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

NOWAYS, 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
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 feasibility 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 made 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 II–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 \( \tau \) 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 \( t_i \in \tau \) is characterized by a tuple \((T_i, \lambda_i, C_i^{LO}, C_i^{HI}, D_i)\), where \( T_i \) is the minimum separation (or period) between successive job releases, \( \lambda_i \in \{LO, HI\} \) is the criticality level, \( C_i^{LO} \) is the LO WCET (worst-case execution time), \( C_i^{HI} \) is the HI WCET, and \( D_i \) is the relative deadline. A task \( t_i \) is said to be a LO and HI task, if \( \lambda_i \) is LO and HI, respectively; let \( T_i^{LO} \) and \( T_i^{HI} \) denote a set of LO and HI tasks in \( \tau \), respectively. We assume that \( C_i^{LO} \leq C_i^{HI} \) for every HI task and \( C_i^{LO} = C_i^{HI} \) for every LO task. We target implicit- and constrained-deadline task systems, respectively, in which \( D_i = T_i \) and \( D_i \geq T_i \) hold for every \( t_i \in \tau \).

**MC Jobs and Scenarios.** Task \( t_i \) generates a potentially infinite sequence of jobs: \( J_i^1, J_i^2, J_i^3, \ldots \). The \( q \)th job of \( t_i \) (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 \( \gamma_i^q \in (0, C_i^{HI}] \) is the execution requirement of the job. \( J_i^q \) has completed its execution if it executes for \( \gamma_i^q \). 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 \( t_i \) is said to be a LO and HI job, if \( \lambda_i \) 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 \( \tau \) mean a collection of release times and execution requirements of jobs of interest invoked by tasks in \( \tau \); 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 \( \gamma_i^q \) units of work before its absolute deadline of \( D_i^q \). The value of \( \gamma_i^q \) 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 \( \tau \) 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 \( \gamma_i^q \) during its execution window \([r_i^q, D_i^q]\) when the system exhibits the LO behavior and ii) every HI job \( J_i^q \) receives execution time \( \gamma_i^q \) during its execution window \([r_i^q, D_i^q]\) when the system exhibits the HI behavior. A task set \( \tau \) is said to be MC-feasible if every scenario is feasible.

According to Definition 1, a task set \( \tau \) is said to be MC-infaseible 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 \( t_{\text{end}} \).

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 \( t_{\text{end}} \), the demand of a SC task \( \tau \), with its WCET of \( C_\tau \), under the scenario of S1, S2 and S3’ with given \( t_{\text{end}} > 0 \) is calculated by \( \text{DBF}_\tau(t_{\text{end}}) \) where

\[
\text{DBF}_\tau(t) = \left( \left\lfloor \frac{t - D_\tau}{T_\tau} \right\rfloor + 1 \right) \cdot C_\tau, \tag{1}
\]

which yields the following lemma.

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

\[
\sum_{\tau_i \in \tau} \text{DBF}_\tau(t_{\text{end}}) \leq m \cdot t_{\text{end}}. \tag{2}
\]

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 FFDBF, \( t_{\text{end}} \) for a multiprocessor platform in [8], which is slightly larger than DBF, \( t_{\text{end}} \) only for some \( t_{\text{end}} \), we use DBF, \( t_{\text{end}} \) 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 \( \{0 < t_{\text{end}} < t_{\text{max}} \mid (t_{\text{end}} - D_i) \mod T_i = 0 \text{ for any } \tau_i \in \tau\} \) where \( t_{\text{max}} \) depends on \( \sum \frac{C_i}{T_i} \) [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 \( t_{\text{end}} \) such that checking Eq. (2) 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 \( \tau \) 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 L0 and H1 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 \( \tau \) is infeasible on a \( m \)-processor platform (where \( m \geq 1 \)), if Eq. (3) is violated for given \( t_{\text{end}} > 0 \):

\[
\sum_{\tau_i \in \tau} \text{DBF}_{\tau_i}(t_{\text{end}}^0) \leq m \cdot t_{\text{end}}^0, \tag{3}
\]

