Necessary Feasibility Analysis for Mixed-Criticality Real-Time Embedded Systems

Hoon Sung Chwa, *Member, IEEE*, Hyeongboo Baek, and Jinkyu Lee, *Senior Member, IEEE*

Abstract—As multiple software components with different safety-criticality levels are integrated on a shared computing platform, a real-time embedded system becomes a mixed-criticality (MC) system, which should provide timing guarantees at all different levels of assurance to software components with different criticality levels. In the real-time systems community, the concept of an MC system is regarded as a promising, emerging solution to solve an inherent challenge of real-time systems: pessimistic reservation of computing resources, which yields a low resource-utilization for the sake of guaranteeing timing requirements. Since a timing guarantee should be provided before a real-time system starts to operate, its feasibility has been extensively studied for single-criticality systems; however, the same cannot be said for MC systems. In this article, we develop necessary feasibility tests for MC real-time embedded systems, which is the first study that yields non-trivial results for MC necessary feasibility on both uniprocessor and multiprocessor platforms. To this end, we investigate characteristics of MC necessary feasibility conditions, and identify new challenges posed by the characteristics. By addressing those challenges, we develop two collective necessary feasibility tests and their simplified versions, which are able to exploit a tradeoff between capability in finding infeasible task sets and time-complexity. The simulation results demonstrate that the proposed tests find a number of additional infeasible task sets for both uniprocessor and multiprocessor platforms, which have been proven neither feasible nor infeasible by any existing studies.

Index Terms—Real-time embedded systems, mixed-criticality systems, necessary feasibility analysis, timing guarantees, uniprocessor and multiprocessor platforms

1 Introduction

Nowadays, we have witnessed an increasing trend in industrial developments towards integrated computing environments, in which multiple software components (i.e., tasks) with different safety-criticality levels are integrated on a shared computing platform. Their prototypical examples are AUTOSAR in the automotive industry [1] and Integrated Modular Avionics (IMA) in the aerospace industry [2]. Such a *mixed-criticality* (MC) property is typically employed in real-time embedded systems, whose correctness depends not only on the functional correctness, but also on the temporal correctness (i.e., timing guarantees) [3], [4]. The real-time systems community has paid attention to the emerging concept of an MC property as it can be promising to solve a fundamental challenge of real-time systems—a low utilization of computing resources due to inevitably pessimistic resource reservation that guarantees timing requirements. The major design challenge for MC real-time embedded systems is to simultaneously provide timing guarantees at all different levels of assurance to tasks with different criticality levels.

The most fundamental issue for the timing guarantees is "feasibility" of timing guarantees—whether every instance of real-time tasks finishes its execution within its deadline under a given setting (e.g., a computing resource such as uni- and multi-processors and a task model such as the Liu and Layland task model [5]). We may classify the research into two categories: (i) developing scheduling algorithms and their schedulability analysis to expand a set of real-time task sets proven schedulable by at least a scheduling algorithm (i.e., addressing sufficient feasibility) [6], [7], and (ii) deriving conditions of task sets that are never schedulable by any scheduling algorithm to reduce a set of task sets that are potentially unschedulable but have not been proven unschedulable so far (i.e., addressing necessary feasibility) [8], [9], [10], [11].

Existing studies belonging to (i) and those belonging to (ii) have matured for single-criticality (SC) task and some of their results have been successfully adapted to MC task systems [3] in which tasks have different criticality levels. However, most studies for MC task systems have focused on (i) including MC-specific scheduling algorithms and/or their schedulability analysis (see [4] for a survey); only a

- *Hoon Sung Chwa is with the Department of Information and Communication Engineering, DGIST, Daegu 42988, South Korea. E-mail: chwahs@dgist.ac.kr.*
- *Hyeongboo Baek is with the Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea. E-mail: hbbaek@inu.ac.kr.*
- *Jinkyu Lee is with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea. E-mail: jinkyu.lee@skku.edu.*

Manuscript received 23 Feb. 2021; revised 22 Sept. 2021; accepted 1 Oct. 2021. Date of publication 7 Oct. 2021; date of current version 1 Nov. 2021.
This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIT) under Grants 2020 R1F1A1076058, 2021R1A2B5B02001758, 2019R1F1A1059663, and 2017M3 A9G8084463 and the Institute for Information & communications Technology Planning & Evaluation (IITP) Grant funded by the Korea government (MSIP) under Grant 2014-3-00065, Resilient Cyber-Physical Systems Research), as well as the DGIST R&D Program of MSIT (20-CoE-IT-01).
(Corresponding author: Jinkyu Lee.)
Recommended for acceptance by B. Veeravalli.
Digital Object Identifier no. 10.1109/TPDS.2021.3118610

few studies have addressed (ii), but all of them have presented trivial results [12], [13], [14]. This paper aims at reducing a set of MC task sets whose feasibility is unknown by existing studies. In particular, we aim at developing *necessary feasibility* tests that prove infeasibility of some of such MC task sets not only for a uniprocessor platform, but also for a multiprocessor one, which is the first study in MC real-time embedded systems (except the trivial results in [12], [13], [14]).

Addressing necessary feasibility for MC task systems is beneficial both from the theoretical and the practical point of view. On the theoretical side, when a schedulability test is unable to verify that a target task set is schedulable, tight necessary feasibility analysis can distinguish the cause of failure: i) the problem is with the task set, which never be schedulable by any scheduling algorithm, or ii) the problem is with the scheduling algorithm and its schedulability test. Thus, it eliminates unnecessary efforts for researchers to try to make task sets schedulable by developing a new scheduling algorithm, if the task sets are proven infeasible by the necessary feasibility analysis. On the practical side, when system software designers set the configuration for scheduling algorithms, task parameters and computing resources, tight necessary feasibility results reduce burden of tuning parameters for the configuration by excluding some infeasible choices of the configuration. In addition, it helps understand MC systems to investigate infeasible task sets proven by a necessary feasibility analysis. For example, we can derive characteristics for combinations of tasks belonging to each infeasible task set. Also, those infeasible task sets give a guideline of how to design MC systems to avoid infeasible task sets.

Before we explain the development of necessary feasibilitiy tests for MC task systems, we present the MC feasibility requirement (that is a de facto standard) as follows. A typical MC task model [3] is to assume more pessimistic worst-case execution times (WCETs) for higher criticality levels for each task. In systems with two levels of criticality (i.e., high and low), the system can be seen as exhibiting two different behaviors at runtime: the low-criticality behavior as long as each job signals its completion without exceeding its low-criticality WCET, and the high-criticality behavior thereafter if any high-criticality job does not signal its completion after executing for its low-criticality WCET. A typical feasibility requirement for MC systems is that 1) all high-criticality tasks always meet their deadlines and 2) all low-criticality tasks meet their deadlines during the low-criticality system behavior, which is formally specified in Section 2.

To develop necessary feasibility tests in accordance with the above MC feasibility definition, we first offer our own interpretation of the existing necessary feasibility tests for SC task systems on a uniprocessor and on a multiprocessor platform (in Section 3.1). We next show that a straightforward extension of such interpretation towards MC task systems is limited to covering only partial cases of MC behaviors, yielding a still huge gap between the region covered by task sets proven feasible and that proven infeasible (in Section 3.2). Utilizing the background, we identify unique issues/challenges specific to MC task systems for developing necessary feasibility tests, based on investigation of their characteristics (in Section 3.3), which can be summarized briefly as follows.

C1. According to the MC-feasibility requirement, (i) a set of jobs whose execution requirement should be guaranteed (i.e., all versus high-criticality jobs) and (ii) high-criticality jobs' amount of execution requirement that should be guaranteed (i.e., no more than versus more than low-criticality WCET) vary depending on the system behavior.

C2. It is impossible to know beforehand which system behavior will be shown and when the system switches from low-criticality to high-criticality behavior (referred to as *mode change*) during runtime.

Such unique characteristics of MC task systems pose new challenges that cannot be resolved by existing techniques for SC task systems, which will be explained I1–I6 in Section 3.3. We establish essential foundations for developing necessary feasibility tests by addressing those challenges, which are summarized in Section 4.2. Putting all the pieces together, we develop two types of collective necessary feasibility tests, considering different target scenarios (including job release patterns) for a task set to check its infeasibility. We also explore a tradeoff between time-complexity and capability in finding infeasible task sets, by developing simplified versions of the two collective tests that have the same complexity as in the SC task system case at the expense of sacrificing the capability.

We demonstrate effectiveness of the proposed necessary feasibility tests in finding infeasible task sets via simulations. The proposed tests are able to newly cover many infeasible task sets which have not been proven neither feasible nor infeasible. In particular, in case of constrained-deadline task sets, a collective necessary feasibility test finds 11,375 (25.9%), 13,894 (24.7%), and 17,778 (22.3%) additional infeasible task sets among 43,972, 56,199, and 79,806 task sets of interests, which have not been proven infeasible by any existing studies, on 1-, 2-, and 4-processor platforms, respectively.

Contribution. The main contributions of this paper can be summarized as follows:

1) We develop necessary feasibility tests for MC task systems, which is the first study that yields non-trivial results for MC necessary feasibility on both uniprocessor and multiprocessor platforms;
2) We pose new challenges for developing necessary feasibility tests specialized for MC task systems (in Section 3);
3) We establish foundations of necessary feasibility tests for MC task systems, by addressing the new challenges one by one (in Section 5);
4) We develop a collective necessary feasibility test to determine infeasibility of MC task sets with as many scenario instances as possible with reasonable (pseudo-polynomial) complexity, by incorporating the foundations (in Section 6);
5) We investigate another scenario (i.e., job release pattern), favorable to finding infeasible MC task sets, by developing an improved version of the collective necessary feasibility test (in Section 7);
6) We explore a tradeoff between time-complexity and capability in finding infeasible task sets, by developing simplified versions of the two types of necessary

feasibility tests that have the same complexity as in the SC task system case (in Sections 8 and 9); and

7) We demonstrate effectiveness of the proposed tests in finding infeasible task sets for both uniprocessor and multiprocessor platforms (in Section 10).

In our preliminary conference version [15], we presented a collective necessary feasibility test and its simplified version for MC task systems on a uniprocessor platform. In this extended version, we generalize all the results in the conference version to a multiprocessor platform, and reveal another scenario favorable to finding infeasible MC task sets with further improvements. Therefore, the entire part of 5) and 6) and the multiprocessor part of 1) and 7) are novel contributions that have not been addressed in the conference version. In addition, we provide an overview of our approach to develop necessary feasibility tests for MC task systems, including new scenario components for MC task systems to consider the unique characteristics thereof (in Section 4.2).

2 System Model, Assumptions and Notations

We consider the problem of scheduling a dual-criticality (high and low, namely HI and LO) task set τ of n sporadic MC tasks on a platform consisting of $m \geq 1$ identical processors. This means, we cover both uniprocessor and multiprocessor platforms; only one job can be executed at any time under the former, while at most m jobs can be executed at any time under the latter. Also, when it comes to a multiprocessor platform, we do not have any restriction for a job to be executed on which processor, meaning that any global and partitioned scheduling algorithm can be applied. Under those settings, this paper aims at finding conditions for necessary feasibility—whether a set of given tasks cannot meet all their job deadlines with *any* scheduling algorithm.

MC Tasks. Each MC task $\tau_i \in \tau$ is characterized by a tuple $(T_i, \chi_i, C_i^{\mathtt{LO}}, C_i^{\mathtt{HI}}, D_i)$, where T_i is the minimum separation (or *period*) between successive job releases, $\chi_i \in \{\mathtt{LO}, \mathtt{HI}\}$ is the *criticality level*, $C_i^{\mathtt{LO}}$ is the LO WCET (*worst-case execution time*), $C_i^{\mathtt{HI}}$ is the HI WCET, and D_i is the *relative deadline*. A task τ_i is said to be a LO and HI task, if χ_i is LO and HI, respectively; let $\tau^{\mathtt{LO}}$ and $\tau^{\mathtt{HI}}$ denote a set of LO and HI tasks in τ, respectively. We assume that $C_i^{\mathtt{LO}} \leq C_i^{\mathtt{HI}}$ for every HI task and $C_i^{\mathtt{LO}} = C_i^{\mathtt{HI}}$ for every LO task. We target implicit- and constrained-deadline task systems, respectively, in which $D_i = T_i$ and $D_i \leq T_i$ hold for every $\tau_i \in \tau$.

MC Jobs and Scenarios. Task τ_i generates a potentially infinite sequence of jobs: $J_i^1, J_i^2, J_i^3, \ldots$. The qth job of τ_i (denoted by J_i^q) is characterized by two parameters: $J_i^q = (r_i^q, \gamma_i^q)$, where r_i^q is the *release time* of the job, and $\gamma_i^q \in (0, C_i^{\mathtt{HI}}]$ is the *execution requirement* of the job; J_i^q has completed its execution if it executes for γ_i^q. The *absolute deadline* of job J_i^q is $d_i^q \overset{\text{def}}{=} r_i^q + D_i$, and we call $[r_i^q, d_i^q]$ the *execution window* of J_i^q. We target sporadic task systems, in which successive jobs are released at least T_i time units apart. A job of τ_i is said to be a LO and HI job, if χ_i is LO and HI, respectively. It is important to notice that neither the release times nor the execution requirements are known in advance. Let a *scenario* for a given task set τ mean a collection of release times and execution requirements of jobs of interest invoked by tasks in τ; there exist infinitely many scenarios for a given task set. In this paper, we consider an independent, sequential task model in which every task (and job) is independent of others, and every job cannot be executed in parallel.

System Behavior and Requirement. Job J_i^q is released at time r_i^q and needs to complete γ_i^q units of work before its absolute deadline of d_i^q. The value of γ_i^q is not known beforehand, but only becomes revealed by actually executing the job until it signals its completion. The values of $\{\gamma_i^q\}$ for a given scenario of a given task set τ define system behavior and its corresponding real-time requirements as follows.

- As long as no job executes for more than its LO WCET for all jobs, the system is regarded as exhibiting the LO *behavior*, and all jobs are required to be completed before their deadlines.
- If any HI job does not signal its completion after executing for its LO WCET at some time instant t^* referred to as *mode change*, the system is regarded as exhibiting the HI *behavior*, and only HI jobs are required to be completed before their deadlines after the mode change; every LO job whose deadline is later than t^* can be discarded in the HI system behavior.[1]

MC-Feasibility. Based on the above two real-time requirements, we define MC-feasibility as follows.

Definition 1 (MC-feasible). *A scenario is said to be* feasible *if there exists a schedule that satisfies i) every job J_i^q receives execution time γ_i^q during its execution window $[r_i^q, d_i^q]$ when the system exhibits the* LO *behavior and ii) every* HI *job J_j^p receives execution time γ_j^p during its execution window $[r_j^p, d_j^p]$ when the system exhibits the* HI *behavior. A task set τ is said to be* MC-feasible *if* every *scenario is feasible.*

According to Definition 1, a task set τ is said to be *MC-infeasible* if there exists *at least one* scenario that is infeasible (i.e., not feasible). Note that determining MC-feasibility for collections of independent dual-criticality periodic and sporadic tasks is known to be NP-hard in the strong sense [16].

