IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 11, 2020, accepted April 30, 2020, date of publication May 6, 2020, date of current version May 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2992868

Response-Time Analysis for Multi-Mode Tasks in Real-Time Multiprocessor Systems

HYEONGBOO BAEK[1], KANG G. SHIN[2], (Life Fellow, IEEE), AND JINKYU LEE[3], (Member, IEEE)

[1]Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea
[2]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA
[3]Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Jinkyu Lee (jinkyu.lee@skku.edu)

This work was supported in part by the National Research Foundation of Korea (NRF) through the Ministry of Science and ICT under Grant 2019R1A2B5B02001794, Grant 2019R1F1A1059663, and Grant 2017H1D8A2031628, in part by the NSF under Grant CNS-1446117 and Grant CNS-1739577, in part by the ONR under Grant N00014-18-1-2141, and in part by LG Chem Ltd.

ABSTRACT Recently, traditional real-time systems that played a dedicated role in a limited environment have been evolving to interact with dynamically varying environments. In modern real-time systems, characteristics of real-time tasks such as computational demand and resource allocation can vary over time according to different circumstances, which is referred to as *mode transition*. In this paper, we focus on the problem of timing guarantees of a set of multi-mode tasks associated with mode transitions and develop an offline schedulability analysis, which does not require any online information; this is an important problem in the real-time systems area. The proposed schedulability analysis not only generalizes an existing framework designed for single-mode tasks, but also significantly improves the state-of-the-art framework designed for multi-mode tasks. Building on the proposed analysis, we also address the problem of enforcing the order of tasks within each mode transition and propose a task-level transition order assignment algorithm, yielding further improvement in schedulability performance. Through simulations, our proposed framework is shown to improve schedulability up to 777.3% over an existing schedulability analysis for multi-mode tasks, depending on the experiment setting under our evaluation environment.

INDEX TERMS Real-time scheduling, multi-mode tasks, schedulability analysis, real-time multiprocessor systems.

I. INTRODUCTION

Traditional real-time systems for dedicated roles with specialized hardware have been evolving to interact with dynamically changing environments through ubiquitous networks and sensor devices. Such modern real-time systems continually sense the physical environment and receive feedback from sensors. As a result, the characteristics of real-time tasks such as computational demand and resource allocation vary over time according to different circumstances, which is referred to as *mode transition*. A compelling example is an unmanned reconnaissance aircraft, involving landing, takeoff, and normal/specialized reconnaissance modes, each representing a different goal under the corresponding environment it faces [1], [2]. Such mode transition may require activation of a new task, and deactivation or changes in existing tasks.

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

Such changing characteristics of real-time systems on varying environments necessitate designing distinct task models that potentially characterize multiple execution modes, referred to as *multi-mode* task models. Multi-mode task models are generalized from a traditional real-time task model specified by a collection of independent recurrent tasks, each of which generates a series of jobs. While a task in the traditional task model [3] is characterized by fixed values for three parameters (i.e., minimum job separation, worst-case execution time (WCET), and relative deadline), multi-mode task models assume multiple values for each parameter of a task. For uniprocessor systems, timing guarantees on the multi-mode task models have been extensively studied by considering various mode-transition protocols, scheduling algorithms, and system domains [5]–[11] (also see the survey in [2]).

In contrast, few studies have been conducted for multiprocessor real-time systems in the classes of *partitioned* and *global* scheduling. For partitioned scheduling, Niz and

Phan considered a criticality transition of tasks [12], whereas Huang and Chen addressed the situation where each task changes its execution independent of other tasks [13]. Regarding global scheduling, a few studies proposed mode-transition protocols that incur additional delay for a mode transition and tried to minimize the delay without any deadline miss [14]–[16]. The mode-transition protocol in our preliminary conference paper [17] is distinct from such studies in that it supports a *system-wide mode transition* where all tasks may switch to their new parameters without additional delay or task drop. Extending an existing popular schedulability analysis framework for single-mode tasks, called the deadline-based schedulability analysis (DA) [18], our previous study succeeded in developing a new schedulability analysis for multi-mode tasks in the presence of a mode transition. Despite the advantages of the transition protocol in our previous study, such advantages have not been fully utilized owing to the limited analytical capability of the DA framework.

In this study, we target the system-wide mode transition protocol without imposing additional transition delay and task drop (to be detailed in Section II-B), and we develop a sufficient, offline response-time analysis (RTA) framework [19] that guarantees timely execution of all tasks in the presence of system-wide mode transitions; RTA significantly improves the analytical capability over DA, as RTA analyzes the worst-case scenario that each task may experience less pessimistically than DA. The key technique of RTA distinguished from DA is the slack reclamation scheme that effectively utilizes the notion of slack (of each task) defined as the minimum interval length between the finishing time and the deadline of any job of a task. We present how the slack reclamation scheme systematically identifies the slack of each task, and incorporate it into RTA to improve the schedulability by reducing the pessimism in analyzing the worst-case scenario that the task may experience. We make RTA independent of online information, such as the time instant when the transition starts, and task release and execution patterns; otherwise, the system would have to monitor/predict the information, and hence the analysis could not provide any offline guarantee.

Also, we further improve the analytical capability of our RTA framework by proposing a mechanism that controls the order for tasks to complete their mode transitions within a system-wide mode transition. In a prior system-wide mode transition protocol, the transitions of tasks occur concurrently in every individual mode transition. Thus, the order for tasks to complete their mode transitions within a system-wide mode transition depends on job release patterns and the mode transition request time. If the transition order of tasks in each system-wide mode transition can be controlled and thus predetermined offline by the system designer, we can further improve schedulability by reducing interference.

We perform extensive simulations to show the effectiveness of the proposed RTA framework. Our proposed framework is shown to improve schedulability up to 777.3%, over an existing schedulability analysis for multi-mode tasks, depending on the experiment setting under our evaluation environment. In particular, schedulability improves as the number of modes increases.

In summary, this paper makes the following contributions.

- Development of a new RTA framework for a mode transition (without any additional transition delay) in real-time multiprocessor systems, which outperforms the DA framework;
- Development of a new interference calculation method to be used for the new RTA framework;
- Development of a slack reclamation scheme to further improve the schedulability;
- Identification of the problem of task-level transition order assignment, and development of a grouping framework for the transition-order assignment using the derived properties under a given condition; and
- Demonstration of the effectiveness of the RTA framework via simulation.

The remainder of this paper is organized as follows. Section II presents our system model and the transition protocol we consider, and recapitulates an existing RTA framework for single-mode tasks. Section III presents challenges and overview of developing RTA framework for multi-mode tasks. Section IV develops a new RTA framework for a mode transition. Section V develops a task-level transition-order assignment framework. Section VI evaluates the effectiveness of this framework. Section VIII discusses related work, and Section IX concludes the paper.

II. BACKGROUND

In this section, we first describe the system model, assumptions, and notations to be used throughout this paper. Then, we summarize an existing RTA framework for single-mode tasks, which will be used as the basis for our RTA framework for multi-mode tasks.

A. SYSTEM MODEL, ASSUMPTIONS AND NOTATIONS

We consider a periodic task model [3], [20] associated with μ different operation modes. We let M^g, M^h, and $M^g \Rightarrow M^h$ denote the g-th mode, the h-th mode, and a mode transition from M^g to M^h where g and h are positive integers that satisfy $1 \leq g \leq h = g + 1 \leq \mu$. A task set τ is denoted by τ^g when the parameters of tasks in τ are associated with M^g. A task τ_i^g in τ^g is specified by $\tau_i^g(p_i^g, e_i^g, d_i^g)$ where p_i^g is the time separation between two successive invocations (called a period), [1] e_i^g is the WCET, and d_i^g is the relative deadline of τ_i^g. Our focus is confined to constrained deadline tasks, each of which satisfies the inequality $d_i^g \leq p_i^g$. Different modes imply not only the change of task parameters (e.g., satisfying at least one of $p_i^g \neq p_i^{g+1}$, $e_i^g \neq e_i^{g+1}$ or $d_i^g \neq d_i^{g+1}$) but also addition/deletion of tasks. For convenience of presentation,

[1] Note that all the analytical results in this paper are also applicable to sporadic tasks in which p_i^g represents the *minimum* separation, not the exact separation.

TABLE 1. Notations and their description.

Notation	Description	Notation	Description
μ	the number of considered modes	p_i^g	the time separation between two successive invocations
M^g	the g-th mode	e_i^g	the worst-case execution time
M^h	the h-th mode	d_i^g	the relative deadline of τ_i^g
$M^g \Rightarrow M^h$	a mode transition from M^g to M^h	J_i	a job of τ_i
τ	a task set	m	the number of processors
τ^g	a task set associated with M^g	ℓ	the length of an interval of interest
τ^h	a task set associated with M^h	s_i^g	a slack value of τ_i^g
τ_i^g	a task in τ^g	$\tau_i^{g \Rightarrow h}$	τ_i in the presence of a transition $M^g \Rightarrow M^h$

we let τ denote the set of all tasks existing in at least one mode; if a task τ_i does not exist in a mode M^g, τ_i^g represents a *dummy* task ($p_i^g = 1, e_i^g = 0, d_i^g = 1$), and does not affect the actual execution of other tasks. Whenever g is irrelevant, we omit it, i.e., using τ_i, p_i, e_i, and d_i instead of τ_i^g, p_i^g, e_i^g, and d_i^g.

A task τ_i^g invokes a series of jobs, each separated from its predecessor by p_i^g time units and is supposed to finish its execution within d_i^g time units taking at most e_i^g time units for its execution. We call the interval between the release time and deadline of a job J_i, the *scheduling window* of J_i. We assume a quantum-based time slot, and consider the length of a quantum as one time unit without loss of generality. All task parameters are specified in multiples of this quantum.

We target multiprocessor systems containing m identical processors. We consider global, preemptive, and work-conserving scheduling algorithms under which the execution of a job can migrate from one processor to another; a job can be preempted at any time, and there should be no idle processors as long as there is a job ready to be scheduled. We also assume that a job cannot be executed in parallel.

B. TRANSITION PROTOCOL

We consider a series of transitions of a task set τ, and each transition is separated from its predecessor and successor transitions, meaning that any time unit in the interval between the start and end of a transition (e.g., $[t_1, t_2)$ for $M^g \Rightarrow M^h$ in Fig. 1) cannot be included in the other transitions' one (e.g., $([t_{-1}, t_0)$ for $M^f \Rightarrow M^g$ or $[t_3, t_4)$ for $M^h \Rightarrow M^i$ in Fig. 1). In other words, the scheduling window of a job with a mode can overlap only with either that of any job with its previous and current modes, or that of any job with its current and next modes. Within a single transition, multiple tasks can change their task parameters; recall that addition or deletion of tasks is also expressed as a change of task parameters by using dummy tasks, as explained in Section II-A. Note that we assume the mode transition protocol itself does not impose additional transition delay and task drop as most existing studies do.

To support a transition that does not result in missing/delaying any task's control update, we follow the protocol in [21], as explained next. For a mode transition $M^g \Rightarrow M^h$, suppose that a mode transition request (MTR) is released at t_1 and the transition is completed at the completion of

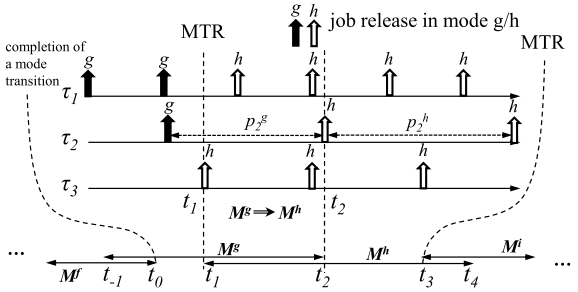


FIGURE 1. A Mode Transition Request (MTR) from M^g to M^h is released at t_1: while the mode transition does not change any task parameter of τ_1, it extends the period of τ_2 and introduces a new task τ_3 (meaning τ_3^g is a dummy task).

at least one job from every task. The transition completion cannot be later than t_2 when it is the earliest time at which jobs of all tasks with M^h are released or in active already, as shown in Fig. 1; M^h starts at no later t_2. We consider two types of tasks: (i) tasks whose parameters are not affected by the transition (i.e., $p_i^g = p_i^h$, $e_i^g = e_i^h$, and $d_i^g = d_i^h$) and (ii) other tasks, which satisfy at least one of $p_i^g \neq p_i^h$, $e_i^g \neq e_i^h$, or $d_i^g \neq d_i^h$. Then, the protocol does not affect release patterns of each task in (i) at all, e.g., τ_1 in the figure. For each task τ_i in (ii), the next release time (i.e., the earliest release time of jobs of τ_i after t_1) is not different from the time without the transition. However, there is a difference in that at the next release time, a job of τ_i^h (associated with a new mode M^h) is released instead of that of τ_i^g (associated with an old mode M^g), e.g., at t_2, a job of τ_2^h is released in the figure. After the release of the job with M^h, jobs of τ_i^h are periodically released until another MTR is released. Note that if τ_i^g is a dummy task, then a job of τ_i^h is released when an MTR is released; for example, τ_3^g in Fig. 1 is a dummy task, so a job of τ_3^h is released as soon as the MTR is released at t_1. Therefore, this transition protocol supports not only each task's transition without missing/delaying its control update, but also immediate task migration from other systems caused by any failure.

While many existing mode transition protocols require discarding unfinished jobs of the old-mode tasks, or must wait until a given time instant for synchronous release of jobs of the new-mode tasks (see the survey in [2]), this protocol does not perform such functions. Therefore, the protocol is suitable for real-time control systems, which require timely control

updates, even in the presence of transitions. For example, consider commercial flight between Detroit and LA (which are run multiple times every day) or robots repeating the same sequence of tasks (assembly, pick and place). In those applications, mode transitions caused by progress of application execution can be predicted based on a history of application execution. When it comes to mode transitions caused by unpredictable events such as component/subsystem failures, the mode transitions will be difficult to deal with, but the mode transitions are prepared (assigning and scheduling tasks) in advance using ''case'' statements, e.g., in case the left engine fails, we invoke new functions while removing existing functions.