where \( \text{DBF}_{\tau_i}(t) = \left( \left\lfloor \frac{t - D_\tau}{T_\tau} \right\rfloor + 1 \right) \cdot C_\tau^0 \). \( \) Note that while the LHS of Eq. (2) is originally FFDBF, \( t_{\text{end}} \) for a multiprocessor platform in [8], which is slightly larger than DBF, \( t_{\text{end}} \) only for some \( t_{\text{end}} \), we use DBF, \( t_{\text{end}} \) 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 \( \{0 < t_{\text{end}} < t_{\text{max}} \mid (t_{\text{end}} - D_i) \mod T_i = 0 \text{ for any } \tau_i \in \tau\} \) where \( t_{\text{max}} \) depends on \( \sum \frac{C_i}{T_i} \) [17].

**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 \( C_\tau \) with \( C_\tau^0 \) for every \( \tau_i \in \tau \). □
Lemma 3. A MC task set $\tau$ is infeasible on a $m$-processor platform (where $m \geq 1$), if Eq. (4) is violated for given $t_{\text{end}} > 0$:

$$\sum_{i \in P_{\text{HI}}} DBP^H_i(t_{\text{end}}) \leq m \cdot t_{\text{end}},$$

where $DBP^H_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \cdot C^H_i$.

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 $C_j$ with $C^H_i$ for every $t_i \in P_{\text{HI}}$ and $C_i$ with 0 for every $t_i \in P_{\text{LO}}$.

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).

Example 1. Consider a task set $\tau$ with the following three tasks: $\tau_1(T_1 = 12, x_1 = \text{HI}, C^D_1 = 3, C^H_1 = 6, D_1 = 12), \tau_2 = \tau_1$, and $\tau_3 = (4, L_0, 1, 1, 4)$. 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 $J_1^2$, $J_2^2$, and $J_3^2$ are 4 and 8, 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 $J_1^2$ 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 $\tau_3$ in $[0,12]$ is 0 because no job of $\tau_3$ has its deadline at or before the mode change instant. Also, one of $J_1^1$ or $J_2^1$ 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 $\tau_3$, the total demand in $[0,12]$ is $C^D_1 + C^H_1 = 12$, 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 $\tau_3$ in $[0,12]$ is 1 because $J_3^1$ has its deadline no later than the mode change instant (while other jobs of $\tau_3$ 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 $\tau_3$ is 1, the total demand in $[0,12]$ is $C^D_1 + C^H_1 + C^H_1 = 13$, 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 $J_1^H$ and $J_2^H$ 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 $J_1^H$ (or $J_2^H$) 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: $τ_1(T_1 = 12, X_1 = HI, C^{LO}_1 = 3, C^{HI}_1 = 6, D_1 = 12), τ_2 = (12, HI, 3, 5, 12)$, and $τ_3 = (2, LO, 1, 1, 2)$. 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 $J_1^H$ or $J_2^H$ 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., $C^{LO}_3$), because $J_2^H$ is the only job whose deadline is no later than $t = 3$. Therefore, the total demand 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 $C^{LO}_1 + C^{HI}_1 + C^{HI}_2 = 12$.

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.

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

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-feasibility in Definition 1, but also to trigger the mode change at $t^* = 3$. 

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 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
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 \([t_k^*(J_k^*), t_k^*(J_k^*)]\), where \( t_k^*(J_k^*) \) and \( t_k^*(J_k^*) \) are respectively the earliest \( (t_k^* + C_k^\text{HI}) \) and the earliest \( (d_k^* - C_k^\text{HI} + C_k^\text{LO}) \) among every HI job \( J_k \) whose release time is no earlier than \( r_k^* \) (i.e., the release time of \( J_k^* \)).

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

Suppose that the mode change occurs at \( t^* < t_k^*(J_k^*) \). By S4 and the definition of \( t_k^*(J_k^*) \), 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 \( t_k^*(J_k^*) \). This contradicts the supposition.