Assumption and Notation. We assume a quantum-based time; let one time unit a quantum length without loss of generality. We also assume that if the system has switched to high-criticality behavior, it will never switch back to low-criticality. Some recent studies have considered returning the system to low-criticality mode (see[4] for a survey), but this is not relevant to MC-feasibility because it does not change the definition of MC-feasibility. Let LHS and RHS denote left-hand-side and right-hand-side, respectively.

3 Challenges

In this section, we consider the following problem: what are unique issues of developing necessary feasibility tests for MC task systems (that do not matter for SC task systems)? To answer the question, we first present our own interpretation of the existing necessary feasibility test for SC task systems. We next present trivial existing results for necessary feasibility of MC task systems. Based on the background, we observe characteristics specialized for MC task systems,

1. Since we have a single computing platform (i.e., either uniprocessor or multiprocessor), the mode change to the HI system behavior is reported and handled to the scheduler, which discards every LO job.

and identify challenges for developing necessary feasibility tests for MC task systems.

3.1 Existing Necessary Feasibility Test for SC Task Systems

In this subsection, we offer our own *interpretation* of the existing necessary feasibility test for SC task systems on a uniprocessor platform [17] and on a multiprocessor platform [8].

A typical way to develop a necessary feasibility test for SC task systems is to focus on a scenario (associated with a given interval of interest, each task's job release pattern, and each job's execution requirement) and to compare the sum of every job's *minimum* execution requirement that should be performed in the interval of interest to avoid its deadline miss (called *demand*), with the time duration in which the computing platform allows jobs to execute within the interval (called *supply*). If the demand is larger than the supply, at least one job in the scenario inevitably misses its deadline, yielding infeasibility of the task set that invokes the scenario. While the demand depends on the scenario's interval of interest, each task's job release pattern and each job's execution time requirement, the existing study for SC task systems focuses on the following scenario S1, S2 and S3′ with given interval length $t^{\text{end}} > 0$.

S1. Target $[0, t^{\text{end}}]$ as an interval of interest, for given interval length $t^{\text{end}} > 0$.

S2. Generate jobs according to the *synchronous* periodic job release pattern from $t = 0$ as follows. The first job of every task is released at 0, and the following jobs of every task are released strictly periodically until each job's absolute deadline is no later than t^{end}.

S3′. Determine the execution requirement of every job as its WCET.

For the scenario of S1, S2 and S3′ with given $t^{\text{end}} > 0$, the existing study calculates demand and supply in the interval of interest, and deems the scenario (and therefore the corresponding task set) infeasible if the demand is strictly larger than the supply. While it is straightforward that the supply under S1 with given $t^{\text{end}} > 0$ amounts to t^{end}, the demand of a SC task τ_i with its WCET of C_i under the scenario of S1, S2 and S3′ with given $t^{\text{end}} > 0$ is calculated by $\texttt{DBF}_i(t^{\text{end}})$ where

$$\texttt{DBF}_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i, \tag{1}$$

which yields the following lemma.

Lemma 1. *A SC task set τ is infeasible on a m-processor platform (where $m \geq 1$), if Eq. (2) is violated for given $t^{\text{end}} > 0$*

$$\sum_{\tau_i \in \tau} \texttt{DBF}_i(t^{\text{end}}) \leq m \cdot t^{\text{end}}. \tag{2}$$

The lemma comes from [17] for a uniprocessor platform, and [8] for a multiprocessor platform. Note that while the LHS of Eq. (2) is originally $\texttt{FFDBF}_i(t^{\text{end}})$ for a multiprocessor platform in [8], which is slightly larger than $\texttt{DBF}_i(t^{\text{end}})$ only for some t^{end}, we use $\texttt{DBF}_i(t^{\text{end}})$ for consistency with the uniprocessor. It has been known that checking Eq. (2) for all $t^{\text{end}} > 0$ is equivalent to checking Eq. (2) for only some of $t^{\text{end}} > 0$, which are $\{0 < t^{\text{end}} < t^{\max} \mid (t^{\text{end}} - D_i) \bmod T_i = 0 \text{ for any } \tau_i \in \tau\}$ where $t^{\max}$ depends on $\sum_{\tau_i \in \tau} C_i/T_i$ [17].

By Lemma 1, it is possible to find an infeasible task set (if Eq. (2) is violated for at least one $t^{\text{end}} > 0$), meaning that a necessary feasibility test is successfully derived.

The next issue is how to systematically check the lemma for the scenario of S1, S2 and S3′ with as many $t^{\text{end}} > 0$ as possible with reasonable time-complexity. The existing study derives an upper bound of t^{end} such that checking Lemma 1 for the scenario of S1, S2 and S3′ with every $t^{\text{end}} > 0$ less than the upper bound is equivalent to checking that with every $t^{\text{end}} > 0$ without the upper bound (up to infinity). Using the upper-bound, the study develops the following collective necessary feasibility test as follows.

- A SC task set τ is infeasible, if there exists at least one $t^{\text{end}} > 0$ that violates Eq. (2) while we repeat to check Lemma 1 with every $t^{\text{end}} > 0$ less than the upper bound.

Note that it has been proven that the collective necessary feasibility test exhibits pseudo-polynomial time-complexity in the task parameters. Also, note that the test is known to be a necessary *and* sufficient feasibility test for SC task systems on a uniprocessor platform [17].

One may misunderstand that the collective necessary feasibility test *should check* Lemma 1 for *every* $t^{\text{end}} > 0$ less the upper bound. Different from the corresponding sufficient feasibility test, the necessary feasibility test can check Lemma 1 for any number of candidates for $t^{\text{end}} > 0$, which can affect capability in finding infeasible task sets, but cannot compromise the correctness of whether task sets deemed infeasible by the test is actually infeasible.

3.2 Trivial Results for Necessary Feasibility of MC Task Systems

Considering the most prominent difference between SC and MC task systems is existence of the mode change and consequences thereof, we may classify scenarios of MC task systems, based on relationship between the mode change instant and the interval of interest. That is, the mode change instant t^* exists after, before, and within the interval of interest, denoted by Cases A, B, and C, respectively.

Since all jobs in the interval of interest *exclusively* experiences the $\texttt{LO}$ and $\texttt{HI}$ system behavior in Cases A and B, respectively, each scenario of Case A and B can be equivalent to a scenario of a SC task system. The following two lemmas correspond Lemma 1 for Cases A and B, respectively (similar conditions were presented in [12], [13], [14] with a different form).

Lemma 2. *A MC task set τ is infeasible on a m-processor platform (where $m \geq 1$), if Eq. (3) is violated for given $t^{\text{end}} > 0$:*

$$\sum_{\tau_i \in \tau} \texttt{DBF}_i^{\texttt{LO}}(t^{\text{end}}) \leq m \cdot t^{\text{end}}, \\ \text{where } \texttt{DBF}_i^{\texttt{LO}}(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i^{\texttt{LO}}. \tag{3}$$

Proof. Consider the scenario of S1, S2 and S3′ with given $t^{\text{end}} > 0$ when the mode change occurs at $t = \infty$. Then, the scenario is the same as Lemma 1 with replacing C_i with $C_i^{\texttt{LO}}$ for every $\tau_i \in \tau$. □

Lemma 3. *A MC task set τ is infeasible on a m-processor platform (where $m \geq 1$), if Eq. (4) is violated for given $t^{\text{end}} > 0$:*

$$\sum_{\tau_i \in \tau^{\text{HI}}} \texttt{DBF}_i^{\text{HI}}(t^{\text{end}}) \leq m \cdot t^{\text{end}},$$
$$\text{where } \texttt{DBF}_i^{\text{HI}}(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i^{\text{HI}}. \tag{4}$$

Proof. Consider the scenario of S1, S2 and S3′ with given $t^{\text{end}} > 0$ when the mode change occurs at $t = -\infty$. Since there is no demand of LO tasks in $[0, t^{\text{end}}]$, the scenario is the same as Lemma 1 with replacing C_i with C_i^{HI} for every $\tau_i \in \tau^{\text{HI}}$ and C_i with 0 for every $\tau_i \in \tau^{\text{LO}}$. □

To find as many infeasible task sets as possible, the collective necessary feasibility test can be developed by applying Lemmas 2 and 3 for every $t^{\text{end}} > 0$ less than its upper-bound (derived by the same technique for SC task systems).[2]

Although Lemmas 2 and 3 successfully address Cases A and B, there is a still huge gap between the region covered by task sets proven feasible and that proven infeasible, as shown in Fig. 4 in Section 10; note that Lemmas 2 and 3 with every $t^{\text{end}} > 0$ prove infeasibility of task sets beyond the region X = 1.0 by Y = 1.0 (colored by grey) in the figure. While tight necessary conditions for feasibility require to consider Case C effectively, deriving the conditions from Case C entails many challenges to be discussed in the next subsection.

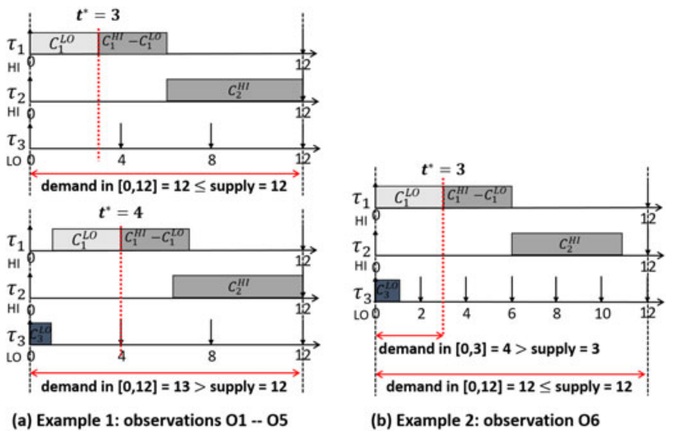
Note that although existing demand-based schedulability tests [18], [19], [20] considered Case C using the notion of demand, all of them are unable to detect infeasible task sets because they are designed to find feasible task sets.

3.3 Challenges for Developing Necessary Feasibility Tests for MC Task Systems

In this subsection, we identify challenges specialized for MC tasks systems to develop necessary feasibility tests, by considering Case C in which the mode change instant t^* exists in the middle of the interval of interest.

We may observe two characteristics of SC task systems to derive necessary conditions for feasibility in Section 3.1. First, the demand in the interval of interest is fixed with the scenario of S1–S3′ with given $t^{\text{end}} > 0$. Second, if the demand is larger than the supply, the scenario (and therefore the corresponding task set) is actually infeasible. The two characteristics, however, do not hold for MC task systems, due to existence of the mode change. We present the following example showing characteristics of MC task systems that affect derivation of necessary conditions for feasibility. Note that the characteristics to be explained are mostly derived from existence of the mode change, which switches (i) a set of jobs whose execution requirement should be guaranteed (i.e., from HI and LO jobs, to HI jobs only) and (ii) HI jobs' amount of execution requirement that should be guaranteed (i.e., from no more than the LO WCET, to more than the LO WCET).

2. The collective necessary feasibility test is reduced to checking violation of either $\sum_{\tau_i \in \tau} C_i^{\text{LO}}/T_i \leq m$ or $\sum_{\tau_i \in \tau^{\text{HI}}} C_i^{\text{HI}}/T_i \leq m$, for implicit-deadline task systems [12], [13], [14].



t* = 3
t* = 4
demand in [0,12] = 12 ≤ supply = 12
demand in [0,12] = 13 > supply = 12
demand in [0,3] = 4 > supply = 3
demand in [0,12] = 12 ≤ supply = 12
(a) Example 1: observations O1 -- O5
(b) Example 2: observation O6

Fig. 1. (a) Illustration of Example 1 for Observations O1–O5, and (b) Illustration of Example 2 for Observation O6.

Example 1. Consider a task set τ with the following three tasks: $\tau_1(T_1 = 12, \chi_1 = \text{HI}, C_1^{\text{LO}} = 3, C_1^{\text{HI}} = 6, D_1 = 12)$, $\tau_2 = \tau_1$, and $\tau_3 = (4, \text{LO}, 1, 1, 4)$. Consider the following scenario as shown in Fig. 1a: (i) the interval of interest is [0,12]; (ii) the synchronous job release pattern from $t = 0$ is applied to all tasks in [0,12], meaning that the release time and deadline of J_1^1 and J_2^1 are 0 and 12, respectively, and those of J_3^1, J_3^2, and J_3^3 are 0 and 4, 4 and 8, and 8 and 12, respectively; and (iii) the execution requirement of every LO and HI job is its LO and HI WCET, respectively. We now consider two cases with different choices of the mode change instant: $t^* = 3$ and 4. In both cases, the following properties hold: (a) one of J_1^1 or J_2^1 should execute for exactly 3 time units (i.e., its LO WCET) in $[0, t^*]$ (i.e., before the mode change) to trigger the mode change, and therefore (b) the job that triggers the mode change should execute for exactly 3 time units (i.e., its HI WCET minus LO WCET) in $[t^*, 12]$ (i.e., after the mode change).

Suppose the mode change instant occurs at $t^* = 3$. The demand of jobs of τ_3 in [0,12] is 0 because no job of τ_3 has its deadline at or before the mode change instant. Also, one of J_1^1 and J_2^1 that does not trigger the mode change cannot execute before the mode change (i.e., in [0,3]) because of (a), and hence the job should execute for 6 time units (i.e., its HI WCET) after the mode change (i.e., in [3,12]). In this case, since there is no demand of τ_3, the total demand in [0,12] is $C_1^{\text{HI}} + C_2^{\text{HI}} = 12$, which is no larger than the supply in [0,12].

Suppose the mode change instant occurs at $t^* = 4$. Then, the demand of jobs of τ_3 in [0,12] is 1 because J_3^1 has its deadline no later than the mode change instant (while other jobs of τ_3 do not). Also, one of J_1^1 and J_2^1 that does not trigger the mode change executes for at most 1 time unit before the mode change (i.e., in [0,4]) because of (a). Considering the sum of execution of the job in [0,4] and that of [4,12] (therefore the sum of the demand in both intervals) should be 6 (its HI WCET), the job should execute for 5 or 6 time units after the mode change (i.e., in [4,12]). In this case, since the demand of τ_3 is 1, the total demand in [0,12] is $C_3^{\text{LO}} + C_1^{\text{HI}} + C_2^{\text{HI}} = 13$, which is larger than the supply in [0,12].

In fact, the scenario is feasible if J_1^1 and J_2^1 are executed in [0,6] and [6,12], respectively. This is because, this schedule yields the mode change at $t^* = 3$, which is before the deadline of all LO jobs (i.e., $t = 4$, 8 or 12); therefore, all LO jobs do not have any demand in [0,12].

We summarize the following observations from Example 1.

O1. The contribution of each LO job to the demand varies with the mode change instant. For example, the demand of LO jobs amounts to 0 and 1 with $t^* = 3$ and $t^* = 4$, respectively.

O2. The constraints for the amount of the contribution of each HI job to the demand varies with the mode change instant. For example, one of J_1^1 and J_2^1 that does not trigger the mode change executes for 0 and at most 1 time unit before the mode change, respectively in case of $t^* = 3$ and $t^* = 4$.

O3. By O1 and O2, it is impossible (or at least very difficult) to calculate demand without specifying the mode change instant.

O4. The demand strictly larger than the supply in a case does not necessarily yield infeasibility of the scenario. For example, the scenario is feasible although the demand is larger than supply with $t^* = 4$.