C. EXPRESSION OF THE SYSTEM MODEL AND TRANSITION PROTOCOL

To provide better understanding of the proposed system model and transition protocol, this subsection expresses them using the existing well-defined semantics and syntax of specification language provided in [22]. The study utilizes notions of multiple types of tasks, jobs, actions and guards; action A presents the operation (e.g., abort, update, etc.) that each job/task should conduct during a mode transition, while guard G does the conditions (e.g., offsets after a mode-transition signal) for such an action to be in effect. To this end, we first categorize types of tasks and jobs under a mode transition of our considered protocol, and then describe their behaviors (during the mode transition) with notions of actions and guards.

A task τ_i under a mode transition $M^g \Rightarrow M^h$ (i.e., an active task in an interval between an MTR and the completion of the mode-transition) of our mode-transition protocol is categorized into the following two types.

- UNCHANGED : Tasks that are active in both M^g and M^h and all task parameters are not changed, and
- CHANGED : Tasks that are active in both M^g and M^h and at least one parameter of τ_i is changed.

Although there exist two more definitions of the task types in [22], which are OLD and NEW (i.e., τ_i that are active in M^g but not in M^h, and vise versa, respectively), τ_i in our mode-transition protocol does not belong to OLD and NEW since we use a dummy task to present τ_i that is not active in one of the two modes. That is, an OLD task in [22] is presented by a CHANGED task whose parameters for M^g and M^h are given and ($p_i^h = 1, e_i^h = 0, d_i^h = 1$), respectively. A NEW task in [22] is also presented by a CHANGED task whose parameters for M^g and M^h are ($p_i^g = 1, e_i^g = 0, d_i^g = 1$) and given, respectively.

In addition, a job J_i under a mode transition $M^g \Rightarrow M^h$ is categorized as follows, which is the same as [22].

- PENDING: Unfinished jobs that are not currently executing.
- EXECUTING: Unfinished jobs that are currently executing.
- NEW: New jobs that will be released after the mode transition request.

Next, we apply the following mode-transition action for PENDING and EXECUTING jobs in [22] to our mode transition protocol.

- CONTINUE: The jobs continue to be scheduled by a given scheduling algorithm with their current task parameters.

In [22], ABORT and UPDATE actions are provided for PENDING and EXECUTING jobs, which aborts the corresponding jobs (if ABORT) and updates the jobs' task parameters (if UPDATE); however, our mode transition protocol does not enforce such actions to PENDING and EXECUTING jobs.

For NEW jobs that should be released after a MTR, the following mode-transition actions can be applied.

- RELEASE: New jobs whose parameters are of M^h (i.e., destination mode) are released immediately as long as associated guards are true
- RELASE_O: New jobs whose parameters are of M^g (i.e., source mode) will be released after an MTR as long as associated guards are true.

Note that the release of the first NEW job with the destination mode's parameters is delayed until an associated guard becomes satisfied. When it comes to action guards provided in [22], our mode-transition protocol uses the following guard only.

- OFFSET_LR: The corresponding action is applied after the pre-defined offset time units elapse from the last release time of the corresponding task.

To summarize, our mode transition protocol $M^g \Rightarrow M^h$ is expressed by the existing well-defined semantics and syntax of specification language provided in [22]. That is, under a mode transition $M^g \Rightarrow M^h$, a task τ_i is expressed by UNCHANGED or CHANGED, and a job J_i is expressed by PENDING, EXECUTING or NEW. The action applied to a PENDING or EXECUTING job always follows CONTINUE without any action guard, meaning that a PENDING or EXECUTING job continues to be scheduled by the given scheduling algorithm until its completion. Then, RELEASE with ''OFFSET_LR: p_i^g'' is applied to a NEW job J_i of a CHANGED task τ_i, and RELASE_O with ''OFFSET_LR: p_j^g'' is applied to a NEW job J_j of a UNCHANGED task τ_j. This implies that a NEW job J_i will be released (after a MTR) when p_i^g time units elapse after from the last release time of the a CHANGED task τ_i. Also, a NEW job J_j will be released (after a MTR) when p_j^g time units elapse after from the last release time of the a UNCHANGED task τ_j.

D. EXISTING RTA FRAMEWORK

To assure no deadline miss of a set of single-mode tasks, many schedulability analysis techniques have been developed. Among them, the RTA (response-time analysis) technique is popular owing to not only its applicability to many scheduling algorithms such as earliest deadline first (EDF)

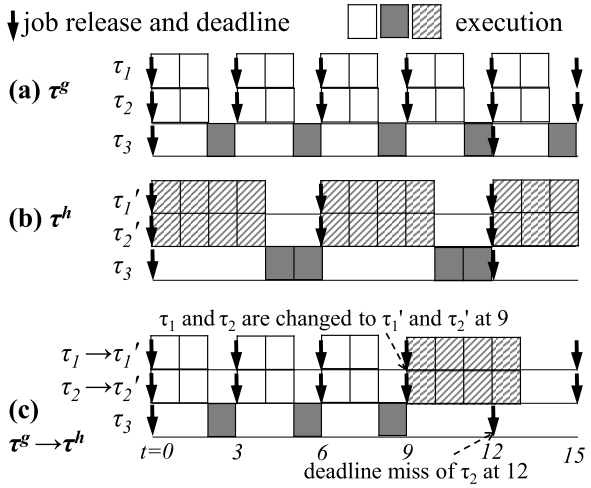


FIGURE 2. $\tau^g = \{\tau_1$**(period and relative deadline=3, execution time=2),** τ_2**(3,2),** τ_3**(12, 4)}** **as well as** $\tau^h = \{\tau_1'$**(6, 4),** τ_2'**(6, 4),** τ_3**(12, 4)}** **is schedulable by FP assuming the priority of** τ_1**,** τ_2**,** τ_1' **and** τ_2' **is the same, but higher than that of** τ_3 **on two processors without any transition. However, the task set is not schedulable in the presence of a transition from** τ^g **to** τ^h **at** $t = 9$**.**

and fixed-priority (FP), but also its schedulability performance, e.g., it is (one of) the best schedulability tests of EDF and FP on a multiprocessor platform [19], [23], [24].

For completeness, we now summarize the response-time analysis technique for single-mode tasks in real-time multiprocessor systems, which was originally described in [19] and has been recapitulated in many papers, e.g., [25]. The technique employs the notion of *interference* [26]. The interference on τ_k in $[a, b)$ (denoted by $I(\tau_k, a, b)$) is defined as the cumulative length of all intervals in $[a, b)$ such that a job of τ_k is ready to execute, but cannot execute owing to other higher-priority jobs' execution. Furthermore, the interference of τ_i on τ_k in $[a, b)$ (denoted by $I(\tau_k \leftarrow \tau_i, a, b)$) is defined as the cumulative length of all intervals in $[a, b)$ such that a job of τ_k cannot execute although it is ready to execute, but a job of τ_i executes instead. Because a job of τ_k cannot execute only when m other jobs execute, the following equation holds under any global work-conserving algorithm [26]:

$$I(\tau_k, a, b) = \frac{\sum_{\tau_i \in \tau \setminus \{\tau_k\}} I(\tau_k \leftarrow \tau_i, a, b)}{m}. \quad (1)$$

A property between $I(\tau_k, a, b)$ and $I(\tau_k \leftarrow \tau_i, a, b)$ has been derived in [26] as follows:

$$I(\tau_k, a, b) \geq x \iff \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left(I(\tau_k \leftarrow \tau_i, a, b), x\right) \geq m \cdot x. \quad (2)$$

Using Eqs. (1) and (2), the technique calculates an upper-bound of the time duration between the release and the completion of any job of τ_k (called the *response time* of τ_k). To do this, we compute the maximum interference of τ_i ($\neq \tau_k$) on τ_k in an interval of length ℓ starting from the release of any job of τ_k (denoted by $\mathbf{I}(\tau_k \leftarrow \tau_i, \ell)$), as follows:

$$\mathbf{I}(\tau_k \leftarrow \tau_i, \ell) \triangleq \max_{t \mid \text{the release time of any job of } \tau_k} I(\tau_k \leftarrow \tau_i, t, t+\ell). \quad (3)$$

Note that we define $\mathbf{I}(\tau_k \leftarrow \tau_i, \ell)$ only for $0 \leq \ell \leq d_k$, because we are interested in satisfying the timing requirements.

If the sum of the execution time of τ_k (i.e., e_k) and the maximum interference on τ_k in an interval of length ℓ ($\leq d_k$) starting from the release time of any job of τ_k is equal to or less than ℓ, any job of τ_k successfully finishes its execution within ℓ time units after its release. This leads to the following response-time analysis framework, using Eqs. (1) and (2).

Lemma 1 (Theorem 6 in [19]): When a set of single-mode tasks τ is scheduled by a global, preemptive, and work-conserving algorithm, an upper-bound of the response time of $\tau_k \in \tau$ is $r_k = r_k^{(x)}$ such that $r_k^{(x+1)} = r_k^{(x)}$ holds in the following expression, starting from $r_k^{(0)} = e_k$:

$$r_k^{(x+1)} \leftarrow e_k + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left(\mathbf{I}(\tau_k \leftarrow \tau_i, r_k^{(x)}), r_k^{(x)} - e_k + 1\right) \right\rfloor. \quad (4)$$

Then, if $r_k \leq d_k$ holds for all $\tau_k \in \tau$, τ is schedulable by the target scheduling algorithm. Note that the iteration of Eq. (4) for τ_k halts if $r_k^{(x)} > d_k$, implying that τ_k is unschedulable.

While the interference $\mathbf{I}(\tau_k \leftarrow \tau_i, \ell)$ varies with target scheduling algorithms, upper-bounds on the interference under EDF, FP, and any work-conserving algorithm were given in [19]. We present their generalizations in Section IV.

III. CHALLENGES AND OVERVIEW OF DEVELOPING RTA FRAMEWORK FOR MULTI-MODE TASKS

Developing the RTA framework for multi-mode tasks is much more complex than the DA framework case, where the main challenges arise in RTA when addressing the following questions:

Q1. How to identify the worst-case scenario inducing the maximum interference on each task in the presence of a mode transition, and how to effectively incorporate it into the RTA framework? (Subsections IV-A, IV-B, and IV-C)

Q2. How to safely upper-bound the slack value of each task (which is a key part of an effective RTA framework) in the presence of a mode transition? (Subsection IV-D)

Fig. 2 demonstrates the difficulty of addressing Q1 by showing that timing guarantees with a transition cannot be achieved even if the timing guarantees of both task sets before and after the transition are made. As shown in Figs. 2(a) and (b), the traditional RTA framework [27] for single-mode tasks guarantees the schedulability of a task set τ^g and that of another task set τ^h on two processors when each task set is scheduled by fixed priority (FP) [20] assuming the priority of τ_1, τ_2, τ_1' and τ_2' is the same, but higher than that of τ_3.

However, if τ^g makes a transition to τ^h at $t = 9$, τ_3 misses its deadline at $t = 12$, as shown in Fig. 2(c). Therefore, we need to develop a new RTA framework that can accommodate mode transitions.

As regards Q2, the slack value of each task (defined by the minimum interval between the finishing time and deadline of any job of the task) is essential to significantly reduce the worst-case interference from higher-priority tasks on a task of interest. For single-mode tasks, the traditional RTA framework conducts the slack reclamation scheme by iteratively reducing the worst-case response time of each task to derive upper-bounded slack values. However, such slack reclamation cannot be directly applied to multi-mode tasks in the presence of mode transitions because the execution pattern of other tasks in the presence of previous transitions affects the response times (as well as slack values) of tasks under the current transition, which does not occur in single-mode systems; we elaborate this phenomenon in detail in Section IV-D. By considering the trade-off between schedulability and computational overhead, we propose two different slack reclamation schemes in which (i) the slack value of each task is derived considering the transition history of previous modes, referred to as *chaining* slack reclamation, or (ii) the slack value is derived independently (but providing restricted analytical capability), referred to as *independent* slack reclamation.

Thus far, Q1 and Q2 focused on a system-wide mode transition protocol where the transitions of tasks occur concurrently in every individual system-wide mode transition. Thus, the order for tasks to complete their mode transitions within a system-wide mode transition depends on job release patterns and the mode transition request time. If the transition order of tasks in each system-wide mode transition can be controlled and thus predetermined offline by the system designer, we can further improve schedulability by reducing interference. That is, while the thus-developed analysis for addressing Q1 and Q2 provides safe guarantees on timing requirements, it can be pessimistic owing to its applicability to any arbitrary transition order of tasks in each system-wide mode transition. This entails the following question to improve schedulability of the RTA framework assuming predetermined task transition order in each system-wide mode transition.

Q3. How to improve schedulability by enforcing a transition order of tasks in a mode transition? (Section V)

Based on the investigation of how interference is reduced when a task-level transition order is enforced (called a *sequential transition*, as opposed to a *concurrent transition*), we address two issues: (i) how to guarantee timing requirements with a specific transition order of tasks? and (ii) how to find a transition order that guarantees the timing requirements of a task set specified by (i)? We achieve (i) by adapting the proposed schedulability analysis to a given transition order of tasks. For (ii), we derive some properties toward an optimal order under a given (restricted) condition, and then develop an effective transition order assignment.

IV. RTA FRAMEWORK FOR A MODE TRANSITION

In this section, we propose a new RTA framework for a mode transition, by generalizing the existing RTA framework considering a single mode in Lemma 1. We first extend it to consider a mode transition for a given task of interest. Then, we calculate upper-bounds of the interference for such a task in the presence of a mode transition under any work-conserving algorithm, EDF or FP. We finally present how to conduct slack reclamation to improve schedulability of the proposed RTA framework.

A. EXTENSION OF THE EXISTING RTA FRAMEWORK

The schedulability analysis framework in Subsection II-D assumes there is only one mode of each task, so we must extend the framework to multiple modes.