Suppose that a mode change occurs at \( t^* > t_k^*(J_k^*) \). By S4 and the definition of \( t_k^*(J_k^*) \), there exists at least one job \( J_k^p \) whose execution requirement equals to its HI WCET, but whose actual execution time performed until \( (d_k^* - C_k^\text{HI} + C_k^\text{LO}) \) is less than \( C_k^\text{LO} \). If there exists no such job, the mode change occurs no later than \( r_k^* \) (i.e., the release time of \( J_k^* \)). 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 infeasible 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 job. 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 \( J_0^* \), and target a single mode change instant \( t^* \) belonging to \([t^*_0,(J_0^*)_0], (J_0^*)_0 \) (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 \( \tau_i \in \tau \) in \([0,t^*]\) is \( \text{DBF}_0^L(t^*) \).
- The demand of jobs of \( \tau_i \in \tau \) 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 \( \text{DBF}_1^L(t) \) in Eq. (3), the former also holds. \( \square \)

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 \( \tau_i \in \tau^\text{HI} \) into three categories:

**CG1.** jobs \( J_p^\text{p} \) whose deadline is no later than \( t^* \) (i.e., \( d_p^\text{p} \leq t^* \)),

**CG2.** at most one job \( J_p^\text{p} \) whose release time is earlier than \( t^* \) but whose deadline is later than \( t^* \) (i.e., \( r_p^\text{p} < t^* < d_p^\text{p} \)),

**CG3.** jobs \( J_p^\text{p} \) whose release time is no earlier than the mode change instant (i.e., \( t^* \leq r_p^\text{p} \)).

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 \( \text{OP}_1^L(t^*) \) and \( \text{OP}_1^L(t^*) \) denote the demand of a job of \( \tau_i \in \tau^\text{HI} \) 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 \( J_2^* \), and given \( t^* \in [t_c(J_2^*),t_0(J_2^*)] \). Note that \( \text{OP}_1^L(t^*) \) and \( \text{OP}_1^L(t^*) \) are 0, if no job of \( \tau_i \in \tau^\text{HI} \) 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 \( J_1^* \), and target a single mode change instant \( t^* \) belonging to
The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG1 in \([0, t^*] \) amounts to \( \text{DBF}^{\text{CG}}_{\tau^*}(t^*) \).

- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG1 in \([t^*, t^\text{end}] \) amounts to 0.

- The demand of at most one job of \( \tau \in \tau^{\text{HI}} \) belonging to CG2 in \([0, t^*] \) amounts to \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \).

- The demand of at most one job of \( \tau \in \tau^{\text{HI}} \) belonging to CG2 in \([t^*, t^\text{end}] \) amounts to \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \).

- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG3 in \([0, t^*] \) amounts to 0.

- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG3 in \([t^*, t^\text{end}] \) amounts to \( \text{DBF}^{\text{CG}}_{\tau^*}(t^\text{end} - [t^*/T_1] \cdot T_1) \).

**Proof.** Since every \( \tau \) job belongs to CG1 executes for its \( \text{LO} \) 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 \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) and \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \). Among jobs of \( \tau \) belonging to CG3, the earliest release time is \( ([t^*/T_1] \cdot T_1) \); considering the definition of \( \text{DBF}^{\text{CG}}_{\tau^*}(t) \) in Eq. (4), the sixth statement holds. \( \square \)

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 consider. Considering the sum of \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) and \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) equals to the execution requirement of the job of \( \tau \) belonging to CG2 (addressing the second part of I4), we can derive the constraints of the contribution of \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) and \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) 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 \( \text{OP}^{\text{CG}}_{\tau^*}(7) \) and \( \text{OP}^{\text{CG}}_{\tau^*}(7) \) for \( J_1 \). If \( J_1 \) triggers the mode change, the job should execute for exactly 3 time units (i.e., \( C_{\text{HI}}^{(7)} \)) in \([0,7] \) and 3 time units (i.e., \( C_{\text{LO}}^{(7)} - C_{\text{HI}}^{(7)} \)) in \([7,12] \), implying \( \text{OP}^{\text{CG}}_{\tau^*}(7) = 3 \) and \( \text{OP}^{\text{CG}}_{\tau^*}(7) = 3 \). We discuss the case where \( J_1 \) does not trigger the mode change, from now on, which is illustrated in Fig. 2c.