O5. Without a concrete schedule determined by the target scheduling algorithm, we may not calculate the exact values for the demand of a HI job in one sub-interval and that in another sub-interval; however, there exists a relationship between the demand in those intervals. For example, the sum of the demand of J_1^1 (or J_2^1) in $[0, t^*]$ and that in $[t^*, 12]$ is 6.

We now present another example that shows necessity of investigating sub-intervals of the interval of interest for higher capability in finding infeasible task sets.

Example 2. Consider a task set τ with the following three tasks: $\tau_1(T_1 = 12, \chi_1 = \texttt{HI}, C_1^{\texttt{LO}} = 3, C_1^{\texttt{HI}} = 6, D_1 = 12)$, $\tau_2 = (12, \texttt{HI}, 3, 5, 12)$, and $\tau_3 = (2, \texttt{LO}, 1, 1, 2)$. Consider the scenario same as Example 1.

As shown in Fig. 1b, we now present a case with the mode change instant $t^* = 3$, in which one of J_1^1 or J_2^1 should execute for exactly 3 time units (i.e., its LO WCET) in [0,3]. The demand of jobs of τ_3 in [0,3] is 1 (i.e., $C_3^{\texttt{LO}}$), because J_3^1 is the only job whose deadline is no later than $t = 3$. Therefore, the total demand[3] in [0,3] is $3 + 1 = 4$, which is larger than the supply in [0,3]; this judges that the mode change cannot occur at $t^* = 3$ without violating i) of Definition 1. However, the same cannot be judged if we focus on the entire interval of [0,12]; this is because the total demand in [0,12] is $C_3^{\texttt{LO}} + C_1^{\texttt{HI}} + C_2^{\texttt{HI}} = 12$.

In fact (after investigating all possible schedules), the scenario is deemed infeasible because there is no schedule that satisfies i) and ii) in Definition 1. Note that both Lemmas 2 and 3 cannot deem the task set infeasible, while our proposed necessary feasibility test to be developed in this paper (i.e., Theorem 2) can.

From Example 2, we have the following observation.

3. In this case, the notion of demand implies the sum of every job's minimum execution requirement that should be performed in the interval of interest not only to satisfy MC-feasility in Definition 1, but also to trigger the mode change at $t^* = 3$.

O6. The inequality of the demand larger than the supply holds in a subset of the interval of interest, while the same does not hold in the entire interval of interest. In Example 2, the inequality holds in [0,3], but does not hold in [0,12].

Considering the unique observations O1–O6, we need to address the following challenges to develop necessary feasibility tests for MC task systems.

I1. For a given scenario, how can we characterize and calculate the demand in an interval that changes depending on the mode change instant? (from O1–O3)

I2. For a given scenario, what is the meaning of the demand larger than the supply in an interval when the mode change instant is given? (from O4)

I3. For a given scenario, how can we derive infeasibility of the scenario from the answer of I2 without assuming the mode change instant is given? (from O3–O4)

I4. For a given scenario, what are good choices of sub-intervals to be targeted for I1–I3? (from O6) How can we utilize the relationship between demand of those sub-intervals? (from O5)

Since I1–I4 need a given scenario, we have the following challenge.

I5. What are additional scenario components for MC task systems other than S1, S2 and S3′, which make it possible to address I1–I4? In particular, how can we make the components specify the mode change instant without targeting a scheduling algorithm? (from O3)

In addition, we have the following challenge for higher capability in finding infeasible task sets.

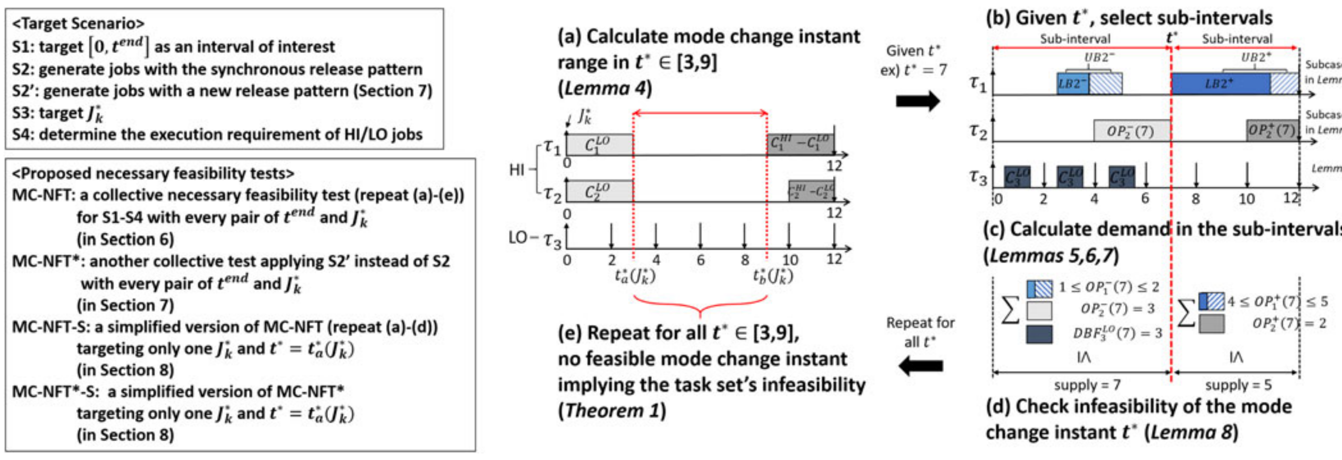
I6. Once we develop a necessary feasibility test for a given scenario by addressing I1–I5, how can we efficiently check the test with as many scenarios as possible with reasonable time-complexity?

4 Approach Overview

In this section, we first define new scenario components for MC task systems to consider the unique characteristics thereof (C1 and C2 in Section 1), addressing I5. Building upon the target scenario tailored to MC task systems, we describe an overview of our approach to develop necessary feasibility tests for MC task systems by addressing I1–I4, I6 and other critical issues, which is illustrated in Fig. 2.

4.1 New Scenario Components for MC Task Systems

Among the scenario of S1, S2 and S3′ for SC task systems, S1 and S2 can be applied as they are, while S3′ should be adapted because each HI job has LO and HI WCETs. Considering the execution requirement of each HI job is relevant to the mode change instant, we need to adapt S3′ for HI jobs so as to (i) determine each HI job's execution requirement as either its LO or HI WCET, and (ii) specify


<Target Scenario>
S1: target $[0, t^{end}]$ as an interval of interest
S2: generate jobs with the synchronous release pattern
S2': generate jobs with a new release pattern (Section 7)
S3: target J_k^*
S4: determine the execution requirement of HI/LO jobs

<Proposed necessary feasibility tests>
MC-NFT: a collective necessary feasibility test (repeat (a)-(e)) for S1-S4 with every pair of t^{end} and J_k^* (in Section 6)
MC-NFT*: another collective test applying S2' instead of S2 with every pair of t^{end} and J_k^* (in Section 7)
MC-NFT-S: a simplified version of MC-NFT (repeat (a)-(d)) targeting only one J_k^* and $t^* = t_a^*(J_k^*)$ (in Section 8)
MC-NFT*-S: a simplified version of MC-NFT* targeting only one J_k^* and $t^* = t_a^*(J_k^*)$ (in Section 8)

(a) Calculate mode change instant range in $t^* \in [3,9]$ (*Lemma 4*)

(b) Given t^*, select sub-intervals

(c) Calculate demand in the sub-intervals (*Lemmas 5,6,7*)

(d) Check infeasibility of the mode change instant t^* (*Lemma 8*)

(e) Repeat for all $t^* \in [3,9]$, no feasible mode change instant implying the task set's infeasibility (*Theorem 1*)

Fig. 2. Overview of our proposed necessary feasibility tests for a task set described in Examples 2 and 3.

the mode change instant without the target scheduling algorithm.

The main idea to address (i) and (ii) is to include a scenario component of J_k^*, which indicates the job with the earliest release time among all HI jobs whose execution requirement is strictly larger than its LO WCET. By the proposed definition, we can specify the range of the mode change instant without targeting a scheduling algorithm, as stated in the following lemma.

Lemma 4. *J_k^* should observe a mode change (triggered by itself or another job) within its execution window; otherwise, the job cannot execute for more than its LO WCET, yielding its deadline miss.*

Proof. Suppose that J_k^* does not observe a mode change within its execution window. We consider two cases: a mode change occurs (i) before and (ii) after the execution window of J_k^*.

In case of (i), there should be another job that observes a mode change before J_k^*'s release time, which contradicts the "the earliest release time" phrase in the definition of J_k^*. In case of (ii), J_k^* cannot have its execution requirement larger than its LO WCET, which contradicts the definition of J_k^* (i.e., the "whose execution requirement is strictly larger than its LO WCET" part). □

Therefore, by targeting given J_k^*, we can restrict the range of the mode change instant for each scenario, to the execution window of J_k^* or a part thereof. Also, by targeting given J_k^*, we can determine each HI job's execution requirement based on its release time. Then, we consider S3 and S4 by adapting/detailing S3′ as follows.

S3. Target a given job J_k^* among all jobs generated by S1 and S2 with given $t^{\text{end}} > 0$, where J_k^* is the job which has the earliest release time among all HI jobs whose execution requirement is strictly larger than LO WCET.

S4. Determine the execution requirement of every LO job as its LO WCET; determine the execution requirement of every HI job whose release time is earlier and no earlier than r_k^* (i.e., J_k^*'s release time) as its LO and HI WCET, respectively.

In the rest of this paper, we target the scenario of S1 and S2 defined in Section 3.1, and S3 and S4 defined in this section, with given $t^{\text{end}} > 0$ and J_k^*. Then, the following lemma calculates a more refined range of the mode change instant t^* associated with given J_k^* as illustrated in Fig. 2a.

Lemma 5. *The scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^* is feasible, only if the mode change occurs within $[t_a^*(J_k^*), t_b^*(J_k^*)]$, where $t_a^*(J_k^*)$ and $t_b^*(J_k^*)$ are respectively the earliest $(r_i^p + C_i^{\text{LO}})$ and the earliest $(d_i^p - C_i^{\text{HI}} + C_i^{\text{LO}})$ among every HI job J_i^p whose release time is no earlier than r_k^* (i.e., the release time of J_k^*).*

Proof. Recall S4; every HI job J_i^p whose release time is no earlier than r_k^* has the execution requirement for its HI WCET.

Suppose that the mode change occurs at $t^* < t_a^*(J_k^*)$. By S4 and the definition of $t_a^*(J_k^*)$, it is impossible for every HI job whose execution requirement equals to its HI WCET to perform its executing for as much as its LO WCET before $t_a^*(J_k^*)$. This contradicts the supposition.

Suppose that a mode change occurs at $t^* > t_b^*(J_k^*)$. By S4 and the definition of $t_b^*(J_k^*)$, there exists at least one job J_i^p whose execution requirement equals to its HI WCET, but whose actual execution time performed until $(d_i^p - C_i^{\text{HI}} + C_i^{\text{LO}})$ is less than C_i^{LO}. If there exists no such job, the mode change occurs no later than $(d_i^p - C_i^{\text{HI}} + C_i^{\text{LO}})$. The former contradicts feasibility, and the latter contradicts supposition. □

4.2 Approach Overview

We summarize essential parts to develop a necessary feasibility test for MC task systems (i.e., how to address I1–I4). Then, we provide an overview of our proposed necessary feasibility tests to utilize the essential parts.

Addressing I4. Under the newly defined target scenario of S1–S4, we determine sub-intervals to be investigated for demand-supply comparison respectively by dividing the interval of interest based on the mode change instant (Fig. 2b). As shown in O6, demand-supply comparison with the interval split is more effective in identifying infeasibile task sets, compared to that without the interval split.

Addressing I1. We calculate the demand of a LO/HI task in the target sub-intervals when the mode change instant is given by considering the relationship between the mode change instant and each job's execution window (as shown in Fig. 2c, and to be stated in Lemmas 6 and 7). For jobs of

HI tasks whose execution window overlaps with the mode change instant, we derive the lower and upper bounds of their demand contribution in the target sub-intervals, respectively (to be stated in Lemma 8).

Addressing I2. Comparing the total demand with the total supply in the target sub-intervals, we derive a necessary condition for feasibility of the mode change instant t^* without missing any job deadline (as shown in Fig. 2d, and to be stated in Lemma 9).

Addressing I3. By associating the necessary conditions with every instant in the range of the mode change instant, we develop a necessary feasibility test for a target scenario (as shown in Fig. 2e, and to be stated in Theorem 1). If there exists no feasible mode change instant, it is impossible for the mode change instance to exist without any job deadline miss, which yields infeasibility of the scenario.

Addressing Other Critical Issues to Complete the Proposed Necessary Feasibility Tests. We develop a collective necessary feasibility test for MC task systems to check the proposed test for a single scenario with as many scenarios as possible with reasonable (pseudo-polynomial) time-complexity (to be presented in Section 6). In addition to the synchronous job release pattern (i.e., S2), we investigate another job release pattern favorable to finding more infeasible MC task sets (to be stated in S2′ in Section 7) and develop an improved version of the collective necessary feasibility test (to be presented in Section 7). Furthermore, we also develop simplified versions of the collective necessary feasibility test and the improved one such that they have the same complexity as in the SC task system case (to be presented in Section 8). Finally, we discuss the relationship among the two collective necessary feasibility tests and their simplified versions, in terms of capability in finding infeasible task sets and time-complexity (to be presented in Section 9).

5 Necessary Feasibility Condition for a Scenario

In this section, we focus on a single scenario associated with S1–S4, and derive a necessary feasibility condition for the scenario by addressing I1–I4. The condition can operate as a necessary feasibility test for MC task systems, because infeasibility of a task set can be verified by a necessary feasibility condition for even a single scenario of the task set. Once this section finishes deriving the condition for a single scenario, Section 6 will present how to check the condition for as many as scenarios as possible within a short time, which enables to find more infeasible task sets.

We first explain how to choose sub-intervals to be investigated (i.e., addressing the first part of I4), which should precede addressing I1–I3. As shown in O1 and O2 in Section 3.3, the mode change instant t^* determines not only whether each LO job contributes to the demand or not, but also how much execution each HI job can contribute to the demand before and after the mode change. Since the mode change instant t^* is a criterion that determines the demand (or its constraints) of LO and HI jobs, we not only target a situation with given t^*, but also divide the interval of interest into two sub-intervals based on the mode change instant t^* (i.e., $[0, t^*]$ and $[t^*, t^{\text{end}}]$), to be targeted for deriving necessary feasibility conditions as illustrated in Fig. 2b. As shown in O6, demand-supply comparison with the interval split yields higher capability in finding infeasible task sets, compared to that without the interval split.

Once we determine the target sub-intervals, we are ready to address I1, which is to calculate the demand in the target sub-intervals when the mode change instant t^* is given. One may wonder whether it is possible to use the techniques of demand calculation in existing demand-based schedulability tests for MC task systems [18], [19], [20]; this is impossible because they were designed for a target scheduling algorithm to derive sufficient feasibility conditions, while this paper derives necessary feasibility conditions meaning that no target scheduling algorithm is assumed.

We first investigate the demand of a LO task. Considering a LO job can contribute the demand only if the job's deadline is no later than t^*, the demand of jobs of a LO task in $[0, t^*]$ can be calculated (and that in $[t^*, t^{\text{end}}]$ is zero), as stated in the following lemma.