A task executing in the presence of a mode transition $M^g \Rightarrow M^h$ $(1 \le g \le h = g+1 \le \mu)$ can interfere with another task with either mode M^g or M^h. To express this, let $\tau_i^{g\Rightarrow h}$ denote τ_i in the presence of a transition $M^g \Rightarrow M^h$. Then, we define interference of $\tau_i^{g\Rightarrow h}$ to τ_k^u (u is either g or h) in $[a, b)$ as the cumulative length of all intervals in $[a, b)$ such that a job of τ_k^u cannot execute although it is ready for execution, because a job of τ_i^g or τ_i^h executes instead. Let $I(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, a, b)$ denote such interference. Similar to Eq. (3), we define $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ as follows.

$$\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell) \triangleq \max_{t \mid \text{the release time of any job of } \tau_k^u} I(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, t, t+\ell). \quad (5)$$

For response time analysis of a given task τ_k in the presence of a transition $M^g \Rightarrow M^h$, we also consider two modes of the task because a task in one mode has different task parameters from the same task in a different mode. This means that we should calculate the response time of both τ_k^g and τ_k^h for a transition $M^g \Rightarrow M^h$. Thus, we generalize the response-time analysis framework in Lemma 1 as follows.

Lemma 2: Suppose that a task set τ makes a transition $M^g \Rightarrow M^h$ and is scheduled by a global, preemptive, and work-conserving algorithm. Then, an upper-bound of the response time of $\tau_k^u \in \tau$ in the presence of $M^g \Rightarrow M^h$ (u is either g or h) is $r_k^u = r_k^{u(x)}$ such that $r_k^{u(x+1)} = r_k^{u(x)}$ holds in the following expression, starting from $r_k^{u(0)} = e_k^u$:

$$r_k^{u(x+1)} \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left(\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, r_k^{u(x)}), r_k^{u(x)} - e_k^u + 1\right) \right\rfloor. \quad (6)$$

If $r_k^u \le d_k^u$ holds for all $\tau_k^u \in \tau$ and $u \in \{g, h\}$, then τ is schedulable by the algorithm.

Note that the iteration of Eq. (6) for τ_k^u halts if $r_k^{u(x)} > d_k^u$, implying that τ_k^u is unschedulable.

Proof: The lemma holds by Lemma 1 and the definitions of $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell)$. □

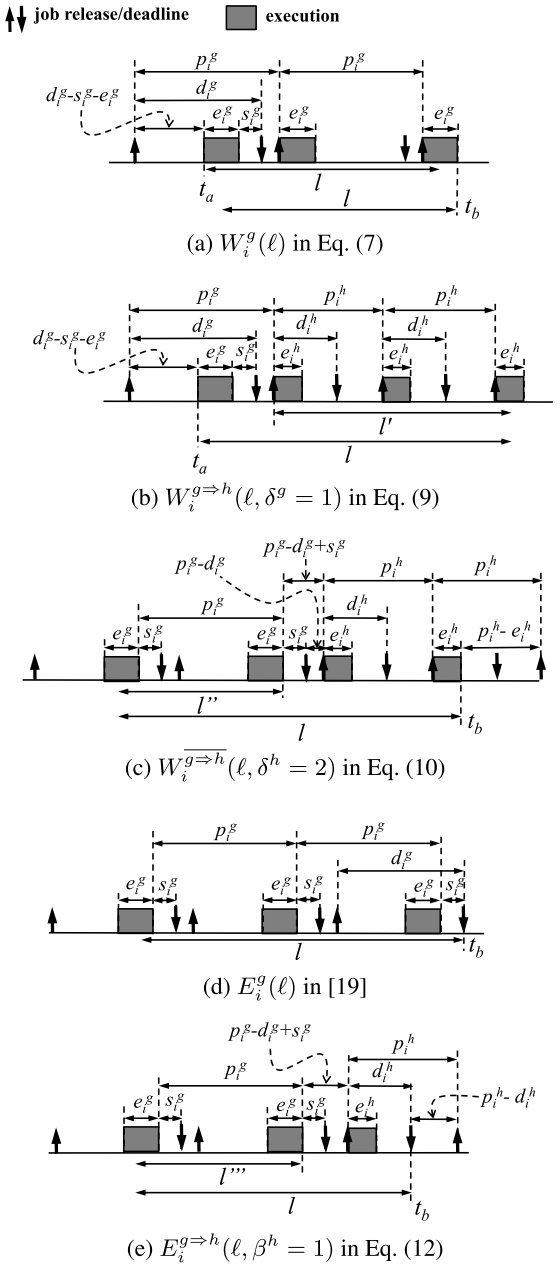


FIGURE 3. Release and execution patterns that derive $W_i^g(\ell)$, $W_i^{g\Rightarrow h}(\ell, \delta^g = 1)$, $W_i^{\overline{g\Rightarrow h}}(\ell, \delta^h = 2)$, $E_i^g(\ell)$ and $E_i^{g\Rightarrow h}(\ell, \beta^h = 1)$, where an interval of interest of length ℓ starts at t_a or ends at t_b.

Then, how to set the upper-bound of interference $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ is the most critical part of the response-time analysis for a mode transition, as addressed in the following subsection.

B. INTERFERENCE CALCULATION

Because the interference depends on the scheduling algorithm, we now calculate two different types of upper-bounds for the interference. First, we compute the maximum amount of execution of jobs of a given task under any work-conserving algorithm. Second, we calculate a tighter interference upper-bound for a specific scheduling algorithm; as examples, we derive upper-bounds for EDF and FP.

1) INTERFERENCE UNDER ANY-WORK CONSERVING ALGORITHM

For single-mode tasks, it has been identified which release and execution patterns maximize the amount of execution of a single task τ_i^g's jobs in an interval of length ℓ. As shown in Fig. 3(a), the patterns are either (i) the first job of τ_i executes as late as possible and starts its execution at the beginning of the interval (see an interval starting at t_a), or (ii) the last job of τ_i executes as early as possible and finishes its execution at the end of the interval (see an interval ending at t_b). Both patterns derive the same amount of execution of τ_i^g's jobs [19], [23], and the amount of execution in case of (i) (denoted by $W_i^g(\ell)$) is calculated by [19] as follows.

$$W_i^g(\ell) \triangleq F_i^g(\ell + d_i^g - s_i^g - e_i^g), \tag{7}$$

where $F_i^g(\ell)$ represents the amount of execution of jobs of τ_i^g in an interval of length ℓ, when the first job is released at the beginning of the interval and all jobs of τ_i^g in the interval execute as early as possible. Then, we can mathematically express $F_i^g(\ell)$ as follows:

$$F_i^g(\ell) \triangleq \begin{cases} \left\lfloor \frac{\ell}{p_i^g} \right\rfloor \cdot e_i^g + \min\left(e_i^g, \ell - \left\lfloor \frac{\ell}{p_i^g} \right\rfloor \cdot p_i^g\right), & \text{if } l > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Further, s_i^g denotes the slack value of τ_i^g, which represents the minimum interval length between the finishing time and the deadline of any job of τ_i^g. Then, any job of τ_i^g completes its execution at least s_i^g time units ahead of its deadline. We will present how to compute s_i^g in Section IV-D.

Regarding multi-mode tasks, it is more difficult to determine the maximum amount of execution of jobs of τ_i because the amount depends not only on release and execution patterns, but also on the time of an MTR. However, we note an interesting property of the maximum amount, as stated in the following observation.

Observation 1: Suppose that τ makes a transition from τ^g to τ^h in an interval of length ℓ, in which the scheduling window of at least one job of τ_i^g and at least one job of τ_i^h (partially or entirely) overlap with the interval as shown in Figs. 3(b) and 3(c). Then, the amount of execution of jobs of both τ_i^g and τ_i^h in the interval is maximized with one of the following release and execution patterns: (i) when the first job of τ_i^g is executed as late as possible and starts its execution at the beginning of the interval of length ℓ, and the last job of τ_i^h is executed as early as possible as shown in Fig. 3(b), and (ii) when the last job of τ_i^h is executed as early as possible and finishes its execution at the end of the interval of length ℓ, and the first job of τ_i^g is executed as late as possible as shown in Fig. 3(c).

Then, the following lemma proves the correctness of Observation 1.

Lemma 3: The amount of execution of jobs of both τ_i^g and τ_i^h in the interval of length ℓ is maximized with one of release and execution patterns (i) and (ii).

Proof: Suppose that a release and execution pattern not belonging to (i) and (ii) makes the larger amount of execution of jobs of both τ_i^g and τ_i^h in the interval of length ℓ than those belonging to (i) and (ii). If a release and execution pattern belongs to neither (i) nor (ii), then shifting the release pattern towards (i) or (ii) yields the larger (or equal) amount of execution for jobs of both τ_i^g and τ_i^h. This contradicts the supposition, and thus the lemma holds. □

A naive approach requires an exhaustive search for the release time of an MTR because we develop a schedulability analysis that an MTR can be released at any time in the interval of interest. By applying Observation 1, we are able to evaluate only the following two types of patterns:

P1. Release and execution patterns of (i) with a situation where the scheduling window of a given number of jobs of τ_i^g (denoted by δ^g) overlaps with the interval of interest.
P2. Release and execution patterns of (ii) with a situation where the scheduling window of a given number of jobs of τ_i^h (denoted by δ^h) overlaps with the interval of interest.

P1 with $\delta^g = 1$ is depicted in Fig. 3(b). The amount of execution of jobs of τ_i^g in the interval of interest (starting at t_a) is equal to $\delta^g \cdot e_i^g$, and that of τ_i^h is calculated by $F_i^g(\ell')$ where $\ell' = \ell + (d_i^g - s_i^g - e_i^g) - \delta^g \cdot p_i^g$ as shown in Fig. 3(b). With the reasoning, the amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ with P1 and a given δ^g is upper-bounded as follows.

Lemma 4: The amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ with P1 and a given δ^g (denoted by $W_i^{g \Rightarrow h}$), is calculated by

$$W_i^{g \Rightarrow h}(\ell, \delta^g) \triangleq \delta^g \cdot e_i^g + F_i^h(\ell'), \tag{9}$$

where $1 \le \delta^g \le \lfloor (\ell + d_i^g - s_i^g - e_i^g)/p_i^g \rfloor$ and $\ell' = \ell + (d_i^g - s_i^g - e_i^g) - \delta^g \cdot p_i^g$.

Proof: For a given δ^g, δ^g jobs of τ_i^g contribute to $W_i^{g \Rightarrow h}(\ell, \delta^g)$. Then, an interval of length ℓ' where the first job of τ_i^h starts its execution at the beginning of ℓ' is calculated by adding $d_i^g - s_i^g - e_i^g$ to ℓ, and deducting $\delta^g \cdot p_i^g$ from it (i.e., $\ell' = \ell + (d_i^g - s_i^g - e_i^g) - \delta^g \cdot p_i^g$). By exploiting $F_i^h(\ell')$, we can derive the amount of jobs of τ_i^h that can contribute to $W_i^{g \Rightarrow h}(\ell, \delta^g)$. Thus, the lemma holds. □

Similarly, P2 with $\delta^h = 2$ is depicted in Fig. 3(c). The amount of execution of jobs of τ_i^h in the interval of interest (ending at t_b) is equal to $\delta^h \cdot e_i^h$, and that of τ_i^g is calculated by $F_i^g(\ell'')$ where $\ell'' = \ell + p_i^h - e_i^h - (p_i^g - d_i^g + s_i^g) - \delta^h \cdot p_i^h$ as shown in Fig. 3(c).

With the reasoning, the amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ in P2 and given δ^h (denoted by $W_i^{\overline{g \Rightarrow h}}$) is upper-bounded as follows.

Lemma 5: The amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ in P2 and given δ^h (denoted by $W_i^{\overline{g \Rightarrow h}}$) is calculated by

$$W_i^{\overline{g \Rightarrow h}}(\ell, \delta^h) \triangleq \delta^h \cdot e_i^h + F_i^g(\ell''), \tag{10}$$

where $1 \le \delta^h \le \lfloor (\ell + p_i^h - e_i^h)/p_i^h \rfloor$ and $\ell'' = \ell + p_i^h - e_i^h - (p_i^g - d_i^g + s_i^g) - \delta^h \cdot p_i^h$.

Proof: For a given δ^h, δ^h jobs of τ_i^h contribute to $W_i^{\overline{g \Rightarrow h}}$. Then, an interval of length ℓ'' where the last job of τ_i^g completes its execution at the end of ℓ'' is calculated by adding $p_i^h - e_i^h$ to ℓ, and deducting $\delta^h \cdot p_i^h$ and $p_i^g - d_i^g + s_i^g$ from it (i.e., $\ell'' = \ell + p_i^h - e_i^h - (p_i^g - d_i^g + s_i^g) - \delta^h \cdot p_i^h$). By exploiting $F_i^g(\ell'')$, we can derive amount of jobs of τ_i^g that can contribute to $W_i^{g \Rightarrow h}(\ell, \delta^g)$. Thus, the lemma holds. □

Then, an upper-bound of the amount of execution of jobs of τ_i^g and τ_i^h in an interval of length ℓ is the maximum among the cases where only jobs of τ_i^g are executed in the interval (when the MTR occurs after the interval) and only jobs of τ_i^h are executed in the interval (when the MTR occurs before the interval), and the cases of P1 and P2 (in which the MTR occurs within the interval). In summary, the upper-bound (denoted by $\mathbf{W}_i^{g \Rightarrow h}(\ell)$) is calculated as follows:

$$\mathbf{W}_i^{g \Rightarrow h}(\ell) \triangleq \max \Bigg\{ W_i^g(\ell), W_i^h(\ell), \max_{1 \le \delta^g \le \lfloor (\ell + d_i^g - s_i^g - e_i^g)/p_i^g \rfloor} W_i^{g \Rightarrow h}(\ell, \delta^g), \max_{1 \le \delta^h \le \lfloor (\ell + p_i^h - e_i^h)/p_i^h \rfloor} W_i^{\overline{g \Rightarrow h}}(\ell, \delta^h) \Bigg\}. \tag{11}$$

As a job can interfere with another job only when the job is executed, $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g \Rightarrow h}, \ell)$ (u is either g or h, and $\ell \le d_k^u$) in Lemma 2 is upper-bounded by $\mathbf{W}_i^{g \Rightarrow h}(\ell)$ under any work-conserving algorithm.