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

Inspired by the example, the following lemma formalizes the constraints of the contribution of \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) and \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \) 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 \([t^*_{\text{m}}(J_k^1), t^*_{\text{m}}(J_k^2)] \) (defined in Lemma 5). Then, the demand of jobs of a \( \tau \) task, in \([0, t^*] \) and \([t^*, t^\text{end}] \) can be calculated/expressed as follows.

- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG1 in \([0, t^*] \) amounts to \( \text{DBF}^{\text{CG}}_{\tau^*}(t^*) \).
- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG1 in \([t^*, t^\text{end}] \) amounts to 0.
- The demand of at most one job of \( \tau \in \tau^{\text{HI}} \) belonging to CG2 in \([0, t^*] \) amounts to \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \).
- The demand of at most one job of \( \tau \in \tau^{\text{HI}} \) belonging to CG2 in \([t^*, t^\text{end}] \) amounts to \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) \).
- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG3 in \([0, t^*] \) amounts to 0.
- The demand of jobs of \( \tau \in \tau^{\text{HI}} \) belonging to CG3 in \([t^*, t^\text{end}] \) amounts to \( \text{DBF}^{\text{CG}}_{\tau^*}(t^\text{end} - [t^*/T_1] \cdot T_1) \).

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

(Cases 2 and 3) The cases can be proved by utilizing \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) + \text{OP}^{\text{CG}}_{\tau^*}(t^*) = C_{\text{LO}}^{(7)} \) and \( \text{OP}^{\text{CG}}_{\tau^*}(t^*) + \text{OP}^{\text{CG}}_{\tau^*}(t^*) = C_{\text{HI}}^{(7)} \), 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. \( \square \)

Combining Lemmas 7 and 8, we can calculate the demand of all \( \tau \) tasks in \([0, t^*] \) and \([t^*, t^\text{end}] \), by adding the demand of jobs of each \( \tau \) task in CG1, CG2 and CG3. By combining the demand of all \( \tau \) tasks and that of all \( \tau \) 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
check all possible combinations of \( OP_{-}^{r}(t^{*}) \) and \( OP_{+}^{r}(t^{*}) \) for every \( r \in \mathbb{R}^{1} \). Since there are multiple options of a HI job's contribution to the demand in \([0, t^{*}]\) and that in \([t^{*}, t^{\text{end}}] \), 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 \( t \in \mathbb{R}^{1} \) do
   2. if \( t \) satisfies (i) and (ii) of Subcase 3A in Lemma 8, and every \( t \in \mathbb{R}^{1} - \{t\} \) belonging to Case 3 can satisfy (i'), (ii') and (iii) of Subcase 3B then
     3. \( \text{SumOP}^{r} \leftarrow C_{k}^{r} \left[ \sum_{t \in \mathbb{R}^{1}(t)} \max \text{OP}_{-}^{r}(t^{*}) \right] \)
     4. \( \text{SumOP}^{r} \leftarrow C_{k}^{r} \left[ \sum_{t \in \mathbb{R}^{1}(t)} \max \text{OP}_{+}^{r}(t^{*}) \right] \)
     5. \( \text{DiffLO} \leftarrow \max \left(0, \sum_{t \in \mathbb{R}^{1}(t)} \text{DBP}^{r}(t^{*}) + \text{SumOP}^{r} - m \cdot \text{t}^{*} \right) \)
     6. \( \text{DiffHI} \leftarrow \max \left(0, \sum_{t \in \mathbb{R}^{1}(t)} \text{DBF}^{r}(t^{*}) + \text{SumOP}^{r} - m \cdot \text{t}^{*} \right) \)
     7. 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 \( t_{k} \in \mathbb{R}^{1} \), …”). By checking there exists a task \( t_{k} \) such that \( t_{k} \) satisfies Conditions (i) and (ii) of Subcase 3A in Lemma 8 (i.e., \( t_{k} \) 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 \( t_{k} \), we conclude that the “if” statement of Lemma 9 is true.