Lemma 6. *Target the scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^*, and target a single mode change instant t^* belonging to $[t_a^*(J_k^*), t_b^*(J_k^*)]$ (defined in Lemma 5). Then, the demand of jobs of a LO task $\tau_i \in \tau$ in $[0, t^*]$ and $[t^*, t^{\text{end}}]$ can be calculated as follows.*

- *The demand of jobs of $\tau_i \in \tau^{\mathtt{LO}}$ in $[0, t^*]$ is $\mathtt{DBF}_i^{\mathtt{LO}}(t^*)$.*
- *The demand of jobs of $\tau_i \in \tau^{\mathtt{LO}}$ τ_i in $[t^*, t^{\text{end}}]$ is 0.*

Proof. If the mode change occurs before the LO job's deadline, we do not need to execute the job, meaning that the demand for the job is zero. Therefore, the latter directly holds. Considering the definition of $\mathtt{DBF}_i^{\mathtt{LO}}(t)$ in Eq. (3), the former also holds. □

The next issue is to calculate the demand of a HI task in the target sub-intervals. Considering the relationship between the mode change instant t^* and each job's execution window, we may classify all jobs of a HI task $\tau_i \in \tau^{\mathtt{HI}}$ into three categories:

CG1. jobs J_i^p whose deadline is no later than t^* (i.e., $d_i^p \le t^*$),

CG2. at most one job J_i^p whose release time is earlier than t^* but whose deadline is later than t^* (i.e., $r_i^p < t^* < d_i^p$),

CG3. jobs J_i^p whose release time is no earlier than the mode change instant (i.e., $t^* \le r_i^p$).

While CG1 and CG3 exclusively contribute to the demand in $[0, t^*]$ and that in $[t^*, t^{\text{end}}]$, respectively, CG2 may contribute to both. Let $\mathtt{OP}_i^-(t^*)$ and $\mathtt{OP}_i^+(t^*)$ denote the demand of a job of $\tau_i \in \tau^{\mathtt{HI}}$ belonging to CG2 in $[0, t^*]$ and that in $[t^*, t^{\text{end}}]$, respectively, under the scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^*, and given $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$. Note that $\mathtt{OP}_i^-(t^*)$ and $\mathtt{OP}_i^+(t^*)$ are 0, if no job of $\tau_i \in \tau^{\mathtt{HI}}$ belongs to CG2.

Then, we can calculate/express the demand of jobs of a HI task in $[0, t^*]$ and that in $[t^*, t^{\text{end}}]$ by CG1, CG2, and CG3 separately, as stated in the following lemma.

Lemma 7. *Target the scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^*, and target a single mode change instant t^* belonging to*

$[t_a^*(J_k^*), t_b^*(J_k^*)]$ (defined in Lemma 5). Then, the demand of jobs of a HI *task τ_i in $[0, t^*]$ and $[t^*, t^{\text{end}}]$ can be calculated/expressed as follows.*

- *The demand of jobs of $\tau_i \in \tau^{\text{HI}}$ belonging to CG1 in $[0, t^*]$ amounts to $\text{DBF}_i^{\text{LO}}(t^*)$.*
- *The demand of jobs of $\tau_i \in \tau^{\text{HI}}$ belonging to CG1 in $[t^*, t^{\text{end}}]$ amounts to 0.*
- *The demand of at most one job of $\tau_i \in \tau^{\text{HI}}$ belonging to CG2 in $[0, t^*]$ amounts to $\text{OP}_i^-(t^*)$.*
- *The demand of at most one job of $\tau_i \in \tau^{\text{HI}}$ belonging to CG2 in $[t^*, t^{\text{end}}]$ amounts to $\text{OP}_i^+(t^*)$.*
- *The demand of jobs of $\tau_i \in \tau^{\text{HI}}$ belonging to CG3 in $[0, t^*]$ amounts to 0.*
- *The demand of jobs of $\tau_i \in \tau^{\text{HI}}$ belonging to CG3 in $[t^*, t^{\text{end}}]$ amounts to $\text{DBF}_i^{\text{HI}}(t^{\text{end}} - \lceil t^*/T_i \rceil \cdot T_i)$.*

Proof. Since every HI job of τ_i belonging to CG1 executes for its LO WCET, the first statement holds. The second and fifth statements hold by the definition of jobs in CG1 and CG3. The third and fourth statements are the definition of $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$. Among jobs of τ_i belonging to CG3, the earliest release time is $(\lceil t^*/T_i \rceil \cdot T_i)$; considering the definition of $\text{DBF}_i^{\text{HI}}(t)$ in Eq. (4), the sixth statement holds. □

While the first, second, fifth and sixth statements in Lemma 7 calculate exact values for the corresponding demand, the third and fourth statements do not. Considering the sum of $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$ equals to the execution requirement of the job of τ_i belonging to CG2 (addressing the second part of I4), we can derive the constraints of the contribution of $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$ to the demand in the target sub-intervals, as shown in the following example.

Example 3. Recall the task set and the scenario in Example 2 in Section 3.3, and target the mode change instant $t^* = 7$. We now explain the constraints of $\text{OP}_1^-(7)$ and $\text{OP}_1^+(7)$ for J_1^1. If J_1^1 triggers the mode change, the job should execute for exactly 3 time units (i.e., C_1^{LO}) in [0,7] and 3 time units (i.e., $C_1^{\text{HI}} - C_1^{\text{LO}}$) in [7,12], implying $\text{OP}_1^-(7) = 3$ and $\text{OP}_1^+(7) = 3$. We discuss the case where J_1^1 does not trigger the mode change, from now on, which is illustrated in Fig. 2c.

Let LB2^- and UB2^- denote lower and upper bounds for $\text{OP}_1^-(7)$, and LB2^+ and UB2^+ denote lower and upper bounds for $\text{OP}_1^+(7)$. Then, J_1^1's execution cannot be larger than its target sub-interval lengths, $\text{UB2}^- \leq 7$ and $\text{UB2}^+ \leq 5$. Also, for J_1^1 not to trigger the mode change, J_1^1's execution in [0,7] should be strictly less than its LO WCET, yielding $\text{UB2}^- = \min(7, C_1^{\text{LO}} - 1 = 2) = 2$. Also, J_1^1's execution in [7,12] should not be larger than its execution requirement, yielding $\text{UB2}^+ = \min(5, C_1^{\text{HI}} = 6) = 5$. Considering the sum of $\text{OP}_1^-(7)$ and $\text{OP}_1^+(7)$ equals to J_1^1's execution requirement, we can calculate $\text{LB2}^- = C_1^{\text{HI}} - \text{UB2}^+ = 6 - 5 = 1$ and $\text{LB2}^+ = C_1^{\text{HI}} - \text{UB2}^- = 6 - 2 = 4$.

Inspired by the example, the following lemma formalizes the constraints of the contribution of $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$ to the demand in the target sub-intervals.

Lemma 8. *Target the scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^*, and target a single mode change instant t^* belonging to $[t_a^*(J_k^*), t_b^*(J_k^*)]$ (defined in Lemma 5). Consider a job of $\tau_i \in \tau^{\text{HI}}$ potentially belonging to CG2, J_i^q, whose release time and deadline are $r_i^q = \lfloor \frac{t^*}{T_i} \rfloor \cdot T_i$ and $d_i^q = r_i^q + D_i$, respectively. Considering J_i^q, we have three cases for calculating $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$. In Case 1, there is no job of τ_i in CG2; J_i^q does not belong to CG2. In Cases 2 and 3, J_i^q is the job of τ_i that belongs to CG2, but the job's execution requirement amounts to its* LO *and* HI *WCET, respectively. Case 3 consists of Subcases 3A and 3B, in which J_i^q does trigger and does not trigger the mode change, respectively. Then, $\text{OP}_i^-(t^*)$ and $\text{OP}_i^+(t^*)$ for every $\tau_i \in \tau^{\text{HI}}$ should satisfy the following constraints.*

- *Case 1: If $r_i^q = t^*$, $d_i^q \leq t^*$ or $d_i^q > t^{\text{end}}$,*
  $\text{OP}_i^-(t^*) = \text{OP}_i^+(t^*) = 0$ holds.
- *Case 2: Otherwise, if $r_i^q < r_k^*$ (recall r_k^* is J_k^*'s release time),*
  (i) $\text{LB}^- \leq \text{OP}_i^-(t^*) \leq \text{UB}^-$,
  (ii) $\text{LB}^+ \leq \text{OP}_i^+(t^*) \leq \text{UB}^+$, and
  (iii) $\text{OP}_i^+(t^*) + \text{OP}_i^-(t^*) = C_i^{\text{LO}}$ hold, where
  $\text{UB}^- = \min(t^* - r_i^q, C_i^{\text{LO}})$, $\text{UB}^+ = \min(d_i^q - t^*, C_i^{\text{LO}})$,
  $\text{LB}^- = C_i^{\text{LO}} - \text{UB}^+$ and $\text{LB}^+ = C_i^{\text{LO}} - \text{UB}^-$.
- *Case 3: Otherwise (i.e., $r_i^q \geq r_k^*$), (Subcase 3A) if $t^* - r_i^q \geq C_i^{\text{LO}}$ and $d_i^q - t^* \geq C_i^{\text{HI}} - C_i^{\text{LO}}$ hold and the job of τ_i triggers the mode change,*
  (i) $\text{OP}_i^-(t^*) = C_i^{\text{LO}}$ and
  (ii) $\text{OP}_i^+(t^*) = C_i^{\text{HI}} - C_i^{\text{LO}}$ hold; (Subcase 3B) otherwise (i.e., the job of τ_i does not trigger the mode change),
  (i) $\text{LB2}^- \leq \text{OP}_i^-(t^*) \leq \text{UB2}^-$,
  (ii) $\text{LB2}^+ \leq \text{OP}_i^+(t^*) \leq \text{UB2}^+$, and
  (iii) $\text{OP}_i^+(t^*) + \text{OP}_i^-(t^*) = C_i^{\text{HI}}$ hold, where
  $\text{UB2}^- = \min(t^* - r_i^q, C_i^{\text{LO}} - 1)$, $\text{UB2}^+ = \min(d_i^q - t^*, C_i^{\text{HI}})$,
  $\text{LB2}^- = C_i^{\text{HI}} - \text{UB2}^+$ and $\text{LB2}^+ = C_i^{\text{HI}} - \text{UB2}^-$.

Proof. (Case 1) If $r_i^q = t^*$ (or $d_i^q \leq t^*$), J_i^q belongs to CG3 (or CG1) and there is no job of τ_i in CG2. Also, if $d_i^q > t^{\text{end}}$, the scenario of S2 does not generate J_i^q.

(Cases 2 and 3) The cases can be proved by utilizing $\text{OP}_i^-(t^*) + \text{OP}_i^+(t^*) = C_i^{\text{LO}}$ and $\text{OP}_i^-(t^*) + \text{OP}_i^+(t^*) = C_i^{\text{HI}}$, respectively. See the details in the supplement, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3118610. □

Combining Lemmas 7 and 8, we can calculate the demand of all HI tasks in $[0, t^*]$ and $[t^*, t^{\text{end}}]$, by adding the demand of jobs of each HI task in CG1, CG2 and CG3. By combining the demand of all HI tasks and that of all LO tasks in Lemma 6, we can calculate the total demand in $[0, t^*]$ and $[t^*, t^{\text{end}}]$. If the total demand is larger than the total supply in $[0, t^*]$ or the same holds in $[t^*, t^{\text{end}}]$, it is impossible for the mode change to occur at t^* without any job deadline miss, which addresses I2. In other words, comparing the total demand with the total supply in $[0, t^*]$ and $[t^*, t^{\text{end}}]$, we can judge the feasibility of the mode change at t^* without missing any job deadline, as stated in the following lemma, which is illustrated in Fig. 2d.

Lemma 9. *Target the scenario of S1–S4 with given $t^{\text{end}} > 0$ and J_k^*, and target a single mode change instant t^* belonging to*

$[t^*_a(J^*_k), t^*_b(J^*_k)]$ (defined in Lemma 5). The mode change instant t^* is infeasible without any job deadline miss, if it is impossible to satisfy both Eqs. (5) and (6) subject to Lemma 8 and the following constraint.

- *There exists at least one $\tau_j \in \tau^{\mathtt{HI}}$ belonging to Subcase 3A of Lemma 8.*

$$\sum_{\tau_i \in \tau^{\mathtt{LO}}} \mathtt{DBF}^{\mathtt{LO}}_i(t^*) + \sum_{\tau_i \in \tau^{\mathtt{HI}}} \Big(\mathtt{DBF}^{\mathtt{LO}}_i(t^*) + \mathtt{OP}^-_i(t^*) \Big) \le m \cdot t^*. \tag{5}$$

$$\sum_{\tau_i \in \tau^{\mathtt{HI}}} \Big(\mathtt{DBF}^{\mathtt{HI}}_i(t^{\mathtt{end}} - \Big\lceil \frac{t^*}{T_i} \Big\rceil \cdot T_i) + \mathtt{OP}^+_i(t^*) \Big) \le m \cdot (t^{\mathtt{end}} - t^*). \tag{6}$$

Proof. By Lemma 8 and the fact that the mode change should be triggered by at least one (but at most m) jobs, the two constraints (i.e., the "subject to" part) hold.

By Lemmas 6 and 7, the LHS of Eq. (5) and that of Eq. (6) calculate the total demand of τ in $[0, t^*]$ and $[t^*, t^{\mathtt{end}}]$, respectively under the scenario of S1–S4 with given $t^{\mathtt{end}} > 0$ and J^*_k. Considering the supply amounts to the interval length, violating either Eqs. (5) or (6) implies that the mode change cannot occur at t^* or at least one job misses its deadline. □

Once we repeat Lemma 9 with every $t^* \in [t^*_a(J^*_k), t^*_b(J^*_k)]$, we know whether there exists at least one mode change instant that does not yield any job deadline miss. If it does not exist, it is impossible for the mode change instance to exist without any job deadline miss, which yields infeasibility of the scenario by Lemma 5. This addresses I3, and is recorded by the following theorem (as also illustrated in Fig. 2e).

Theorem 1. *A MC task set τ is infeasible, if every mode change instant $t^* \in [t^*_a(J^*_k), t^*_b(J^*_k)]$ associated with the scenario of S1–S4 with given $t^{\mathtt{end}} > 0$ and J^*_k makes the "if" statement in Lemma 9 true.*

Proof. By Lemma 9 and the range of $t^* \in [t^*_a(J^*_k), t^*_b(J^*_k)]$, the impossibility to satisfy both Eqs. (5) and (6) subject to Lemma 8 and the constraint in Lemma 9 implies that the scenario of S1–S4 with given $t^{\mathtt{end}} > 0$ and J^*_k yields no existence of the mode change instant without any job deadline miss. According to Lemma 5, existence of the mode change instant in $[t^*_a(J^*_k), t^*_b(J^*_k)]$ is a necessary feasibility condition for the scenario. Therefore, the theorem holds. □

6 Collective Necessary Feasibility Test

In this section, we address I6 in Section 3.3—how to check the proposed test with as many scenarios as possible with less time-complexity. To this end, we develop (i) how to test Lemma 9 (and therefore Theorem 1) with low time-complexity and (ii) how to apply the necessary feasibility test in Theorem 1 to the scenario of S1–S4 with every pair of $t^{\mathtt{end}} > 0$ and J^*_k with reasonable time-complexity.

