Beyond Implicit-Deadline Optimality: A Multiprocessor Scheduling Framework for Constrained-Deadline Tasks

Hyeongboo Baek*, Hoon Sung Chwa† and Jinkyu Lee*
*Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea
†Department of Electrical Engineering and Computer Science, The University of Michigan, U.S.A.
hbbaek@skku.edu, hchwa@umich.edu, jinkyu.lee@skku.edu

***Abstract*—In the real-time systems community, many studies have addressed how to efficiently utilize a multiprocessor platform so as to accommodate as many periodic/sporadic real-time tasks as possible without violating any timing constraints. The scheduling theory has sufficiently matured for a set of implicit-deadline tasks (the relative deadline equal to the period), yielding a class of optimal scheduling algorithms. However, the same does not hold for a set of constrained-deadline tasks (the relative deadline no larger than the period) in that those task sets have been fully covered by neither existing implicit-deadline optimal scheduling algorithms nor heuristic scheduling algorithms.**

In this paper, we propose a scheduling framework that not only takes advantage of both existing implicit-deadline optimal and heuristic algorithms, but also surpasses both in finding schedulable constrained-deadline task sets. The proposed framework logically divides a given task set into the higher- and lower-priority classes and schedules the classes using an implicit-deadline optimal algorithm and a heuristic algorithm, respectively. Then, while the proposed framework guarantees schedulability of tasks in the higher-priority class by the target implicit-deadline optimal algorithm, we need to address the following technical issues for enabling tasks in the lower-priority class to efficiently reclaim remaining processor capacity while guaranteeing their schedulability: (i) division of a given task set into the two classes, (ii) selection/development of scheduling algorithms for the two classes, and (iii) development of a schedulability test for the framework with given (i) and (ii). We present a general case showing how to address (i)–(iii), and then a specific case addressing how to further improve schedulability by utilizing characteristics of the specific case. Our simulation results demonstrate that the proposed framework outperforms all existing scheduling algorithms in covering schedulable task sets; in particular, if we focus on task sets with the system density larger than the number of processors, the framework finds up to 446.3% additional schedulable task sets, compared to task sets covered by at least one of existing scheduling algorithms.

I. Introduction

The real-time systems community has sought answers of the following fundamental questions regarding a set of periodic/sporadic real-time tasks $\tau_i \in \tau$, each of which is specified by the minimum inter-arrival time or period (T_i), the worst-case execution time (C_i) and the relative deadline (D_i). Can we develop a scheduling algorithm that does not yield any single job deadline miss for all possible legal job arrival patterns invoked by a task set τ? If so, how can we guarantee there is no job deadline miss under the scheduling algorithm?

On a uniprocessor platform, the two questions have been fully addressed. EDF (Earliest Deadline First) [1] has been proven optimal for not only a task set consisting of implicit-deadline tasks ($D_i = T_i$) but also that of constrained-deadline tasks ($D_i \leq T_i$). Also, EDF has exact (necessary and sufficient) schedulability tests for both implicit- and constrained-deadline task sets [1, 2], e.g., $\sum_{\tau_i \in \tau} C_i/T_i \leq 1$ for a set of implicit-deadline tasks τ.

When it comes to a multiprocessor platform consisting of m (≥ 2) identical processors, a class of optimal scheduling algorithms have achieved 100% utilization for a set of implicit-deadline tasks, meaning a task set τ is schedulable if and only if $\sum_{\tau_i \in \tau} C_i/T_i \leq m$ holds. Starting from P-Fair [3], various optimal scheduling algorithms have been developed in order to reduce the number of preemptions/migrations and/or accommodate new environments (e.g., supporting sporadic releases) such as ER-Fair, LLREF, EKG, DP-Wrap, RUN, U-EDF, QPS [4–10]. While most of the implicit-deadline optimal scheduling algorithms can cover constrained-deadline task sets whose system density is no larger than the number of processors m (i.e., $\sum_{\tau_i \in \tau} C_i/D_i \leq m$), *few of them* guarantee the schedulability of the other constrained-deadline task sets (i.e., $\sum_{\tau_i \in \tau} C_i/D_i > m$).

On the other hand, there exist heuristic scheduling algorithms that significantly improve existing simple scheduling algorithms EDF and FP (Fixed Priority) [1], such as EDZL, FPZL, LLF, EDF-CF, SPDF, EQDF, EQDZL [11–17]. Due to the difficulty of developing tight schedulability tests, all existing schedulability tests for those heuristic algorithms (including even EDF and FP) are only sufficient. Therefore, empirical results demonstrate that those scheduling algorithms with their best schedulability tests give the schedulability guarantee to only some of task sets belonging to $\sum_{\tau_i \in \tau} C_i/D_i \leq m$ and *a few of task sets* belonging to $\sum_{\tau_i \in \tau} C_i/D_i > m$, as shown in Fig. 1.

In this paper, we aim at not only taking advantage of both existing implicit-deadline optimal scheduling algorithms (covering *all task sets* satisfying $\sum_{\tau_i \in \tau} C_i/D_i \leq m$) and heuristic scheduling algorithms (covering *a few task sets* satisfying $\sum_{\tau_i \in \tau} C_i/D_i > m$), but also outperforming both and finding additional task sets that have *not been proven schedulable by any existing scheduling algorithm*. To this end, we propose a two-level scheduling framework, which logically divides a given task set τ into the higher- and lower-priority classes τ^{HI} and τ^{LO}, and schedules them by an implicit-deadline optimal algorithm and a heuristic algorithm, respectively. Then, while the proposed framework guarantees schedulability of tasks in τ^{HI} as long as $\sum_{\tau_i \in \tau^{\mathsf{HI}}} C_i/D_i \leq m$ holds, we need to address the following issues for enabling tasks in τ^{LO} to efficiently reclaim remaining processor capacity while guaranteeing their schedulability.

I1. How to divide a task set τ into τ^{HI} and τ^{LO}?

1052-8725/17/31.00 ©2017 IEEE
DOI 10.1109/RTSS.2017.00038

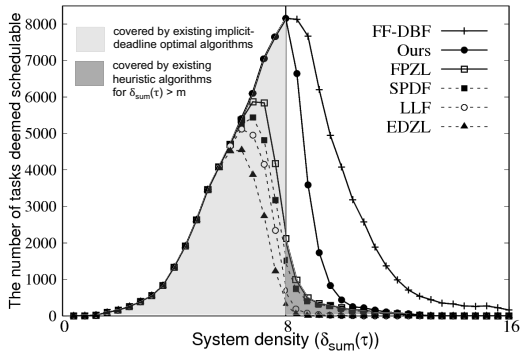


Fig. 1. The number of task sets deemed schedulable by schedulability tests of the proposed scheduling framework (Ours) and existing well-known heuristic scheduling algorithms (FPZL, SPDF, LLF and EDZL) according to the system density $\delta_{sum}(\tau) = \sum_{\tau_i \in \tau} C_i/D_i$ for m = 8; here, FF-DBF means the total number of generated task sets, and detailed simulation settings will be explained in Section VI.

I2. How to determine/develop scheduling algorithms for τ^{HI} and τ^{LO}?

I3. How to guarantee schedulability of tasks in τ^{LO}, under the given task set division and scheduling algorithms?

To address I1–I3, we focus on the framework employing any implicit-deadline optimal scheduling algorithm and FP for τ^{HI} and τ^{LO}, respectively. We first develop a schedulability test for the framework for given task priority/class assignment for τ^{LO}. We then present an *Optimal* task Priority/Class Assignment (OPCA) policy under the schedulability test. In spite of success in addressing I1–I3, the framework with the current settings cannot take account of the target scheduling algorithm for τ^{HI} when we decide the scheduling algorithm for τ^{LO} and develop its schedulability test, entailing the following issue to further improve schedulability.

I4. How to address I1–I3 by utilizing the characteristics of a target scheduling algorithm for τ^{HI}?

As to I4, we target fluid scheduling [18] and fluid FP scheduling (that is fluid scheduling with task-level fixed priority, to be explained in Section V) for τ^{HI} and τ^{LO}, respectively, and allow each task to be split into both τ^{HI} and τ^{LO}. While fluid (FP) scheduling as it is cannot work in an actual system because of employing a fractional processor, we can easily construct a feasible schedule on an actual system from its schedule, e.g., using DP-Wrap [7] that employs McNaughton's wrap-around rule [19]. For given execution rate assignment for every task splitting to τ^{HI} and τ^{LO} and task priority assignment for τ^{LO}, we develop a new schedulability test, which tightly calculates other tasks' interference by utilizing characteristics of fluid (FP) scheduling. We then develop a sub-optimal execution rate and task priority assignment policy by deriving necessary conditions for optimal execution rate assignment and applying a part of OPCA, yielding significant schedulability improvement.

To demonstrate the effectiveness of our framework in covering constrained-deadline task sets, we compare our framework to existing scheduling algorithms as shown in Fig. 1. Our simulation results demonstrate that the proposed framework not only covers all task sets satisfying $\sum_{\tau_i \in \tau} C_i/D_i \leq m$ (which is comparable to implicit-deadline optimal scheduling algorithms), but also finds a number of additional task sets satisfying $\sum_{\tau_i \in \tau} C_i/D_i > m$, which have not been proven schedulable by any existing scheduling algorithms. In particular, if we focus on task sets satisfying $\sum_{\tau_i \in \tau} C_i/D_i > m$, the framework finds up to 446.3% additional schedulable task sets, compared to task sets covered by at least one of existing scheduling algorithms.

In summary, this paper makes the following contributions.

- We propose a scheduling framework for constrained-deadline task sets, which not only generalizes existing implicit-deadline optimal scheduling algorithms, but also efficiently reclaims remaining processor capacity.
- We develop a schedulability test and OPCA that can be used for the framework employing any implicit-deadline optimal scheduling algorithm and FP.
- We incorporate fluid scheduling into the framework, and develop a tight schedulability test and an execution rate and task priority assignment policy specialized for fluid scheduling, yielding significant schedulability improvement.
- We demonstrate the effectiveness of the framework in finding a number of additional schedulable task sets, which have not been proven schedulable by any existing scheduling algorithm.

The rest of this paper is organized as follows. Section II presents our system model with notations and assumptions. Section III designs the proposed scheduling framework. Section IV presents the framework employing any implicit-deadline optimal scheduling algorithm and FP. Section V presents the framework with fluid scheduling. Section VI evaluates the schedulability performance of the proposed framework, and Section VII concludes this paper with discussion.