On the other hand, under a multiprocessor platform (i.e., \( m \geq 2 \)), 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 \( 1 \leq x \leq m \)) 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 UB2' and LB2' in Conditions (i) and (ii) of Subcase 3B are replaced with UB2' and LB2', respectively as follows:

\( (i') \text{ LB2}^{-} \preceq \text{OP}_{-}^{r}(t^{*}) \preceq \text{UB2}^{-}, \) and
\( (ii') \text{ LB2}^{r} \preceq \text{OP}_{+}^{r}(t^{*}) \preceq \text{UB2}^{r}, \) where
\( \text{UB2}^{r} = \min \left( t^{*} - t^{*}, C_{k}^{r} \right), \) and \( \text{LB2}^{r} = C_{k}^{r} - \text{UB2}^{r}. \)

By changing the upper-limit of \( \text{OP}_{-}^{r}(t^{*}) \) from \( C_{k}^{r} - 1 \) in UB2 to \( C_{k}^{r} \) in 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
return FALSE in Line 13, if every $t_k \in r_{HI}$ 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 $r$ in $[0, t_{end}]$. The details are shown in the supplement, available online.

The time-complexity of Algorithm 1 is $O(n^2)$, and there are at most $\max_{r \in \text{grid}} D_r$ choices of $t$ for any given $J'_k$ in Theorem 1. Therefore, we test Theorem 1 using Algorithm 1, the time-complexity of Theorem 1 is $O(n^2 \cdot \max_{r \in \text{grid}} D_r)$.

While Theorem 1 focuses on the scenario of $S_1$–$S_4$ with given $t_{end} > 0$ and $J'_k$, we can find more infeasible task sets if we apply every pair of $t_{end} > 0$ and $J'_k$. The following lemma calculates an upper-bound of $t_{end} > 0$ 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 $\left(\sum_{i \in \text{grid}} (T_i - D_i) \cdot C_{LO_i} / T_i + \sum_{i \in \text{grid}} C_{HI_i} \right) / (m - \sum_{i \in \text{grid}} C_{HI_i} / T_i)$. Also, if Eq. (6) is violated with any $(t_{end} - t')$, it is also violated with some $(t_{end} - t')$ which is smaller than $\left(\sum_{i \in \text{grid}} (T_i - D_i) \cdot C_{HI_i} / T_i + C_{HI_i} / (m - \sum_{i \in \text{grid}} C_{HI_i} / T_i)\right)$.

**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 $S_1$–$S_4$ with all possible pairs of $t_{end} > 0$ and $J'_k$ with reasonable time-complexity, yielding the following collective necessary feasibility test.

**Theorem 2.** A MC task set $\tau$ is infeasible, if the following collective necessary feasibility test finds at least one pair of $t_{end} > 0$ 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 $t_{end} > 0$ and $J'_k$ that satisfies (i) $t_{end}$ is less than the sum of upper bounds of $t'$ and $(t_{end} - t')$ 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 $t_{end} > 0$ and $J'_k$” in Theorem 2. First, by adding two upper-bounds for $t'$ and $(t_{end} - t')$ in Lemma 11, we have an upper-bound for $t_{end}$. Second, we target every job $J'_k$ whose execution window overlaps with $[0, t_{end}]$. 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.
For a given \( t^{\text{end}} > 0 \) and \( J^*_k \), Theorem 1 conducts Algorithm 1 at most \( \max_{t, e} D_l \) times, resulting in \( O(n^2 \cdot \max_{t, e} D_l) \) as explained. As proved in Lemma 11, the upper-bounds on \( t^{\text{end}} \) and \( (t^{\text{end}} - t^*) \) are a pseudo-polynomial in the size of task parameters (if the following condition holds: \( \sum_{i \in T} C_i^{LO}/T_i \) and \( \sum_{i \in HI} C_i^{HI}/T_i \) are upper-bounded by some constant strictly less than \( n \)). 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: \( t_1(T_1 = 12, x_1 = HI, C_1^{LO} = 3, C_1^{HI} = 6, D_1 = 12) \), \( t_2 = t_1 \), and \( t_3 = (L0, 2, 2, 4) \).

