Journal of Systems Architecture 135 (2023) 102808

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Tight necessary feasibility analysis for recurring real-time tasks on a multiprocessor

Hoon Sung Chwa [a], Jinkyu Lee [b,*]

[a] Department of Electrical Engineering and Computer Science, DGIST, Republic of Korea
[b] Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea

ARTICLE INFO

Keywords:
Necessary feasibility analysis
Multiprocessor platform
Real-time systems
Recurring real-time tasks

ABSTRACT

One of the important design issues for time-critical embedded systems is to derive necessary conditions that meet all job deadlines invoked by a set of recurring real-time tasks under a computing resource (called *feasibility*). To this end, existing studies focused on how to derive a tight lower-bound of execution requirement (i.e., *demand*) of a target set of real-time tasks. In this paper, we address the following question regarding the *supply* provided by a multiprocessor resource: is it possible for a real-time task set to *always* utilize all the provided supply? We develop a systematic approach that i) calculates the amount of supply proven unusable, ii) finds a partial schedule that yields a necessary condition to minimize the amount of unusable supply, and iii) uses the partial schedule to further reclaim unusable supply. While the systematic approach can be applied to most (if not all) recurring real-time task models, we show two examples how the approach can yield tight necessary feasibility conditions for the sequential task model and the gang scheduling model. We demonstrate the proposed approach finds a number of additional infeasible task sets which have not been proven infeasible by any existing studies for the task models.

1. Introduction

Nowadays, we have witnessed an increasing trend in system developments towards multiprocessor computing environments. One of the most important issues for time-critical system design is how to utilize a multiprocessor computing resource efficiently so as to accommodate as many real-time tasks subject to timing constraints as possible [1,2], which has been influenced by the increasing demand for safety-critical systems such as autonomous vehicles.

To address the issue, the real-time systems community have focused on "feasibility" analysis to determine whether every instance of recurring real-time tasks finishes its execution within its deadline under a computing resource and have sought two research directions: (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) [3], 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) [4–10].

The goal of this paper is to reduce a set of task sets whose feasibility is unknown by existing studies. In particular, we aim at developing *necessary feasibility* tests that prove infeasibility of task sets on a multiprocessor (also called *infeasibility* tests).

Addressing necessary feasibility for real-time task systems is beneficial both from the theoretical and the practical point of view [9,10]. On the theoretical side, tight necessary feasibility analysis 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 designers set the configuration for scheduling algorithms, task parameters, and computing resources, tight necessary feasibility results reduce the burden of tuning parameters for the configuration by excluding some infeasible choices of the configuration.

A typical way to derive necessary feasibility conditions is *demand–supply* comparison, which is, to compare a lower-bound of execution requirements (i.e., *demand*) of a target set of real-time tasks in an interval, with the amount of execution capability (i.e., *supply*) provided by a target computing resource in the interval; if the former is larger than the latter, the task set is deemed infeasible by any scheduling algorithm on the resource. Existing studies have focused on finding a tighter lower-bound of *demand* of a task set (e.g., demand bound function [4], forced-forward demand bound function [7]), but they assume that the *supply* is consistently provided by a computing resource (i.e., $m \times t$ amount of supply provided in an interval of length t by m processors) [4–8,10]. The approach was successful for a uniprocessor

* Corresponding author.
E-mail addresses: chwahs@dgist.ac.kr (H.S. Chwa), jinkyu.lee@skku.edu (J. Lee).

https://doi.org/10.1016/j.sysarc.2022.102808
Received 20 June 2022; Received in revised form 4 November 2022; Accepted 14 December 2022
Available online 20 December 2022
1383-7621/

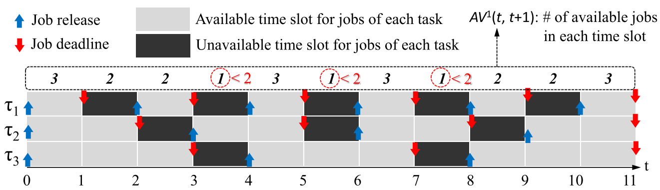

Fig. 1. Available time slots for jobs of each sequential task $\tau_1(T_1 = 2, C_1 = 1, D_1 = 1)$, $\tau_2(3, 2, 2)$ and $\tau_3(4, 2, 3)$, and the number of available jobs in each time slot.

resource in that a necessary *and* sufficient feasibility condition was derived for the most popular real-time task model [4]. However, the approach cannot address the following important question regarding the supply provided by a multiprocessor resource: is it possible for a task set to *always* utilize all the provided supply?

The question can be answered by an example in Fig. 1, which illustrates time slots for each task that is *available* to be executed and the number of *available* tasks in each time slot[1]; a task is said to be *available* in a time slot, if there exists the task's job whose execution window between its release time and deadline subsumes the time slot. In the time slots $[3, 4)$, $[5, 6)$ and $[7, 8)$, the number of available tasks is only one, meaning that the supply provided by two processors cannot be fully utilized by the task set because other tasks cannot have their jobs that are available in those slots. Therefore, among $2 \times 11 = 22$ amount of supply provided by two processors in $[0, 11)$, the task set can utilize at most $22 - 3 = 19$ amount of supply. Since tasks $\tau_1(\text{period} = 2, \text{relative deadline} = 1, \text{execution time} = 1)$, $\tau_2(3, 2, 2)$ and $\tau_3(4, 2, 3)$ need to execute 1×6 jobs=6, 2×4 jobs=8, and 2×3 jobs=6 amount of executions in $[0, 11)$, respectively, the amount of demand (i.e., 6+8+6=20) is strictly larger than an upper-bound of the supply (i.e., 19), making the task set infeasible on two processors regardless of scheduling algorithms. However, no existing necessary feasibility conditions have proven the infeasibility because they have failed to consider such unusable supply. This paper proposes the first approach to derive a tighter upper-bound of the supply that can be utilizable by a task set on a multiprocessor resource.

To this end, this paper addresses the following technical issues (in Section 3):

- Q1. How to calculate the amount of supply trivially proven unusable,
- Q2. How to find a set of jobs to be necessarily executed in each time slot in order to maximally utilize the supply,
- Q3. How to use the set of jobs in each time slot to further reclaim unusable supply, and
- Q4. How to develop a systematic way to fully utilize the synergy of the answers of Q2 and Q3.

While the proposed approach can be applicable to most (if not all) real-time recurring task models, we demonstrate how the proposed approach improves the state-of-the-art necessary feasibility conditions for two popular task models: the sequential task model [11] and the gang task model [12] (in Section 4). Our simulation results show that the proposed approach finds 297,622 (25%) and 980,205 (82%) additional infeasible task sets among each of 1,200,000 generated sequential and gang task sets, respectively, which have not been proven infeasible by any existing studies (in Section 5).

This paper makes the following contributions.