II. System Model, Notations, and Assumptions

We consider a sporadic/periodic task model [20], where a task $\tau_i \in \tau$ is specified by the minimum inter-arrival time or period (T_i), the worst-case execution time (C_i) and the relative deadline (D_i). A task τ_i is assumed to have a constrained deadline, i.e., $C_i \leq D_i \leq T_i$, and invokes potentially infinite jobs, each of which should finish its execution within D_i time from its release. We consider *a legal job release pattern* for every task $\tau_i \in \tau$, implying a time interval between release times of two consecutive jobs of a task τ_i is at least T_i. We assume a single job cannot be executed in parallel. We call a job *active* if the job has remaining execution. We let $\delta_{sum}(\tau)$ denote the system density of τ, calculated by $\delta_{sum}(\tau) = \sum_{\tau_i \in \tau} C_i/D_i$. Also, let n denote the number of tasks in τ.

We target a platform with m (≥ 2) identical processors. We consider *preemptive*, *global* scheduling algorithms, in which a currently-executing lower-priority job can be preempted by a higher-priority job at any time, and a job can be executed in any processor and allowed to migrate from one processor to another.

A task set τ is referred to as *schedulable* by a scheduling algorithm on a platform, if there is no job deadline miss for all possible legal job release patterns invoked by τ. This paper aims at judging whether a task set τ is *schedulable*, for given

information of task parameters of τ, a target scheduling algorithm, and a target platform. For example, we can guarantee the schedulability of a task set τ on a m-processor platform by a class of implicit-deadline optimal scheduling algorithms [4–9], if the task set satisfies $\delta_{sum}(\tau) \leq m$. Note that a target scheduling algorithm itself may need limited online information, e.g., DP-Wrap [7] requires online information of the upcoming time instant at which any job has its deadline or release time; even in this case, we can check the schedulability of a task set only with task parameters, i.e., $\delta_{sum}(\tau) \leq m$. We also note that, by the definition of "schedulability", it is intractable to check schedulability by a method that requires information of all future job release times for judging whether there is no job deadline miss or not, e.g., [21].

III. Design of a Scheduling Framework for Constrained-Deadline Task Sets

In this section, we propose a new scheduling framework for constrained-deadline task sets, which logically divides a given task set τ into the higher- and lower-priority classes τ^{HI} and τ^{LO}. The framework employs an existing implicit-deadline optimal scheduling algorithm for τ^{HI}, while it enables tasks in τ^{LO} to reclaim processor capacity by employing/developing a heuristic scheduling algorithm.

Algo. 1 describes the proposed two-level scheduling framework $\mathsf{TL}(\tau,\mathsf{Algo_A},\mathsf{Algo_B})$, where $\mathsf{Algo_A}$ and $\mathsf{Algo_B}$ denote the algorithms that schedule tasks in τ^{HI} and τ^{LO}, respectively. $\mathsf{TL}(\tau,\mathsf{Algo_A},\mathsf{Algo_B})$ divides the given task set τ into τ^{HI} and $\tau^{\mathsf{LO}} = \tau \setminus \tau^{\mathsf{HI}}$, such that $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$ holds (Line 1). Here, we consider not only the division where a task belongs to either τ^{HI} or τ^{LO}, but also the division where a portion of a task belongs to τ^{HI} and the remaining portion of the same task belongs to τ^{LO}. $\mathsf{Algo_A}$ and $\mathsf{Algo_B}$ determine applicability of such two different division types; Section IV and V will show how we apply the former and the latter, respectively.

Then, for each scheduling interval $[t_a, t_b)$, tasks in τ^{HI} are scheduled by $\mathsf{Algo_A}$ with higher priorities than all tasks in τ^{LO} that are scheduled by $\mathsf{Algo_B}$ (Lines 2–5). Here, each scheduling interval depends on the target scheduling algorithm; for example, each scheduling interval in DP-Wrap [7] is set to an interval between any two consecutive time instants at which any job is released or has its deadline. When it comes to qualification of $\mathsf{Algo_A}$ and $\mathsf{Algo_B}$, $\mathsf{Algo_A}$ can be any implicit-deadline optimal scheduling algorithm as long as it can handle constrained-deadline tasks.[1] On the other hand, $\mathsf{Algo_B}$ can be any scheduling algorithm.

By the principle of the two-level scheduling and the qualification of $\mathsf{Algo_A}$, we do not need to care for schedulability of tasks in τ^{HI}, recorded as follows.

Lemma 1: Suppose that we apply the two-level scheduling framework $\mathsf{TL}(\tau,\mathsf{Algo_A},\mathsf{Algo_B})$ in Algo. 1, where $\mathsf{Algo_A}$ is any implicit-deadline optimal scheduling algorithm (as long as it can handle constrained-deadline tasks). Then, every job invoked by tasks in τ^{HI} cannot miss its deadline.

Proof: By the common feasibility condition of any implicit-deadline optimal scheduling algorithm, a task set τ is schedulable by the algorithm if $\delta_{sum}(\tau) \leq m$ holds. According to the principle of $\mathsf{TL}(\tau,\mathsf{Algo_A},\mathsf{Algo_B})$, $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$ holds, and tasks in τ^{HI} have a higher priority than all tasks in τ^{LO}. Thus, τ^{HI} is schedulable by the proposed scheduling framework. ■

[1] While most of implicit-deadline optimal scheduling algorithms can handle constrained-deadline tasks, some of them cannot, e.g., RUN [10].

Algorithm 1 Two-level scheduling framework: $\mathsf{TL}(\tau,\mathsf{Algo_A},\mathsf{Algo_B})$

1: Determine τ^{HI} and τ^{LO} satisfying $\tau^{\mathsf{HI}} \cup \tau^{\mathsf{LO}} = \tau$, $\tau^{\mathsf{HI}} \cap \tau^{\mathsf{LO}} = \emptyset$, and $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$;
2: **for** each scheduling interval $[t_a, t_b)$ **do**
3: Given processor capacity $m \cdot (t_b - t_a)$, determine the schedule of jobs of tasks in τ^{HI} in $[t_a, t_b)$ by $\mathsf{Algo_A}$;
4: Given remaining processor capacity consumed by tasks in τ^{HI}, determine the schedule of jobs of tasks in τ^{LO} in $[t_a, t_b)$ by $\mathsf{Algo_B}$;
5: **end for**

Differently from tasks in τ^{HI}, the framework itself does not guarantee the schedulablity of tasks in τ^{LO}. Therefore, in order to allow tasks in τ^{LO} to efficiently reclaim remaining processor capacity while guaranteeing their schedulability, we need to address I1–I3, to be presented in the next section.

IV. The Framework Employing any Implicit-Deadline Optimal Scheduling and FP

Among many choices of scheduling algorithms for the proposed framework in Algo. 1, this section considers $\mathsf{TL}(\tau,\ \mathsf{Any},\ \mathsf{FP})$. That is, τ^{HI} is scheduled by any implicit-deadline optimal scheduling algorithm (as long as it can handle constrained-deadline tasks), while τ^{LO} is scheduled by FP (Fixed Priority) [1] with given task priority assignment (addressing I2). The remaining issues are (i) how to guarantee schedulability of tasks in τ^{LO} under given task priority/class assignment (addressing I3), and (ii) how to find the optimal task priority/class assignment under (i) (addressing I1), both of which are explained now.

We now develop a schedulability test for a task in τ^{LO} under $\mathsf{TL}(\tau,\ \mathsf{Any},\ \mathsf{FP})$. Here, we assume every task τ_i has its own priority P_i; the smaller P_i, the higher the priority.

Let $W_i(\ell)$ denote the upper-bound of the amount of execution of jobs of τ_i in an interval of length ℓ (assuming a legal job release pattern and no job deadline miss of τ_i) under any scheduling algorithm [22]. Fig. 2 illustrates the scenario that results in $W_i(\ell)$. Here, the first job of τ_i starts its execution at the beginning of the interval of interest of length ℓ and finishes the execution at its absolute deadline, which fully executes for C_i; thereafter, following jobs of τ_i are scheduled as soon as possible. By considering the jobs fully executing for C_i and the last job executing for at most C_i, we can calculate $W_i(\ell)$ as follows [22]:

$$W_i(\ell) = \left\lfloor \frac{\ell + D_i - C_i}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \ell + D_i - C_i - \left\lfloor \frac{\ell + D_i - C_i}{T_i} \right\rfloor \cdot T_i\right). \quad (1)$$

Now, we would like to analyze whether a job of τ_k of interest (denoted by J_k) finishes its execution within its deadline. Let t and $t + D_k$ denote the release time and deadline of J_k. We focus on a set of intervals (not necessarily continuous) over $[t, t + D_k)$ in which J_k is not executed due to execution of jobs of other tasks on all m processors. Let

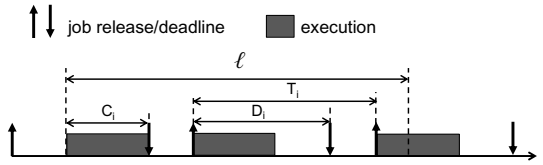


Fig. 2. Scenario where the amount of execution of jobs of a task τ_i is maximized in a given interval of length ℓ under any scheduling algorithm

Γ denote a subset of the intervals, which limits its cumulative length to $D_k - C_k$. That is, if the cumulative length of a set of the intervals is at most $D_k - C_k$, Γ denotes a set of all the intervals; otherwise, Γ represents a subset of the intervals, whose cumulative length is exactly $D_k - C_k$. Then, for J_k to execute for C_k, we need to calculate the amount of execution of jobs of tasks whose priority is higher than τ_k in $[t, t + D_k)$, which can be classified into the following two types.

- $\mathsf{E_{np}}$: Execution of tasks whose priority is higher than τ_k in Γ, and
- $\mathsf{E_p}$: Execution of tasks whose priority is higher than τ_k in $[t, t + D_k) \setminus \Gamma$,

Fig. 3 describes how the interval of $[t,\ t + D_k)$ is separated by the notion of Γ with a scenario in which J_k finishes its execution exactly at $t + D_k$. As seen in Fig. 3, J_k cannot be executed in the interval of Γ due to the execution of higher-priority jobs (the execution in the dotted rectangles representing $\mathsf{E_{np}}$), but the job fully executes in $[t,\ t + D_k)\backslash\Gamma$ in conjunction with other jobs.

To guarantee schedulability of J_k, we use two conditions, whose main idea is the same as the existing deadline analysis [23]. First, the amount of execution that potentially contributes to $\mathsf{E_{np}}$ should not be larger than $m\cdot(D_k - C_k)$; otherwise, the job of τ_k may not execute for C_k in $[t,\ t + D_k)\backslash\Gamma$. Second, the number of tasks that potentially contribute to $\mathsf{E_p}$ at any given time instant of $[t,\ t + D_k)\ \backslash\Gamma$ is at most $m - 1$; otherwise, if such tasks' jobs execute at the same time, there exists no room for J_k to execute for C_k in $[t,\ t + D_k)\backslash\Gamma$.