Consider the scenario of S1–S4 with \( t^{\text{end}} = 12 \) and \( J^*_k = J^*_1 \). According to Example 1 where its \( t_1 \) and \( t_2 \) are the same as those in this example, \( t_3^*(J_3^*) = 3 \) and \( t_3^*(J_3) = 9 \) 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., \( t_2 \)). \( t_3 \) invokes its jobs strictly-periodically, but one of the jobs has its deadline at \( t = 3 \). Then, \( r_3^2 \) and \( d_3^2 \) for \( J^*_3 \) are \( -1 \) and \( 3 \); \( r_3^3 \) and \( d_3^2 \) for \( J^*_3 \) are \( 3 \) and \( 7 \); and \( r_3^3 \) and \( d_3^3 \) for \( J^*_3 \) 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 \( J^*_3 \) (i.e., \( t = 3 \)) is no later than the mode change instant and therefore \( J^*_3 \) contributes to the demand in \([0, 3]\).

### 7.2 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 \( t^{\text{end}} \). Second, every LO task has a job with its absolute deadline of \( t = t^*_0(J^*_k) \) 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 \( t^{\text{end}} \).

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., \( \sum_{t \in LO} \text{DBF}_{LO}(t^*) \)) 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 \([t^*_0(J^*_k), t^*]\) and \([0, t^*_0(J^*_k)]\). Since there exists a job of \( t_i \in t^{LO} \), whose deadline is \( t^*_0(J^*_k) \), the next job’s release time is \( t^*_0(J^*_k) + (T_i - D_i) \). Therefore, the demand in \([t^*_0(J^*_k), t^*]\) is calculated by \( \text{DBF}_{LO}(0, t^* - t^*_0(J^*_k) - (T_i - D_i))^\infty \). The demand of all jobs of \( t_i \in t^{LO} \) whose release time is no earlier than \( t = 0 \) in \([0, t^*_0(J^*_k)]\) is simply calculated by \( \sum_{t \in t^{LO}} \text{DBF}_{LO}(t^*) \), and the demand of a job (if any) of \( t_i \in t^{LO} \) whose release time is earlier than \( t = 0 \) but whose deadline is later than \( t = 0 \) is calculated by \( \sum_{t \in t^{LO}} \left( \text{DBF}_{LO}(0, t^* - t^*_0(J^*_k) - (T_i - D_i))^\infty \right) \).

### 7.3 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 \( a[^{\text{max}}(a, b, c)] \) to denote \( \max(\min(a, b, c)) \).
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 \( t' \in [0, t^*] \) 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. \( \Box \)

**Theorem 3 (Another necessary feasibility condition).** A MC task set \( \tau \) 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. \( \Box \)

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 \( \tau \) is infeasible, if the following combined 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) \( t^{\text{end}} \) 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. \( \Box \)

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 \( t^{\text{end}}\) in \([0, t^{\text{end}}]\) is upper bounded by \( \sum_{t_i \in HI} DBF^{HI}(t^{\text{end}}) \), which is independent of \( J'_k \). Second, we only check \( t' = t^*_a(J'_k) \); considering \( DBF^{HI}(t') \) 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 \( \min_{t_i \in HI} C_i^{LO} \). By adding Eqs. (5) and (6) under the scenario of S1–S4 with given \( t^{\text{end}} \), the targeted \( J'_k \), and \( t' = t^*_a(J'_k) \), we derive the following combined inequality:

\[
\sum_{t_i \in HI} DBF^{HI}(t^*_a) + \sum_{t_i \in HI} DBF^{HI}(t^{\text{end}}) \leq m \cdot t^{\text{end}},
\]

where \( t^*_a = \min_{t_i \in HI} C_i^{LO} \). Then, the following theorem holds.

