27] G. Nelissen, J. Marcè i Igual, and M. Nasri, "Response-time analysis for non-preemptive periodic moldable gang tasks," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2022, pp. 12:1–12:22.
[28] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2005, pp. 209–218.
[29] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.
[30] J. Lee and K. G. Shin, "Controlling preemption for better schedulability in multi-core systems," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2012, pp. 29–38.
[31] J. Lee and K. G. Shin, "Improvement of real-time multi-core schedulability with forced non-preemption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1233–1243, 2014.
[32] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, no. 1-2, pp. 9–20, 2006.
[33] S. Lee, S. Lee, and J. Lee, "Response time analysis for real-time global gang scheduling," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2022, to appear.
[34] N. Ueter, M. Gunzel, G. von der Bruggen, and J.-J. Chen, "Hard real-time stationary GANG-scheduling," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2021, pp. 10:1–10:19.
[35] W. Ali, R. Pellizzoni, and H. Yun, "Virtual gang scheduling of parallel real-time tasks," in *Proceedings of Design Automation and Test in Europe (DATE)*, 2021, pp. 270–275.