While we successfully developed a necessary feasibility test in Theorem 1 using Lemma 9, we did not discuss how to check the "if" statement of the lemma. A simple way is to check all possible combinations of $\mathtt{OP}^-_i(t^*)$ and $\mathtt{OP}^+_i(t^*)$ for every $\tau_i \in \tau^{\mathtt{HI}}$. Since there are multiple options of a HI job's contribution to the demand in $[0, t^*]$ and that in $[t^*, t^{\mathtt{end}}]$, the number of combinations of all HI jobs' contribution to those demands can be an exponential function of the number of HI jobs. This entails to find a way that tests Lemma 9 in a time-efficient manner, without investigating all possible combinations.

Algorithm 1. Efficient Test for Lemma 9

```
1:  for $\tau_k \in \tau^{\mathtt{HI}}$ do
2:      if $\tau_k$ satisfies (i) and (ii) of Subcase 3A in Lemma 8, and
        every $\tau_i \in \tau^{\mathtt{HI}} - \{\tau_k\}$ belonging to Case 3 can satisfy (i'),
        (ii') and (iii) of Subcase 3B then
3:          $\mathtt{SumOP}^- \leftarrow C^{\mathtt{LO}}_k + \sum_{\tau_j \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}} \mathrm{maxOP}^-_i(t^*)$
4:          $\mathtt{SumOP}^+ \leftarrow C^{\mathtt{HI}}_k - C^{\mathtt{LO}}_k + \sum_{\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}} \mathrm{maxOP}^+_i(t^*)$
5:          $\mathtt{DiffOP} \leftarrow \sum_{\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}} \mathrm{maxOP}^-_i(t^*) - \sum_{\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}}$
        $\mathrm{minOP}^-_i(t^*)$-
6:          $\mathtt{DiffLO} \leftarrow \max\Big(0, \sum_{\tau_i \in \tau} \mathtt{DBF}^{\mathtt{LO}}_i(t^*) + \mathtt{SumOP}^- - m \cdot t^*\Big)$
7:          $\mathtt{DiffHI} \leftarrow \max\big(0, \sum_{\tau_i \in \tau^{\mathtt{HI}}} \mathtt{DBF}^{\mathtt{HI}}_i\big(t^{\mathtt{end}} - \lceil t^*/T_i \rceil \cdot T_i\big) +$
        $\mathtt{SumOP}^+ - m \cdot (t^{\mathtt{end}} - t^*)\big)$
8:          if $\mathtt{DiffLO} + \mathtt{DiffHI} \le \mathtt{DiffOP}$ then
9:              Return TRUE
10:         end if
11:     end if
12: end for
13: Return FALSE
```

Under a uniprocessor platform (i.e., $m = 1$), there exists only one task (job) that triggers the mode change. Therefore, it is possible to check the constraint in Lemma 9 (i.e., "there exists at least one $\tau_j \in \tau^{\mathtt{HI}} \ldots$") by checking there exists a task τ_k such that τ_k satisfies Conditions (i) and (ii) of Subcase 3A in Lemma 8 (i.e., τ_k triggers the mode change) and the other tasks belonging to Case 3 satisfy Conditions (i)–(iii) of Subcase 3B (i.e., the other tasks do not trigger the mode change). Then, if there exists no such τ_k, we conclude that the "if" statement of Lemma 9 is true.

On the other hand, under a multiprocessor platform (i.e., $m \ge 2$), the number of tasks (jobs) that trigger the mode change can be between 1 and m, which necessitates checking all combinations of choosing x tasks (where $1 \le x \le m$) such that the x tasks satisfy Conditions (i) and (ii) of Subcase 3A and the other tasks belonging to Case 3 satisfy Conditions (i)–(iii) of Subcase 3B. Since the number of combinations is an exponential function of the number of tasks belonging to Case 3, we need to develop an efficient way to check all the combinations. To this end, we will use Conditions (i') and (ii') of Subcase 3B, in which $\mathtt{UB2}^-$ and $\mathtt{LB2}^+$ in Conditions (i) and (ii) of Subcase 3B are replaced with $\mathtt{UB2}^{-'}$ and $\mathtt{LB2}^{+'}$, respectively as follows:

(i') $\mathtt{LB2}^- \le \mathtt{OP}^-_i(t^*) \le \mathtt{UB2}^{-\prime}$, and
(iI') $\mathtt{LB2}^{+\prime} \le \mathtt{OP}^+_i(t^*) \le \mathtt{UB2}^+$, where
$\mathtt{UB2}^{-\prime} = \min(t^* - r^q_i, C^{\mathtt{LO}}_i)$, and $\mathtt{LB2}^{+\prime} = C^{\mathtt{HI}}_i - \mathtt{UB2}^{-\prime}$.

By changing the upper-limit of $\mathtt{OP}^-_i(t^*)$ from $(C^{\mathtt{LO}}_i - 1)$ in $\mathtt{UB2}^-$ to $C^{\mathtt{LO}}_i$ in $\mathtt{UB2}^{-'}$, a task that satisfies Conditions (i'), (ii') and (iii) of Subcase 3B *may* or *may not* trigger the mode change. Then, we can efficiently check all combinations of choosing x tasks that trigger the mode change (where

$1 \le x \le m$) as follows: we check there exists a task τ_k such that 1) τ_k is *one of tasks* that trigger the mode change, which is checked by Conditions (i) and (ii) of Subcase 3A, and 2) the other tasks belonging to Case 3 satisfy Conditions (i'), (ii') and (iii) of Subcase 3B (i.e., the other tasks *may* or *may not* trigger the mode change). Then, instead of checking all combinations of choosing x tasks that trigger the mode change (where $1 \le x \le m$) one by one, we only need to check 1) and 2) for every τ_k belonging to Case 3.

Using the method explained so far, we develop Algorithm 1. We repeat the following steps in Lines 2–11 for every $\tau_k \in \tau^{\mathtt{HI}}$ (Line 1). We first check whether it is possible for the job of τ_k to trigger the mode change by checking 1) τ_k satisfies the conditions (i) and (ii) of Subcase 3A in Lemma 8, and 2) every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ belonging to Case 3 can satisfy the conditions (i'), (ii') and (iii) of Subcase 3B (by having at least one possible value for both $\mathtt{OP}_i^-(t^*)$ and $\mathtt{OP}_i^+(t^*)$) in Line 2.[4] From Line 3, we use the following notations. Let $\max\mathtt{OP}_i^-(t^*)$ and $\min\mathtt{OP}_i^-(t^*)$ denote the upper bound and the lower bound of $\mathtt{OP}_i^-(t^*)$, which are calculated by 0 and 0, $\mathtt{UB}^-$ and $\mathtt{LB}^-$, and $\mathtt{UB2}^{-'}$ and $\mathtt{LB2}^-$, respectively when the job of τ_i belongs to Case 1, Case 2, and Subcase 3B. Likewise, let $\max\mathtt{OP}_i^+(t^*)$ denote the upper bound of $\mathtt{OP}_i^+(t^*)$ which is calculated by 0, $\mathtt{UB}^+$, and $\mathtt{UB2}^+$, respectively when the job of τ_i belongs to Case 1, Case 2, and Subcase 3B. In Line 3, we calculate the sum of $C_k^{\mathtt{LO}}$ (i.e., $\mathtt{OP}_k^-(t^*)$ for τ_k) and the upper bounds of $\mathtt{OP}_i^-(t^*)$ (i.e., $\max\mathtt{OP}_i^-(t^*)$) for all $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$. Likewise, we calculate the sum of $(C_k^{\mathtt{HI}} - C_k^{\mathtt{LO}})$ (i.e., $\mathtt{OP}_k^+(t^*)$ for τ_k) and the upper bounds of $\mathtt{OP}_i^+(t^*)$ for all $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ in Line 4. We also calculate the difference between the sum of the upper bound of $\mathtt{OP}_i^-(t^*)$ (i.e., $\max\mathtt{OP}_i^-(t^*)$) for $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ and the sum of the lower bound of $\mathtt{OP}_i^-(t^*)$ (i.e., $\min\mathtt{OP}_i^-(t^*)$) for $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$, called `DiffOP` in Line 5. Note that `DiffOP` is the same as the sum of the followings for every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$: the sum of the difference between $\max\mathtt{OP}_i^-(t^*)$ and the actual assignment of $\mathtt{OP}_i^-(t^*)$, and the difference between $\max\mathtt{OP}_i^+(t^*)$ and the actual assignment of $\mathtt{OP}_i^+(t^*)$. See the proof of Lemma 10 about why `DiffOP` is the same as the above value.

Then, we calculate `DiffLO`, the LHS minus the RHS of Eq. (5) assuming the maximum (upper bound) of $\mathtt{OP}_i^-(t^*)$ for every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ (Line 6); in other words,`DiffLO` means the minimum amount to be reduced to satisfy Eq. (5) when $\mathtt{OP}_i^-(t^*)$ for every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ has the maximum. Similarly, we calculate `DiffHI`, the LHS minus RHS of Eq. (6) assuming the maximum (upper bound) of $\mathtt{OP}_i^+(t^*)$ for every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ (Line 7); in other words,`DiffHI` means the minimum amount to be reduced to satisfy Eq. (6) when $\mathtt{OP}_i^+(t^*)$ for every $\tau_i \in \tau^{\mathtt{HI}} \setminus \{\tau_k\}$ has the maximum. Then, to satisfy Eqs. (5) and (6), we need to reduce the sum of $\mathtt{OP}_i^-(t^*)$ by as much as `DiffLO` and the sum of $\mathtt{OP}_i^+(t^*)$ by as much as `DiffHI`, while the total budget we can reduce the former or latter sum is as much as `DiffOP`. Therefore, if $\mathtt{DiffLO} + \mathtt{DiffHI} \le \mathtt{DiffOP}$ holds, we return `TRUE` (Lines 8 and 9), meaning that it may be possible for the mode change to occur at t^* without missing any job deadline. Finally, we return `FALSE` in Line 13, if every $\tau_k \in \tau^{\mathtt{HI}}$ cannot yield `TRUE`, meaning that it is impossible for the mode change to occur at t^* without missing any job deadline.

Algorithm 1 can replace Lemma 9, as in the following lemma.

Lemma 10. *If Algorithm 1 returns* `FALSE`*, Theorem 1 (and Lemma 9) judges that for given t^* it is impossible to satisfy both Eqs. (5) and (6) subject to Lemma 8 and the constraint in Lemma 9.*

Proof. The lemma can be proved by showing contradiction of the following statement: Algorithm 1 returns `FALSE` but Lemma 9 cannot judge that a mode change cannot occur at given $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$ without any deadline miss of jobs invoked by τ in $[0, t^{\mathrm{end}}]$. The details are shown in the supplement, available online. □

The time-complexity of Algorithm 1 is $O(n^2)$, and there are at most $\max_{\tau_i \in \tau^{\mathtt{HI}}} D_i$ choices of t^* for any given J_k^* in Theorem 1. Therefore, if we test Theorem 1 using Algorithm 1, the time-complexity of Theorem 1 is $O(n^2 \cdot \max_{\tau_i \in \tau^{\mathtt{HI}}} D_i)$.

While Theorem 1 focuses on the scenario of S1–S4 with given $t^{\mathrm{end}} > 0$ and J_k^*, we can find more infeasible task sets if we apply every pair of $t^{\mathrm{end}} > 0$ and J_k^*. The following lemma calculates an upper-bound of $t^{\mathrm{end}} > 0$ as well as t^* associated with J_k^*.

Lemma 11. *If Eq. (5) is violated with any t^*, it is also violated with some t^* which is smaller than $\left(\sum_{\tau_i \in \tau}(T_i - D_i) \cdot C_i^{\mathtt{LO}}/T_i + \sum_{\tau_i \in \tau^{\mathtt{HI}}} C_i^{\mathtt{LO}}\right)/\left(m - \sum_{\tau_i \in \tau} C_i^{\mathtt{LO}}/T_i\right)$. Also, if Eq. (6) is violated with any $(t^{\mathrm{end}} - t^*)$, it is also violated with some $(t^{\mathrm{end}} - t^*)$ which is smaller than $\left(\sum_{\tau_i \in \tau^{\mathtt{HI}}}(T_i - D_i) \cdot C_i^{\mathtt{HI}}/T_i + C_i^{\mathtt{HI}}\right)/\left(m - \sum_{\tau_i \in \tau^{\mathtt{HI}}} C_i^{\mathtt{HI}}/T_i\right)$.*

Proof. We can prove the lemma using a similar technique to [21], which will be detailed in the supplement, available online. □

Applying Lemma 11, we can test the scenario of S1–S4 with all possible pairs of $t^{\mathrm{end}} > 0$ and J_k^* with reasonable time-complexity, yielding the following collective necessary feasibility test.

Theorem 2. *A MC task set τ is infeasible, if the following collective necessary feasibility test finds at least one pair of $t^{\mathrm{end}} > 0$ and J_k^* that makes it impossible to satisfy both Eqs. (5) and (6) subject to Lemma 8 and the constraint in Lemma 9.*

- *Repeat Theorem 1 using Algorithm 1 for every pair of $t^{\mathrm{end}} > 0$ and J_k^* that satisfies (i) t^{end} is less than the sum of upper bounds of t^* and $(t^{\mathrm{end}} - t^*)$ in Lemma 11 and (ii) J_k^*'s release time is earlier than the upper-bound of t^* in the lemma.*

Proof. By Algorithm 1 and Theorem 1, the theorem holds. □

One may wonder how to "Repeat Theorem 1 using Algorithm 1 for every pair of $t^{\mathrm{end}} > 0$ and J_k^*" in Theorem 2. First, by adding two upper-bounds for t^* and $(t^{\mathrm{end}} - t^*)$ in Lemma 11, we have an upper-bound for t^{end}. Second, we target every job J_k^* whose execution window overlaps with $[0, t^{\mathrm{end}})$. For each job J_k^*, we calculate $[t_a^*(J_k^*), t_b^*(J_k^*)]$ using Lemma 5, and then check all $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$ using Algorithm 1.

4. If there is only one processor, only one task can trigger the mode change. Therefore, on a uniprocessor platform (i.e., $m = 1$), we can apply Conditions (i) and (ii) of Subcase 3B as they are, instead of Conditions (i') and (ii').

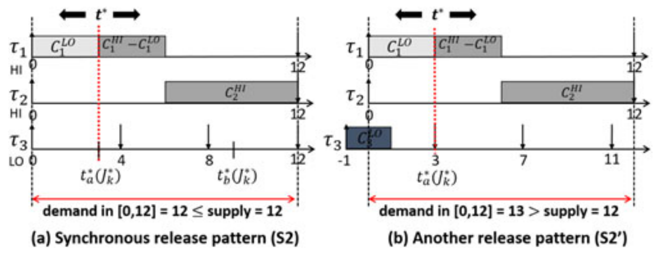

t*
τ1 HI
C1^LO
C1^HI − C1^LO
τ2 HI
C2^HI
τ3 LO
0
4
8
12
-1
3
7
11
t_a*(J_k*)
t_b*(J_k*)
demand in [0,12] = 12 ≤ supply = 12
demand in [0,12] = 13 > supply = 12
(a) Synchronous release pattern (S2)
(b) Another release pattern (S2')

Fig. 3. Synchronous job release pattern and another job release pattern.