Then, the challenging issue is how to upper-bound the amount of execution that potentially contribute to $\mathsf{E_{np}}$ and the number of tasks that potentially have $\mathsf{E_p}$ offline. To this end, we consider the worst-case scenario where execution of each task τ_i is performed as much as possible in $[t,\ t + D_k)$, and it potentially contributes to $\mathsf{E_{np}}$ (i.e., executes in Γ) as much as possible. We use $W_i(D_k)$ since it upper-bounds the amount of execution of τ_i (in either τ^{HI} or τ^{LO}) within any interval of length D_k regardless of which scheduling algorithm is used (as well as which task/job priority is assigned) by the definition of $W_i(\ell)$. Also, considering Γ's interval length is at most $D_k - C_k$ by the definition of Γ, we can upper-bound the amount of execution of τ_i in Γ by $\min(W_i(D_k), D_k - C_k)$. By the definition of Γ, J_k never miss its deadline if the length of Γ is less than $D_k - C_k$ since this implies that the length of interval that J_k's execution is hindered by higher-priority jobs is less than $D_k - C_k$. In this case, since the schedulability of J_k is already guaranteed, we do not need to count the number of tasks that contributes to $\mathsf{E_p}$. Therefore, considering we assume that τ_i executes in Γ as much as possible and the length of Γ is exactly $D_k - C_k$, only task that holds $W_i(D_k) > D_k - C_k$ can contribute to $\mathsf{E_p}$ under our assumption.

Based on the above approach, the following lemma can guarantee the schedulability of τ under $\mathsf{TL}(\tau,\ \mathsf{Any},\ \mathsf{FP})$.

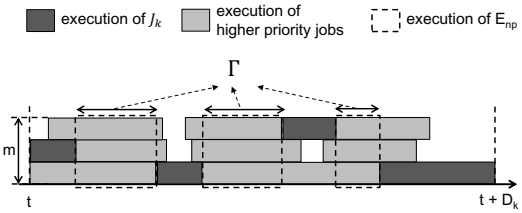


Fig. 3. Scenario where the cumulative length of Γ is equal to $D_k - C_k$, and thus the amount of execution that contribute to $\mathsf{E_{np}}$ is equal to $m\cdot(D_k - C_k)$

Lemma 2: For given τ^{HI} and τ^{LO} satisfying $\tau^{\mathsf{HI}} \cup \tau^{\mathsf{LO}} = \tau$ and $\tau^{\mathsf{HI}} \cap \tau^{\mathsf{LO}} = \emptyset$, suppose that $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$ holds, and every $\tau_k \in \tau^{\mathsf{LO}}$ satisfies the following two conditions Eqs. (2) and (3). Then, a task set τ is schedulable by $\mathsf{TL}(\tau,\ \mathsf{Any},\ \mathsf{FP})$.

$$\sum_{\tau_i \in \tau^{\mathsf{LO}} | P_i < P_k} \min\left(W_i(D_k), D_k - C_k\right) + \sum_{\tau_i \in \tau^{\mathsf{HI}}} \min\left(W_i(D_k), D_k - C_k\right) \leq m \cdot (D_k - C_k), \quad (2)$$

$$\sum_{\tau_i \in \tau^{\mathsf{LO}} | P_i < P_k \ \&\ W_i(D_k) > D_k - C_k} 1 + \sum_{\tau_i \in \tau^{\mathsf{HI}} | W_i(D_k) > D_k - C_k} 1 \leq m - 1. \quad (3)$$

Proof: Every task τ_i in τ^{HI} under $\mathsf{TL}(\tau,\ \mathsf{Any},\ \mathsf{FP})$ does not miss its deadline if $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$ holds by Lemma 1. Then, the remaining step is to prove that every task τ_k in τ^{LO} does not miss its deadline if Eqs. (2) and (3) hold.

Focusing on $[t, t + D_k]$ for the job of interest of J_k, we show that J_k is schedulable in the following two cases.

(Case i) LHS<RHS of Eq. (2) and LHS≤RHS of Eq. (3) hold: Suppose that J_k is not schedulable in this case. By the supposition, the length of Γ should be $D_k - C_k$; otherwise J_k is schedulable by the definition of Γ, which contradicts the supposition. Since $W_i(D_k)$ upper-bounds the amount of execution of τ_i in any interval of length D_k regardless of τ_i' class and priority, the LHS of Eq. (2) implies an upper-bound of the amount of execution of tasks each of whose priority is higher than τ_k in Γ. Considering m higher-priority jobs are needed for J_k not to execute at a time instant, LHS<RHS of Eq. (2) implies that the length of Γ is less than $D_k - C_k$, which means J_k never miss its deadline; thus, this contradicts the supposition.

Note that satisfying LHS<RHS of Eq. (2) implies satisfying Eq. (3). This is because, violating Eq. (3) implies that there are at least m tasks that hold $W_i(D_k) > D_k - C_k$; this always yields LHS≥RHS of Eq. (2), which contradicts LHS<RHS of Eq. (2).

(Case ii) LHS=RHS of Eq. (2) and LHS≤RHS of Eq. (3) hold: Suppose that J_k is not schedulable in this case. With the same reasoning as Case i, LHS=RHS of Eq. (2) implies that the length of Γ is exactly $D_k - C_k$. Therefore, satisfying Eq. (2) is insufficient to judge the schedulability of J_k. If Eq.(3) holds and the length of Γ is exactly $D_k - C_k$, there are at most $m - 1$ tasks each of whose priority is higher than τ_k and whose execution is performed in $[t, t + D_k] \setminus \Gamma$ (as well as in Γ). Since m jobs are needed for J_k not to execute in an interval, J_k can always fully execute in $[t, t + D_k] \setminus \Gamma$. Thus,

Algorithm 2 Optimal task priority/class assignment (OPCA) for TL(τ, Any, FP)

1: For every $\tau_i \in \tau$, $P_i \leftarrow -1$;
2: **for** P from n to 1 decreasing by 1 **do**
3: **if** $\delta_{sum}(\{\tau_i \in \tau | P_i = -1\}) \leq m$ **then**
4: Set τ^{HI} and τ^{LO} to $\{\tau_i \in \tau | P_i = -1\}$ and $\{\tau_i \in \tau | P_i \neq -1\}$, respectively;
5: Return Schedulable;
6: **end if**
7: **for** $\tau_k \in \tau | P_k = -1$ **do**
8: **if** τ_k satisfies Eqs. (2) and (3) in Lemma 2 with $P_k = P$ **then**
9: $P_k \leftarrow P$;
10: Go to Step 14;
11: **end if**
12: **end for**
13: Return Unschedulable;
14: **end for**

if LHS=RHS of Eq. (2) and LHS≤RHS of Eq. (3) hold, J_k does not miss its deadline, which contradicts the supposition. ■

Then, Algo. 2 presents an Optimal task Priority/Class Assignment (OPCA) policy, which determines whether every task in τ belongs to τ^{HI} or τ^{LO} and assigns a task priority to every task in τ^{LO} so as to make τ schedulable by Lemma 2 (if such priority/class assignment exists). To this end, the algorithm selects tasks which belong to τ^{LO} as well as determines their priorities from the lowest; then, the remaining tasks whose priority is unassigned belong to τ^{HI}. As an initial step, the highest priority (expressed by -1) is temporarily assigned to every task $\tau_i \in \tau$ (Line 1). Then, we repeat to assign the priorities from the lowest (i.e., $P = n$) to the highest (i.e., $P = 1$). For each assignment, we test whether there exists a task τ_k that passes Eqs. (2) and (3) in Lemma 2 with the priority P assuming all unassigned tasks have a higher priority than τ_k; if there exists at least one task τ_k that passes the conditions, P is assigned to τ_k, i.e., $P_k \leftarrow P$ (Lines 7–12). On the other hand, if there is no task to which we can assign P during each repetition, τ is deemed unschedulable (Line 13). Before each assignment, we test whether all remaining tasks with $P_i = -1$ can be included in τ^{HI}; if so, τ is deemed schedulable (Lines 3–6).

Algo. 2 is similar to OPA (Optimal task Priority Assignment) for FP with deadline analysis [24]. The main difference is that we do not need to assign the priority to every task; once a set of tasks whose priority is unassigned (i.e., $\tau' = \{\tau_i \in \tau | P_i = -1\}$) satisfies $\delta_{sum}(\tau') \leq m$, we can set τ^{HI} to τ'. This is because we do not need to care for the schedulability of tasks in τ^{HI} by virtue of Lemma 1. We now prove the optimality of OPCA in Algo. 2 in the following lemma.

Lemma 3: Suppose that τ is scheduled by TL(τ, Any, FP). If OPCA in Algo. 2 deems τ unschedulable, there exists no task priority/class assignment such that Lemma 2 deems τ schedulable.

Proof: We prove this lemma by using two important properties of Lemma 2: (i) the schedulability of a higher-priority task τ_k in τ^{LO} is not affected by any lower-priority task; and (ii) the schedulability of a lower-priority task τ_k in τ^{LO} is not affected by the priority ordering of its higher-priority tasks. (i) trivially holds since every $\tau_k \in \tau^{\mathsf{LO}}$ is scheduled by FP scheduling under TL(τ, Any, FP). (ii) also holds since $W_i(D_k)$ of $\tau_i \in (\tau^{\mathsf{HI}} \cup \tau^{\mathsf{LO}} | P_i < P_k)$ is the fixed value regardless of which priority and class are assigned to τ_i by the definition of $W_i(\ell)$; thus, the LHS of Eqs. (2) and (3) for a given τ_k are the fixed values regardless of the priority ordering of higher-priority tasks of τ_k.

Then, suppose that OPCA in Algo. 2 succeeds to assign task priorities up to $n - x + 1$ (or no task priority assigned if $x = 0$) but fails to assign the task priority of $n - x$ ($x >= 0$), while there exists another priority assignment policy OPCA2 that succeeds to assign task priorities up to $n - x$. We show such supposition results in contradiction for two cases: (Case 1) a set of tasks each whose priority is from n to $n-x+1$ by OPCA is the same as that by OPCA2, and (Case 2) otherwise.

(Case 1) By (i) and (ii), to successfully assign the task priority of $n - x$ to a task by OPCA2 implies that it is also possible to do that by OPCA since the schedulability of the task whose priority is $n - x$ is not affected by lower-priority tasks and the priority ordering of remaining higher-priority tasks whose priorities are not assigned yet, which contradicts the supposition.

(Case 2) Let τ_j denote one of tasks whose priority is from n to $n-x+1$ by OPCA2 but whose priority is not assigned by OPCA (if there are many choices of τ_j, we select the lowest-priority task assigned by OPCA2). Then, if we compare a set of higher-priority tasks of τ_j by OPCA2 and that by OPCA, the former subsumes the latter. By (i) and (ii), this implies that OPCA can assign a priority to τ_j, which contradicts the supposition.

By Cases 1 and 2, the lemma holds. ■

Due to the principle of the proposed framework and Lines 3–6 in Algo. 2, TL(τ, Any, FP) with OPCA in Algo. 2 dominates any implicit-deadline optimal scheduling algorithm, recorded in the following lemma.