- Development of the first approach to derive tight upper-bounds of the supply for necessary feasibility conditions,
- Establishment of a general methodology for identifying unusable supply, which can be incorporated into existing necessary feasibility tests for recurring real-time task models,
- Development of improved necessary feasibility tests for two representative task models by applying our approach, and
- Demonstration of effectiveness of the proposed tests in finding infeasible task sets and thus in narrowing the area of uncertainty between the task sets proven feasible and those proven infeasible.

[1] The example adopts the sequential task model [11] to be detailed in Section 2.1.

Table 1
Summary of notation.

τ; $\mathcal{J}$	Task set; Job set
m	# of processors
T_i	Task period
D_i	Task relative deadline
C_i	Task execution time
v_i	# of threads to be concurrently executed
$AV^x(t, t+1)$	# of x-depth-available jobs in $[t, t+1)$
$PJ^x(t, t+1)$	Partial job set in $\mathcal{J}$ in $[t, t+1)$, associated with $AV^x(t, t+1)$ (see Definition 5)
$CJ(t, t+1)$	Complete job set in $\mathcal{J}$ scheduled in $[t, t+1)$
VS ; IVS	Valid schedule; Invalid schedule

2. System model and background

2.1. System model

We describe our system model and notation, summarized in Table 1. This paper develops tight upper-bounds of supply provided by a resource of $m \geq 2$ processors (to be detailed in Section 3), which can be incorporated into existing necessary feasibility conditions for most (if not all) recurring real-time task models, including the sequential task model [11], the parallel task model [8], the gang task model [12], and the mixed-criticality sequential task model [9,10]. In those task models, a task τ_i in a task set τ invokes a series of sporadic jobs, and two consecutive jobs of τ_i are separated by at least T_i time units (called the period or the minimum inter-job separation).

Among the task models, we choose the sequential task model and the gang task model, to demonstrate how the new supply upper-bounds improve the state-of-the-art necessary feasibility conditions in Section 4. In the sequential task model, τ_i has two more parameters in addition to T_i: the relative deadline D_i and the execution time C_i. Once a job of τ_i is released at r_i, it should execute for at most C_i time units until $r_i + D_i$; whenever a job of τ_i is executed, the job occupies only one processor. On the other hand, τ_i subject to the gang task model has one more parameter in addition to T_i, D_i and C_i, which is v_i, the number of threads of each job of τ_i. Whenever a job of τ_i is executed, its threads are concurrently executed on v_i processors. Note that we do not have any restriction between T_i and D_i for both sequential and gang task models (i.e., we consider arbitrary-deadline tasks).

In this paper, we focus on global preemptive scheduling; a job can be executed on any processor, and a higher-priority job can preempt the execution of any lower-priority job. We assume that each preemption does not incur any overhead, and the quantum length is 1, meaning that all task parameters are integer values.

For recurring real-time task models, in general, each task set may legally generate infinitely many different job arrival sequences since the actual inter-arrival time between any two consecutive jobs of each task can vary (due to the inter-job separation of *at least* T_i time units). A task set is deemed *infeasible* if there exists a single job arrival sequence that cannot meet at least one job deadline under global preemptive scheduling; and it is deemed *feasible* otherwise.

2.2. Background

There have been proposed many multiprocessor "feasibility" analysis techniques that determine whether every job of real-time tasks finishes its execution within its deadline (see [3] for a survey). A typical way to develop a feasibility test is to compare the sum of every job's 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*). A task set is feasible on a multiprocessor computing platform if and only if the demand of the task set does not exceed the supply of the computing platform.

Horn [13] derived the exact necessary and sufficient condition for the feasibility of the *implicit-deadline* sequential task model, where each task has its relative deadline equal to its period (i.e., $D_i = T_i$), as follows:

$$u_{sum} \leq m, \tag{1}$$

where $u_{sum} = \sum_{\tau_i \in \tau} C_i/T_i$. Eq. (1) implies that a task set is feasible on a m-processor platform if and only if the resource demand of the task set (captured by u_{sum}) does not exceed the resource supply of the platform (captured by m).

Necessary feasibility analysis. For the *constrained-deadline* sequential task model (i.e., $D_i \leq T_i$), the term u_{sum} is still a lower bound on resource demand, but not an upper bound. That is, Eq. (1) is a necessary condition for the feasibility, but it is not sufficient.

Baruah and Fisher [14] proposed a tighter necessary condition with the concept of the processor load (denoted by $\mathsf{LOAD}(\tau)$) as follows:

$$\mathsf{LOAD}(\tau) \leq m, \tag{2}$$

where $\mathsf{LOAD}(\tau)$ represents the maximum cumulative resource demand normalized by the interval length. $\mathsf{LOAD}(\tau)$ can be expressed as

$$\mathsf{LOAD}(\tau) = \max_{\forall t>0}\left(\frac{\sum_{\tau_i \in \tau} \mathsf{DBF}(\tau_i, t)}{t}\right), \tag{3}$$

where the demand bound function of τ_i for an interval of length t (denoted by $\mathsf{DBF}(\tau_i, t)$) is

$$\mathsf{DBF}(\tau_i, t) = \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \times C_i. \tag{4}$$

Fisher et al. [6] showed how to approximate the processor load in an efficient way.

Baker and Cirinei [7] further improved upon the necessary feasibility condition in Eq. (2) by considering a synchronous arrival sequence with the force-forward demand bound function (FFDBF) as stated in Eq. (8) to be presented. This is the state-of-the-art necessary feasibility analysis for the sequential task model.

Note that most studies for multiprocessor necessary feasibility analysis focused on the sequential task model explained so far, but there exist some studies for the gang task model [12], the parallel task model [8], and mixed-criticality task model [10].

All of the existing studies focused on finding a tighter lower-bound of *demand* of a task set, but they assumed that the *supply* provided by a computing resource is stationary (i.e., $m \times t$ amount of supply provided in an interval of length t by m processors). On the other hand, this paper proposes the first approach to derive a tighter upper-bound of the supply that can be utilizable by a task set on a multiprocessor platform.

3. Deriving tight supply upper-bounds

Generalizing the motivational example in Section 1, this section proposes a systematic way to calculate tight upper-bounds of the amount of supply to be actually utilizable by a set of jobs $\mathcal{J}$, a single instance of job arrival sequences generated by a set of recurring real-time tasks τ. Although we target $\mathcal{J}$ subject to the sequential task model in the examples to be explained, all the theories to be developed in this section are valid for $\mathcal{J}$ subject to any recurring real-time task models mentioned in Section 2.1. As the first step, we define the notion of *availability* for each job.

Definition 1 (1-*Depth-Availability*)**.** Consider a job set $\mathcal{J}$ invoked by a task set τ. A job is said to be *available* or 1-*depth-available* in $[t, t+1)$, if the job satisfies the following condition.

- The time slot $[t, t+1)$ belongs to the interval between the job's release time and deadline.