For a given $t^{\text{end}} > 0$ and J_k^*, Theorem 1 conducts Algorithm 1 at most $\max_{\tau_i \in \tau^{\text{HI}}} D_i$ times, resulting in $O(n^2 \cdot \max_{\tau_i \in \tau^{\text{HI}}} D_i)$ as explained. As proved in Lemma 11, the upper-bounds on t^{end} and $(t^{\text{end}} - t^*)$ are a pseudo-polynomial in the size of task parameters (if the following condition holds: $\sum_{\tau_i \in \tau} C_i^{\text{LO}}/T_i$ and $\sum_{\tau_i \in \tau^{\text{HI}}} C_i^{\text{HI}}/T_i$ are upper-bounded by some constant strictly less than m). Therefore, the overall time complexity of the above collective necessary feasibility test in Theorem 2 is also pseudo-polynomial in the size of task parameters (if the same condition holds).

We would like to emphasize that the collective necessary feasibility test in Theorem 2 can check Theorem 1 for any number of pairs of $t^{\text{end}} > 0$ and J_k^*, which can affect capability in finding infeasible task sets, but cannot compromise the correctness of whether task sets deemed infeasible by the test is actually infeasible. Therefore, if time-complexity matters, we can limit the number of pairs to be checked, at the expense of sacrificing capability in finding infeasible task sets.

7 Improved Version of Necessary Feasibility Test

Sections 5 and 6 developed necessary feasibility tests by considering the synchronous job release pattern (i.e., S2). Then, we need to know whether the synchronous job release pattern is the best in finding infeasible task sets or not, which is addressed by the following example.

Example 4. There are three tasks: $\tau_1(T_1 = 12, \chi_1 = \text{HI}, C_1^{\text{LO}} = 3, C_1^{\text{HI}} = 6, D_1 = 12)$, $\tau_2 = \tau_1$, and $\tau_3 = (4, \text{LO}, 2, 2, 4)$. Consider the scenario of S1–S4 with $t^{\text{end}} = 12$ and $J_k^* = J_1^1$. According to Example 1 where its τ_1 and τ_2 are the same as those in this example, $t_a^*(J_1^1) = 3$ and $t_b^*(J_1^1) = 9$ hold and Lemma 9 concludes that the "if" statement is false for $t^* = 3$, as shown in Fig. 3a. This means that Lemma 9 cannot prove that the mode change cannot occur at $t^* = 3$ without missing any job deadline, and therefore Theorem 1 cannot deem the task set infeasible.

Now, consider another scenario in which all other conditions are the same as the previous scenario, except the job release pattern of the LO task (i.e., τ_3). τ_3 invokes its jobs strictly-periodically, but one of the jobs has its deadline at $t = 3$. Then, r_3^1 and d_3^1 for J_3^1 are -1 and 3; r_3^2 and d_3^2 for J_3^2 are 3 and 7; and r_3^3 and d_3^3 for J_3^3 are 7 and 11. Then, different from the previous scenario, it is impossible to trigger a mode change at any $t^* \in [3, 9]$ without any job deadline miss, because the deadline of J_3^1 (i.e., $t = 3$) is no later than the mode change instant and therefore J_3^1 contributes to the demand in [0,3]. That is, if J_3^1 does not execute in [0,3], the job should execute for two time units in $[-1, 0]$, which is impossible; therefore, J_3^1 should execute for at least one time unit in [0,3], as shown in Fig. 3b. Considering the first jobs of τ_1 and τ_2 respectively contribute six time units to the demand in [0,12], the total demand in [0,12] is $6 + 6 + 1 = 13$, yielding infeasibility of the scenario as well as the task set.

Motivated by the example, we may consider another job release pattern of LO tasks favorable to finding infeasible task sets without changing that of HI tasks, stated in S2′ that corresponds to S2.

S2′. Generate jobs using the synchronous periodic job release pattern, as follows. First, the first job of every HI task is released at 0, and the following jobs of each HI task are released strictly periodically until each job's absolute deadline is no later than t^{end}. Second, every LO task has a job with its absolute deadline of $t = t_a^*(J_k^*)$ for given J_k^* in S3. The preceding jobs of each LO task are generated backwards such that their deadlines are determined strictly periodically until each job's absolute deadline is later than $t = 0$. The following jobs of each LO tasks are generated forwards such that their deadlines are determined strictly periodically until each job's absolute deadline is no later than t^{end}.

Then, the difference between S2 and S2′ that affects Lemma 9 is the contribution of LO tasks to the demand in Eq. (5), in particular its first term (i.e., $\sum_{\tau_i \in \tau^{\text{LO}}} \text{DBF}_i^{\text{LO}}(t^*)$); since LO tasks cannot contribute to the demand after the mode change, Eq. (6) is irrelevant to the change of S2 to S2′. Then, we calculate the demand of LO tasks in $[0, t^*]$ for S2′, by splitting the interval into $[t_a^*(J_k^*), t^*]$ and $[0, t_a^*(J_k^*)]$. Since there exists a job of $\tau_i \in \tau^{\text{LO}}$, whose deadline is $t_a^*(J_k^*)$, the next job's release time is $t_a^*(J_k^*) + (T_i - D_i)$. Therefore, the demand in $[t_a^*(J_k^*), t^*]$ is calculated by $\text{DBF}_i^{\text{LO}}\big([0, t^* - t_a^*(J_k^*) - (T_i - D_i)]_0^\infty\big)$.[5] The demand of all jobs of $\tau_i \in \tau^{\text{LO}}$ whose release time is no earlier than $t = 0$ in $[0, t_a^*(J_k^*)]$ is simply calculated by $\text{DBF}_i^{\text{LO}}\big(t_a^*(J_k^*)\big)$, and the demand of a job (if any) of $\tau_i \in \tau^{\text{LO}}$ whose release time is earlier than $t = 0$ but whose deadline is later than $t = 0$ is calculated by $\big[(t_a^*(J_k^*) + T_i - D_i) \bmod T_i - (T_i - C_i^{\text{LO}})\big]_0^\infty$. For example, in Fig. 3b, the demand of the first job of τ_3 in [0,3] is $\big[(3 + 4 - 4) \bmod 4 - (4 - 2)\big]_0^\infty = \big[3 - 2\big]_0^\infty = 1$. Therefore, for the scenario of S1, S2′, S3 and S4 with given t^{end} and J_k^*, Eq. (5) for $t_a^*(J_k^*) \le t^* \le t_b^*(J_k^*)$ is changed as follows:

$$\sum_{\tau_i \in \tau^{\text{LO}}} \Big(\text{DBF}_i^{\text{LO}}\big([0, t^* - t_a^*(J_k^*) - (T_i - D_i)]_0^\infty\big) + \text{DBF}_i^{\text{LO}}\big(t_a^*(J_k^*)\big) + \big[(t_a^*(J_k^*) + T_i - D_i) \bmod T_i - (T_i - C_i^{\text{LO}})\big]_0^\infty\Big) + \sum_{\tau_i \in \tau^{\text{HI}}} \Big(\text{DBF}_i^{\text{LO}}(t^*) + \text{OP}_i^-(t^*)\Big) \le m \cdot t^*. \quad (7)$$

Applying S2′ instead of S2 needs to replace Eq. (5) in Lemma 9 with Eq. (7) only. Therefore, we develop a new necessary feasibility test and prove its properties

5. We use $[a]_c^b$ to denote $\max\big(\min(a, b), c\big)$.

corresponding to Lemma 9 and Theorem 1, as stated in the following lemma and theorem.

Lemma 12. *Target the scenario of S1, S2′, S3 and S4 with given $t^{\text{end}} > 0$ and J_k^*, and target a single mode change instant t^* belonging to $[t_a^*(J_k^*), t_b^*(J_k^*)]$ (defined in Lemma 5). The mode change instant t^* is infeasible without any job deadline miss, if it is impossible to satisfy both Eqs. (7) and (6) subject to Lemma 8 and the constraint in Lemma 9.*

Proof. By Lemma 8 and the fact that the mode change is triggered by at least one (but at most m) jobs, the two constraints (i.e., the "subject to" part) hold.

The LHS of Eq. (7) and that of Eq. (6) calculate the total demand of τ in $[0, t^*]$ and $[t^*, t^{\text{end}}]$, respectively under the scenario of S1, S2′, S3 and S4 with given $t^{\text{end}} > 0$ and J_k^*. Considering the supply amounts to the interval length, violating either Eqs. (7) or (6) implies that the mode change cannot occur at t^* or at least one job misses its deadline. □

Theorem 3 (Another necessary feasibility condition). *A MC task set τ is infeasible, if every mode change instant $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$ associated with the scenario of S1, S2′, S3, and S4 with given $t^{\text{end}} > 0$ and J_k^* makes the "if" statement in Lemma 12 true.*

Proof. By Lemma 12 and the range of $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$, the impossibility to satisfy both Eqs. (7) and (6) subject to Lemma 8 and the constraint in Lemma 9 implies that the scenario of S1, S2′, S3 and S4 with given $t^{\text{end}} > 0$ and J_k^* yields no existence of the mode change instant without any job deadline miss. Since Lemma 5 also holds for the scenario of S1, S2′, S3 and S4, existence of the mode change instant in $[t_a^*(J_k^*), t_b^*(J_k^*)]$ is a necessary feasibility condition for the scenario. Therefore, the theorem holds. □

We note that all the theories in Section 6 hold for Lemma 12 and Theorem 3, after slight modification; here, we present a new theorem corresponding to Theorem 2 as follows.

Theorem 4. *A MC task set τ is infeasible, if the following collective necessary feasibility test finds at least one pair of $t^{\text{end}} > 0$ and J_k^* that makes it impossible to satisfy both Eqs. (7) and (6) subject to Lemma 8 and the constraint in Lemma 9.*

- *Repeat Theorem 3 using Algorithm 1 for every pair of $t^{\text{end}} > 0$ and J_k^* that satisfies (i) t^{end} is less than the sum of upper bounds of t^* and $(t^{\text{end}} - t^*)$ in a lemma[6] corresponding to Lemma 11 and (ii) J_k^*'s release time is earlier than the upper-bound of t^* in the lemma.*

Proof. By Algorithm 1 and Theorem 3, the theorem holds. □

8 Simplified Version of Necessary Feasibility Test

Although we successfully developed two types of necessary feasibility tests, one may desire simpler versions of the tests that not only exhibit the same (or similar) time-complexity as the existing necessary feasibility test for SC task systems, but also make it possible to easily compare them with Eqs. (3) and (4) in terms of capability in finding infeasible task sets. In this section, we develop such simpler versions of Theorems 2 and 4.

Compared with the existing test for SC task systems presented in Lemma 1, our proposed test presented in Theorem 2 additionally requires to i) target every job J_k^* whose execution window overlaps with $[0, t^{\text{end}})$ and ii) check all $t^* \in [t_a^*(J_k^*), t_b^*(J_k^*)]$. To develop a simpler version of Theorem 2 that exhibits the same time-complexity as Lemma 1, we first restrict to target J_k^* as the *first* job of any HI task, meaning that the execution requirement of every HI job amounts to its HI WCET. Then, the total demand of $\tau^{\texttt{HI}}$ in $[0, t^{\text{end}})$ is upper-bounded by $\sum_{\tau_i \in \tau^{\texttt{HI}}} \texttt{DBF}_i^{\texttt{HI}}(t^{\text{end}})$, which is independent of J_k^*. Second, we only check $t^* = t_a^*(J_k^*)$; considering $\texttt{DBF}_i^{\texttt{LO}}(t^*)$ non-decreases as t^* increases, it is sufficient to check the combined inequality only with $t^* = t_a^*(J_k^*)$ for Lemma 9, and the earliest $t_a^*(J_k^*)$ is calculated by $\min_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{LO}}$. By adding Eqs. (5) and (6) under the scenario of S1–S4 with given t^{end}, the targeted J_k^*, and $t^* = t_a^*(J_k^*)$, we derive the following combined inequality:

$$\sum_{\tau_i \in \tau^{\texttt{LO}}} \texttt{DBF}_i^{\texttt{LO}}(t_a^*) + \sum_{\tau_i \in \tau^{\texttt{HI}}} \texttt{DBF}_i^{\texttt{HI}}(t^{\text{end}}) \leq m \cdot t^{\text{end}}, \tag{8}$$

where $t_a^* = \min_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{LO}}$. Then, the following theorem holds.

Theorem 5. *A task set τ is infeasible, if there exists $t^{\text{end}} \geq t_a^* = \min_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{LO}}$ which makes Eq. (8) false.*

Proof. In the target scenario, it is impossible for any HI job to trigger the mode change (by executing for its LO WCET) before t_a^*. Therefore, LO jobs should perform their execution for as much as the first term of Eq. (8) in $[0, t^{\text{end}}]$ with $t^{\text{end}} \geq t_a^*$; otherwise, there exists a LO job's deadline miss. Also, HI jobs should perform their execution for as much as the second term of Eq. (8) in $[0, t^{\text{end}}]$ with $t^{\text{end}} > 0$; otherwise, there exists a HI job deadline miss. □

Similarly, if we consider the scenario of S1, S2′, S3 and S4 instead of S1–S4, we have the following combined inequality from Eqs. (7) and (6), which corresponds to Eq. (8):

$$\begin{aligned}\sum_{\tau_i \in \tau^{\texttt{LO}}} \Big(\texttt{DBF}_i^{\texttt{LO}}(t_a^*) + \big[(t_a^* + T_i - D_i) \bmod T_i - (T_i - C_i^{\texttt{LO}})\big]_0^\infty \Big) \\ + \sum_{\tau_i \in \tau^{\texttt{HI}}} \texttt{DBF}_i^{\texttt{HI}}(t^{\text{end}}) \quad \leq \quad m \cdot t^{\text{end}},\end{aligned} \tag{9}$$

which yields the following theorem.

Theorem 6. *A task set τ is infeasible, if there exists $t^{\text{end}} \geq t_a^* = \min_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{LO}}$ which makes Eq. (9) false.*

Proof. The proof is the same as that of Theorem 5 except the demand of LO tasks. □

It is trivial that the time-complexity of testing Theorem 5 or 6 with all possible t^{end} is equivalent to that of testing Lemma 1 with all possible t^{end}. Also, it is easily observed that Theorems 5 and 6 exhibit better capability in finding infeasible task sets, than Lemma 3, which is recorded in the following lemma.

6. To derive a condition for Eq. (7) corresponding to the condition for Eq. (5) in Lemma 11, it is sufficient to add $\sum_{\tau_i \in \tau^{\texttt{LO}}} C_i^{\texttt{LO}}$ in the numerator for the upper bound of t^*.

Lemma 13. *If a task set τ is deemed infeasible by Lemma 3, τ is also deemed infeasible by Theorems 5 and 6. However, the converse does not always hold.*

Proof. Since the LHS of Eq. (4) is no larger than that of Eq. (8) and that of Eq. (9), the lemma trivially holds. □

9 Relationship Among Proposed Tests

So far, we have presented four necessary feasibility tests: two collective necessary feasibility tests with different target job release patterns, and two simplified versions thereof. We now discuss their relationship in terms of capability in finding infeasible task sets and time-complexity.