2) INTERFERENCE UNDER FP AND EDF

FP schedules jobs according to pre-determined task-level priorities. Therefore, under FP, if τ_i has a higher priority than τ_k, then $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g \Rightarrow h}, \ell)$ is upper-bounded by $\mathbf{W}_i^{g \Rightarrow h}(\ell)$; otherwise, any job of τ_i cannot interfere with the jobs of τ_k, meaning that $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g \Rightarrow h}, \ell) = 0$.

EDF determines jobs' priorities based on their deadlines; a job with an earlier deadline has a higher priority than a job with a later deadline. Therefore, a job J_A can interfere with another job J_B only when the deadline of J_A is no later than that of J_B. We derive a property that can be used to derive an upper-bound of the interference under EDF, as stated in the following observation.

Observation 2: Suppose that τ makes a transition from τ^g to τ^h in an interval of length ℓ, in which at least one job of τ_i^g and at least one job of τ_i^h (partially or entirely) overlap with the interval as shown in Fig. 3(e). Then, the amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ is maximized in the following scenario: (iii) deadlines of jobs

of τ_i^g and τ_i^h are no later than the end of the interval (t_b in Fig. 3(e)), and the deadline of the last job of τ_i^h is equal to the end of the interval, and the first job of τ_i^g executes as late as possible as shown in Fig. 3(e).

Then, the following lemma proves the correctness of Observation 2.

Lemma 6: The amount of execution of jobs of both τ_i^g and τ_i^h in the interval of length ℓ under EDF is maximized with the release and execution patterns (iii).

Proof: Suppose that a release and execution pattern not belonging to (iii) makes the larger amount of execution of jobs of both τ_i^g and τ_i^h under EDF in the interval of length ℓ than those belonging to (iii). If we shift the job releases of (iii) to slightly later, the last job's deadline is later than the end of the interval, which means that the last job cannot interfere with a job whose deadline is the end of the interval under EDF. Shifting the job releases to the other direction also does not increase the amount of interference. This contradicts the supposition, and thus the lemma holds. □

Let β^h denote the number of jobs of τ_i^h whose scheduling windows overlap the interval of interest when the release and execution patterns accord with Observation 2, e.g., $\beta^h = 1$ in Fig. 3(e). Then, the amount of execution of jobs of τ_i^h in the interval is equal to $\beta^h \cdot e_i^h$, and that of τ_i^g is calculated by $F_i(\ell''')$ where $\ell''' = \ell + p_i^h - d_i^h - (p_i^g - d_i^g + s_i^g) - \beta^h \cdot p_i^h$.

With the reasoning, the amount of execution of jobs of τ_i^g and τ_i^h in an interval of length ℓ whose deadlines are no later than the end of the interval (denoted by $E_i^{g\Rightarrow h}(\ell, \beta^h)$), is upper-bounded as follows

Lemma 7: The amount of execution of jobs of τ_i^g and τ_i^h in the interval of length ℓ whose deadlines are no later than the end of the interval (denoted by $E_i^{g\Rightarrow h}(\ell, \beta^h)$), is calculated by

$$E_i^{g\Rightarrow h}(\ell, \beta^h) \triangleq \beta^h \cdot e_i^h + F_i^g(\ell'''), \tag{12}$$

where $1 \le \beta^h \le \lfloor(\ell + p_i^h - d_i^h)/p_i^h\rfloor$ and $\ell''' = \ell + p_i^h - d_i^h - (p_i^g - d_i^g + s_i^g) - \beta^h \cdot p_i^h$.

Proof: For a given δ^h, δ^h jobs of τ_i^h contribute to $W_i^{g\Rightarrow h}$. Then, an interval of length ℓ''' where the last job of τ_i^g completes its execution at the end of ℓ''' is calculated by adding $p_i^h - d_i^h$ to ℓ, and deducting $\delta^h \cdot p_i^h$ and $p_i^g - d_i^g + s_i^g$ from it (i.e., $\ell'' = \ell + p_i^h - d_i^h - (p_i^g - d_i^g + s_i^g) - \beta^h \cdot p_i^h$). By exploiting $F_i^g(\ell''')$, we can derive amount of jobs of τ_i^g that can contribute to $W_i^{g\Rightarrow h}(\ell, \delta^g)$. Thus, the lemma holds. □

Similar to $\mathbf{W}_i^{g\Rightarrow h}(\ell)$, an upper-bound of the amount of execution of jobs of τ_i^g and τ_i^h in an interval of length ℓ whose deadlines are no later than the end of the interval is the maximum among the cases when the MTR occurs outside of the interval (either only the jobs of τ_i^g or those of τ_i^h execute in the interval, which is calculated by $E_i^g(\ell) \triangleq F_i^g(\ell - s_i^g)$ or $E_i^h(\ell) \triangleq F_i^h(\ell - s_i^h)$ [19], as shown in Fig. 3(d)), and the case when the MTR occurs in the interval $E_i^{g\Rightarrow h}(\ell, \beta^h)$ with different β^h. In summary, the upper-bound (denoted by $\mathbf{E}_i^{g\Rightarrow h}(\ell)$) is calculated by

$$\mathbf{E}_i^{g\Rightarrow h}(\ell) \triangleq \max\left(E_i^g(\ell), E_i^h(\ell), \max_{1\le\beta^h\le\left\lfloor(\ell+p_i^h-d_i^h)/p_i^h\right\rfloor} E_i^{g\Rightarrow h}(\ell, \beta^h)\right). \tag{13}$$

Because a job with a later deadline cannot interfere with a job with an earlier deadline, $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ (u is either g or h, and $\ell \le d_k^u$) in Lemma 2 is upper-bounded by $\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$ under EDF. Incorporating this into an upper-bound of the amount of execution under any work-conserving algorithm, then $\mathbf{I}(\tau_k^u \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ under EDF is finally upper-bounded by $\min\left(\mathbf{W}_i^{g\Rightarrow h}(\ell), \mathbf{E}_i^{g\Rightarrow h}(d_k^u)\right)$.

C. NEW RTA FRAMEWORK FOR FP AND EDF

Using Lemma 2 and the derived upper-bounds on the amount of interference, we derive the RTA framework for a mode transition under FP and EDF as follows:

Theorem 1: Suppose that a task set τ makes a transition $M^g \Rightarrow M^h$ and is scheduled by FP (likewise EDF). Then, an upper-bound of the response time of $\tau_k^u \in \tau$ in the presence of $M^g \Rightarrow M^h$ (u is either g or h) is $r_k^u = r_k^{u(x)}$ such that $r_k^{u(x+1)} = r_k^{u(x)}$ holds in Eq. (14) (likewise Eq. (15)), starting from $r_k^{u(0)} = e_k^u$:

$$r_k^{u(x+1)} \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i\in\tau\setminus\{\tau_k\}} \min\left(\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)}), r_k^{u(x)} - e_k^u + 1\right)\right\rfloor, \tag{14}$$

$$r_k^{u(x+1)} \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i\in\tau\setminus\{\tau_k\}} \min\left(\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)}), \mathbf{E}_i^{g\Rightarrow h}(d_k^u), r_k^{u(x)} - e_k^u + 1\right)\right\rfloor. \tag{15}$$

Then, if $r_k^{u(x)} \le d_k^{u(x)}$ holds for all $\tau_k^u \in \tau$ and $u \in \{g, h\}$, τ is schedulable by FP (likewise EDF). The iteration of Eq. (14) (likewise Eq. (15)) for τ_k halts if $r_k^{u(x)} > d_k^{u(x)}$, implying τ_k^u is unschedulable. Note that Eq. (14) holds only when τ_i has a higher priority than τ_k; otherwise, $\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)})$ should be replaced with 0.

Proof: The theorem holds by Lemma 2 and the derivation of $\mathbf{W}_i^{g\Rightarrow h}(\ell)$ and $\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$. □

D. SLACK RECLAMATION SCHEMES

In the previous subsection, we developed the RTA framework for a mode transition in Theorem 1 and upper-bounds on the interference under any work-conserving, FP and EDF. In this subsection, we present the mechanism deriving slack values s_i^g and s_i^h for the computation of $\mathbf{W}_i^{g\Rightarrow h}(\ell)$ and $\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$ in an iterative manner, which is referred to as *slack reclamation*.

By definition, we can calculate the slack value by the relative deadline minus the response time, e.g., $s_i^g = d_i^g - r_i^g$.

Algorithm 1 RTA Framework With CSR

1: $S_i = \infty$ for all $\tau_i \in \tau$
2: **for** every mode transition $M^x \Rightarrow M^y$ from $M^1 \Rightarrow M^2$ to $M^g \Rightarrow M^h$ **do**
3: $s_i^x \leftarrow 0$ and $s_i^y \leftarrow 0$ for all $\tau_i \in \tau$.
4: **for** every task τ_i **do**
5: **while** true **do**
6: $r_i^x \leftarrow d_i^x - s_i^x$ and $r_i^y \leftarrow d_i^y - s_i^y$ for all $\tau_i \in \tau$.
7: **for** $\tau_i^x \in \tau^x$ **do**
8: Calculate new r_i^x using Theorem 1 with the upper-bounds according to the scheduling algorithm.
9: **if** $r_i^x < d_i^x$ **then**
10: $s_i^x \leftarrow \min(d_i^x - r_i^x, S_i)$
11: **end if**
12: **end for**
13: **for** $\tau_i^y \in \tau^y$ **do**
14: Calculate new r_i^y using Theorem 1 with the upper-bounds according to the scheduling algorithm.
15: **if** $r_i^y < d_i^y$ **then**
16: $s_i^y \leftarrow d_i^y - r_i^y$ (*no upper-bound*)
17: **end if**
18: **end for**
19: **if** there is no update of s_i^x and s_i^y for all $\tau_i \in \tau$ in Steps 7 and 13 **then**
20: **if** $r_i^x \le d_i^x$ and $r_i^y \le d_i^y$ hold for all $\tau_i \in \tau$ **then**
21: $S_i = s_i^y$
22: goto Step 26.
23: **end if**
24: Return UNSCHEDULABLE.
25: **end if**
26: **end while**
27: **end for**
28: **end for**
29: Return SCHEDULABLE.

This means, however, that when we compute τ_k's response time, we need the response times of other tasks τ_i ($\neq \tau_k$) as shown in Eqs. (14) and (15). To solve this chicken and egg problem, the response time analysis for single-mode tasks [19] employs iterations for slack reclamation. In the first loop, all slack values are assumed to be zero, and we calculate all tasks' response times. After the calculation, the slack value is updated only if the response time is strictly less than the relative deadline (meaning a positive slack value). Then, with the updated slack values, this process is repeated until all response times are no larger than their relative deadlines (schedulable) or there is no more update of slack values (unschedulable).

This slack reclamation process is correct because of two reasons. First, the response time analysis finds a task that triggers the first deadline miss; therefore, when we calculate the response time of a task, we can assume that all other tasks do not miss their deadlines, i.e., non-negative slack values.

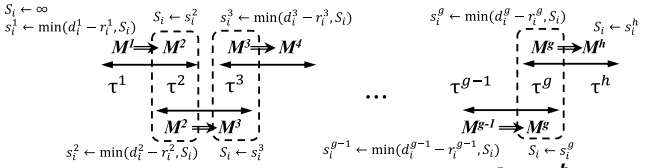


FIGURE 4. Procedure of chaining slack reclamation for τ^g and τ^h in the presence of a mode transition $M^g \Rightarrow M^h$.

Second, there exists only one mode of each task; the response time of a task is only affected by task parameters of a given task set with a single, fixed mode.

However, such slack reclamation cannot be directly applied to multi-mode tasks because the slack values of tasks in each mode transition $M^g \Rightarrow M^{g+1}$ are affected by the behavior of the previous transitions, which is explained as follows. Suppose that a task set τ experiences two transitions $M^{g-1} \Rightarrow M^g$ and $M^g \Rightarrow M^{g+1}$, and we want to utilize a slack value s_k^g of τ_k^g in the presence of a mode transition $M^g \Rightarrow M^{g+1}$. By applying Theorem 1 separately for $M^{g-1} \Rightarrow M^g$ and $M^g \Rightarrow M^{g+1}$, we obtain two response times of τ_k^g (i.e., r_k^g), and two slack values of τ_k^g (i.e., $s_k^g = d_k^g - r_k^g$). Therefore, to utilize a slack value s_k^g of τ_k^g in the presence of a mode transition $M^g \Rightarrow M^{g+1}$, we must choose the lowest of the two slack values of s_k^g, and use it for calculating τ_k^g's interference to other tasks τ_i ($\neq \tau_k$) in Eqs. (11) and (13). This phenomenon also occurs for s_k^{g+1}, meaning that s_k^{g+1} is derived by considering both $M^g \Rightarrow M^{g+1}$ and $M^{g+1} \Rightarrow M^{g+2}$.