**Theorem 5.** A task set \( \tau \) is infeasible, if there exists \( t^{\text{end}} \geq t^*_a = \min_{t_i \in HI} C_i^{LO} \) 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. \( \Box \)

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):

\[
\sum_{t_i \in HI} (DBF^{HI}(t^*_a) + [(t^*_a + T_i - D_i) \mod T_i - (T_i - C_i^{LO})]_{0}^{\infty})
\]

\[
+ \sum_{t_i \in HI} DBF^{HI}(t^{\text{end}}) \leq m \cdot t^{\text{end}},
\]

which yields the following theorem.

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

**Proof.** The proof is the same as that of Theorem 5 except the demand of LO tasks. \( \Box \)

It is trivial that the time-complexity of testing Theorem 5 or 6 with all possible \( t^{\text{end}} \) is equivalent to that of testing Lemma 1 with all possible \( t^{\text{end}} \). 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.

\[\text{Proof.}\]

6. To derive a condition for Eq. (7) corresponding to the condition for Eq. (5) in Lemma 11, it is sufficient to add \( \sum_{t_i \in LO} C_i^{LO} \) in the numerator for the upper bound of \( t' \).
Lemma 13. If a task set \( \tau \) is deemed infeasible by Lemma 3, \( \tau \) 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. \( \square \)

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 \( \tau \) 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 \( \tau_3 \) with \( \tau_3' = (4, L0, 1, 1, 4); \) also, we consider \( t' = 4 \). Then, the former yields the demand of one time unit by \( t' \) in \([0,4]\) (which is one \( L0 \) WCET of \( \tau_3 \)). On the other hand, the latter yields the demand of zero time unit by \( t' \) in \([0,4]\) as follows. Since the first job of \( \tau_3' \) (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 \( \tau_3' \) 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. \( \square \)

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 \( \tau \) deemed infeasible by Theorem 5 is also deemed infeasible by Theorem 6.

Proof. Suppose that there is a task set \( \tau \) deemed infeasible by Theorem 5. Then, according to Theorem 5, there exists \( t^\text{end} \geq t^*_u = \min_{t \in \mathbb{N}^+} C_l^{L0} \) which makes Eq. (8) false. Since the LHS of Eq. (9) is always no smaller than that of Eq. (8), such \( t^\text{end} \) also makes Eq. (9) false. Therefore, the lemma holds. \( \square \)

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 \( \tau \) 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 \( J_k^i \) and \( t^* \) for given \( \theta^\text{end} \), 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 \( J_k^i \) 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. \( \square \)

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 \( \tau \), having \( \chi_i = \mathbb{H} \in \{0.3, 0.5, 0.7\} \), (iv) the maximum ratio (CF) of each \( \mathbb{H} \) task \( \tau \)'s \( \mathbb{H} \) WCET to \( L0 \) WCET, i.e., \( CF = C_l^{L0} / C_l^{H0} \in \{1.25, 1.5, 2, 3, 4\} \), (v) \( L0 \) total utilization \( U^{L0} = \sum_{t \in \mathbb{N}^+} C_l^{L0} / T_t \), and (vi) \( \mathbb{H} \) total utilization \( U^{H0} = \sum_{t \in \mathbb{N}^+} C_l^{H0} / T_t \). We choose \( U^{L0} \) and \( U^{H0} \) from \((m - 0.55)\) to \( m \) with an incremental step of 0.05 resulting in 12 choices, respectively.