Capability in Finding Infeasible Task Sets. A collective necessary feasibility test with S2 and that with S2′ complement each other in finding infeasible task sets, since we cannot say that either the LHS of Eq. (5) or that of Eq. (7) is always no smaller than the other, recorded in the following lemma.

Lemma 14 (Incomparability between Theorems 2 and 4). *There exists a task set τ deemed infeasible by Theorem 2, not by Theorem 4, and the converse is true.*

Proof. The lemma can be proven by showing the existence of two opposite situations: (i) the latter (i.e., Theorem 4) is more favorable in finding infeasible task sets than the former (i.e., Theorem 2), meaning that the latter yields a larger demand than the former, and (ii) the vice versa.

First, (i) is already presented in Example 4.

Second, for (ii), we consider the task set in Example 4, with replacing τ_3 with $\tau_3' = (4, \texttt{LO}, 1, 1, 4)$; also, we consider $t^* = 4$. Then, the former yields the demand of one time unit by τ_3' in [0,4] (which is one LO WCET of τ_3). On the other hand, the latter yields the demand of zero time unit by τ_3' in [0,4] as follows. Since the first job of τ_3' (whose deadline is $t = 3$) can be executed in $[-1, 0]$, there is no demand by the job in [0,3]; also, the next jobs of τ_3' cannot yield any demand because their deadlines are later than $t^* = 4$.

Due to the existence of the two opposite situations (i) and (ii), the lemma holds. □

Different from the two collective necessary feasibility tests with different release patterns, there is a dominance relationship between their simplified versions, since the LHS of Eq. (9) is always no smaller than that of Eq. (8), recorded in the following lemma.

Lemma 15 (Dominance between Theorems 5 and 6). *Any task set τ deemed infeasible by Theorem 5 is also deemed infeasible by Theorem 6.*

Proof. Suppose that there is a task set τ is deemed infeasible by Theorem 5. Then, according to Theorem 5, there exists $t^{\text{end}} \geq t_a^* = \min_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{LO}}$ which makes Eq. (8) false. Since the LHS of Eq. (9) is always no smaller than that of Eq. (8), such t^{end} also makes Eq. (9) false. Therefore, the lemma holds. □

The relationship between collective necessary feasibility tests and their simplified versions is recorded in the following lemma.

Lemma 16 (Dominance between individual collective tests and their simplified versions). *Any task set τ deemed infeasible by Theorem 5* (likewise *Theorem 6) is also deemed infeasible by Theorem 2* (likewise *Theorem 4).*

Proof. Theorem 2 checks all possible pairs of J_k^* and t^* for given t^{end}, and any pair that violates Eqs. (5) or (6) yields infeasibility of the target task set. On the other hand, Theorem 5 checks only one pair of J_k^* and t^*, and the pair belongs to a set of pairs checked by Theorem 2. This implies the dominance between Theorems 2 and 5 holds. With the same reason, the dominance between Theorem 2 with replacing S2 with S2′ and Theorem 6 holds. □

Also, it is easily observed that Theorems 5 and 6 exhibit better capability in finding infeasible task sets, than Lemma 3 (i.e., one of trivial extensions of the necessary feasibility test for SC task systems), without incurring additional time-complexity.

Time-Complexity. Although all proposed necessary feasibility tests are of pseudo-polynomial time complexity in the task parameters, the simplified versions of the collective necessary feasibility tests (i.e., Theorems 5 and 6) have a much lower degree of the pseudo-polynomial—even with the same complexity as the one for SC task systems presented in Lemma 1.

10 Evaluation

In this section, we demonstrate the capability of the proposed feasibility tests in covering a broader range of infeasible MC task sets on both uniprocessor and multiprocessor platforms.

10.1 Generation of Task Sets

We generate a synthetic task set similarly in [22], [23], [24], which can be summarized as follows. We have six input parameters: (i) the number of processors $m \in \{1, 2, 4\}$, (ii) the number of tasks $n \in \{4, 6, 8, 10\}$, (iii) the probability (CP) of each task τ_i having $\chi_i = \texttt{HI} \in \{0.3, 0.5, 0.7\}$, (iv) the maximum ratio (CF) of each HI task τ_i's HI WCET to LO WCET, i.e., $\mathsf{CF} \overset{\text{def}}{=} C_i^{\texttt{HI}}/C_i^{\texttt{LO}} \in \{1.25, 1.5, 2, 3, 4\}$, (v) LO total utilization $U^{\texttt{LO}} \overset{\text{def}}{=} \sum_{\tau_i \in \tau} C_i^{\texttt{LO}}/T_i$, and (vi) HI total utilization $U^{\texttt{HI}} \overset{\text{def}}{=} \sum_{\tau_i \in \tau^{\texttt{HI}}} C_i^{\texttt{HI}}/T_i$. We choose $U^{\texttt{LO}}$ and $U^{\texttt{HI}}$ from $(m - 0.55)$ to m with an incremental step of 0.05 (resulting in 12 choices), respectively.

Given a 6-tuple $(m, n, \mathsf{CP}, \mathsf{CF}, U^{\texttt{LO}}, U^{\texttt{HI}})$ for a task set, each task parameter is determined as follows: T_i is uniformly chosen in [1,1000]; χ_i is selected as HI with probability CP (and as LO with probability $(1.0 - \mathsf{CP})$); $C_i^{\texttt{LO}}$ is determined by using the UUniFast-Discard algorithm [25], [26]; $C_i^{\texttt{HI}}$ is uniformly chosen in $[C_i^{\texttt{LO}} + 1, \mathsf{CF} \cdot C_i^{\texttt{LO}} + 1]$, if $\chi_i = \texttt{HI}$ (and set to $C_i^{\texttt{LO}}$, otherwise); and D_i is set to T_i for implicit-deadline task sets. Using the above task parameters, we first generate 1,000 implicit-deadline task sets whose LO and HI total utilizations are in $[U^{\texttt{LO}} - 0.05, U^{\texttt{LO}}]$ and $[U^{\texttt{HI}} - 0.05, U^{\texttt{HI}}]$ for given $U^{\texttt{LO}}$ and $U^{\texttt{HI}}$, respectively, resulting in a total of 144,000 task sets for given m, n, CP, and CF. Accordingly, constrained-deadline task sets are generated to have the same task parameter values as implicit-deadline ones except

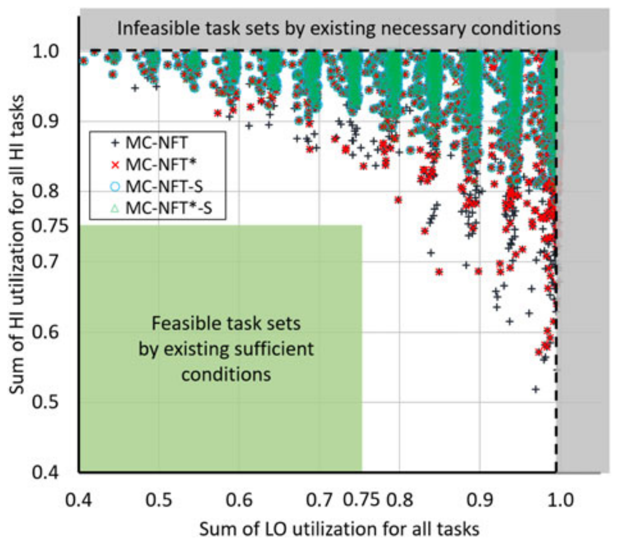


Fig. 4. All task sets in the region between X=0.75 by Y=0.75 and X=1.0 by Y=1.0 have not been proven infeasible by any existing studies. The necessary feasibility tests proposed in this paper proves infeasibility of the task sets marked in the region.

choosing D_i uniformly in $[C_i^{\mathrm{HI}}, T_i]$. Among the generated constrained-deadline task sets, we exclude all task sets that can be proven infeasible by trivial necessary feasibility tests shown in Lemmas 2 and 3 and consider the remaining task sets as task sets of interest.

With the generated task sets, we compare the following four proposed necessary feasibility tests:

- MC-NFT: the collective necessary feasibility test in Theorem 2 (with the scenario of S1–S4),
- MC-NFT*: the collective necessary feasibility test in Theorem 4 (with the scenario of S1, S2′, S3 and S4),
- MC-NFT-S: the simplified version of MC-NFT in Theorem 5,
- MC-NFT*-S: the simplified version of MC-NFT* in Theorem 6, and
- MC-NFT-ALL: the necessary feasibility test in which a task set is deemed infeasible by either MC-NFT or MC-NFT*.

Note that we also compare MC-NFT-ALL. Since MC-NFT and MC-NFT* are incomparable as stated in Lemma 14, combining the results of MC-NFT and MC-NFT* can find more infeasible task sets than those proven infeasible by either test alone.

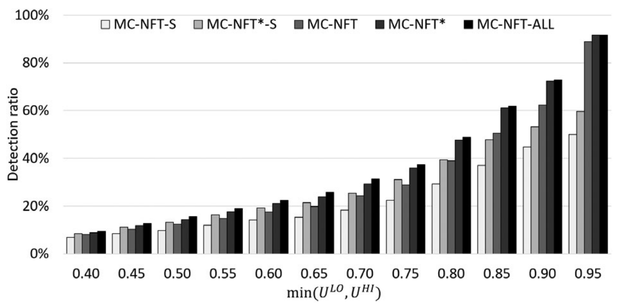


Fig. 5. Detection ratio of the four proposed necessary feasibility tests for constrained-deadline task sets with different ranges of $\min(U^{\mathrm{LO}}, U^{\mathrm{HI}})$ when $m = 1$, $n = 4$, CP $= 0.3$ and CF $= 3$.

TABLE 1
Average Running Times of the Four Proposed Necessary Feasibility Tests for Constrained-Deadline Task Sets With Different Numbers of Tasks (n) When $m = 1$, CP $= 0.3$, and CF $= 3$

	n	4	6	8	10
Avg. running time (μs)	MC-NFT	19	41	77	75
	MC-NFT*	16	55	79	137
	MC-NFT-S	4	5	9	5
	MC-NFT*-S	4	4	4	5

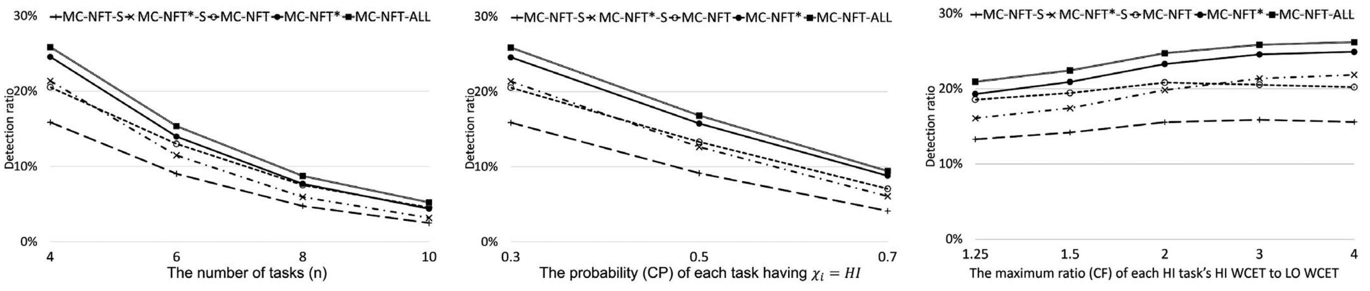
10.2 Simulation Results

We show the results of implicit-deadline task sets and constrained-deadline task sets for different combinations of input parameters (i.e., m, n, CP, CF, U^{LO}, and U^{HI}). In order to show how many infeasible task sets can be found by our proposed tests, we use *detection ratio* as performance metric, defined as the percentage of task sets that are deemed infeasible by each individual necessary feasibility test to the total number of task sets of interest. First, we focus on the uniprocessor case and discuss i) how the detection ratios of the proposed necessary feasibility tests vary with different values of U^{LO} and U^{HI} for implicit-deadline task sets (with Fig. 4) and then those for constrained-deadline task sets (with Fig. 5), respectively, ii) how other input combinations (i.e., n, CP, CF) influence the overall detection ratios (with Fig. 6), and iii) how scalable the proposed tests are with respect to the number of tasks (with Table 1). Next, we show the results of multiprocessor cases (with Figs. 7 and 8).

For Implicit-Deadline Task Sets. Fig. 4 shows the implicit-deadline task sets (marked in the region between X=0.75 by Y=0.75 and X=1.0 by Y=1.0)—which were not proven infeasible by any existing studies but which were newly proven infeasible by the five proposed necessary feasibility tests—in (U^{LO}, U^{HI})-plane when $m = 1$, $n = 4$, CF $= 3$ and CP $= 0.3$.[7] In (U^{LO}, U^{HI})-plane, we focus on the region where $0.75 < \max(U^{\mathrm{LO}}, U^{\mathrm{HI}}) \leq 1.0$, yielding 95,000 task sets of interest among 144,000 generated task sets. This is because all task sets with $\max(U^{\mathrm{LO}}, U^{\mathrm{HI}}) \leq 0.75$ and those with $\max(U^{\mathrm{LO}}, U^{\mathrm{HI}}) > 1.00$ are already proven feasible [27] and infeasible [12], [13], [14], respectively.

We have the following observations. First, the proposed necessary feasibility tests can newly find a number of infeasible task sets over a wider range of LO and HI total utilization. Second, they identify significantly more infeasible task sets as LO and HI total utilization become close to 1.0; for example, among 1,000 task sets with $0.95 \leq U^{\mathrm{LO}} \leq 1.0$ and $0.95 \leq U^{\mathrm{HI}} \leq 1.0$, MC-NFT and MC-NFT* find 443 (44.3%) and 489 (48.9%) infeasible task sets, respectively. Third, MC-NFT-S and MC-NFT*-S rarely find infeasible task sets in the region where $0.75 \leq U^{\mathrm{LO}} \leq 1.0$ and $0.4 \leq U^{\mathrm{HI}} \leq 0.75$, whereas MC-NFT and MC-NFT* can do. This is because MC-NFT-S and MC-NFT*-S restrict to target J_k^* as the first job of any HI task, meaning that most of the interval of interest exhibits the HI behavior, so it is difficult to determine infeasibility when exhibiting the LO behavior. On the other

7. We do not plot task sets proven infeasible by MC-NFT-ALL in Fig. 4, as the result of MC-NFT-ALL is a combination of all the task sets marked by at least one of MC-NFT, MC-NFT*, MC-NFT-S and MC-NFT*-S.



(a) Varying n (b) Varying CP (c) Varying CF

Fig. 6. Detection ratio of the four proposed necessary feasibility tests for constrained-deadline task sets with (a) varying n when CP $= 0.3$ and CF $= 3$, (b) varying CP when $n = 4$ and CF $= 3$, and (c) varying CF when $n = 4$ and CP $= 0.3$.

hand, MC-NFT and MC-NFT* consider many choices of J_k^* and separately investigate the two sub-intervals of each interval of interest exhibiting LO and HI behavior, resulting in higher capability in finding infeasible task sets over a wider range of LO and HI total utilization. Fourth, MC-NFT and MC-NFT* are shown to dominate MC-NFT-S and MC-NFT*-S, respectively, and MC-NFT*-S is shown to dominate MC-NFT-S. Meanwhile, MC-NFT and MC-NFT* are incomparable; 361 task sets are proven infeasible by MC-NFT, not by MC-NFT*, and 402 task sets are proven infeasible by MC-NFT*, not by MC-NFT.