Let $AV^1(t, t+1)$ denote the number of 1-depth-available jobs in $\mathcal{J}$ in $[t, t+1)$.

For example, τ_3 in Fig. 1 has available jobs in $[t, t+1)$ where $t = 0, 1, 2, 4, 5, 6, 8, 9$ and 10. Also, $AV^1(t, t+1) = 1$ holds for $t = 3, 5$ and 7, while $AV^1(t, t+1) \geq 2$ holds for other t in the figure. Since $AV^1(t, t+1)$ depends on the job release patterns of $\mathcal{J}$ but does not depend on scheduling algorithms, we can utilize the notion for deriving an upper-bound of supply regardless of scheduling algorithms.

Using the property that $AV^1(t, t+1) < m$ implies the job set $\mathcal{J}$ cannot fully utilize m processors in $[t, t+1)$, the following theorem calculates an upper-bound of the amount of supply to be utilized by $\mathcal{J}$.

Theorem 1. *Consider a job set $\mathcal{J}$ invoked by a task set τ on an m-processor resource. If $AV^1(t, t+1) < m$, the supply able to service $\mathcal{J}$ in $[t, t+1)$ is at most $AV^1(t, t+1)$, not m. Therefore, the amount of supply able to service $\mathcal{J}$ in $[0, t)$ by an m-processor platform is upper-bounded by $SB^1(t)$, where*

$$SB^1(t) \stackrel{def}{=} m \times t - \sum_{t_a=0}^{t-1} \max\left(0, m - AV^1(t_a, t_a+1)\right). \tag{5}$$

Proof. Suppose $\mathcal{J}$ which satisfies $AV^1(t_a, t_a+1) < m$ occupies more than $AV^1(t_a, t_a+1)$ processors in $[t_a, t_a+1)$. By the definition of $AV^1(t_a, t_a+1)$ with the definition of the 1-depth-availability of a job, this contradicts that a job can be executed between its release time and deadline. Therefore, $\mathcal{J}$ can occupy at most $AV^1(t_a, t_a+1)$ processors when $AV^1(t_a, t_a+1) < m$ and at most m processors when $AV^1(t_a, t_a+1) \geq m$, respectively. By applying t_a to 0, 1, 2, ... $t-1$, the theorem holds. □

While most existing necessary feasibility conditions that employ demand–supply comparison [4–10] assume $m \times t$ amount of supply for an interval of length t, Theorem 1 derives the amount of supply no larger than $m \times t$, yielding tighter necessary feasibility conditions for recurring real-time task models to be detailed in Section 4. However, there is room for further improvement to be explained in the following example.

Example 1. We add the fourth task to the task set of the example in Fig. 1. That is, we consider four sequential tasks $\tau_1(T_1 = 2, C_1 = 1, D_1 = 1)$, $\tau_2(3, 2, 2)$, $\tau_3(4, 2, 3)$ and $\tau_4(8, 1, 6)$, to be scheduled on a two-processor resource. From $t = 0$, the four tasks invoke a series of jobs periodically as shown in Fig. 2(a). In this case, $[7, 8)$ is the only time slot in which the task set cannot fully utilize a two-processor resource (i.e., $AV^1(7, 8)$=1). Therefore, the four tasks can utilize at most $2 \times 11 - 1 = 21$ amount of supply in $[0, 11)$. Since τ_1, τ_2, τ_3 and τ_4 need to execute 1×6 jobs=6, 2×4 jobs=8, 2×3 jobs=6, 1×1 job=1 amount of executions in $[0, 11)$, respectively, the amount of demand (i.e., $6+8+6+1 = 21$) is not larger than the upper-bound of the supply (i.e., 21). Therefore, we cannot judge that the task set is infeasible on two processors; note that no existing necessary feasibility conditions have proven the infeasibility as well.

However, we need to scrutinize whether all the 21 amount of supply in $[0, 11)$ can be actually utilizable by the four tasks. This necessitates the following conditions: if $AV^1(t, t+1) \leq m$, all the available tasks in $[t, t+1)$ should be executed in $[t, t+1)$. For example, since $AV^1(3, 4) = 2$ holds, the job of τ_2 and that of τ_4 should be executed in $[3, 4)$; otherwise, we cannot fully utilize the two-processor resource in $[3, 4)$. Once we decide to execute the job of τ_4 in $[3, 4)$, the job cannot be executed in other time slots (due to $C_4 = 1$), meaning that we can deduct the contribution of the job of τ_4 from $AV^1(t, t+1)$ for $[0, 1)$, $[1, 2)$, $[2, 3)$, $[4, 5)$ and $[5, 6)$. Then this deduction helps to further reclaim the unusable supply, yielding a tighter upper-bound of supply.

(a) $x = 1$ (b) $x = 2$

Fig. 2. x-depth-available slots for jobs of each sequential task $\tau_1(T_1 = 2, C_1 = 1, D_1 = 1)$, $\tau_2(3, 2, 2)$, $\tau_3(4, 2, 3)$ and $\tau_4(8, 1, 6)$, slots for jobs that belong to $PJ^x(t, t+1)$, and calculation of $AV^x(t, t+1)$, when $m = 2$.

To derive tighter supply upper-bounds than Theorem 1 using the motivational example, we need to develop a systematic way of addressing the following issues:

I1. How to identify a set of jobs in $\mathcal{J}$ to be necessarily executed in each time slot in order to maximally utilize the supply, and
I2. How to reduce the number of available jobs in each time slot by considering I1 in order to further reclaim the unusable supply.

To address I1, we define $PJ^1(t, t+1)$, a unique *partial* set of jobs in $\mathcal{J}$ that should be executed in $[t, t+1)$ in order to maximally utilize the supply as follows.

Definition 2. Let $PJ^1(t, t+1)$ denote a set of jobs in $\mathcal{J}$, each of which satisfies the two following conditions; note that $PJ^1(t, t+1)$ is calculated for $t = 0, 1, 2, 3, \ldots$ in a sequential manner.

- The job is 1-depth-available in $[t, t+1)$ where $AV^1(t, t+1) \le m$ holds.
- The number of time slots $[t', t'+1)$ where the job belongs to $PJ^1(t', t'+1)$ for $t_r \le t' \le t-1$ is less than the execution time of the job, where t_r denotes the release time of the job.

Note that the second condition guarantees that the number of different time slots such that the job of interest belongs to $PJ^1(t^*, t^*+1)$ for any $t^* > 0$ cannot be larger than the execution time of the job of interest, by checking $PJ^1(t', t'+1)$ for $t' \ge 0$ smaller than t in a sequential manner (but we do not need to investigate $PJ^1(t', t'+1)$ if $[t', t'+1)$ is before the release time of the job of interest).