Therefore, we must track each slack value of each task for every transition from $M^1 \Rightarrow M^2$ to $M^g \Rightarrow M^h$ sequentially to derive upper-bounds of s_k^g and s_k^h, which we call *chaining* slack reclamation (CSR). Alg. 1 describes how the RTA framework works for a given mode transition $M^g \Rightarrow M^h$ with CSR, where we denote the upper-bound of s_i^g by S_i^g that is derived in the calculation for a previous mode. It first sets infinity to the upper-bound of each slack value S_i for all tasks $\tau_i \in \tau$ (Line 1). Then, it sequentially considers each transition $M^x \Rightarrow M^y$ ($1 \le x \le y = x + 1 \le h$) from $M^1 \Rightarrow M^2$ to $M^g \Rightarrow M^h$ (Line 2). In each sequence, the slack values of each task τ_i for mode M^x and M^y are initialized to 0 (Line 3) and each task τ_i is considered one by one (Line 4). For each task τ_i, s_i^x (*likewise*, s_i^y) is set to the difference between the response time derived by Theorem 1 and the absolute deadline of τ_i with the upper-bound for M^x, i.e., S_i (*likewise*, without the upper-bound for M^y) as shown in Lines 7–12 (*likewise*, Lines 13–18). If there is no update for both modes M^x and M^y, the schedulability of τ_i is judged; if τ_i is deemed schedulable, S_i for the next mode is updated by the slack value of τ_i^y (Lines 19–25). Then, the task set is deemed schedulable if every task is deemed schedulable for every mode transition (Line 29). Note that the RTA framework with CSR does not require any information involving release or execution patterns of jobs and the release time of an MTR; instead, it only requires the order of system-wide mode transitions.

Fig. 4 presents how slack values s_i^x and s_i^y are updated while Alg. 1 is conducted. As shown in Fig. 4, the RTA is conducted g times each with its corresponding mode transition

$M^x \Rightarrow M^y$ $(1 \leq x \leq y = x+1 \leq h)$ (as Line 2 in Alg. 1 indicates) and s_i^g is upper-bounded by S_i that is derived in the previous iteration. For example, s_i^3 in the present of a mode transition $M^3 \Rightarrow M^4$ is upper-bounded by S_i that is obtained from the previous iteration for a mode transition $M^2 \Rightarrow M^3$. Because s_i^3 is calculated with the consideration of interference from $\tau_i^2 \in \tau^2$ in the presence of a mode transition $M^2 \Rightarrow M^3$, it needs an upper-bounded value of s_i^2. Also, it holds for s_i^2 that is upper-bounded by S_i obtained from the previous iteration for a mode transition $M^1 \Rightarrow M^2$. Such chaining effect requires tracking slack value of each task for every transition from $M^1 \Rightarrow M^2$ to $M^g \Rightarrow M^h$ sequentially to derive upper-bounds of s_k^g and s_k^h.

The time complexity of Algorithm 1 is derived as follows. In Line 2, Algorithm 1 considers each transition $M^x \Rightarrow M^y$ $(1 \leq x \leq y = x+1 \leq h)$ from $M^1 \Rightarrow M^2$ to $M^g \Rightarrow M^h$ $(O(\mu))$. Then, each task τ_i is considered one by one in Line 4 $(O(n)$ where is n is the number of tasks in $\tau)$. Lines 5–26 can be conducted until there is no update of s_i^x and s_i^y in Line 19, and values of s_i^x and s_i^y are in $[0, maxD)$ where $maxD$ is the largest value among all d_i of tasks in $\tau^x \cup \tau^y$ $(O(maxD))$. In Lines 7–12, Theorem 1 is utilized $(O(n))$ for each task $\tau_i^x \in \tau^x$ $(O(n))$; the same is conducted for $\tau_i^y \in \tau^y$ in Lines 13–18. Therefore, the time complexity of Algorithm 1 is $O(\mu) \cdot O(n) \cdot O(maxD) \cdot 2 \cdot O(n) = O(\mu \cdot maxD \cdot n^2)$.

As shown in Alg. 1, the RTA framework with CSR is conducted for every mode transition $M^x \Rightarrow M^y$, which may require high computational overhead for a large value of g. To relieve such overhead, we propose another slack reclamation scheme, called *independent* slack reclamation (ISR). The RTA framework with ISR assumes no information of the tasks of the previous modes in τ^g, and s_i^g is fixed to zero meaning that we only use the fundamental assumption that there is no deadline miss so far. Thus, the RTA framework with ISR is conducted for the current transition $M^g \Rightarrow M^h$ and reclaims a slack value of τ^h only, not that of τ^g. Hence, Alg. 1 is simplified when ISR is applied instead of CSR; the change is as follows. $M^x \Rightarrow M^y$ indicates $M^g \Rightarrow M^h$ only (Line 2). s_i^x is not updated (Lines 9–11). Also, S_i needs not to be updated by s_i^y for the next mode transition's s_i^x because s_i^x is fixed to zero under ISR (Line 21).

Now, we discuss which information is necessary to the system designer in order to utilize the proposed RTA framework. Since we aim at developing offline schedulability analysis for online scheduling algorithm for multi-mode tasks, we assume that only offline information of the target system is available. Basically, task parameters (i.e., p_i, e_i, and d_i) for each mode M^g $(1 \leq g \leq \mu)$ are only the given information before using our RTA framework. When we use ISR, no further information is necessary; on the other hand, the sequence of mode transitions should be known when we apply CSR. When it comes to task parameters, p_i and d_i are naturally determined by a system designer, and e_i should be safely derived (i.e., upper-bounded) in the course of program analysis via various static or dynamic techniques; for example, the worst-case execution time estimation upon multiple inputs. Then, the system designer can utilize our RTA framework.

V. RTA FRAMEWORK FOR A SEQUENTIAL MODE TRANSITION

In this section, we discuss how to improve schedulability of the proposed RTA framework by enforcing a constraint in order of individual tasks' mode transitions when each mode transition $M^g \Rightarrow M^h$ is performed, and develop an effective transition order assignment framework based on deriving key observations.

A. SEQUENTIAL TRANSITION PROTOCOL

In the previous section, we presented the proposed RTA framework for multi-mode tasks in the presence of a mode transition conducted by the mode-transition protocol described in Section II-B. The protocol assumes that tasks' transitions occur concurrently starting from the MTR in every individual system-wide mode transition, and thus tasks will complete their mode transitions in arbitrary order, which cannot be controlled by the system designer offline. Because of this property, the proposed RTA framework should be independent of job release and execution patterns and the release time of an MTR, but it over-estimates the interference, to be discussed below.

In Fig. 1, after the release of an MTR for a transition $M^g \Rightarrow M^h$ at t_1, the first jobs of τ_1^h and τ_3^h are released earlier than t_2, while that of τ_2^h is released at t_2. Therefore, the scheduling window of any job of τ_2^h does not overlap with that of any job of tasks in M^g. If we enforce such a transition order of tasks within a single transition (e.g., M^g to M^h), we can rule out other tasks in mode M^g in the calculation of the amount of interference to τ_2^h during the transition $M^g \Rightarrow M^h$. This reduces the interference and increases the possibility of τ_2^h's schedulability.

To exploit the merit of such interference reduction, we propose a new mode-transition protocol that allows only one task's transition at a time (whose order can be predetermined by the system designer offline), which is referred to as a *sequential transition*. The sequential transition is opposed to the transition protocol described in Section II-B in that it allows multiple tasks to transit from one mode to another concurrently, which we call a *concurrent transition*. A sequential transition can improve the schedulability performance over a corresponding concurrent transition, and this improvement can be achieved at the expense of increasing the system's transition time. Within a sequential transition, a task's transition may be delayed until completion of the preceding task's transition, thereby prolonging the system's transition time.

There are two challenges in making timing guarantees under the sequential transition: (i) how to guarantee schedulability with a given sequential transition, and (ii) how to find the effective transition order of tasks, whose schedulability is guaranteed by (i), which will be addressed in Subsections V-B and V-C, respectively.

B. NEW RTA FRAMEWORK FOR A SEQUENTIAL TRANSITION

We aim at developing a new RTA framework to accommodate a sequential transition by modifying the one for a concurrent transition proposed in Theorem 1. Suppose that a task set τ makes a sequential transition $M^g \Rightarrow M^h$ in a given order. To express the relative transition order of tasks, let $\tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h}$ denote a situation where τ_k's transition $M^g \Rightarrow M^h$ is performed before τ_i's transition, meaning that the scheduling window of any job of τ_k^g cannot overlap with that of any job of τ_i^h, e.g., $\tau_1^{g\Rightarrow h} \prec \tau_2^{g\Rightarrow h}$ holds in Fig. 1. Therefore, if $\tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h}$ holds, then any job of τ_i^h cannot interfere with any job of τ_k^g. This implies that the interference of $\tau_i^{g\Rightarrow h}$ on τ_k^g is reduced by that of τ_i^g on τ_k^g. Hence, if $\tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h}$ holds, the upper-bound of $\mathbf{I}(\tau_k^g \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ under FP (when τ_i has a higher priority than τ_k) in Theorem 1 changes from $\mathbf{W}_i^{g\Rightarrow h}(\ell)$ to ${}^{\mathsf{g}}\mathbf{W}_i^{g\Rightarrow h}(\ell)$, where

$$ {}^{\mathsf{g}}\mathbf{W}_i^{g\Rightarrow h}(\ell) \triangleq \begin{cases} W_i^g(\ell), & \text{if } \tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h} \text{ (reduced)}, \\ \mathbf{W}_i^{g\Rightarrow h}(\ell), & \text{if } \tau_k^{g\Rightarrow h} \succ \tau_i^{g\Rightarrow h} \text{ (no change)}. \end{cases} \tag{16} $$

On the other hand, any job of τ_i^g cannot interfere with any job of τ_k^h, if $\tau_k^{g\Rightarrow h} \succ \tau_i^{g\Rightarrow h}$ holds. Therefore, the upper-bound of $\mathbf{I}(\tau_k^h \leftarrow \tau_i^{g\Rightarrow h}, \ell)$ by FP (when τ_i has a higher priority than τ_k) in Theorem 1 changes from $\mathbf{W}_i^{g\Rightarrow h}(\ell)$ to ${}^{\mathsf{h}}\mathbf{W}_i^{g\Rightarrow h}(\ell)$, where

$$ {}^{\mathsf{h}}\mathbf{W}_i^{g\Rightarrow h}(\ell) \triangleq \begin{cases} \mathbf{W}_i^{g\Rightarrow h}(\ell), & \text{if } \tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h} \text{ (no change)}, \\ W_i^h(\ell), & \text{if } \tau_k^{g\Rightarrow h} \succ \tau_i^{g\Rightarrow h} \text{ (reduced)}. \end{cases} \tag{17} $$

Note that all theories developed in this section can be applied to EDF by applying the same modification for $\mathbf{E}_i^{g\Rightarrow h}(\ell)$. Our description is confined to FP because the interference reduction described so far is trivially applicable to EDF. That is, we can define ${}^{\mathsf{g}}\mathbf{E}_i^{g\Rightarrow h}(\ell)$ and ${}^{\mathsf{h}}\mathbf{E}_i^{g\Rightarrow h}(\ell)$ by replacing $\mathbf{W}_i^{g\Rightarrow h}(\ell)$, $W_i^g(\ell)$ and $W_i^h(\ell)$ with $\mathbf{E}_i^{g\Rightarrow h}(\ell)$, $E_i^g(\ell)$ and $E_i^h(\ell)$, respectively, with the same reasoning.

We can now derive a new RTA framework for a sequential transition, as stated in the following theorem.

Theorem 2: Suppose that a task set τ undergoes a sequential transition $M^g \Rightarrow M^h$ with a given order under FP (likewise EDF). An upper-bound of the response time of $\tau_k \in \tau$ is $r_k^u = r_k^{u(x)}$ (u is either g or h) such that $r_k^{u(x+1)} = r_k^{u(x)}$ holds in Eq. (18) (likewise Eq. (19)), starting from $r_k^{u(0)} = e_k^u$:

$$ r_k^{u(x+1)} \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left({}^{\mathsf{u}}\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)}), r_k^{u(x)} - e_k^u + 1 \right) \right\rfloor, \tag{18} $$

$$ r_k^{u(x+1)} \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left({}^{\mathsf{u}}\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)}), {}^{\mathsf{u}}\mathbf{E}_i^{g\Rightarrow h}(d_k^u), r_k^{u(x)} - e_k^u + 1 \right) \right\rfloor. \tag{19} $$

Note that Eq. (18) holds only when τ_i has a higher priority than τ_k; otherwise, ${}^{\mathsf{u}}\mathbf{W}_i^{g\Rightarrow h}(r_k^{u(x)})$ should be replaced with 0.

Proof: The theorem holds by Lemma 2 and the derivation of ${}^{\mathsf{u}}\mathbf{W}_i^{g\Rightarrow h}(\ell)$ and ${}^{\mathsf{u}}\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$; recall u is either g or h. □

From the definition of $\mathbf{W}_i^{g\Rightarrow h}(\ell)$, the upper-bound of the amount of interference with a sequential transition (i.e., ${}^{\mathsf{g}}\mathbf{W}_i^{g\Rightarrow h}(\ell)$ or ${}^{\mathsf{h}}\mathbf{W}_i^{g\Rightarrow h}(\ell)$) is always less than or equal to that with a concurrent transition (i.e., $\mathbf{W}_i^{g\Rightarrow h}(\ell)$). Therefore, the following observation is useful in developing the transition order assignment framework discussed in the next subsection.

Observation 3: Suppose that τ makes a transition $M^g \Rightarrow M^h$. If τ_k^g and τ_k^h are deemed schedulable with a concurrent transition (proven by Theorem 1), then they are also deemed schedulable for any sequential transition order (proven by Theorem 2).

C. TASK-LEVEL TRANSITION ORDER ASSIGNMENT

As we stated in Subsection V-A, the transition order of tasks within each sequential transition can be predetermined offline by the system designer, and as shown in Eqs. (16) and (17), enforcing a task-level transition order will reduce (or at least stay) the interference on each task, yielding the possibility of finding additional schedulable task sets that are not deemed schedulable with a concurrent transition. Then, an important question arises: "How can we find an effective transition order of tasks to improve schedulability for a sequential transition?"