Lemma 4: A task set deemed schedulable by the common feasibility condition of any implicit-deadline optimal scheduling is also schedulable by TL(τ, Any, FP) with OPCA in Algo. 2.

Proof: While the common feasibility condition of any implicit-deadline optimal scheduling is $\delta_{sum}(\tau) \leq m$, Lines 3–6 in Algo. 2 assigns a set of tasks satisfying $\delta_{sum}(\tau) \leq m$ to τ^{HI}. Since all tasks in τ^{HI} have a higher priority than tasks in τ^{LO}, the lemma holds. ■

Although Algo. 2 yields not only an optimal task priority/class assignment for TL(τ, Any, FP) under Lemma 2 but also a dominance relation over any implicit-deadline optimal scheduling, we still cannot fully utilize the proposed framework in Algo. 1 because of the following three reasons. First, the current schedulability test can be applied to any implicit-deadline optimal scheduling algorithm for τ^{HI}, meaning that it can be improved if we develop a new schedulability test specialized for the target scheduling algorithm. Second, FP is probably not the best scheduling algorithm that enables tasks in τ^{LO} to effectively reclaim remaining processor capacity in conjunction with a target scheduling algorithm for τ^{HI}. Third, TL(τ, Any, FP) cannot exploit the full capability of accommodating tasks in τ^{HI}, because it is usually impossible

to make τ^{HI} satisfy exactly $\delta_{sum}(\tau^{\mathsf{HI}}) = m$. In the next section, we will address the issues by employing fluid scheduling for τ^{HI}.

V. The Framework with Fluid Scheduling

In this section, we target fluid scheduling, and significantly improve schedulability of the proposed framework by addressing I4. To this end, we first recapitulate DP-Wrap to show how to translate the schedule generated by fluid-scheduling into a schedule feasible on an actual system. Then, we propose TL(τ, Fluid, Fluid-FP) and its wrapping algorithm, which employ fluid scheduling for τ^{HI} and allow a single task to split into τ^{LO} and τ^{HI}. We next develop a tight schedulability test specialized for the framework with fluid scheduling. We derive necessary conditions for optimal execution rate assignment of each task, and present a final sub-optimal execution rate and task priority assignment policy that utilizes the necessary conditions and OPCA developed in Section IV.

A. Recapitulation: fluid scheduling with DP-Wrap

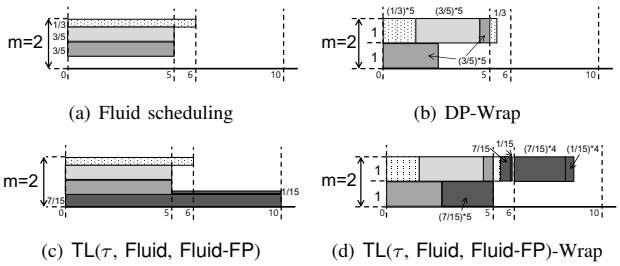
Before we adopt fluid scheduling as a scheduling algorithm for τ^{HI}, we review how fluid scheduling works in conjunction with DP-Wrap. Fluid scheduling executes each job on a fractional processor at all time instants [18]. We consider fluid scheduling such that an active job of τ_i executes with C_i/D_i rate. While fluid scheduling as it is cannot operate in an actual system because it needs infinitesimal quantum length, we can easily translate the schedule generated by fluid scheduling, into the feasible schedule on an actual system using DP-Wrap [7] that employs McNaughton's wrap-around rule [19].

DP-Wrap partitions an interval of interest into multiple intervals, such that there does not exist any job release and job deadline in the middle of each interval. Let $[t_j, t_{j+1})$ denote the jth partitioned interval. Then, every active job of τ_i in $[t_j, t_{j+1})$ under fluid scheduling executes with C_i/D_i rate, yielding $C_i/D_i \cdot (t_{j+1} - t_j)$ amount of execution in the interval. To avoid a job's execution on more than one processor, DP-Wrap utilizes McNaughton's wrap-around rule [19] as shown in the following example.

Example 1: Consider a task set τ consisting of three tasks that is scheduled by TL(τ, Any, FP) on a two-processor platform: $\tau_1(T_1{=}10, C_1{=}2, D_1{=}6)$, $\tau_2(12, 3, 5)$, $\tau_3(12, 3, 5)$; all tasks invoke their jobs periodically from $t{=}0$. Under fluid scheduling on a two-processor platform, τ_1 executes in $[0, 6)$ with $C_1/D_1{=}1/3$ rate, and τ_2 and τ_3 execute in $[0, 5)$ with $C_2/D_2{=}3/5$ rate, as shown in Fig. 4(a). DP-Wrap partitions an interval $[0, 6)$ into $[0, 5)$ and $[5, 6)$ because there is a job deadline at $t = 5$. In $[0, 5)$, τ_1 has $1/3{\cdot}5{=}5/3$ amount of execution, while τ_2 and τ_3 have $3/5{\cdot}5{=}3$ amount of execution under fluid scheduling; therefore, τ_1 and τ_2 occupy the first processor in $[0, 5/3)$ and $[5/3, 14/3)$, while τ_3 occupies the first processor in $[14/3, 5)$ and the second processor in $[0, 8/3)$ under DP-Wrap, as shown in Fig. 4(b). In $[5, 6)$, τ_1 solely has its execution amount to $1/3{\cdot}1{=}1/3$, occupying the first processor in $[5, 16/3)$.

B. TL(τ, Fluid, Fluid-FP) and TL(τ, Fluid, Fluid-FP)-Wrap

We now introduce our scheduling algorithm TL(τ, Fluid, Fluid-FP), in which each task $\tau_i(T_i, C_i, D_i) \in \tau$ is split into two subtasks which belong to τ^{HI} and τ^{LO} as follows.



(a) Fluid scheduling (b) DP-Wrap

(c) TL(τ, Fluid, Fluid-FP) (d) TL(τ, Fluid, Fluid-FP)-Wrap

Fig. 4. Schedules for Example 1 by fluid scheduling and DP-Wrap, and those for Example 2 by TL(τ, Fluid, Fluid-FP) and TL(τ, Fluid, Fluid-FP)-Wrap

- $\tau_i^{\mathsf{HI}}\ (T_i, C_i^{\mathsf{HI}}, D_i, R_i^{\mathsf{HI}}) \in \tau^{\mathsf{HI}}$, and
- $\tau_i^{\mathsf{LO}}\ (T_i, C_i^{\mathsf{LO}}, D_i, R_i^{\mathsf{LO}}, P_i^{\mathsf{LO}}) \in \tau^{\mathsf{LO}}$,

which should satisfy the following constraints:

C1. $C_i^{\mathsf{HI}} + C_i^{\mathsf{LO}} = C_i$,
C2. $R_i^{\mathsf{HI}} = C_i^{\mathsf{HI}}/D_i$,
C3. $C_i^{\mathsf{LO}}/D_i \le R_i^{\mathsf{LO}} \le 1.0 - R_i^{\mathsf{HI}}$, and
C4. $\delta_{sum}(\tau^{\mathsf{HI}}) = \sum_{\tau_i \in \tau} C_i^{\mathsf{HI}}/D_i = \sum_{\tau_i \in \tau} R_i^{\mathsf{HI}} \le m$.

where R_i^{HI} and R_i^{LO} denote the execution rate of τ_i^{HI} and τ_i^{LO}, Also, P_i^{LO} denotes the task priority of τ_i^{LO}; the smaller P_i^{LO}, the higher the priority. We also define $X_i^{\mathsf{HI}} = C_i^{\mathsf{HI}}/R_i^{\mathsf{HI}}$ and $X_i^{\mathsf{LO}} = C_i^{\mathsf{LO}}/R_i^{\mathsf{LO}}$, meaning the execution duration of a job of τ_i^{HI} and τ_i^{LO} when the job executes with R_i^{HI} and R_i^{LO} rate, respectively, to be used in the next subsection.

For given τ^{HI} and τ^{LO} with subtasks, TL(τ, Fluid, Fluid-FP) works as follows. Tasks in τ^{HI} are scheduled by fluid scheduling; an active job of τ_k^{HI} in τ^{HI} performs its execution with exactly R_k^{HI} rate. On the other hand, tasks in τ^{LO} are scheduled by fluid FP scheduling; an active job of τ_k^{LO} in τ^{LO} performs its execution with *up to* R_k^{LO} rate if there is remaining processor capacity after executing active jobs of all tasks τ_i^{HI} in τ^{HI} and all tasks τ_i^{LO} in τ^{LO} satisfying $P_i^{\mathsf{LO}} < P_k^{\mathsf{LO}}$. Here, what we mean by "up to R_k^{LO} rate" is as follows. Since τ_k^{LO} reclaims remaining processor capacity, it may not fully execute with R_k^{LO} rate. For example, suppose that we have one more task τ_y with the lowest priority and $R_y^{\mathsf{LO}} = 2/3$ in Fig. 4(a), and it has an active job in $[0, 5)$. Since other three higher-priority tasks occupy $(1/3{+}3/5{+}3/5{=}23/15)$ rate in $[0, 5)$, the remaining rate is only $(2{-}23/15{=}7/15)$. Therefore, an active job of τ_y executes with $7/15$ rate in $[0, 5)$ in spite of its rate of $2/3$.

Let us discuss the constrains C1–C4. C1 is straightforward because we need to fully execute every job of τ_i for C_i. C2 is also straightforward as τ^{HI} is scheduled by fluid scheduling. C3 prevents a task τ_i from occupying more than one processor at the same time by enforcing $R_i^{\mathsf{LO}} \le 1.0 - R_i^{\mathsf{HI}}$; that is, the summation of execution rates of τ_i^{LO} and τ_i^{HI} never be larger than 1 at any time since $R_i^{\mathsf{LO}} + R_i^{\mathsf{HI}} \le 1$ holds, which prevents a single task τ_i from executing in parallel on multiple processors at the same time. Therefore, the range of R_i^{LO} can be from C_i^{LO}/D_i (similar to fluid scheduling) to $1.0 - R_i^{\mathsf{HI}}$ (the largest rate without occupying more than one processor by τ_i^{HI} and τ_i^{LO}). Finally, C4 should hold by the principle of the proposed framework in Algo. 1.

Differently from TL(τ, Any, FP), TL(τ, Fluid, Fluid-FP) allows a task τ_i to be split into τ_i^{HI} and τ_i^{LO}. This enables the

proposed framework to take advantage of existing implicit-deadline optimal scheduling algorithms as much as possible because it is possible to achieve $\delta_{sum}(\tau^{\mathsf{HI}}) = m$ while it is usually impossible without the task split. Note that such a task split is inherently impossible for TL(τ, Any, FP), because TL(τ, Any, FP) itself cannot prevent a task from occupying more than one processor by an implicit-deadline optimal scheduling algorithm and FP at the same time.