For example, since $[3, 4)$, $[5, 6)$ and $[7, 8)$ are the time slots in which $AV^1(t, t+1) \le m$ holds in Fig. 2(a), $PJ^1(3, 4)$, $PJ^1(5, 6)$ and $PJ^1(7, 8)$ include the jobs of τ_2 and τ_4, the job of τ_3, and the job of τ_2, respectively, as shown in the figure. Note that the first job of τ_4 with $C_4 = 1$ is included in $PJ^1(3, 4)$, so it is excluded from $PJ^1(5, 6)$ due to the second condition of Definition 2.

Definition 3 (*Valid Schedule*)**.** A schedule of $\mathcal{J}$ in $[0, t^*)$ is said to be *valid*, if every job in $\mathcal{J}$ can execute for its full execution time between its release time and deadline. Also, a schedule of $\mathcal{J}$ can be expressed by $CJ(t, t+1)$ for every $t \ge 0$, where $CJ(t, t+1)$ denotes a *complete* set of jobs in $\mathcal{J}$ selected to be executed in $[t, t+1)$.

With the notion of a valid schedule, the following lemma states and proves that $PJ^1(t, t+1)$ addresses I1.

Lemma 1. *If there exists a valid schedule of $\mathcal{J}$, the following statement holds. For any arbitrary valid schedule VS, we can construct a valid schedule VS^* such that $CJ(t, t+1) \supset PJ^1(t, t+1)$ holds for every $t \ge 0$.*

Otherwise, the following statement holds. For any arbitrary invalid schedule IVS, we can construct an invalid schedule IVS^* such that $CJ(t, t+1) \supset PJ^1(t, t+1)$ holds for every $t \ge 0$ and the amount of execution of a job J scheduled by IVS^* within its execution window is equal to or larger than that by IVS for every job J.

Proof. Suppose that there exists a VS, but there is no VS^*. If VS does not execute a job J^* in $[t, t+1)$ where $J^* \in PJ^1(t, t+1)$, we can guarantee the execution of J^* in $[t, t+1)$ by migrating the execution of J^* from another time slot to $[t, t+1)$. This is because, $J^* \in PJ^1(t, t+1)$ implies (i) $AV^1(t, t+1) \le m$ (i.e., a processor is availability for J^*) and (ii) J^* is 1-depth-available in $[t, t+1)$ (i.e., $[t, t+1)$ is between J^*'s release time and deadline). Therefore, by migrating the execution of J^* from another time slot to $[t, t+1)$ where $J^* \in PJ^1(t, t+1)$, we can make a valid schedule of $\mathcal{J}$ such that $CJ(t, t+1) \supset PJ^1(t, t+1)$ holds for every $t \ge 0$. This contradicts the supposition, which proves the lemma for the valid schedule case. The proof of the invalid schedule case is similar to the valid schedule case. □

Thanks to Lemma 1, it suffices to consider a valid schedule VS^* where all jobs in $PJ^1(t, t+1)$ should be executed in $[t, t+1)$ (if there exists any valid schedule), and therefore, we can pin the execution of the jobs in $PJ^1(t, t+1)$ on $[t, t+1)$, as shown in Fig. 2(a). Then, if there is no remaining execution of a job after pinning, we declare that the job is not available after pinning. This can yield the following notion of more restricted availability for each job, addressing I2.

Definition 4 (*2-Depth-Availability*)**.** Consider a job set $\mathcal{J}$ invoked by a task set τ. A job is said to be *2-depth-available* in $[t, t+1)$, if the job satisfies the following conditions.

- The time slot $[t, t+1)$ belongs to the interval between job's release time and deadline.
- The job belongs to $PJ^1(t, t+1)$, or the number of $t' \ge 0$ such that $PJ^1(t', t'+1)$ includes the job is less than the execution time of the job.

Let $AV^2(t, t+1)$ denote the number of 2-depth-available jobs in $\mathcal{J}$ in $[t, t+1)$.

Note that the second condition disallows a job to be 2-depth-available in a time slot if the job already belongs to $PJ^1(t, t+1)$ for other C time slots for any $t > 0$, where C is the execution time of the job. However, if a job belongs to $PJ^1(t, t+1)$ itself, the job is 2-depth-available in $[t, t+1)$ because the job can be executed in $[t, t+1)$.

For example, while the first job of τ_4 is 1-depth-available in $[t, t+1)$ where $t = 0, 1, 2, 3, 4$ and 5 in Fig. 2(a), the job is 2-depth-available in $[3, 4)$ only in Fig. 2(b) because the job belongs to $PJ^1(3, 4)$ and $C_4 = 1$. Also, it is straightforward that $AV^2(t, t+1) \le AV^1(t, t+1)$ holds for every t. Using the notion of $AV^2(t, t+1)$, we derive a tighter supply upper-bound as follows.

Theorem 2. *Consider a job set $\mathcal{J}$ invoked by a task set τ on an m-processor resource. If $AV^2(t, t+1) < m$, the supply able to service $\mathcal{J}$ in $[t, t+1)$ is at most $AV^2(t, t+1)$, not m. Therefore, the amount of supply able to service $\mathcal{J}$ in $[0, t)$ by an m-processor platform is upper-bounded by $SB^2(t)$, where*

$$SB^2(t) \stackrel{def}{=} m \times t - \sum_{t_a=0}^{t-1} \max\left(0, m - AV^2(t_a, t_a+1)\right). \tag{6}$$

Proof. The theorem holds by Lemma 1, which proves that we can transform any arbitrary schedule (either valid or invalid) to the corresponding schedule such that $CJ(t_a, t_a+1) \supset PJ^1(t_a, t_a+1)$ holds for every $t_a \geq 0$. Thanks to the lemma, to determine the feasibility, we only need to check the schedules that satisfy $CJ(t_a, t_a+1) \supset PJ^1(t_a, t_a+1)$ for every $t_a \geq 0$, instead of checking all possible schedules. Therefore, it is possible to focus on the amount of supply utilized by the schedules that satisfy $CJ(t_a, t_a+1) \supset PJ^1(t_a, t_a+1)$ for every $t_a \geq 0$. Since $AV^2(t_a, t_a+1)$ is the number of jobs that can be available to be executed in $[t, t+1)$ after pinning the execution of jobs in $PJ^1(t_a, t_a+1)$ in $[t_a, t_a+1)$ for every $t_a \geq 0$, the remaining proof is the same as that of Theorem 1 by applying t_a to $0, 1, 2, \ldots t-1$. □

We can easily check that $SB^2(t)$ in Theorem 2 is no larger than $SB^1(t)$ in Theorem 1, for every $t > 0$.

Similar to deriving a tighter upper-bound of supply from $SB^1(t)$ to $SB^2(t)$ using the notions of $PJ^1(t, t+1)$ and 2-depth-availability, we can generalize $PJ^{x-1}(t, t+1)$, x-depth-availability and $SB^x(t)$ for $x = 3, 4, 5, 6, \ldots$ in a sequential manner.