The most critical factor to address this question is to exploit a slack value s_k of a task τ_k on our RTA framework. That is, s_k is affected by the amount of the worst-case interference on τ_k from other tasks $\tau_i \in \tau \setminus \tau_k$ (e.g., $\mathbf{W}_i^{g\Rightarrow h}(\ell)$) as we discussed in Subsection IV-D, and the amount of such interference on τ_k is determined by the transition order of other tasks in a sequential transition as we also discussed in Subsection V-A. This indicates that even if τ_k is deemed schedulable in a given task-level transition order, it may not be unschedulable if the order is changed. For example, for three given tasks, τ_a and two higher priority tasks τ_b and τ_c in the presence of a transition $M^g \Rightarrow M^h$, τ_a suffers from different amount of interference (and different slack values of τ_a) depending on whether $\tau_a^{g\Rightarrow h} \succ \tau_b^{g\Rightarrow h} \succ \tau_c^{g\Rightarrow h}$ or $\tau_a^{g\Rightarrow h} \succ \tau_c^{g\Rightarrow h} \succ \tau_b^{g\Rightarrow h}$. This is because slack values of τ_b and τ_c are determined by such a different task-level transition order, which changes the values of ${}^{\mathsf{u}}\mathbf{W}_b^{g\Rightarrow h}(\ell)$ and ${}^{\mathsf{u}}\mathbf{W}_c^{g\Rightarrow h}(\ell)$ for τ_a, where u is either g or h.

Such phenomenon on slack values may enforce us to use an exhaustive search, which requires investigating $O(|\tau|!)$ transition orders, and thus, we need to develop an efficient means to find an effective task-level transition order to improve schedulability. To achieve this goal, we consider the DA framework proposed in our preliminary conference

paper [17], where slack values are not used and useful properties are derived, and we use the properties for our RTA framework.

Note that if an interval of length ℓ is fixed to d_k^u and slack values s_i^g and s_i^h are fixed to 0, our RTA framework becomes the existing DA framework for multi-mode tasks [17]. That is, the following equation is used instead of Eqs. (14) and (15) in Theorem 1.

$$r_k^u \leftarrow e_k^u + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \tau \setminus \{\tau_k\}} \min\left({}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u), d_k^u - e_k^u + 1 \right) \right\rfloor, \tag{20}$$

where ${}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u)$ is ${}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ with $s_i^g = s_i^h = 0$ for FP, and ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$ with $s_i^g = s_i^h = 0$ or EDF, where u is either g or h.

Following the same reasoning, our RTA framework in Theorem 2 is equivalent to the DA framework for a sequential transition, i.e., Eq. (20) with ${}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u) = {}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ or ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$ (where u is either g or h for τ_i^g or τ_i^h) is used instead of Eqs. (18) and (19) in Theorem 2.

Note that the DA framework is only exploited for assigning a transition order of tasks, and we apply our RTA framework using Theorem 2 to judge schedulability after a transition order of tasks is determined using useful properties under the DA framework. We then investigate how the transition order of a given task affects the interference of other tasks on the task itself under the DA framework, as stated in the following observation.

Observation 4: Suppose that τ makes a sequential transition $M^g \Rightarrow M^h$ under the DA framework. If τ_k's transition order is placed first (likewise last), the right-hand side of Eq. (20) with $\mathbf{X}_i^{g \Rightarrow h}(d_k^u) = {}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ or ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$ for τ_k^g is minimized (likewise maximized) while that for τ_k^h is maximized (likewise minimized).

As shown in Eq. (16), if $\tau_k^{g \Rightarrow h} \prec \tau_i^{g \Rightarrow h}$ holds, the upper-bound on $\mathbf{I}(\tau_k^g \leftarrow \tau_i^{g \Rightarrow h})$ is reduced from ${}^{\mathrm{g}}\mathbf{W}_i^{g \Rightarrow h}(d_k^g)$ to $W_i^g(d_k^g)$. Therefore, the right-hand side of Eq. (20) with ${}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u) = {}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ or ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$ for τ_k^g is minimized, if the order of τ_k's transition is the earliest. Likewise, the case for τ_k^g and both cases for τ_k^h also hold.

Using the above observation, we now derive some properties of our task-level transition order assignment under the DA framework. Let us focus only on the schedulability of τ_k (i.e., both τ_k^g and τ_k^h), rather than on other tasks. If τ_k^h is schedulable with a concurrent transition under the DA framework (proven by Theorem 1 with Eq. (20) in which ${}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u)$ is ${}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ or ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$), placing τ_k's transition order in the earliest position not only maximizes the possibility of the schedulability of τ_k^g as shown in Observation 4, but also guarantees the schedulability of τ_k^h by Observation 3 (regardless of the transition order). However, such a favorable assignment for τ_k's schedulability may increase the interference of τ_k on other tasks. To address this, we introduce two notations.

First, we use $\tau_k^g \overset{I(\tau')}{>} \tau_k^h$ (*likewise*, $\tau_k^g \overset{I(\tau')}{<} \tau_k^h$), if $\min({}^{\mathrm{u}}\mathbf{W}_k^{g \Rightarrow h}(d_i^u), d_i^u - e_i^u + 1) = \min(W_k^g(d_i^u), d_i^u - e_i^u + 1)$ (*likewise*, $\min({}^{\mathrm{u}}\mathbf{W}_k^{g \Rightarrow h}(d_i^u), d_i^u - e_i^u + 1) = \min(W_k^h(d_i^u), d_i^u - e_i^u + 1)$) holds for all $\tau_i \in \tau' \setminus \{\tau_k\}$ and $u \in \{g, h\}$ under the DA framework. Its physical meaning is that the interference of $\tau_k^{g \Rightarrow h}$ on other tasks is dominated by that of τ_k^g. Thus, $\min({}^{\mathrm{u}}\mathbf{W}_k^{g \Rightarrow h}(d_i^u), d_i^u - e_i^u + 1)$ in Eq. (20) is fixed as $\min(W_k^g(d_i^u), d_i^u - e_i^u + 1)$ regardless of the transition order.

Second, let τ^* denote a set of tasks in τ, which are schedulable with a concurrent transition under the DA framework; then, any task $\tau_k \in \tau^*$ with M^g and that with M^h are schedulable with a sequential transition for any task-level transition order according to Observation 3.

Let us focus on a task τ_k which satisfies that (i) τ_k^h is schedulable with a concurrent transition under the DA framework, and (ii) $\tau_k^g \overset{I(\tau \setminus \tau^*)}{>} \tau_k^h$ holds. Then, we determine the order of τ_k by considering two aspects: (a) τ_k is schedulable or not; and (b) τ_k makes other tasks schedulable or not. For (a), we should place τ_k's transition in the earliest position, because this placement maximizes the chance of the schedulability of τ_k^g by Observation 4 and τ_k^h is schedulable with any task-level transition order by Observation 3. For (b), placing τ_k's transition first yields a lower $\min({}^{\mathrm{h}}\mathbf{W}_k^{g \Rightarrow h}(d_i^h), d_i^h - e_i^h + 1)$ $(= \min(W_k^h(d_i^h), d_i^h - e_i^h + 1))$ for a given τ_i^h while $\min({}^{\mathrm{g}}\mathbf{W}_k^{g \Rightarrow h}(d_i^g), d_i^g - e_i^g + 1)$ for a given τ_i^g is independent of the task-level transition order. Therefore, in terms of (i) and (ii), τ_k's transition should be performed earliest. Note that we do not consider tasks in τ^* for (b) because they are schedulable for any sequential order based on Observation 3.

With this reasoning, we develop a task-level transition-order assignment framework that identifies three groups in Alg. 2. In Steps 2 and 3, the algorithm identifies two groups of tasks, whose transition orders should be placed the earliest ($\tau(1)$) and latest ($\tau(3)$) as shown in Steps 5 and 7, respectively; the remaining tasks belong to the second group ($\tau(2)$) as shown in Step 6.

The remaining step is then to determine a transition order of tasks within individual groups $\tau(1)$, $\tau(2)$, and $\tau(3)$ in Steps 5–7 in Alg. 2. The following lemma finds an optimal transition order for tasks in $\tau(1)$ and $\tau(3)$.

Lemma 8: Suppose that τ makes a sequential transition $M^g \Rightarrow M^h$ with a given transition order compliant with Alg. 2. The relative transition order of tasks within $\tau(1)$ (likewise, $\tau(3)$) in Alg. 2 does not change the schedulability of any task in τ under the DA framework.

Proof: Before proving this lemma, we introduce a property to be used, as stated in the following observation.

Observation 5: Suppose that τ makes a sequential transition $M^g \Rightarrow M^h$ with a given order. Then, the right-hand side of Eq. (20) with ${}^{\mathrm{u}}\mathbf{X}_i^{g \Rightarrow h}(d_k^u) = {}^{\mathrm{u}}\mathbf{W}_i^{g \Rightarrow h}(d_k^u)$ or ${}^{\mathrm{u}}\mathbf{E}_i^{g \Rightarrow h}(d_k^u)$ for a given task τ_k^u (u is either g or h) is not affected by the relative transition order of tasks in $\tau' \triangleq \{\tau_i | \tau_k^{g \Rightarrow h} \prec \tau_i^{g \Rightarrow h}\}$ and $\tau'' \triangleq \{\tau_i | \tau_k^{g \Rightarrow h} \succ \tau_i^{g \Rightarrow h}\}$, but is affected by the elements of τ' and τ''.

Algorithm 2 Task-Level Transition Order Assignment Framework

1: **for** $\tau_k \in \tau$ **do**
2: Check whether $\tau_k^g \overset{I(\tau\setminus\tau^*)}{>} \tau_k^h$ and τ_k^h is schedulable with a concurrent transition (i.e., Theorem 1 with Eq. (20) with ${}^{\cup}\mathbf{X}_i^{g\Rightarrow h}(d_k^u) = {}^{\cup}\mathbf{W}_i^{g\Rightarrow h}(d_k^u)$ or ${}^{\cup}\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$). If so, add τ_i to $\tau(1)$.
3: Check whether $\tau_k^g \overset{I(\tau\setminus\tau^*)}{<} \tau_k^h$ and τ_k^g is schedulable with a concurrent transition (i.e., Theorem 1 with Eq. (20) with ${}^{\cup}\mathbf{X}_i^{g\Rightarrow h}(d_k^u) = {}^{\cup}\mathbf{W}_i^{g\Rightarrow h}(d_k^u)$ or ${}^{\cup}\mathbf{E}_i^{g\Rightarrow h}(d_k^u)$). If so, add τ_i to $\tau(3)$.
4: **end for**
5: Determine the first $|\tau(1)|$ transition orders of tasks in $\tau(1)$.
6: Determine the next $|\tau(2)|$ transition orders of tasks in $\tau(2) \triangleq \tau \setminus (\tau(1) \cup \tau(3))$.
7: Determine the last $|\tau(3)|$ transition orders of tasks in $\tau(3)$.

The observation holds because ${}^{\cup}\mathbf{W}_i^{g\Rightarrow h}(d_k^u)$ depends only on whether $\tau_k^{g\Rightarrow h} \prec \tau_i^{g\Rightarrow h}$ or $\tau_k^{g\Rightarrow h} \succ \tau_i^{g\Rightarrow h}$ holds.

Suppose that τ is schedulable in the presence of a transition from τ^g to τ^h by the DA framework, with a given task-level transition order compliant with Alg. 2. Now, we investigate how a change of the transition order of tasks in $\tau(1)$ affects the schedulability of two groups of tasks: (i) tasks in $\tau(1)$ and (ii) tasks in $\tau(2) \cup \tau(3)$.

For (i), the transition order of a task τ_k in $\tau(1)$ does not affect the schedulability of τ_i^g for all $\tau_i \in \tau(1) \setminus \{\tau_k\}$, because $\min({}^{\cup}\mathbf{W}_k^{g\Rightarrow h}(d_i^g), d_i^g - e_i^g + 1) = \min(W_k^g(d_i^g), d_i^g - e_i^g + 1)$ holds regardless of the relative transition order of τ_k and τ_i (by the definition of $\tau_k^g \overset{I(\tau)}{>} \tau_k^h$). As to τ_i^h, it is also schedulable with any task-level transition order by Observation 3, meaning that the transition order of a task τ_k in $\tau(1)$ does not affect the schedulability of τ_i^h for all $\tau_i \in \tau(1) \setminus \{\tau_k\}$.

The schedulability of tasks in $\tau(2) \cup \tau(3)$ is also not affected by the relative order of tasks in $\tau(1)$ according to Observation 5.

In summary, the relative transition order of tasks in $\tau(1)$ does not change the schedulability of every task in τ. This holds for $\tau(3)$ with the same reasoning. Therefore, the lemma holds. □

By applying Alg. 2 with any arbitrary order for tasks in $\tau(1)$ and $\tau(3)$, we can derive an optimal task-level transition order under the DA framework except for the relative transition order of tasks in $\tau(2)$. To determine a transition order for tasks in $\tau(2)$, we may apply an exhaustive search (if $|\tau(2)|$ is small) or a heuristic transition order assignment algorithms.

So far, we have discussed how to effectively assign a task-level transition order under the DA framework to simplify the interference relationships among tasks. Then we apply our original RTA framework in Theorem 2 based on the task-level transition order determined by Alg. 2. This heuristic approach is quite effective owing to the effectiveness of Alg. 2 under the DA framework, whose performance will be demonstrated via simulation results in the next section.

VI. EVALUATION

In this section, we demonstrate the effectiveness of the proposed RTA frameworks in which a concurrent transition and a sequential transition are associated with CSR and ISR.

We randomly generate task sets for a multiprocessor platform using a well-known task set generation method, referred to as UUnifast-discard [28]. Typically, three input parameters are considered for UUnifast-discard: (i) the number of processors m (2, 4, 8, and 16), (ii) the number of tasks n ($m+1$, 1.5 m, 2.0 m, 2.5 m, 3.0 m, 3.5 m, 4.0 m, 4.5 m, and 5.0 m), and (iii) the task set utilization $U = \sum_{\tau_i \in \tau} e_i/p_i$ (0.1 m, 0.2 m, 0.3 m, 0.4 m, 0.5 m, 0.6 m, 0.7 m, and 0.8 m). As we consider multi-mode systems, we additionally consider the number of modes μ (from 2 to 10 for positive integer values). We generate 1000 task sets for every four-tuple (m, n, U, μ). For a given task utilization for τ_i (u_i), p_i is uniformly selected in [1, 1000]; C_i is computed based on the given utilization and p_i (i.e., $e_i = p_i \cdot u_i$); and d_i is set to p_i. The randomly generated task parameters are known to sufficiently cover the general cases of real systems such as antenna controller software in an unmanned aerial vehicle as shown in the previous studies [35], [36].