Once TL(τ, Fluid, Fluid-FP) generates a schedule for τ, we can translate the schedule into a feasible schedule on an actual system as DP-Wrap does; we call the scheduling algorithm that generates the feasible schedule on an actual system, TL(τ, Fluid, Fluid-FP)-Wrap.

We now present an example how TL(τ, Fluid, Fluid-FP) works and how TL(τ, Fluid, Fluid-FP)-Wrap translates the schedule from TL(τ, Fluid, Fluid-FP).

Example 2: Consider a task set τ where $\tau_4(15, 5, 10)$ is added from the task set considered in Example 1; all tasks invoke their jobs periodically from $t = 0$. Suppose that C_1^{HI}=C_1=2, C_2^{HI}=C_2=3, C_3^{HI}=C_3=3 and C_4^{HI}=14/3 (implying C_4^{LO}=1/3), meaning that τ_4^{LO} is the only task belonging to τ^{LO}. Also, suppose R_4^{LO}=1/15. Under TL(τ, Fluid, Fluid-FP) on a two-processor platform, the schedules of τ_1^{HI}, τ_2^{HI} and τ_3^{HI} in $[0, 10)$ are the same as τ_1, τ_2 and τ_3 in Example 1, as shown in Figs. 4(a) and (c). τ_4^{HI} executes in $[0, 10)$ with $R_4^{\mathsf{HI}} = C_4^{\mathsf{HI}}/D_4$=7/15 rate, and τ_4^{LO} executes in $[5, 10)$ with R_4^{LO}=1/15 rate. Then, TL(τ, Fluid, Fluid-FP)-Wrap partitions $[0, 10)$ into $[0, 5)$, $[5, 6)$ and $[6, 10)$, since there are job deadlines at t=5 and t=6. In $[0, 5)$, all tasks other than τ_4 exhibit the same schedules as Example 1, while τ_4 occupies the second processor in $[8/3, 5)$ because it has 7/15·5=7/3 amount of execution. In $[5, 6)$, τ_1^{HI} exhibits the same schedule of τ_1 in Example 1, occupying the first processor in $[5, 16/3)$. τ_4^{HI} and τ_4^{LO} have 7/15·1=7/15 and 1/15·1=1/15 amount of execution, respectively, in total 8/15 amount of execution, thereby occupying the first processor in $[16/3, 88/15)$. Similarly, in $[6, 10)$, τ_4^{HI} and τ_4^{LO} occupy the first processor in $[6, 122/15)$.

As shown in the example, TL(τ, Fluid, Fluid-FP)-Wrap can translate a schedule feasible on fractional processors generated by TL(τ, Fluid, Fluid-FP), into a schedule feasible on an actual system. Therefore, the next subsections focus on TL(τ, Fluid, Fluid-FP), and develop its tight schedulability analysis, and execution rate and task priority assignment policies.

C. Tight schedulability analysis tailored to fluid scheduling

While most (if not all) existing interference-based schedulability tests assume that each task at a time instant either fully occupies a processor or does not occupy any processor (e.g., Lemma 2), few studies have addressed how to tightly analyze the interference under fluid scheduling where each task occupies a fractional processor. Hence, we develop a new schedulability test for TL(τ, Fluid, Fluid-FP) by deriving two conditions corresponding to Eqs. (2) and (3) in Lemma 2, and explain why our new schedulability test employing the conditions yields significant schedulability improvement compared to Lemma 2.

The schedulability analysis for TL(τ, Any, FP) calculates the amount of execution of jobs of τ_i in an interval of length ℓ, because the amount also implies the cumulative length of execution. On the other hand, the schedulability analysis for TL(τ, Fluid, Fluid-FP) needs to calculate the cumulative length of executions of jobs of τ_i in an interval of length ℓ. Let $L_i^{\mathsf{Class}}(\ell)$ denote the maximum cumulative length of execution of jobs of τ_i^{Class} in an interval of length ℓ if τ_i^{Class} executes with exactly R_i^{Class} rate, where Class = {HI, LO}; therefore each job of τ_i^{Class} executes for $X_i^{\mathsf{Class}} = C_i^{\mathsf{Class}}/R_i^{\mathsf{Class}}$. Fig. 5 describes the scenario for $L_i^{\mathsf{Class}}(\ell)$. Similar to the scenario of $W_i(\ell)$, the first job of τ_i in Fig. 5 begins its execution at the beginning of the interval of interest, and following jobs are scheduled as soon as possible. Using the scenario for $L_i^{\mathsf{Class}}(\ell)$, we derive the upper-bound of $L_i^{\mathsf{Class}}(\ell)$ as follows:

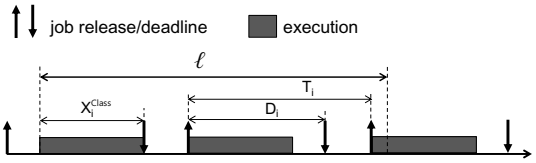


Fig. 5. Scenario where the cumulative execution length of a task τ_i is maximized in a given interval of length ℓ under fluid scheduling.

$$L_i^{\mathsf{Class}}(\ell) = \left\lfloor \frac{\ell + D_i - X_i^{\mathsf{Class}}}{T_i} \right\rfloor \cdot X_i^{\mathsf{Class}} + \min\left(X_i^{\mathsf{Class}}, \ell + D_i - X_i^{\mathsf{Class}} - \left\lfloor \frac{\ell + D_i - X_i^{\mathsf{Class}}}{T_i} \right\rfloor \cdot T_i\right). \quad (4)$$

Thus, the cumulative length of intervals when jobs of τ_i^{Class} execute in an interval of length ℓ is at most $L_i^{\mathsf{Class}}(\ell)$.

By the definition of $L_i^{\mathsf{Class}}(\ell)$, the amount of execution of jobs of τ_i^{Class} in an interval of length ℓ is upper-bounded by $R_i^{\mathsf{Class}} \cdot L_i^{\mathsf{Class}}(\ell)$ when τ_i^{Class} executes with exactly R_i^{Class} rate. We then need to show that this property also holds even if τ_i^{Class} executes with a rate lower than R_i^{Class}, which is a useful property to derive a schedulability condition under TL(τ, Fluid, Fluid-FP) to be presented in Lemma 5. This is quite trivial since if τ_i^{Class} executes with a rate lower than R_i^{Class}, the longer ℓ may be required to accommodate the same amount of execution compared to the case where τ_i^{Class} executes with exactly R_i^{Class} rate. We leave the proof of this property in Section A of the supplement file [25].

Let J_k denote a job of τ_k^{LO} of interest, and t and $t + D_k$ denote its release time and deadline, respectively. We now analyze whether J_k finishes its execution within its deadline under TL(τ, Fluid, Fluid-FP). To this end, we first redefine the notions of Γ, $\mathsf{E_{np}}$ and $\mathsf{E_p}$ of Section IV for fluid scheduling (denoted by Γ^{Fluid}, $\mathsf{E_{np}^{Fluid}}$ and $\mathsf{E_p^{Fluid}}$, respectively) as follows. We focus on a set of intervals (not necessarily continuous) over $[t, t + D_k)$ in which J_k is executed with a rate strictly lower than R_k^{LO} (or not executed at all) due to execution of jobs of other higher-priority tasks on more than $m - R_k^{\mathsf{LO}}$ processors. Let Γ^{Fluid} denote a subset of the intervals, which limits its cumulative length to $D_k - X_k^{\mathsf{LO}}$. That is, if the cumulative length of a set of the intervals is at most $D_k - X_k^{\mathsf{LO}}$, Γ^{Fluid} denotes a set of all the intervals; otherwise, Γ^{Fluid} represents a subset of the intervals, whose cumulative length is exactly $D_k - X_k^{\mathsf{LO}}$. Then, $\mathsf{E_{np}^{Fluid}}$ and $\mathsf{E_p^{Fluid}}$ are defined as execution of tasks whose priority is higher than τ_k in Γ^{Fluid} and $[t, t + D_k) \setminus \Gamma^{\mathsf{Fluid}}$ respectively.

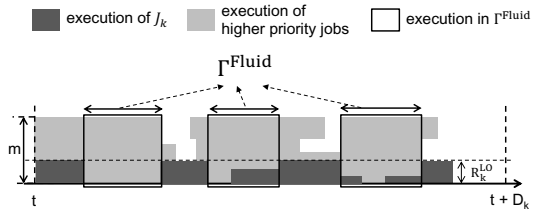


Fig. 6. Scenario where the cumulative length of Γ^{Fluid} is equal to $D_k - X_k^{\mathsf{LO}}$, but the amount of higher-priority jobs' execution performed in Γ^{Fluid} is less than $m \cdot (D_k - X_k^{\mathsf{LO}})$

Fig. 6 illustrates how the interval of $[t, t+D_k)$ is separated by the notion of Γ^{Fluid} with a scenario where the cumulative length of Γ^{Fluid} is equal to $D_k - X_k^{\mathsf{LO}}$, but the amount of higher-priority jobs' execution performed in Γ^{Fluid} is less than $m \cdot (D_k - X_k^{\mathsf{LO}})$. As shown in Fig. 6, J_k can execute with R_k^{LO} rate in $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$ while J_k executes with a rate lower than R_k^{LO} or does not execute due to jobs of higher-priority tasks in Γ^{Fluid}.

We can observe from Fig. 6 that J_k never miss its deadline as long as the amount of execution interfered by jobs of higher-priority tasks in $[t, t+D_k)$ is no larger than $R_k^{\mathsf{LO}} \cdot (D_k - X_k^{\mathsf{LO}})$; otherwise, J_k can execute for $R_k^{\mathsf{LO}} \cdot X_k^{\mathsf{LO}}$. To judge schedulability of J_k based on the observation, we consider the similar reasoning that is used with notions of $\mathsf{E_{np}}$ and $\mathsf{E_p}$ in the previous section as follows. First, the amount of execution that potentially contributes to $\mathsf{E}_{\mathsf{np}}^{\mathsf{Fluid}}$ should not be larger than $m \cdot (D_k - X_k^{\mathsf{LO}})$; otherwise, the amount of execution of J_k interfered by jobs of higher-priority tasks is larger than $R_k^{\mathsf{LO}} \cdot (D_k - X_k^{\mathsf{LO}})$ in $[t, t+D_k)$, thereby missing its deadline. Second, the summation of execution rates of tasks whose execution potentially contribute to $\mathsf{E}_{\mathsf{p}}^{\mathsf{Fluid}}$ should be smaller than $m - R_k^{\mathsf{LO}}$; otherwise, J_k is not guaranteed to execute with R_k^{LO} rate in $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$, and thus we cannot guarantee that J_k executes for $R_k^{\mathsf{LO}} \cdot X_k^{\mathsf{LO}}$ in $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$.