Definition 5. Let $PJ^{x-1}(t, t+1)$ denote a set of jobs in $\mathcal{J}$, each of which satisfies the two following conditions (where $x \geq 3$); note that $PJ^{x-1}(t, t+1)$ is calculated for $t = 0, 1, 2, 3, \ldots$ in a sequential manner.

- The job is $(x-1)$-depth-available in $[t, t+1)$ where $AV^{x-1}(t, t+1) \leq m$ holds.
- The number of time slots $[t', t'+1)$ where the job belongs to $PJ^{x-1}(t', t'+1)$ for $t_r \leq t' \leq t-1$ is less than the execution time of the job, where t_r denotes the release time of the job.

Definition 6 (*x-Depth-Availability*). Consider a job set $\mathcal{J}$ invoked by a task set τ. A job is said to be *x-depth-available* in $[t, t+1)$ where $x \geq 3$, if the job satisfies the following conditions.

- The time slot $[t, t+1)$ belongs to the interval between the job's release time and deadline.
- The job belongs to $PJ^{x-1}(t, t+1)$, or the number of $t' \geq 0$ such that $PJ^{x-1}(t', t'+1)$ includes the job is less than the execution time of the job.

Let $AV^x(t, t+1)$ denote the number of x-depth-available jobs in $\mathcal{J}$ in $[t, t+1)$.

The relationship between $PJ^{x-1}(t, t+1)$ and x-depth-availability is the same as that between $PJ^1(t, t+1)$ and 2-depth-availability. Also, it is easily checked that $AV^{x+1}(t, t+1) \leq AV^x(t, t+1)$ and $PJ^{x+1}(t, t+1) \supset PJ^x(t, t+1)$ hold for every t where $x \geq 1$. Similar to Lemma 1, we can prove that $PJ^{x-1}(t, t+1)$ for $x \geq 3$ also addresses I1.

Lemma 2. *Lemma 1 holds by replacing $PJ^1(t, t+1)$ with $PJ^{x-1}(t, t+1)$, for $x \geq 3$.*

Proof. (Base case) In Lemma 1, we already proved that we can transform any arbitrary schedule (either valid or invalid), to the corresponding schedule such that $CJ(t, t+1) \supset PJ^1(t, t+1)$ holds for every $t \geq 0$.

(Inductive case) Suppose that Lemma 1 holds for $PJ^{x-1}(t, t+1)$. By the proof of Lemma 1 by replacing $PJ^{x-1}(t, t+1)$ with $PJ^x(t, t+1)$, we can transform any arbitrary schedule such that $CJ(t, t+1) \supset PJ^{x-1}(t, t+1)$ for every $t \geq 0$, to the corresponding schedule such that $CJ(t, t+1) \supset PJ^x(t, t+1)$ holds for every $t \geq 0$.

By the base and inductive cases, the lemma holds. □

Finally, we can derive a tighter upper-bound of supply by using the notions of $PJ^{x-1}(t, t+1)$ and x-depth-availability, stated in the following theorem, which subsumes Theorems 1 and 2 by assigning $x = 1$ and $x = 2$, respectively.

Theorem 3. *Consider a job set $\mathcal{J}$ invoked by a task set τ on an m-processor resource. If $AV^x(t, t+1) < m$ (where $x \geq 1$), the supply able to service $\mathcal{J}$ in $[t, t+1)$ is at most $AV^x(t, t+1)$, not m. Therefore, the amount of supply able to service $\mathcal{J}$ in $[0, t)$ by an m-processor platform is upper-bounded by $SB^x(t)$, where*

$$SB^x(t) \overset{def}{=} m \times t - \sum_{t_a=0}^{t-1} \max\big(0, m - AV^x(t_a, t_a+1)\big). \quad (7)$$

Proof. Applying the idea of how to derive Theorem 2 from Lemma 1, we can derive the theorem for $x \geq 3$ from Lemma 2. The theorem for $x = 1$ and $x = 2$ was already proven in Theorems 1 and 2, respectively. □

Similar to the relationship of $SB^2(t) \leq SB^1(t)$ for every $t > 0$, $SB^{x+1}(t) \leq SB^x(t)$ holds for every $t > 0$ where $x \geq 2$.

4. Developing tight necessary feasibility conditions

A typical form of existing necessary feasibility conditions for τ on an m-processor platform is the amount of demand of τ in an interval of length t no larger than the amount of supply in the interval. Focusing on a single job arrival sequence where τ invokes a series of jobs periodically from $t = 0$, existing studies have found a tight lower-bound of the amount of demand in an interval $[0, t)$, but they have deployed $m \times t$ as the amount of supply, assuming that the job arrival sequence of τ can *always* utilize m processors, e.g., those for the sequential task model [6,7], the parallel task model [8], the gang task model [12], and the mixed-criticality sequential task model [9,10]. While our approach can be applied to all of the mentioned necessary feasibility conditions, this section shows two examples.

Sequential task model. The state-of-the-art necessary feasibility condition for the sequential task model is as follows [7].

$$\sum_{\tau_i \in \tau} \mathsf{FFDBF}(\tau_i, t) \leq m \times t, \ \textit{where} \quad (8)$$

$$\mathsf{FFDBF}(\tau_i, t) = \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \times C_i + \max\Big(0, (t - D_i) \bmod T_i - T_i + C_i\Big).$$

If we apply Theorem 3 by replacing the RHS of Eq. (8) with Eq. (7), we can derive a tighter necessary feasibility condition.

Theorem 4. *For given $x \geq 1$, a task set τ of sequential tasks is feasible on m processors, only if the following condition holds for every $t \geq 0$.*

$$\sum_{\tau_i \in \tau} \mathsf{FFDBF}(\tau_i, t) \leq m \times t - \sum_{t_a=0}^{t-1} \max\big(0, m - AV^x(t_a, t_a+1)\big) \quad (9)$$

Proof. The theorem holds by Eq. (8) with Theorem 3, detailed as follows. Theorem 4 and Eq. (8) focus on the single job arrival sequence where τ invokes a series of jobs periodically from $t = 0$. Under that sequence, the study in [7] proves the LHS of Eq. (8) is a lower-bound of the amount of demand in $[0, t)$. Also, Theorem 3 proves Eq. (7) is an upper-bound of the amount of supply in $[0, t)$. Therefore, violation of Eq. (9) implies that a lower-bound of the amount of demand in $[0, t)$ is larger than an upper-bound of the amount of supply in $[0, t)$, yielding at least one missed job deadline. □

While $AV^x(t, t+1)$ in Eq. (9) can be calculated according to Definition 6 with $t = 0, 1, 2, 3, \ldots$ in a sequential manner, one may wonder a specific way to calculate $AV^x(t, t+1)$ for the sequential task model, which is now explained in Algorithm 1. Note that Algorithm 1 is described for constrained-deadline tasks (i.e., $D_i \leq T_i$ holds for every