In total, MC-NFT, MC-NFT*, MC-NFT-S, MC-NFT*-S, and MC-NFT-ALL find 2,989, 3,030, 1,556, 1,952, and 3,391 infeasible task sets, respectively, among 95,000 generated task sets. Note that although, among generated task sets, there might also exist task sets proven feasible by existing sufficient schedulability tests under some scheduling algorithms, such as PLRS [18], EDF-VD [28], GREEDY [19], and ECDF [20], we did not identify them in our evaluation. This is because the number of task sets proven infeasible by this paper is independent of those sufficient feasibility tests. Instead, if we exclude those feasible task sets from task sets of interest, the detection ratio presented in this section will increase, implying that the presented detection ratio as of now exhibits the minimum capability of the proposed necessary feasibility tests in finding infeasible task sets.

For Constrained-Deadline Task Sets. Recall that, among the generated task sets, we exclude all task sets that can be proven infeasible by trivial necessary feasibility tests shown in Lemmas 2 and 3 and consider the remaining task sets as task sets of interest, which are 43,972 task sets for $m = 1$.

Fig. 5 plots the detection ratio by the five proposed necessary feasibility tests while varying $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ from [0.40,0.45) to $[0.95, 1.0]$ on a uniprocessor platform (i.e., $m = 1$). We have the following observations. First, all the proposed tests exhibit high capability in finding infeasible task sets in that MC-NFT, MC-NFT*, MC-NFT-S, MC-NFT*-S, and MC-NFT-ALL find 9,028 (20.5%), 10,805 (24.6%), 6,981 (15.9%), 9,395 (21.4%), and 11,375 (25.9%) infeasible task sets, respectively, among 43,972 task sets of interests. Such high capability can be interpreted as the benefit of dealing with unique issues pertaining to MC task systems. In particular, higher capability for constrained-deadline task sets (than that for implicit-deadline task sets) mainly comes from accurate calculation on the execution contribution of HI to the sub-intervals and precise constraints thereof, in that we generate a constrained-deadline task set by reducing the relative deadline of tasks in the corresponding implicit-deadline task set. Second, all the proposed tests find more infeasible task sets as $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ increases; for example, using MC-NFT*, 8.8% and 91.7% of the task sets are proven infeasible with $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ in [0.4,0.45) and [0.95,1.0], respectively. This is due to the difficulty in meeting all job deadlines of a task set with high $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$. Third, MC-NFT and MC-NFT* are shown to outperform MC-NFT-S and MC- NFT*-S, respectively, for all values of $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$. This is because MC-NFT and MC-NFT* (i) derive a tighter bound on the demand of HI jobs by considering the relationship between the mode change instant and each HI job's execution window, and (ii) test many choices of J_k^* (while MC-NFT-S and MC-NFT*-S test one choice of J_k^*). Nevertheless, MC-NFT-S and MC-NFT*-S have the same time complexity as in the SC task system case, while finding some infeasible task sets. Fourth, MC-NFT* and MC-NFT*-S outperform MC-NFT and MC-NFT-S, respectively, to some extent (by up to 10.6% and

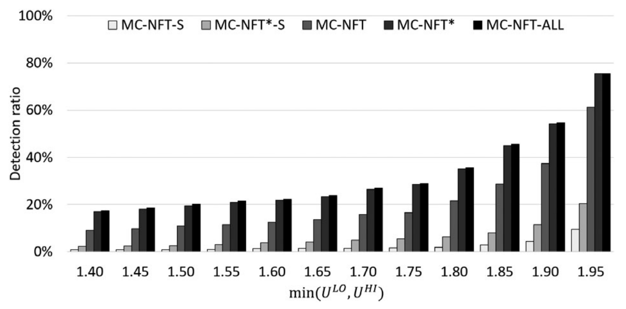


Fig. 7. Detection ratio of the four proposed necessary feasibility tests for constrained-deadline task sets with different ranges of $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ when $m = 2$, $n = 8$, CP $= 0.3$ and CF $= 3$.

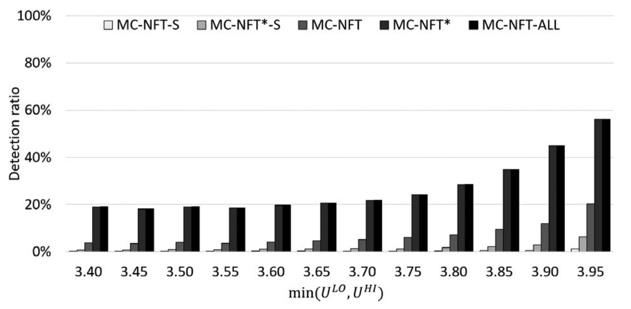


Fig. 8. Detection ratio of the four proposed necessary feasibility tests for constrained-deadline task sets with different ranges of $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ when $m = 4$, $n = 16$, CP $= 0.3$ and CF $= 3$.

10.7% detection ratio, respectively) for all ranges of $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$. Such an improvement can be interpreted as the benefit of considering another job release pattern favorable to finding infeasible MC task sets, compared to the synchronous one.

Recall that MC-NFT and MC-NFT* are incomparable in terms of capability in finding infeasible task sets as stated in Lemma 14. Thus, these two tests can be used to complement each other in discovering more infeasible task sets. By combining the results of MC-NFT and MC-NFT*, 11,375 (25.9%) infeasible tasks sets were proven infeasible by MC-NFT-ALL, which finds 1.3% more infeasible task sets than the maximum performance of MC-NFT and MC-NFT*.

Note that, in our preliminary conference version [15], there was a minor implementation error in the simulation results of MC-NFT for constrained-deadline task sets; the corrected results are shown in the supplement, available online.

For Other Input Combinations. We now discuss how each input parameter influences the overall detection ratio of the five necessary feasibility tests. Fig. 6 plots the detection ratio by five proposed necessary feasibility tests while (a) varying n from 4 to 10, (b) varying CP from 0.3 to 0.7, and (c) varying CF from 1.25 to 4 when $m = 1$. The overall detection ratio of all the proposed tests is decreased as n or CP increases. For example, using MC-NFT*, 24.6% and 4.4% of the task sets are proven infeasible with $n = 4$ and 10, and 24.6% and 8.8% of the task sets are proven infeasible with $\mathsf{CP} = 0.3$ and 0.7, respectively. As n or CP increases, the number of HI jobs whose execution windows overlap with the mode change instant increases, yielding more combinations for constraints. If at least one combination yields a feasible schedule, the necessary feasibility tests cannot deem the task sets infeasible. On the other hand, the overall detection ratio of all the proposed tests is slightly increased as CF increases. For example, using MC-NFT*, 19.3% and 25.0% of the task sets are proven infeasible with $\mathsf{CF} = 1.25$ and 4, respectively.

Scalability With Respect to the Number of Tasks. We now briefly show the scalability of the proposed feasibility tests with respect to the number of tasks. Table 1 shows the average running time of the proposed feasibility tests to identify an infeasible constrained-deadline task set with different numbers of tasks when $m = 1$, $\mathsf{CP} = 0.3$, and $\mathsf{CF} = 3$. The running time of MC-NFT and MC-NFT* increases as n increases by reflecting their analytical procedures while the running time of MC-NFT-S and MC-NFT*-S is relatively stable to the number of tasks and an order of magnitude shorter than that of MC-NFT and MC-NFT* . In particular, as n increases from 4 to 10, the average running time of MC-NFT* is increased from 16μs to 137μs, while the average running time of MC-NFT*-S is almost the same for all values of n.[8] Note that the running time of the proposed feasibility tests is independent of the number of processors as shown in their time complexity.

For Multiprocessor Platforms. Figs. 7 and 8 plot the detection ratio by the four proposed necessary feasibility tests while varying $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ from $[m - 0.6, m - 0.55)$ to $[m - 0.05, m]$ when $m = 2$ and 4 ($n = 4m$), respectively. We have the following observations. First, we confirmed that the similar tendency is observed for multiprocessor platforms with that for uniprocessor platforms as $\min(U^{\mathtt{LO}}, U^{\mathtt{HI}})$ increases. Second, the performance gap between MC-NFT (MC-NFT*) and MC-NFT-S (MC-NFT*-S) becomes larger as m increases. We can interpret such a gap as the benefit of separate demand-supply comparison with the interval split based on the mode change instant, compared to that without the interval split. As m increases, the number of HI jobs whose execution windows overlap with the mode change instant increases, imposing more constraints of the contribution of those jobs to demand in each sub-interval. MC-NFT and MC-NFT* more accurately capture those constraints and derive a tighter bound on the execution contribution of HI tasks to the demand in the sub-intervals separately, yielding higher capability in finding infeasible task sets with a larger value of m (i.e., a larger value of n).

In total, MC-NFT, MC-NFT*, MC-NFT-S, MC-NFT*-S, and MC-NFT-ALL find 7,942 (14.1%), 13,627 (24.2%), 758 (1.3%), 2,269 (4.0%), and 13,894 (24.7%) infeasible task sets, respectively, among 56,199 task sets of interests, when $m = 2$. Also, MC-NFT, MC-NFT*, MC-NFT-S, MC-NFT*-S, and MC-NFT-ALL find 4,017 (5.0%), 17,751 (22.2%), 158 (0.2%), 944 (1.2%), and 17,778 (22.3%) infeasible task sets, respectively, among 79,806 task sets of interests, when $m = 4$. Note that, for $m = 8$ or more, it was very time-consuming to generate enough valid task sets by using the UUniFast-Discard algorithm as mentioned in [26]; recent studies [29], [30] proposed more efficient algorithms for task set generation to resolve the scalability issue.

11 Conclusion

In this paper, we investigated characteristics of MC necessary feasibility conditions and identified challenges for deriving those conditions. By resolving the challenges, we successfully established the following foundations for necessary feasibility tests for MC task systems: i) how to select a scenario, favorable to derive necessary conditions, ii) how to calculate demand, when the mode change occurs, iii) how to determine the target sub-intervals, for demand-supply comparison, and iv) how to derive an infeasibility condition from demand-supply comparisons with different possible mode change instants, v) how to reduce time-complexity of the proposed tests, and vi) which job release pattern is favorable to finding infeasible task sets. We then completed to develop a collection of necessary feasibility tests, by considering different job release patterns as a target scenario and a tradeoff between time-complexity and capability in finding infeasible task sets, which are the first study that yield non-trivial results for MC necessary feasibility for both uniprocessor and multiprocessor platforms. We demonstrated that the proposed necessary feasibility tests find a number of infeasible task sets which have been proven neither feasible nor infeasible by any existing studies.

References

[1] AUTOSAR, 2021. [Online]. Available: autosar.org

[2] T. Gaska, C. Watkin, and Y. Chen, “Integrated modular avionics - past, present, and future,” *IEEE Aeros. Electron. Syst. Mag.*, vol. 30, no. 9, pp. 12–23, Sep. 2015.

8. The average running time was measured on a machine that has Intel i7-8700K CPU.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. 28th IEEE Int. Real-Time Syst. Symp.*, 2007, pp. 239–243.

[4] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–37, Jan. 2017.

[5] 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.

[6] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 1–44, Oct. 2011.

[7] L. Sha *et al.*, "Real time scheduling theory: A historical perspective," *Real-Time Syst.*, vol. 28, pp. 101–155, 2004.

[8] T. P. Baker and M. Cirinei, "A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks," in *Proc. 27th IEEE Int. Real-Time Syst. Symp.*, 2006, pp. 178–190.

[9] S. Baruah and N. Fisher, "The feasibility analysis of multiprocessor real-time systems," in *Proc. 18th Euromicro Conf. Real-Time Syst.*, 2006, pp. 85–96.

[10] N. Fisher, S. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Proc. 18th Euromicro Conf. Real-Time Syst.*, 2006, pp. 118–127.

[11] V. Bonifaci and A. Marchetti-Spaccamela , "Feasibility analysis of sporadic real-time multiprocessor task systems," *Algorithmica*, vol. 101, pp. 763–780, 2012.

[12] S. Baruah *et al.*, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *J. ACM*, vol. 62, no. 2, pp. 14:1–14:33, Apr. 2015.

[13] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: Simplified and optimally quantified," in *Proc. IEEE Real-Time Syst. Symp.*, 2015, pp. 327–337.

[14] S. Ramanathan, X. Gu, and A. Easwaran, "The feasibility analysis of mixed-criticality systems," in *Proc. Real-Time Scheduling Open Problems Seminar*, 2016, pp. 5–6.

[15] H. S. Chwa, H. Baek, and J. Lee, "Necessary feasibility analysis for mixed-criticality task systems on uniprocessor," in *Proc. IEEE Real-Time Syst. Symp.*, 2019, pp. 446–457.

[16] K. Agrawal and S. Baruah, "Intractability issues in mixed-criticality scheduling," in *Proc. 30th Euromicro Conf. Real-Time Syst.*, 2018, pp. 11:1–11:21.

[17] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. 11th Real-Time Syst. Symp.*, 1990, pp. 182–190.

[18] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *Proc. IEEE 32nd Real-Time Syst. Symp.*, 2011, pp. 13–23.

[19] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, 2012, pp. 135–144.

[20] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *Proc. IEEE 34th Real-Time Syst. Symp.*, 2013, pp. 78–87.

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

[22] R. M. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, 2012, pp. 309–320.

[23] H. Baek, N. Jung, H. S. Chwa, I. Shin, and J. Lee, "Non-preemptive scheduling for mixed-criticality real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1766–1779, Aug. 2018.

[24] H. S. Chwa, K. G. Shin, H. Baek, and J. Lee, "Physical-state-aware dynamic slack management for mixed-criticality systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2018, pp. 129–139.

[25] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1/2, pp. 129–154, May 2005.

[26] R. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, 2009, pp. 398–409.

[27] S. Baruah *et al.*, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. 24th Euromicro Conf. Real-Time Syst.*, 2012, pp. 145–154.

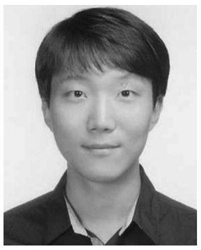
[28] S. Baruah, V. Bonifaci, G. D'Angelo, A. MarchettiSpaccamela, S. van der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *Proc. 19th Annu. Eur. Symp. Algorithms*, 2011, pp. 555–566.

[29] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. Int. Workshop Anal. Tools Methodol. Embedded Real-Time Syst.*, 2010, pp. 6–11.

[30] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 76–88.

Hoon Sung Chwa (Member, IEEE) received the BS, MS, and PhD degrees from KAIST, Daejeon, South Korea, in 2009, 2011, and 2016, respectively, all in computer science. He is currently an assistant professor with the Department of Information and Communication Engineering, DGIST, Daegu, South Korea. He has been a research fellow with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, until 2018. His research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems. He was a recipient of the Best Paper Award from 33rd IEEE Real-Time Systems Symposium in 2012 and the IEEE International Conference on Cyber-Physical Systems, Networks, and Applications in 2014.

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



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