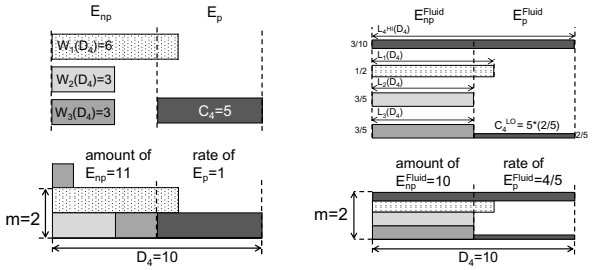
Similar to Lemma 2, we develop our schedulability analysis for TL(τ, Fluid, Fluid-FP) using two conditions regarding $\mathsf{E}_{\mathsf{np}}^{\mathsf{Fluid}}$ and $\mathsf{E}_{\mathsf{p}}^{\mathsf{Fluid}}$ that correspond to Eqs. (5) and (6) as follows.

Lemma 5: Suppose that every task $\tau_i \in \tau$ is split into $\tau_i^{\mathsf{HI}} \in \tau^{\mathsf{HI}}$ and $\tau_i^{\mathsf{LO}} \in \tau^{\mathsf{LO}}$, satisfying C1–C4. Then, τ is schedulable by TL(τ, Fluid, Fluid-FP) if every $\tau_k^{\mathsf{LO}} \in \tau^{\mathsf{LO}}$ with $C_k^{\mathsf{LO}} > 0$ satisfies the following two conditions Eqs. (5) and (6).

$$\sum_{\tau_i \in \tau | P_i^{\mathsf{LO}} < P_k^{\mathsf{LO}}} R_i^{\mathsf{LO}} \cdot \min\left(L_i^{\mathsf{LO}}(D_k), D_k - X_k^{\mathsf{LO}}\right) + \sum_{\tau_i \in \tau} R_i^{\mathsf{HI}} \cdot \min\left(L_i^{\mathsf{HI}}(D_k), D_k - X_k^{\mathsf{LO}}\right) \leq m \cdot (D_k - X_k^{\mathsf{LO}}), \quad (5)$$

$$\sum_{\tau_i \in \tau | P_i^{\mathsf{LO}} < P_k^{\mathsf{LO}} \,\&\, L_i^{\mathsf{LO}}(D_k) > D_k - X_k^{\mathsf{LO}}} R_i^{\mathsf{LO}} + \sum_{\tau_i \in \tau | L_i^{\mathsf{HI}}(D_k) > D_k - X_k^{\mathsf{LO}}} R_i^{\mathsf{HI}} \leq m - R_k^{\mathsf{LO}}. \quad (6)$$

Proof: Since $\delta_{sum}(\tau^{\mathsf{HI}}) \leq m$ (i.e., C4) holds under TL(τ, Fluid, Fluid-FP), every task τ_i^{HI} in τ^{HI} is schedulable by Lemma 1. Then, suppose that a job J_k of a task τ_k^{LO} in τ^{LO} is not schedulable even if Eqs. (5) and (6) hold. By the definition of Γ^{Fluid}, the supposition implies that the length of Γ^{Fluid} should be $D_k - X_k^{\mathsf{LO}}$ since J_k is schedulable otherwise. Therefore, we only consider the case where the length of Γ^{Fluid} is $D_k - X_k^{\mathsf{LO}}$.



(a) Lemma 2 (b) Lemma 5

Fig. 7. Schedulablity analysis comparison for Example 3: Lemmas 2 and 5 for TL(τ, Any, FP) and TL(τ, Fluid, Fluid-FP), respectively

We prove this lemma by showing contradiction of the supposition for (Case i) a special case where jobs of higher-priority tasks execute in Γ^{Fluid} as much as possible and (Case ii) a general case other than Case 1, where some execution of a higher-priority task τ_i performed in Γ^{Fluid} under Case 1 moves to $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$.

(Case i) By the property of Eq. (4), the amount of execution of τ_i^{Class} performed in $[t, t+D_k)$ is upper-bounded by $R_i^{\mathsf{Class}} \cdot L_i^{\mathsf{Class}}(D_k)$. Considering the length of Γ^{Fluid} is $D_k - X_k^{\mathsf{LO}}$ by the supposition, the amount of execution of τ_i^{Class} performed in Γ^{Fluid} is upper-bounded by $R_i^{\mathsf{Class}} \cdot \min(L_i^{\mathsf{Class}}(D_k), D_k - X_k^{\mathsf{LO}})$. Since execution of each higher-priority task τ_i is performed as much as possible in Γ^{Fluid} in this case and Eq. (5) holds, only tasks that hold $L_i^{\mathsf{Class}}(D_k) > D_k - X_k^{\mathsf{LO}}$ can contribute to $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$. Thus, if Eqs. (5) and (6) hold, J_k never miss its deadline since J_k' execution is not interfered at all in $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$, which contradicts the supposition.

(Case ii) Let α be the space in Γ^{Fluid}, occupied by higher-priority tasks under Case i but not under Case ii. Since Case i implies the largest possible execution of higher-priority tasks in Γ^{Fluid}, the amount of execution in α should be performed by J_k unless J_k already executes with R_k^{LO} rate in α or J_k does not have any remaining execution in α. The former contradicts the definition of Γ and the latter implies that J_k is schedulable. Therefore, even if the amount of execution in α moves to $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$ and it fully prevents J_k's execution in $[t, t+D_k) \setminus \Gamma^{\mathsf{Fluid}}$, J_k can compensate it in Γ^{Fluid}. This implies that J_k executes its full execution in $[t, t+D_k)$, which contradicts the supposition. ■

Now, the following example presents how Lemma 5 works.

Example 3: Recall the same task set as Example 2 where τ_1 increases its execution time by 1 as follows: $\tau_1(T_1{=}10, C_1{=}3, D_i{=}6)$. Suppose that $C_1^{\mathsf{HI}}{=}C_1{=}3$, $C_2^{\mathsf{HI}}{=}C_2{=}3$, $C_3^{\mathsf{HI}}{=}C_3{=}3$ and $C_4^{\mathsf{HI}}{=}3$ (implying $C_4^{\mathsf{LO}}{=}2$), meaning that τ_4^{LO} is the only task in τ^{LO}. Also, suppose that $X_4^{\mathsf{LO}}{=}5$. As shown in Fig. 7(a), under TL(τ, Any, FP) on a two-processor platform, Lemma 2 cannot deem τ schedulable because the summation of $W_1(D_4)$, $W_2(D_4)$ and $W_3(D_4)$ (i.e., 11) is larger than $m \cdot (D_4 - C_4)$ (i.e., 10), meaning the amount of $\mathsf{E}_{\mathsf{np}}^{\mathsf{Fluid}}$ is larger than that τ_4 can accommodate to avoid deadline miss. On the other hand, under TL(τ, Fluid, Fluid-FP), Lemma 5 deems τ schedulable, because amount

of $\mathsf{E}_{\mathsf{np}}^{\mathsf{Fluid}}$ is not larger than $m \cdot (D_k - X_k^{\mathsf{LO}})$, and the amount of execution rate satisfying $L_i^{\mathsf{HI}}(D_4) > D_4 - X_4^{\mathsf{LO}}$ is smaller than $m - R_4^{\mathsf{LO}}$ (i.e., $3/10+1/2 < 2-2/5$), as shown in Fig. 7(b).

Let us discuss why the schedulability performance of Lemma 5 is better than that of Lemma 2; recall that the two lemmas target different scheduling algorithms TL(τ, Fluid, Fluid-FP) and TL(τ, Any, FP). First, if we compare $R_i^{\mathsf{HI}} \cdot L_i^{\mathsf{HI}}(\ell)$ (or $R_i^{\mathsf{LO}} \cdot L_i^{\mathsf{LO}}(\ell)$) in Eq. (5) and $W_i(\ell)$ in Eq. (2), the former is always no larger than the latter; for example, if we focus on an interval of length 3 in Example 3, $R_2^{\mathsf{HI}} \cdot L_2^{\mathsf{HI}}(3)=3/5 \cdot 3=9/5$ is smaller than $W_2(3)=3$. Also, since $L_i^{\mathsf{HI}}(\ell)$ (or $L_i^{\mathsf{LO}}(\ell)$) is no smaller than $W_i(\ell)$, the "min" operation in Eq. (5) reduces the interference more than that in Eq. (2). Therefore, the LHS of Eq. (5) is always no larger than that of Eq. (2) even with the same RHS, i.e., $X_k^{\mathsf{LO}} = C_k$. On the other hands, Eq. (6) is more pessimistic than Eq. (3) even with $X_k^{\mathsf{LO}} = C_k$, due to the the relationship of $L_i^{\mathsf{HI}}(\ell)$ (or $L_i^{\mathsf{LO}}(\ell)) \geq W_i(\ell)$, which implies that Lemma 5 cannot dominate Lemma 2. However, Lemma 5 is usually tighter than Lemma 2, because the advantage of Lemma 5 outperforms the pessimism of the lemma, to be demonstrated in Section VI. For better schedulability of TL(τ, Fluid, Fluid-FP), we can apply both lemmas, because Lemma 2 holds for TL(τ, Fluid, Fluid-FP)-wrap due to its wrapping algorithm.

D. Necessary conditions for optimal execution rate assignment

In this subsection, we consider a problem of optimal execution rate assignment under the schedulability test in Lemma 5, which determines R_i^{HI} and R_i^{LO} for every $\tau_i \in \tau$, and derive Necessary Conditions for Optimal execution Rate Assignment (NC-ORA), which is presented in Algo. 3. While the main idea of NC-ORA is applicable to TL(τ, Fluid, Fluid-FP) with any valid $R_i^{\mathsf{LO}} \in [C_i^{\mathsf{LO}}/D_i, 1 - R_i^{\mathsf{HI}}]$, we focus on the case where R_i^{LO} is set to C_i^{LO}/C_i for every $\tau_i^{\mathsf{LO}} \in \tau^{\mathsf{LO}}$; we present how to extend NC-ORA to any valid R_i^{LO} in the next subsection. Therefore, we now deal with a situation where $R_i^{\mathsf{HI}} = C_i^{\mathsf{HI}}/D_i$ and $R_i^{\mathsf{LO}} = C_i^{\mathsf{LO}}/C_i$ (implying $X_i^{\mathsf{HI}} = D_i$ and $X_i^{\mathsf{LO}} = C_i$), and focus on the problem of determining R_i^{HI} (thereby R_i^{LO}), which is equivalent to determining C_i^{HI} (thereby C_i^{LO}). Since the sum of execution rates of tasks in τ^{HI} is limited by m (i.e., $\delta_{sum}(\tau^{\mathsf{HI}}) = \sum_{\tau_i \in \tau} C_i^{\mathsf{HI}}/D_i = \sum_{\tau_i \in \tau} R_i^{\mathsf{HI}} \leq m$) under TL($\tau$, Fluid, Fluid-FP), it is important to effectively assign execution rate R_i^{HI} of every $\tau_i^{\mathsf{HI}} \in \tau^{\mathsf{HI}}$ so as to make τ schedulable.