Algorithm 1 Calculating $AV^x(t, t+1)$ for the sequential task set τ

1: $A_i^1(t, t+1) \leftarrow \mathcal{F}$ for every $\tau_i \in \tau$
2: For every $\tau_i \in \tau$, **if** $(t+1) \bmod T_i \le D_i$, **then** $A_i^1(t, t+1) \leftarrow \mathcal{T}$.
3: $AV^1(t, t+1) \leftarrow$ # of tasks $\tau_i \in \tau$ that satisfy $A_i^1(t, t+1) = \mathcal{T}$
4: **if** $x = 1$, **then** return $AV^1(t, t+1)$
5: **for** $y = 2, 3, ..., x$ **do**
6: $PJ^{y-1}(t, t+1) \leftarrow \emptyset$
7: For every $\tau_i \in \tau$, **if (a)** $A_i^{y-1}(t, t+1) = \mathcal{T}$ **AND (b)** $AV^{y-1}(t, t+1) \le m$ **AND (c)** # of $PJ^{y-1}(t', t'+1) \ni \tau_i$ for $t' = \lfloor t/T_i \rfloor, ..., t-1$ is less than C_i, **then** $PJ^{y-1}(t, t+1) \leftarrow PJ^{y-1}(t, t+1) \cup \{\tau_i\}$
8: $A_i^y(t, t+1) \leftarrow \mathcal{F}$ for every $\tau_i \in \tau$
9: For every $\tau_i \in \tau$, **if** $\Big($**(d)**$(t+1) \mod T_i \le D_i\Big)$ **AND** $\Big($**(e)** $\tau_i \in PJ^{y-1}(t, t+1)$ **OR (f)** # of $PJ^{y-1}(t', t'+1) \ni \tau_i$ for $t' = \lfloor t/T_i \rfloor, ..., \lfloor t/T_i \rfloor + D_i$ is less than $C_i\Big)$, **then** $A_i^y(t, t+1) \leftarrow \mathcal{T}$.
10: $AV^y(t, t+1) \leftarrow$ # of tasks $\tau_i \in \tau$ that satisfy $A_i^y(t, t+1) = \mathcal{T}$
11: **end for**
12: Return $AV^x(t, t+1)$

$\tau_i \in \tau$), but the algorithm can be easily adapted to arbitrary-deadline tasks. Recall that Theorem 4 focuses on the single job arrival sequence where τ invokes a series of jobs periodically from $t = 0$, as Eq. (8) does. This job arrival pattern will be used to test some properties; for example, a task $\tau_i \in \tau$ has a 1-depth-available job in $[t, t+1)$ if $(t+1)\ mod\ T_i \le D_i$ holds.

In Lines 1–3, we check whether every task τ_i has a 1-depth-available job in $[t, t+1)$. We mark the 1-depth-availability of τ_i as $\mathcal{T}$ (i.e., TRUE) for $A_i^1(t, t+1)$, and calculate $AV^1(t, t+1)$ as the number of $\tau_i \in \tau$ that satisfies $A_i^1(t, t+1) = \mathcal{T}$. In Line 4, if $x = 1$, return $AV^1(t, t+1)$. In Lines 5–11, we will calculate $PJ^{y-1}(t, t+1)$ and $AV^y(t, t+1)$ for $y = 2, 3, \dots, x$ in a sequential manner. Line 7 checks whether τ_i should be included in $PJ^{y-1}(t, t+1)$; (a) and (b) correspond to check the first item in Definition 5, and (c) corresponds to the second item in the definition. Since $\lfloor t/T_i \rfloor$ is the release time of the job of τ_i whose release time is no later than t but whose deadline is no earlier than $t+1$, we only check $t' = \lfloor t/t_i \rfloor, \dots, t-1$ for (c). Line 9 checks whether τ_i has a y-depth-available job in $[t, t+1)$ by checking whether (d) is satisfied (corresponding to the first item in Definition 6) *and* (e) or (f) is satisfied within a job's execution window $[\lfloor t/t_i \rfloor, \lfloor t/t_i \rfloor + D_i)$ (corresponding to the second item in the definition).

Gang task model. The demand bound function $\mathsf{DBF}(\tau_i, t)$ of τ_i shown in Eq. (4) for the sequential task model can be extended for the gang task model to compute a lower-bound of the amount of demand in an interval $[0, t)$, parallelized on v_i processors. This yields the following necessary feasibility conditions for the gang task model.

$$\sum_{\tau_i \in \tau} \mathsf{DBF}^G(\tau_i, t) \le m \times t, \ \ where \tag{10}$$

$$\mathsf{DBF}^G(\tau_i, t) = \max\Big(0, \Big\lfloor \frac{t - D_i}{T_i} \Big\rfloor + 1\Big) \times C_i \times v_i.$$

If we apply Theorem 3 by replacing the RHS of Eq. (10) with Eq. (7), we can derive a tighter necessary feasibility condition.

Theorem 5. *For given $x \ge 1$, a task set τ of gang tasks is feasible on m processors, only if the following condition holds for every $t \ge 0$.*

$$\sum_{\tau_i \in \tau} \mathsf{DBF}^G(\tau_i, t)$$
$$\le m \times t - \sum_{t_a = 0}^{t-1} \max\big(0, m - AV^x(t_a, t_a + 1)\big) \tag{11}$$

Proof. Since $\mathsf{DBF}^G(\tau_i, t)$ is equal to $v_i \cdot \mathsf{DBF}(\tau_i, t)$ in Eq. (4) for the sequential task model, $\mathsf{DBF}^G(\tau_i, t)$ is a lower-bound of the amount of demand of a gang task τ_i. Then, the theorem holds by Eq. (10) with Theorem 3. □

Note that we can calculate $AV^x(t, t+1)$ for the gang task model, by modifying Algorithm 1 such that Lines 3 and 10 count the number of threads (v_i) of tasks $\tau_i \in \tau$, instead of the number of tasks $\tau_i \in \tau$.

Dominance relation. We now discuss the relationship between the existing infeasibility tests and those associated with our approach, in terms of finding infeasible task sets, recorded in the following lemma.