As a baseline schedulability analysis, we take the DA framework for multi-mode tasks proposed in our preliminary conference paper [17]. For all generated task sets with the four-tuple (i.e., $1000 \cdot 4 \cdot 9 \cdot 8 \cdot 9$ task sets), we compare the number of schedulable task sets with $\mu - 1$ different transitions $M^g \Rightarrow M^h$ ($1 \le g \le h = g+1 \le \mu$) using their corresponding schedulability analyses[2]:

- A concurrent transition tested by DA, RTA with CSR and RTA with ISR (denoted by $\mathsf{DA}_{\mathsf{con}}$, $\mathsf{RTA(C)}_{\mathsf{con}}$, and $\mathsf{RTA(I)}_{\mathsf{con}}$, respectively);
- A sequential transition tested by DA, RTA with CSR and RTA with ISR – the entire order is determined randomly (denoted by $\mathsf{DA}_{\mathsf{seq}}$, $\mathsf{RTA(C)}_{\mathsf{seq}}$, and $\mathsf{RTA(I)}_{\mathsf{seq}}$, respectively); and
- A sequential transition tested by DA, RTA with CSR and RTA with ISR – the entire order is grouped by Alg. 2, and then the relative order of tasks in each group is determined randomly (denoted by $\mathsf{DA}^*_{\mathsf{seq}}$, $\mathsf{RTA(C)}^*_{\mathsf{seq}}$, and $\mathsf{RTA(I)}^*_{\mathsf{seq}}$, respectively).

Here, $\mathsf{DA}_{\mathsf{con}}$ is conducted by Theorem 1 in [17], and $\mathsf{RTA(C)}_{\mathsf{con}}$ and $\mathsf{RTA(I)}_{\mathsf{con}}$ are conducted by Theorem 1 in Subsection IV-C. Further, $\mathsf{DA}_{\mathsf{seq}}$ and $\mathsf{DA}^*_{\mathsf{seq}}$ follow the analysis presented in Theorem 2 in [17], and the others are conducted by Theorem 2 in Subsection V-B. We present the simulation results for implicit-deadline tasks scheduled by FP only, as those for implicit-deadline tasks scheduled by EDF (or those for constrained-tasks scheduled by FP or EDF) exhibit a similar trend.

[2] A task set is deemed schedulable if it passes a corresponding schedulability test for every transition $M^g \Rightarrow M^h$ with a given μ.

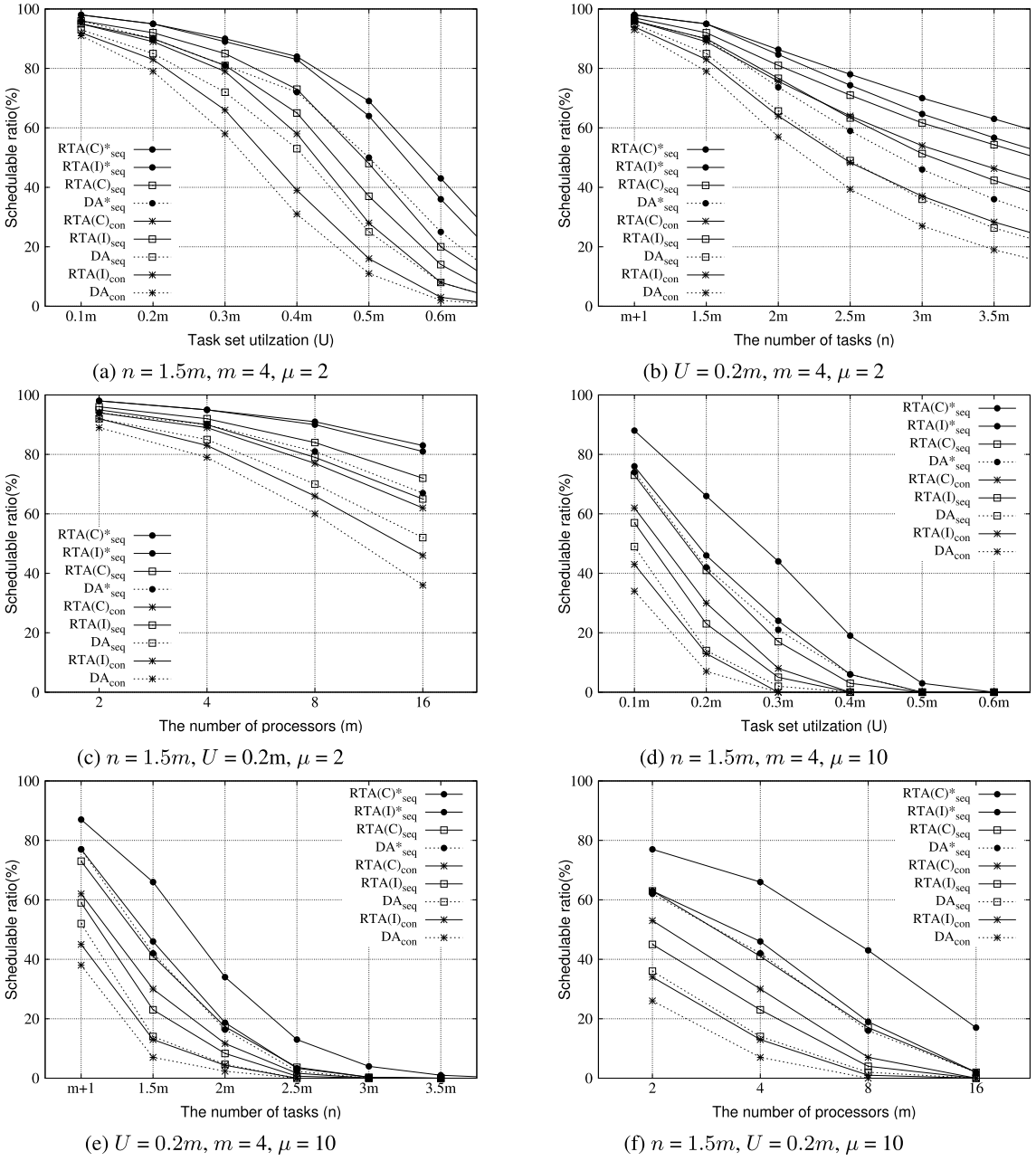


FIGURE 5. Schedulability tests for implicit deadline task sets.

TABLE 2. Schedulable ratio for $n = 1.5m$, $U = 0.2m$ and $\mu = 10$.

m	DA_{con}	$RTA(I)_{con}$	DA_{seq}	$RTA(I)_{seq}$	$RTA(C)_{con}$	DA^*_{seq}	$RTA(C)_{seq}$	$RTA(I)^*_{seq}$	$RTA(C)^*_{seq}$
2	26.1%	34.1%	36.4%	45.2%	53.8%	62.5%	63.3%	63.7%	77.8%
4	7.8%	13.6%	14.6%	23.1%	30.0%	42.9%	41.1%	46.0%	66.6%
8	0.3%	1.4%	2.4%	4.7%	7.4%	16.1%	17.1%	19.5%	43.3%
16	0.0%	0.2%	0.1%	0.2%	0.3%	2.2%	2.0%	2.7%	17.1%

Fig. 5 plots the percentage of generated task sets that are deemed schedulable by each schedulability analysis (referred to as the schedulable ratio) according to varying values of input parameters (i.e., m, n, U, and μ). Fig. 5(a) shows it for varying values of U over fixed values of $n = 1.5m$, $m = 4$, and $\mu = 2$, and Fig. 5(c) does it for varying values of n over fixed $U = 0.2m$, $m = 4$, and $\mu = 2$, respectively. Fig. 5(e) plots it for varying number of processors m over fixed values

of $n = 1.5m$, $U = 0.2$, and $\mu = 2$. Fig. 5(b), (d), and (f) show it of (a), (c), and (e) for $\mu = 10$ instead of $\mu = 2$. From the figures, we note the following.

O1. Each schedulability analysis of the **RTA(C)** series (i.e., $\textsf{RTA(C)}_{\textsf{con}}$, $\textsf{RTA(C)}_{\textsf{seq}}$, and $\textsf{RTA(C)}^{*}_{\textsf{seq}}$) outperforms each corresponding schedulability analysis of the **RTA(I)** series, which also holds between **RTA(I)** series and **DA** series.
O2. As the value of μ increases, the performance gap between the **RTA(C)** series and the **RTA(I)** series becomes pronounced.
O3. For an increasing value of U, schedulability analyses utilizing task order assignment in Algo. 2 (i.e., $-^{*}_{\textsf{seq}}$) shows less performance degradation than the others.
O4. As the number of tasks n increases, the schedulable ratio of **DA** series decreases at a rate higher than the other series.

O1 holds owing to different analytic capabilities between the existing DA framework, and the two proposed RTA frameworks with CSR and ISR. That is, although three analyses commonly exploit interference calculations presented in Subsection IV-B, the DA framework does not utilize a slack value of each task while the RTA framework with CSR uses it for both τ_i^g and τ_i^h (and the RTA framework with ISR only uses it for τ_i^h). Further, the DA framework considers interference on τ_k within an interval of length d_k while the others gradually increase such interval length from e_k up to d_k during judging schedulability. Because of such different capabilities, each schedulability analysis of the **RTA(C)** series dominates the corresponding **RTA(I)** series in terms of schedulable ratio, which also holds between **RTA(I)** and **DA**. For example, as shown in Fig 5(a), the schedulable ratio of $\textsf{RTA(C)}^{*}_{\textsf{seq}}$, $\textsf{RTA(I)}^{*}_{\textsf{seq}}$, and $\textsf{DA}^{*}_{\textsf{seq}}$ for $0.5m$ are 69.2%, 64.4%, and 50.3%, respectively.

O2 demonstrates the superiority of CSR compared to ISR when it comes to multiple modes, meaning a larger number of schedulable task sets are found by the RTA framework with CSR than the RTA framework with ISR. This implies that it is crucially important to capture how slack values of tasks in two consecutive modes are correlated and to incorporate them into the analysis for improving schedulability for multi-mode tasks, which is successfully fulfilled by CSR but not ISR. For example, $\textsf{RTA(C)}^{*}_{\textsf{seq}}$ and $\textsf{RTA(I)}^{*}_{\textsf{seq}}$ for $n = 1.5m$, $U = 0.2m$, $m = 4$, and $\mu = 2$ show the nearly same performance (in Fig. 5(c)) while there is about 20% performance gap between $\textsf{RTA(C)}^{*}_{\textsf{seq}}$ and $\textsf{RTA(I)}^{*}_{\textsf{seq}}$ for $\mu = 10$ (in Fig. 5(f)) (66.6% vs. 46.0%).

O3 stems from the advantage of interference reduction by utilizing task ordering assignment. High-utilization tasks may have fewer slacks while inducing a large amount of interference to lower priority tasks. Thus, reducing interference from ${}^{9}\mathbf{W}_i^{g\Rightarrow h}(\ell)$ to $W_i^g(\ell)$ is a more effective approach for achieving better schedulability rather than exploiting small amount of slacks. For example in Fig. 5(a), $\textsf{DA}^{*}_{\textsf{seq}}$ performs better than $\textsf{RTA(C)}_{\textsf{seq}}$ for $U = 0.5m$ even though $\textsf{RTA(C)}_{\textsf{seq}}$ utilizes slack values of both τ^g and τ^h while $\textsf{DA}^{*}_{\textsf{seq}}$ does not (51.2% vs. 50.5%).

In contrast, O4 holds owing to the advantage of slack reclamation subject to low-utilization tasks. The increasing number of tasks in a task set with fixed system utilization results in decreasing average task utilization. Low task utilization possibly induces much slack for each task, and thus it can compensate pessimistic interference calculation of a large number of tasks in a task set. For example in Fig. 5(b), the performance gap between $\textsf{RTA(C)}_{\textsf{seq}}$ and $\textsf{DA}^{*}_{\textsf{seq}}$ becomes larger even though $\textsf{DA}^{*}_{\textsf{seq}}$ exploits task order assignment in Algo. 2 while $\textsf{RTA(C)}_{\textsf{seq}}$ does not; the schedulable ratio of $\textsf{RTA(C)}_{\textsf{seq}}$ is about 37% while that of $\textsf{DA}^{*}_{\textsf{seq}}$ is 16% for $n = 5m$.

Table 2 shows the schedulable ratio of considered schedulability tests for $n = 1.5m$, $U = 0.2m$, $\mu = 10$ (corresponding to Fig. 5(f)). As shown the table, $\textsf{RTA(C)}^{*}_{\textsf{seq}}$ (our best performing test) improves $\textsf{DA}^{*}_{\textsf{seq}}$ (the existing one) by 17.1/2.2 = 777.3%.

VII. DISCUSSION

The underlying assumption under Liu and Layland's task model is that the worst-case execution time C_i of each task encompasses the worst-case timing overheads stemming from preemptions/migrations. However, considering such timing overheads in every case is pessimistic since such a case happens rarely in practice. This pessimism can be mitigated by upper-bounding the number of preemptions that a job can experience. Since we consider FP and EDF scheduling classified as a task- or job-level fixed priority assignment scheme, the job priority never changes once it is assigned. Instead, the priority order of jobs of currently running jobs can be changed due to a newly-released job. Therefore, a preemption occurs only when a job is released *and* the job's priority is higher than the currently-executing jobs. Also, a migration occurs only when the preempted job resumes its execution on a core different from the one on which the job was executed before the preemption. Thus, the number of preemptions that a job can experience cannot be larger than the number of higher-priority jobs released during the job's execution. Also, the number of migrations that the job can experience cannot be larger than the number of preemptions that the job undergoes. Therefore, we can upper-bound the number of migrations that a job can experience, by the number of higher-priority jobs released during the job's execution.