In the beginning, NC-ORA investigates a simple condition for C_k^{HI}. If $\sum_{\tau_i \in \tau \setminus \{\tau_k\}} C_i/D_i < m$ holds for τ_k, we can accommodate all tasks other than τ_k in τ^{HI} without any task split, implying that τ_k^{HI} can consume $m - \sum_{\tau_i \in \tau \setminus \{\tau_k\}} C_i/D_i$ execution rate without compromising the accommodation of all other tasks in τ^{HI}. This is addressed in Lines 1–8 in Algo. 3, where C_k^{HI} is upper-bounded by C_k and C_k^{LO} is set to $C_k - C_k^{\mathsf{HI}}$.

We next derive non-trivial necessary conditions for optimal execution rate assignment. Suppose that there exists an *optimal* execution rate assignment R_i^{HI} for every $\tau_i^{\mathsf{HI}} \in \tau^{\mathsf{HI}}$, such that τ with the execution rate assignment and some task-priority assignment for τ^{LO} is schedulable by the schedulability analysis in Lemma 5. We now calculate the largest possible execution rate of R_i^{LO} of each task τ_i^{LO}, assuming i) the execution rate assignment of other tasks in τ^{HI}, which is the most favorable to τ_i^{LO} and ii) τ_i^{LO} having the highest priority among tasks in τ^{LO}. This implies that if R_i^{LO} is strictly larger than the calculated execution rate, τ_i^{LO} cannot be schedulable by Lemma 5, with any execution rate assignment of other tasks and any priority assignment of τ_i^{LO} itself as well as other tasks. In other words, this calculates the essential execution rate of τ_i^{HI} that should be included in R_i^{HI} in order to make τ_i^{LO} schedulable.

Algorithm 3 NC-ORA(τ)

1: **for** $\tau_k \in \tau$ **do**
2: **if** $\sum_{\tau_i \in \tau \setminus \{\tau_k\}} C_i/D_i < m$ **then**
3: $C_k^{\mathsf{HI}} \leftarrow \min\left(C_k, D_k \cdot \left(m - \sum_{\tau_i \in \tau \setminus \{\tau_k\}} C_i/D_i\right)\right)$;
4: **else**
5: $C_k^{\mathsf{HI}} \leftarrow 0$;
6: **end if**
7: $C_k^{\mathsf{LO}} \leftarrow C_k - C_k^{\mathsf{HI}}$;
8: **end for**
9: **for** $\tau_k^{\mathsf{LO}} | C_k^{\mathsf{LO}} > 0 \in \tau^{\mathsf{LO}}$ **do**
10: **if** τ_k^{LO} is deemed unschedulable by Lemma 5, with the execution rate and priority assignment according to Lemma 6 **then**
11: Using binary search, find and assign the largest C_k^{LO} which makes τ_k^{LO} deemed schedulable by Lemma 5, with the execution rate and priority assignment according to Lemma 6;
12: $C_k^{\mathsf{HI}} \leftarrow C_k - C_k^{\mathsf{LO}}$;
13: **end if**
14: **end for**

We formally present the most favorable execution rate and priority assignment for τ_i^{LO} to be schedulable by TL(τ, Fluid, Fluid-FP).

Lemma 6: For given $R_k^{\mathsf{HI}} = C_k^{\mathsf{HI}}/D_k$ and $R_k^{\mathsf{LO}} = C_k^{\mathsf{LO}}/C_k$ (therefore given C_k^{HI} and C_k^{LO}), consider an execution rate assignment of R_i^{HI} for every $\tau_i \in \tau \setminus \{\tau_k\}$ as follows. We first sort every task $\tau_i \in \tau \setminus \{\tau_k\}$ in a non-decreasing order of $L_i^{\mathsf{HI}}(D_k)$. We then sequentially select a task τ_i from the sorted list and set R_i^{HI} to C_i/D_i until $\sum_{\tau_i \in \tau^{\mathsf{B}}(\tau_k)} R_i^{\mathsf{HI}} \leq m - R_k^{\mathsf{HI}}$ holds, where $\tau^{\mathsf{B}}(\tau_k)$ denotes a set of selected tasks. Next, we select one more task τ_j from the sorted list and then set R_j^{HI} to $\left(m - R_k^{\mathsf{HI}} - \sum_{\tau_i \in \tau^{\mathsf{B}}(\tau_k) \setminus \{\tau_j\}} R_i^{\mathsf{HI}}\right)$, which yields $\sum_{\tau_i \in \tau^{\mathsf{B}}(\tau_k)} R_i^{\mathsf{HI}} = m - R_k^{\mathsf{HI}}$. For the remaining tasks $\tau_i \notin \tau^{\mathsf{B}}(\tau_k)$, we set R_i^{HI} to zero. Since R_i^{HI} for every task $\tau_i \in \tau$ is determined, R_i^{LO} can be calculated.

Suppose that Lemma 5 does not deem τ_k^{LO} schedulable when we apply the above execution rate assignment and give τ_k^{LO} the highest task priority among every $\tau_i^{\mathsf{LO}} \in \tau^{\mathsf{LO}}$. Then, Lemma 5 does not deem τ_k^{LO} schedulable with any execution rate assignment and task priority assignment.

Proof: We show that above execution rate assignment induces the smallest value for both LHS of Eqs. (5) and (6).

Due to the sorting policy for selecting tasks, we have two observations for the above execution rate assignment policy. First, if we add (*likewise* subtract) ϵ to R_i^{HI}, the second term of the LHS of Eq. (5) is increased (*likewise* decrease) by $\epsilon \cdot \min(L_i^{\mathsf{HI}}(D_k), D_k - X_k^{\mathsf{LO}})$, which depends on $L_i^{\mathsf{HI}}(D_k)$. Second, we cannot find $\tau_x \in \tau^{\mathsf{B}}(\tau_k)$ and $\tau_y \in \tau \setminus \tau^{\mathsf{B}}(\tau_k)$ such that $L_x^{\mathsf{HI}}(D_k) > D_k - X_k^{\mathsf{LO}}$ and $L_y^{\mathsf{HI}}(D_k) \leq D_k - X_y^{\mathsf{LO}}$ hold. Therefore, if we subtract ϵ to R_x^{HI} for $\tau_x \in \tau^{\mathsf{B}}(\tau_k)$ and

add ϵ to R_y^{HI} for $\tau_y \in \tau \setminus \tau^{\mathsf{B}}(\tau_k)$, we cannot decrease the LHS of Eqs. (5) and (6), which proves the lemma. ■

Using Lemma 6, we can judge whether given rate assignment for τ_k (i.e., given R_k^{HI} and R_k^{LO}) makes τ_k^{LO} schedulable assuming the most favorable execution rate and task priority assignment. Therefore, if we find the largest R_k^{LO} that makes τ_k^{LO} schedulable with the assumption, we also find the essential execution rate of τ_k^{HI}; if R_k^{HI} is less than the essential rate, τ_k^{LO} cannot be schedulable with any execution rate and task priority assignment. Lines 9–14 in Algo. 3 present how NC-ORA finds the largest possible value of R_k^{LO} (or equivalently C_k^{LO}) with the most favorable situation identified in Lemma 6. That is, if τ_k^{LO} with current C_k^{LO} (therefore R_k^{LO}) is unschedulable with the execution rate and task priority assignment in Lemma 6, we can recalculate the largest C_k^{LO} using binary search. We can apply such binary search because if Lemma 5 deems τ_k^{LO} schedulable with given R_k^{LO} and the execution rate and task priority assignment by Lemma 6, then the lemma deems τ_k^{LO} schedulable with a decreased R_k^{LO} and the execution rate and task priority assignment by Lemma 6. In Sections B and C of the supplement file [25], we prove that such binary search can be done without backtracking in Line 11 of Algo. 3 and also illustrate how NC-ORA works with an example.

We now prove that NC-ORA in Algo. 3 derives necessary conditions for optimal execution rate assignment with the following lemma.

Lemma 7: Consider τ is scheduled by TL(τ, Fluid, Fluid-FP). Suppose that there is an optimal execution rate and task priority assignment of τ (denoted by X) that is deemed schedulable by Lemma 5, and we let Y denote an execution rate assignment by NC-ORA in Algo. 3. Then, (i) R_i^{HI} under X is no smaller than that under Y for every $\tau_i^{\mathsf{HI}} \in \tau^{\mathsf{HI}}$, implying that the execution rate assigned by NC-ORA is a necessary condition for the optimal execution rate assignment. Therefore, (ii) if $\delta_{sum}(\tau^{\mathsf{HI}}) > m$ under Y holds, τ cannot be deemed schedulable by Lemma 5 with any execution rate and task priority assignment.

Proof: As we discussed, if R_k^{HI} is less than the one assigned by NC-ORA, τ_k^{LO} cannot be deemed schedulable by Lemma 5 with any execution rate and task priority assignment of other tasks. Therefore, (i) holds. By (i), (ii) trivially holds. ■

E. Final sub-optimal execution rate and task priority assignment

Now, we decide the execution rate and task priority of every task using three steps: applying (a part of) OPCA in Algo. 2, NC-ORA in Algo 3, and heuristic policies for remaining execution rate and task priority assignment.

We first apply a priority assignment part of OPCA in Algo. 2 developed for TL(τ, Any, FP). Algo. 2 checks whether a task with the lowest priority among all unassigned tasks can be schedulable by Lemma 2. If a task is assigned the lowest priority by Algo. 2, the task can be schedulable with any ordering of other higher-priority tasks as long as they are scheduled on an actual system. Therefore, under TL(τ, Fluid, Fluid-FP)-Wrap (but not TL(τ, Fluid, Fluid-FP) itself), we can also guarantee schedulability. Hence, we apply OPCA in Algo. 2, and find as many as tasks which can be assigned the lowest priorities. For those tasks, we set C_i^{LO} and C_i^{HI} to C_i and 0, respectively. This task priority (and execution rate) assignment is always beneficial to schedulablity, since both split subtasks (i.e., τ_i^{HI} and τ_i^{LO}) of those tasks do not affect the schedulability of other tasks.

Then, for the remaining tasks, we can apply NC-ORA in Algo. 3. To improve schedulability, we may consider more general X_k^{LO}, i.e., $X_k^{\mathsf{LO}} \in [C_k^{\mathsf{LO}}/(1 - R_i^{\mathsf{HI}}), D_k]$ (from C3 in Section V-B). To this end, we can apply given Z candidates by linearly chopping the interval $[C_k^{\mathsf{LO}}/(1 - R_i^{\mathsf{HI}}), D_k]$. That is, when we find the largest C_k^{LO} using binary search in Line 11 of Algo. 3, we can try Z candidates with different value of X_k^{LO} described above. Such an approach yields a tighter lower-bound of C_i^{HI} of each task that should be included in τ^{HI}.