Lemma 3. *Any task set τ deemed infeasible by the infeasibility test with Eq. (8) (likewise that with Eq. (10)) is also deemed infeasible by Theorem 4 (likewise Theorem 5).*

Proof. Suppose that there is a task set τ is deemed infeasible by the infeasibility test with Eq. (8). Then, according to the infeasibility test with Eq. (8), there exists t^* which makes Eq. (8) false. Since the RHS of Eq. (9) is always no larger than that of Eq. (8), such t^* also makes Eq. (9) false. This implies the dominance between the infeasibility test with Eq. (8) and Theorem 4 holds. With the same reason, the dominance between the infeasibility test with Eq. (10) and Theorem 5 also holds. □

Time-complexity for Theorems 4 and 5. In Theorem 4, the set of values of t to be checked has pseudo-polynomial size [15]. For a given value of t, the LHS of Eq. (9) can be computed with $O(n)$ complexity [4]. Note that the values of t to be checked are monotonically increasing. Thus, when computing the RHS of Eq. (9), i.e., $SB^x(t)$, for a given value of t with $x = 1$, we store the value of $SB^1(t-1)$ that can be used to calculate $SB^1(t)$ as $SB^1(t) = SB^1(t-1) + m - \max\big(0, m - AV^1(t-1, t)\big)$. Computing $AV^1(t-1, t)$ requires $O(n)$, so computing the RHS of Eq. (9) requires $O(n)$ for a given value of t with $x = 1$. In order to compute $SB^x(t)$ for a given value of t with $x \ge 2$, we store the values of $AV^{x-1}(t-1, t)$ and $A_i^{x-1}(t-1, t)$ for every $\tau_i \in \tau$. Then, $AV^x(t-1, t)$ with $x \ge 2$ can be also computed with $O(n)$ complexity. Thus, the total complexity of Theorem 4 is pseudo-polynomial, which is equivalent to that of the corresponding necessary feasibility test in Eq. (8). The same holds for Theorem 5.

Discussion. The proposed approach utilizes the time slots in which no job of a target task can be executed. Therefore, it cannot improve existing studies in terms of finding infeasible task sets, if every task τ_i in the target task set satisfies $D_i \ge T_i$. However, as long as there exist some constrained-deadline (i.e., $D_i < T_i$) tasks in the target task set, the proposed approach potentially finds the infeasibility of the task set, which is not proven by existing studies. Note that when every task τ_i in the target task set satisfies $D_i \ge T_i$, Eq. (8) for the sequential model is equivalent to $\sum_{\tau_i \in \tau} C_i/T_i \le m$, which is necessary *and* sufficient feasibility test; therefore, no one for the sequential model can improve the existing study in terms of finding infeasible task sets.

5. Evaluation

We now demonstrate the capability of the proposed feasibility tests in covering a broader range of infeasible task sets for the sequential and gang sporadic task models. We generate a synthetic task set by using the Dirichlet-Rescale (DRS) algorithm [16], known as an efficient task set generation method. We have four input parameters: (i) the number of processors $m \in \{2, 4, 8\}$, (ii) the number of tasks $n \in \{m+1, 3/2m+1, 2m+1, 5/2m+1, 3m+1\}$, (iii) total utilization $u_{sum} \overset{\text{def}}{=} \sum_{\tau_i \in \tau} C_i/T_i$, and (iv) total density $\delta_{sum} \overset{\text{def}}{=} \sum_{\tau_i \in \tau} C_i/D_i$.

Given a 4-tuple $(m, n, u_{sum}, \delta_{sum})$ for a sequential task set, each task parameter is determined as follows: T_i is uniformly chosen in $[1, 5000]$; its utilization ($u_i = C_i/T_i$) and density ($\delta_i = C_i/D_i$) are randomly generated using the DRS algorithm such that $\sum u_i = u_{sum}$ and $\sum \delta_i = \delta_{sum}$, respectively; C_i and D_i ($\le T_i$) are computed as $u_i \cdot T_i$ and C_i/δ_i, respectively. For a gang task set, one additional task parameter v_i is uniformly chosen in $[1, m]$, and the other parameters are chosen similarly with the sequential one by replacing C_i to $v_i \cdot C_i$.

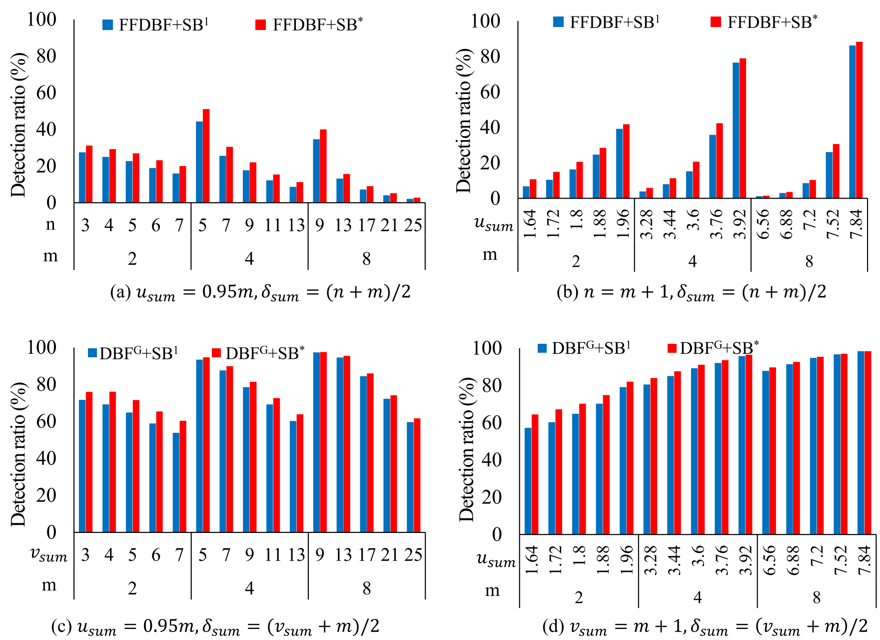

Fig. 3. Detection ratio of the proposed necessary feasibility tests for sequential task sets ((a) and (b)) and gang task sets ((c) and (d)).

In order to show how many infeasible task sets can be found by our proposed tests over the existing state-of-the-art necessary feasibility analysis, we generate sequential and gang task sets which are not proven infeasible by Eq. (8) and by Eq. (10), respectively. We emphasize that all the generated task sets have not proven infeasible by any existing necessary feasibility analysis for the corresponding task model. We compare the following four proposed necessary feasibility tests:

- FFDBF+SB¹ and FFDBF+SB*: the necessary feasibility test for the sequential task model in Theorem 4, respectively with $x = 1$ and the smallest $x \geq 1$ such that $SB^{x+1}(t) = SB^{x}(t)$ for every $t > 0$;
- DBF^{G}+SB¹ and DBF^{G}+SB*: the necessary feasibility test for the gang task model in Theorem 5, respectively with $x = 1$ and the smallest $x \geq 1$ such that $SB^{x+1}(t) = SB^{x}(t)$ for every $t > 0$;

We use *detection ratio* as a performance metric, defined as the percentage of task sets that are deemed infeasible by each individual necessary feasibility test to the total number of generated task sets.