Given a 6-tuple \((m, n, CP, CF, U^{L0}, U^{H0})\) for a task set, each task parameter is determined as follows: \( T_i \) is uniformly chosen in \([1,1000]\); \( \chi_i \) is selected as \( \mathbb{H} \) with probability \( CP \) (and as \( L0 \) with probability \((1.0 - CP)\)); \( C_l^{L0} \) is determined by using the UUniFast-Discard algorithm [25], [26]; \( C_l^{H0} \) is uniformly chosen in \([C_l^{L0} + 1, CF \cdot C_l^{L0} + 1]\); \( \mathbb{H} \) is \( \mathbb{H} \) (and set to \( C_l^{L0} \), 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 \( L0 \) and \( \mathbb{H} \) total utilizations are in \([U^{L0} - 0.05, U^{L0}]\) and \([U^{H0} - 0.05, U^{H0}]\) for given \( U^{L0} \) and \( U^{H0} \), 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
choosing $D_i$ uniformly in $[c_{i1}, T_i]$. 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.

### 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 $CP=0.3$. In $(U^{LO}, U^{HI})$-plane, we focus on the region where $0.75 < \max(U^{LO}, U^{HI}) \leq 1.0$, yielding 95,000 task sets of interest among 144,000 generated task sets. This is because all task sets with $\max(U^{LO}, U^{HI}) < 0.75$ and those with $\max(U^{LO}, U^{HI}) > 1.00$ 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 $0.95 \leq U^{LO} \leq 1.0$ and $0.95 \leq U^{HI} \leq 1.0$, 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 $0.75 \leq U^{LO} \leq 1.0$ and $0.4 \leq U^{HI} \leq 0.75$, 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 hand, MC-NFT-S and MC-NFT*-S are largely dependent on the interval of $HI$, and they do not always behave well when $HI$ behavior is not significant. Therefore, we find that MC-NFT-S and MC-NFT*-S are often not competitive with other tests when dealing with tasks having both $LO$ and $HI$ behaviors.

### 10.2.1 Comparison of Detection Ratios

We compare the detection ratios of all necessary feasibility tests for implicit-deadline task sets with different combinations of $m$, $n$, $CP$, $CF$, $U^{LO}$, and $U^{HI}$ in $(U^{LO}, U^{HI})$-plane. The results are shown in Table 1.

### Table 1

<table>
<thead>
<tr>
<th>$n$</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC-NFT</td>
<td>19</td>
<td>41</td>
<td>77</td>
<td>75</td>
</tr>
<tr>
<td>MC-NFT*</td>
<td>16</td>
<td>55</td>
<td>79</td>
<td>137</td>
</tr>
<tr>
<td>MC-NFT-S</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>MC-NFT*-S</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**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 $CF=3$.**

**Fig. 4.** Detection ratio of the four proposed necessary feasibility tests for constrained-deadline task sets with different ranges of $min(U^{LO}, U^{HI})$ when $m=1$, $n=4$, $CP=0.3$ and $CF=3$. A new test is needed if the existing test fails to prove infeasibility.
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 L0 and H1 behavior, resulting in higher capability in finding infeasible task sets over a wider range of L0 and H1 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 \(\min(ULO, UHI)\) 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 H1 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 \(\min(ULO, UHI)\) increases; for example, using MC-NFT\(^*\), 88.8% and 91.7% of the task sets are proven infeasible with \(\min(ULO, UHI)\) 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 \(\min(ULO, UHI)\). Third, MC-NFT and MC-NFT\(^*\) are shown to outperform MC-NFT-S and MC-NFT\(^*\)-S, respectively, for all values of \(\min(ULO, UHI)\). This is because MC-NFT and MC-NFT\(^*\) (i) derive a tighter bound on the demand of H1 jobs by considering the relationship between the mode change instant and each H1 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
have the following observations. First, we confirmed that the similar tendency is observed for multiprocessor platforms with that for uniprocessor platforms as \( \min(U^{L}, U^{R}) \) 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 \( n = 8 \) or more, it was very time-consuming to generate enough valid task sets by using the UUniFast-Disard 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