The Liu and Layland model also assumes that the worst-case execution time C_i of each task implicitly includes the worst-case blocking time resulted from mutual exclusion of critical sections on shared resources such as main memory, memory bus, and shared cache. The resource-locking protocol ensures that one job never enters its critical section at the instance when another concurrent job enters its own critical section, which has been extensively studied in a number of existing work for real-time systems [39], [40]. The resource-locking protocol to be developed for our proposed RTA framework for multi-mode tasks potentially improves

analytic capability as it reduces response time of τ_k by relieving a pessimistic assumption of the Liu and Layland's task model regarding worst-case execution time C_i of each task.

In addition to the theoretical upper-bound, we also discuss how many migrations can occur during the considered scheduling via well-known experimental results. As shown in the previous experimental results in [37], the average actual number of preemptions incurred by each implicit-deadline (i.e., $d_i = p_i$) task set scheduled by EDF during 100,000 time units is 1,068.3 for $m = 2$ and 2,319.8 for $m = 8$, indicating that possibility of the occurrence of a preemption at each time unit ranges from 1.0% to 2.3%, depending on the number of processors. Although our proposed RTA framework does not explicitly include any method to upper-bound the number of preemptions that occur during each job's execution, such experimental results imply that it is acceptable to ignore preemption cost when the migration cost is not very high.

In this paper, we considered the system in which tasks' parameters undergo multiple transitions, while the scheduling policy (i.e., job prioritization policy) does not change. When it comes to dynamic scheduling in which the scheduling policy changes according to online schedulability analysis, we may consider another system model proposed in [38] where task parameters and the scheduling policy change during operation. Also, the applicability of the current version of our proposed RTA framework is limited to the mode-transition protocol for predetermined (fixed) sequence of mode transitions (CSR case), or the analytical capability of that is sacrificed when the sequence of mode transitions is not known (ISR case). It would be worth relieving such the limitations to improve the applicability of our RTA framework while improving the schedulability.

VIII. RELATED WORK

Starting from [29], a number of studies proposed task models considering potential mode transitions for uniprocessor platforms, such as the generalized multiframe (GMF) model [30], the digraph real-time (DRT) model [6], the acyclic task model [31], and the variable rate-dependent behavior (VRB) task model (also known as the adaptive variable-rate (AVR) model) [1], [8], [32], [33]. In the GMF model, each task invokes static types of jobs sequentially, each with potentially different WCETs and relative deadlines. The DRT model is a more expressive one, in which the release of different types of jobs are represented by a directed graph; each vertex in the directed graph indicates a specific type of job labeled with an ordered pair of WCETs of the corresponding job and relative deadline, and each edge represents the order in which jobs are generated by the corresponding task. A job in the acyclic task model has any arbitrary execution time, which assumes that the absolute deadline and the arrival time of the next job of a task are determined by the arrival time of the job plus the utilization multiplied by the execution time of the job. Tasks in the VRB model are related to angular velocity of a specific device (e.g., the crankshaft, gears, or wheels). Such a device's activation rate is proportional to the angular velocity thereof, which determines the execution mode assuming different corresponding task parameters.

For multiprocessor platforms, there have been few studies addressing multi-mode tasks for partitioned and global scheduling, respectively. For partitioned scheduling, Niz and Phan addressed the system-wide mode transition for multi-mode tasks in which each task's criticality may change during a mode transition while the other parameters (i.e., p_i, e_i and d_i) do not change [12]. They considered zero-slack rate-monotonic scheduling [34] and proposed a partitioned scheduling scheme maximizing the schedulable utilization while ensuring the absence of criticality violation. Huang and Chen addressed a mode transition (which is different from our protocol) for single-criticality tasks and proposed bin-packing task allocating algorithms guaranteeing utilization bound of tasks [13]. For global scheduling, three studies [14]–[16] developed system-wide mode-transition protocols aiming at minimizing the transition delay occurring in the presence of a mode transition without deadline miss under a target global scheduling. The first work [14] suggested the notion of a relative enablement deadline for each task and proved the correctness of the proposed transition protocol by showing that every transition delay is no larger than the enablement deadline of each task. The other two studies aimed at incorporating single-mode tasks (i.e., independent of mode transitions) into the transition protocol [15] and reducing transition delay by exploiting a rate-based global dynamic-priority scheduling algorithm [16]. Unlike such studies, our study in the preliminary conference paper [17] supports a transition without imposing additional transition delay or task drop. Our previous study extended the existing DA framework developed for single-mode tasks on multiprocessor platforms to judge schedulability of multi-mode tasks in the presence of a given single mode transition.

IX. CONCLUSION

In this paper, we focused on the problem of guaranteeing the timing requirements of task sets with system-wide mode transitions in real-time multiprocessor systems. We developed an offline schedulability analysis that does not require any online information, and generalizes the existing RTA framework designed for single-mode tasks and the DA schedulability analysis designed for multi-mode tasks. To improve the analysis, we enforced the transition order of tasks, and proposed a transition sequence assignment algorithm by deriving the useful properties of an effective transition order under a given restricted condition. Our proposed RTA framework is shown to improve performance up to 777.3% depending on the experiment setting under our evaluation environment. In the future, we would like to extend our study towards the mechanism providing less computational complexity or supporting other platforms/frameworks such as heterogeneous multiprocessor platforms and hierarchical scheduling frameworks.

ACKNOWLEDGMENT

An earlier (shorter) version of this paper was presented at the IEEE RTSS 2013 [17].

REFERENCES

[1] R. I. Davis, T. Feld, V. Pollex, and F. Slomka, "Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 51–62.

[2] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Syst.*, vol. 26, no. 2, pp. 161–197, Mar. 2004.

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

[4] B. Andersson, "Uniprocessor EDF scheduling with mode change," in *Principles of Distributed Systems*, vol. 5401. Berlin, Germany: Springer, 2008, pp. 572–577.

[5] Q. Guangming, "An earlier time for inserting and/or accelerating tasks," *Real-Time Syst.*, vol. 41, no. 3, pp. 181–194, Apr. 2009.

[6] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Proc. 17th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2011, pp. 71–80.

[7] M. Ahmed, N. Fisher, and D. Grosu, "A parallel algorithm for EDF-schedulability analysis of multi-modal real-time systems," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2012, pp. 154–163.

[8] J. Kim, K. Lakshmanan, and R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *Proc. IEEE/ACM 3rd Int. Conf. Cyber-Phys. Syst.*, Apr. 2012, pp. 55–64.

[9] T. Kloda, B. Ausbourg, and L. Santinelli, "Towards EDF schedulability analysis of an extended timing definition language," *ACM SIGBED Rev.*, vol. 11, no. 3, pp. 44–49, 2014.

[10] T. Kloda, B. Ausbourg, and L. Santinelli, "EDF schedulability analysis of an extended timing definition language," in *Proc. IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, 2014, pp. 30–40.

[11] W.-H. Huang and J.-J. Chen, "Techniques for schedulability analysis in mode change systems under fixed-priority scheduling," in *Proc. IEEE 21st Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2015, pp. 176–186.

[12] D. de Niz and L. T. X. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 111–122.

[13] W.-H. Huang and J.-J. Chen, "Utilization bounds on allocating rate-monotonic scheduled multi-mode tasks on multiprocessor systems," in *Proc. 53rd Annu. Design Autom. Conf. (DAC)*, 2016, pp. 1–6.

[14] V. Nelis, J. Goossens, and B. Andersson, "Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms," in *Proc. 21st Euromicro Conf. Real-Time Syst.*, Jul. 2009, pp. 151–160.

[15] V. Nelis, B. Andersson, J. Marinho, and S. M. Petters, "Global-EDF scheduling of multimode real-time systems considering mode independent tasks," in *Proc. 23rd Euromicro Conf. Real-Time Syst.*, Jul. 2011, pp. 205–214.

[16] P. Rattanatamrong and J. A. B. Fortes, "Mode transition for online scheduling of adaptive real-time systems on multiprocessors," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2011, pp. 25–32.

[17] J. Lee and K. G. Shin, "Schedulability analysis for a mode transition in real-time multi-core systems," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 11–20.

[18] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, Apr. 2009.

[19] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 149–160.

[20] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[21] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority preemptively scheduled systems," in *Proc. Real-Time Syst. Symp.*, 1992, pp. 100–109.

[22] T. Chen and L. T. X. Phan, "SafeMC: A system for the design and evaluation of mode-change protocols," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 105–116.

[23] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 387–397.

[24] M. Bertogna and S. Baruah, "Tests for global EDF schedulability analysis," *J. Syst. Archit.*, vol. 57, no. 5, pp. 487–497, May 2011.

[25] J. Lee and K. G. Shin, "Controlling preemption for better schedulability in multi-core systems," in *Proc. IEEE 33rd Real-Time Syst. Symp.*, Dec. 2012, pp. 29–38.

[26] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *Proc. 17th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2005, pp. 209–218.

[27] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-timeäscheduling: The deadline-monotonic approach," in *Proc. IEEE Workshop Real-Time Oper. Syst. Softw.*, 1991, pp. 127–132.

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

[29] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Syst.*, vol. 1, no. 3, pp. 243–264, Dec. 1989.

[30] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Syst.*, vol. 17, no. 1, pp. 5–22, 1999.

[31] T. F. Abdelzaher, V. Sharma, and C. Lu, "A utilization bound for aperiodic tasks and priority driven scheduling," *IEEE Trans. Comput.*, vol. 53, no. 3, pp. 334–350, Mar. 2004.

[32] A. Biondi, A. Melani, M. Marinoni, M. D. Natale, and G. Buttazzo, "Exact interference of adaptive variable-rate tasks under fixed-priority scheduling," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 165–174.

[33] G. C. Buttazzo, E. Bini, and D. Buttle, "Rate-adaptive tasks: Model, analysis, and design issues," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, pp. 1–6.

[34] D. D. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 291–300.

[35] H. Baek and J. Lee, "Improved schedulability analysis of the contention-free policy for real-time systems," *J. Syst. Softw.*, vol. 154, pp. 112–124, Aug. 2019.

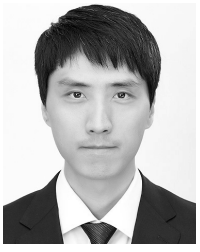
[36] H. Baek and J. Lee, "Task-level re-execution framework for improving fault tolerance on symmetry multiprocessors," *Symmetry*, vol. 11, no. 5, p. 651, 2019.

[37] J. Lee, A. Easwaran, and I. Shin, "Contention-free executions for real-time multiprocessor scheduling," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 2s, pp. 1–25, Jan. 2014.

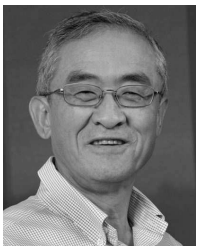
[38] H. S. Chwa, K. G. Shin, H. Baek, and J. Lee, "Physical-state-aware dynamic slack management for mixed-criticality systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 129–139.

[39] F. Rabee, A. Onaizah, and A. F. Mahdi, "Shift-exchange synchronization protocol (SESP) in hard real time system," in *Proc. IEEE 2nd Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Mar. 2017, pp. 304–310.

[40] G. von der Brüggen, J.-J. Chen, W.-H. Huang, and M. Yang, "Release enforcement in resource-oriented partitioned scheduling for multiprocessor systems," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, Oct. 2017, pp. 287–296.



HYEONGBOO BAEK received the B.S. degree in computer science and engineering from Konkuk University, South Korea, in 2010, and the M.S. and Ph.D. degrees in computer science from KAIST, South Korea, in 2012 and 2016, respectively. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Incheon National University (INU), South Korea. His research interests include cyber physical systems, real time embedded systems, and system security. Dr. Baek received the Best Paper Award from the 33rd IEEE Real Time Systems Symposium (RTSS), in 2012.



KANG G. SHIN (Life Fellow, IEEE) is currently the Kevin & Nancy O'Connor Professor of computer science with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. His current research focuses on QoS-sensitive computing and networking as well as on embedded real-time and cyber-physical systems. He has supervised the completion of 78 Ph.D.'s. He has authored or coauthored more than 830 technical articles, a textbook and more than 30 patents or invention disclosures. Dr. Shin received numerous best paper awards, including the Best Paper Awards from the 2011 ACM International Conference on Mobile Computing and Networking (MobiCom'11), the 2011 IEEE International Conference on Autonomic Computing, the 2010 and 2000 USENIX Annual Technical Conferences, the 2003 IEEE Communications Society William R. Bennett Prize Paper Award, and the 1987 Outstanding IEEE Transactions of Automatic Control Paper Award. He has also received several institutional awards, including the Research Excellence Award, in 1989, Outstanding Achievement Award, in 1999, the Distinguished Faculty Achievement Award, in 2001, and the Stephen Attwood Award, in 2004, from the University of Michigan (the highest honor bestowed to Michigan Engineering Faculty), the Distinguished Alumni Award of the College of Engineering, Seoul National University, in 2002, the 2003 IEEE RTC Technical Achievement Award, and the 2006 Ho-Am Prize in Engineering (the highest honor bestowed to Korean-Origin Engineers). He was a Co-Founder of a couple of startups and also licensed some of his technologies to industry.

JINKYU LEE (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2004, 2006, and 2011, respectively. He has been a Research Fellow/Visiting Scholar with the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. He joined the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), South Korea, in 2014, where he is currently an Associate Professor. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real time embedded systems and cyber physical systems. Dr. Lee received the Best Student Paper Award from the 17th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), in 2011, and the Best Paper Award from the 33rd IEEE Real Time Systems Symposium (RTSS), in 2012.