Figs. 3(a) and 3(b) plot the detection ratios by FFDBF+SB¹ and FFDBF+SB* for 1,200,000 sequential task sets while (a) varying the number of tasks n when $u_{sum} = 0.95m$ and $\delta_{sum} = (n+m)/2$ and (b) varying total utilization u_{sum} from $0.82m$ to $0.98m$ when $n = m+1$ and $\delta_{sum} = (n+m)/2$, respectively, for $m = 2, 4$, and 8. The proposed necessary feasibility tests can newly find a number of infeasible task sets for all n and u_{sum} values when $m = 2, 4$ and 8. They identify more infeasible task sets as the number of tasks becomes smaller and the total utilization becomes larger. For example, in Fig. 3(a), FFDBF+SB* finds 51% and 11% infeasible task sets with $n = 5$ and 13, respectively, and, in Fig. 3(b), 6% and 79% infeasible task sets with $u_{sum} = 3.28$ and 3.92, respectively, when $m = 4$. In total, FFDBF+SB¹ and FFDBF+SB* find 256,873 (21%) and 297,622 (25%) infeasible task sets, respectively, among 1,200,000 generated task sets. FFDBF+SB* exhibits 16% higher capability in finding infeasible task sets than FFDBF+SB¹. Such an improvement can be interpreted as the benefit of reclaiming more amount of unusable supply by identifying an optimal way to utilize the supply, yielding tighter supply upper-bounds.

Figs. 3(c) and 3(d) plot the detection ratios by DBF^{G}+SB¹ and DBF^{G}+SB* for 1,200,000 gang task sets while (c) varying the total number of threads $v_{sum} \overset{\text{def}}{=} \sum_{\tau_i \in \tau} v_i$ when $u_{sum} = 0.95m$ and $\delta_{sum} = (v_{sum}+m)/2$ and (d) varying total utilization u_{sum} from $0.82m$ to $0.98m$ when $v_{sum} = m+1$ and $\delta_{sum} = (v_{sum}+m)/2$, respectively, for $m = 2, 4$, and 8. Similar trends have observed for gang task sets as the total number of threads and the total utilization increase, but both DBF^{G}+SB¹ and DBF^{G}+SB* exhibit much higher capability in finding infeasible task sets in that DBF^{G}+SB¹ and DBF^{G}+SB* find 943,285 (79%) and 980,205 (82%) total infeasible task sets, respectively, among 1,200,000 generated task sets. Such higher capability for gang task sets (than that for sequential task sets) mainly comes from the accurate calculation on the amount of unusable supply, in that the threads that belong to a single gang task share their execution windows, yielding more unavailable time slots for each gang task, although the total number of threads in a gang task set is same as the number of tasks in a sequential task set.

Note that although, among generated task sets, there might also exist task sets proven feasible by existing sufficient schedulability tests under some scheduling algorithms, we did not identify them in our evaluation, implying that the presented detection ratio as of now exhibits the minimum capability of the proposed necessary feasibility tests in finding infeasible task sets.

Table 2
Running times (ms) with respect to the number of tasks.

# of tasks (n)	5	9	13	17	21	25
FFDBF+SB¹	0.2	0.2	0.4	0.5	0.5	0.7
FFDBF+SB*	4.3	7.5	16.8	24.2	29.9	33.9

Running time. We measure the average running times of the proposed necessary feasibility tests to identify an infeasible task set with respect to the number of tasks shown in Table 2. When n is increased from 5 to 25, the average running times of FFDBF+SB¹ and FFDBF+SB* increase from 0.2ms to 0.7ms and from 4.3ms to 33.9ms, respectively. Under FFDBF+SB*, the supply upper-bound converges at 2-depth-availability on average with 8-depth-availability at the maximum. Note that DBF^{G}+SB¹ and DBF^{G}+SB* show less average running time than FFDBF+SB¹ and FFDBF+SB*, respectively, because they are performed with a smaller number of tasks.

6. Conclusion

In this paper, we propose a new general methodology for deriving tight supply upper-bounds to be utilizable by a set of jobs on multiprocessor platforms and demonstrate its utility and power in finding infeasible task sets by developing necessary feasibility conditions for two representative task models. In the future, we would like to extend our approach to develop a collective way of identifying the amount of unusable supply in the interval length of more than one in order to further narrow the area of uncertainty between the task sets proven feasible and those proven infeasible for a wide range of task models on multiprocessor platforms.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant (NRF-2021R1A2B5B02001758, NRF-2022R1A4A3018824, NRF-2017M3A9G8084463, NRF-2018R1A5A1060031, NRF-2020R1F1A1076058) and Institute of Information & communications Technology Planning & Evaluation (IITP), South Korea grant (IITP-2022-0-01053) funded by the Korea government (MSIT).

References

[1] M. Bertogna, S. Baruah, Tests for global EDF schedulability analysis, J. Syst. Archit. 57 (5) (2011) 487–497.

[2] S. Chang, J. Sun, Z. Hao, Q. Deng, N. Guan, Computing exact WCRT for typed DAG tasks on heterogeneous multi-core processors, J. Syst. Archit. 124 (2022) 102385, 1–11.

[3] R.I. Davis, A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Comput. Surv. 43 (4) (2011) 1–44.

[4] S. Baruah, A. Mok, L. Rosier, Preemptively scheduling hard-real-time sporadic tasks on one processor, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 1990, pp. 182–190.

[5] S. Baruah, N. Fisher, The feasibility analysis of multiprocessor real-time systems, in: Proceedings of Euromicro Conference on Real-Time Systems, ECRTS, 2006, pp. 85–96.

[6] N. Fisher, T.P. Baker, S. Baruah, Algorithms for determining the demand-based load of a sporadic task system, in: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, 2006.

[7] T.P. Baker, M. Cirinei, A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 2006, pp. 178–190.

[8] B. Andersson, G. Raravi, Scheduling constrained-deadline parallel tasks on two-type heterogeneous multiprocessors, in: Proceedings of International Conference on Real-Time Networks and Systems, RTNS, 2016, pp. 247–256.

[9] H.S. Chwa, H. Baek, J. Lee, Necessary feasibility analysis for mixed-criticality task systems on uniprocessor, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 2019, pp. 446–457.

[10] H.S. Chwa, H. Baek, J. Lee, Necessary feasibility analysis for mixed-criticality real-time embedded systems, IEEE Trans. Parallel Distrib. Syst. 33 (7) (2022) 1520–1537.

[11] A. Mok, Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment (Ph.D. thesis), Massachusetts Institute of Technology, 1983.

[12] S. Kato, Y. Ishikawa, Gang EDF scheduling of parallel task systems, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 2009, pp. 459–468.

[13] W.A. Horn, Some simple scheduling algorithms, Nav. Res. Logist. Q. 21 (1) (1974) 177–185.

[14] S. Baruah, N. Fisher, The partitioned multiprocessor scheduling of sporadic task systems, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 2005, pp. 321–329.

[15] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, Implementation of a speedup-optimal global EDF schedulability test, in: Proceedings of Euromicro Conference on Real-Time Systems, ECRTS, 2009, pp. 259–268.

[16] D. Griffin, I. Bate, R.I. Davis, Generating utilization vectors for the systematic evaluation of schedulability tests, in: Proceedings of IEEE Real-Time Systems Symposium, RTSS, 2020, pp. 76–88.