After applying OPCA and NC-ORA, the remaining issue is how to assign execution rate for τ^{HI} so as to satisfy $\delta_{sum}(\tau^{\mathsf{HI}}) = m$, and how to assign task priorities in τ^{LO}. We consider DM (Deadline Monotonic) [26] and LD (Largest Density first), which give a higher priority to a task with a smaller relative deadline D_i and a larger C_i/D_i, respectively. If the relative deadline of τ_k of interest is smaller than a higher-priority task τ_i^{HI}, $L_i^{\mathsf{HI}}(D_k)$ is equal to D_k, implying that τ_i^{HI} fully contributes to both $\mathsf{E_{np}}$ and $\mathsf{E_p}$, which is the worst-case situation. DM can prevent such a situation, by assigning a higher priority to a task with a smaller relative deadline. Also, if we consider the schedulability of τ_i^{LO}, the larger C_i/D_i implies that we have less choices for X_i^{LO}, yielding lower probability to guarantee the schedulability of τ_i^{LO}. Therefore, LD can be a reasonable heuristic. While we can apply either DM or LD to both execution rate and task priority assignment, we also consider combinations of DM and LD, e.g., apply DM to execution rate assignment and LD to task priority assignment (denoted by DM/LD), or vise versa (denoted by LD/DM). Such combinations bring a significant synergy since τ_k with a shorter relative deadline and a larger C_k/D_k has both disadvantages we mentioned so far. By giving a higher priority to such τ_k, the combinations of DM and LD significantly improve schedulability of the entire task set. Section VI will demonstrate effectiveness of DM, LD and combinations thereof via simulations.

Note that a schedulability test in Lemma 5 does not dominate that in Lemma 2 (although the former is better than the latter in most cases), as we discussed in Section V-C. Therefore, when we check the schedulability of a task set with the final sub-optimal execution rate and task priority assignment, we use both lemmas, which slightly improves the schedulability compared to using Lemma 5 only.

VI. Evaluation

In this section, we present simulation results to evaluate the proposed scheduling framework and compare its performance to existing schedulability tests for implicit-deadline optimal and heuristic scheduling algorithms.

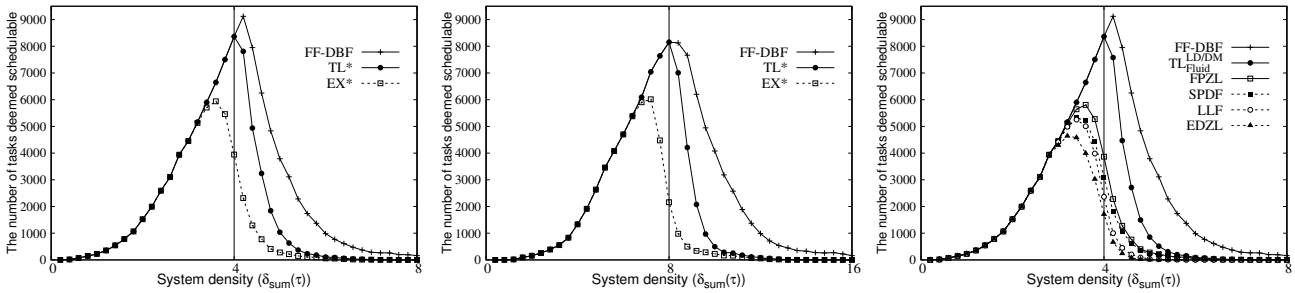
We randomly generate 100,000 constrained-deadline task sets for each $m \in \{2, 4, 8, 16\}$, based on a technique proposed in [27] used in many studies, e.g., [13, 23]; the detailed task set generation procedure is described in Section D of the supplement file [25]. We only take account of task sets that pass a *necessary feasibility condition* presented in [28]

TABLE I. THE NUMBER OF TASK SETS DEEMED SCHEDULABLE BY INDIVIDUAL SCHEDULABILITY TESTS (FOR τ SATISFYING $\delta_{sum}(\tau) \leq m$)

m	EDF-CF	EQDF	EDZL	LLF	EQDZL	SPDF	FPZL	$\mathsf{TL}_{\mathsf{Any}}$	$\mathsf{TL}^{\mathsf{LD}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{DM}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{DM/LD}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$	OPT	EX*	TL*	$\frac{\mathsf{TL}^*}{\mathsf{EX}^*}$ (%)	FF-DBF
2	27172	41003	46899	49392	50148	49335	51230	55373	55373	55373	55373	55373	55373	51821	55373	106.86%	55373
4	20437	26790	38586	42382	43823	43842	46519	54414	54414	54414	54414	54414	54414	47010	54414	115.75%	54414
8	16951	18703	32881	37609	39124	40561	43766	54756	54756	54756	54756	54756	54756	44360	54756	123.44%	54756
16	15458	14792	29591	34735	35754	38803	42060	54677	54677	54677	54677	54677	54677	42742	54677	127.92%	54677

TABLE II. THE NUMBER OF TASK SETS DEEMED SCHEDULABLE BY INDIVIDUAL SCHEDULABILITY TESTS (FOR τ SATISFYING $\delta_{sum}(\tau) > m$)

m	EDF-CF	EQDF	EDZL	LLF	EQDZL	SPDF	FPZL	$\mathsf{TL}_{\mathsf{Any}}$	$\mathsf{TL}^{\mathsf{LD}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{DM}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{DM/LD}}_{\mathsf{Fluid}}$	$\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$	OPT	EX*	TL*	$\frac{\mathsf{TL}^*}{\mathsf{EX}^*}$ (%)	FF-DBF
2	342	4480	6468	8353	10148	12281	13585	11041	20692	23822	22504	25131	0	14231	26686	187.52%	44627
4	6	299	1005	1833	2605	4734	5742	4677	13648	16204	17258	18572	0	5833	20546	352.23%	45586
8	0	4	61	330	355	2361	2942	2584	9126	10944	12702	14160	0	2943	15585	529.56%	45244
16	0	0	3	113	53	1817	2268	2054	6421	7401	9127	11688	0	2268	12390	546.29%	45323



(a) m=4 (b) m=8 (c) m=4

Fig. 8. The number of task sets deemed schedulable by individual schedulability tests, according to the system density $\delta_{sum}(\tau)$

(denoted by FF-DBF). Since it has not been identified how many task sets passing FF-DBF are actually schedulable by any scheduling algorithm, FF-DBF can be interpreted as a (probably unachievable) upper-bound of schedulability tests.

For our evaluation, we consider the following schedulability tests of different scheduling algorithms.

- $\mathsf{TL}_{\mathsf{Any}}$: a schedulability test for TL(τ, Any, FP) employing OPCA in Algo. 2 in Section IV,
- $\mathsf{TL}^{\mathsf{DM}}_{\mathsf{Fluid}}$, $\mathsf{TL}^{\mathsf{LD}}_{\mathsf{Fluid}}$, $\mathsf{TL}^{\mathsf{DM/LD}}_{\mathsf{Fluid}}$ and $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$: schedulability tests for TL(τ, Fluid, Fluid-FP) applying DM/DM, LD/LD, DM/LD and LD/DM for assigning remaining execution rate for τ^{HI} and assigning task priorities for τ^{LO}, respectively, after applying OPCA and NC-ORA described in Section V-E,[2]
- OPT: a schedulability test of any existing implicit-deadline optimal scheduling algorithm (i.e., $\delta_{sum}(\tau) \leq m$),
- EDZL, LLF, EDF-CF, EQDF, EQDZL, SPDF and FPZL: schedulability tests for EDZL, LLF, EDF-CF, EQDF, EQDZL, SPDF and FPZL [11–17],[3] and
- TL* and EX*: a schedulability test to check whether a given task set is deemed schedulable by at least one of the five schedulability tests for the proposed framework, and that by at least one of the seven schedulability tests of existing heuristic algorithms.

Tables I and II show the number of task sets deemed schedulable by individual schedulability tests, among task sets satisfying $\delta_{sum}(\tau) \leq m$ (about 55,000 task sets for each m) and task sets satisfying $\delta_{sum}(\tau) > m$ (about 45,000 task sets for each m), respectively. Fig. 1 in the introduction and Fig. 8 plot the number of tasks deemed schedulable by individual schedulability tests, according to the system density $\delta_{sum}(\tau)$.

[2]For applying NC-ORA, we set B and Z to 10 and 1000, respectively.

[3]Among some schedulability tests for a single heuristic scheduling algorithm, we choose the best one with polynomial time-complexity.

We first investigate how many additional task sets are covered by our two-level scheduling framework compared to the existing implicit-deadline optimal and heuristic scheduling algorithms (i.e., TL* versus OPT and EX*). As seen in Figs. 8(a) and 8(b), performance of EX* sharply decreases before a point of $\delta_{sum}(\tau) = m$, and nearly converges to 0 when $\delta_{sum}(\tau)$ is about $1.25 \cdot m$. That is, EX* covers 86.4% and 81.0% task sets with $\delta_{sum}(\tau) \leq m$, and 12.8% and 6.5% task sets with $\delta_{sum}(\tau) > m$ for m = 4 and 8, respectively; these numeric values can be calculated by Tables I and II. Also, OPT only covers task sets with $\delta_{sum}(\tau) \leq m$ (shown as the vertical line in each figure), and does not cover any task sets with $\delta_{sum}(\tau) > m$ as shown in the tables. Differently from heuristic and implicit-deadline optimal scheduling algorithms, TL* not only covers all task sets satisfying $\delta_{sum}(\tau) \leq m$, but also finds from 59.8% to 27.3% schedulable task sets satisfying $\delta_{sum}(\tau) > m$ for m increasing from 2 to 16, which are from 87.5% to 446.3% improvement over EX* as shown in Table II. This demonstrates the proposed framework not only generalizes existing implicit-deadline optimal scheduling algorithms but also effectively exploits characteristics thereof.

We now compare our single best schedulability test $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$, with individual schedulability tests for existing heuristic algorithms. As shown in Figs. 8(c) and 1 for $m = 4$ and 8, $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$ and individual schedulability tests of existing heuristic algorithms show the similar trends to TL* and EX* in Figs. 8(a) and 8(b). As performance of TL* and EX* are mainly contributed by $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$ and FPZL, respectively (as shown in Tables I and II), $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$ significantly outperforms FPZL, yielding up to 30.0% and 415.3% improvement for task sets satisfying $\delta_{sum}(\tau) \leq m$ and $\delta_{sum}(\tau) > m$, respectively.

We next compare performance of schedulability tests for the proposed framework. We first check that the four schedulability tests $\mathsf{TL}_{\overline{\mathsf{Fluid}}}$ surpasses $\mathsf{TL}_{\mathsf{Any}}$, which is straightforward. We now discuss how DM and LD (for assigning remaining execution rate for τ^{HI} and assigning task priorities for τ^{LO})

influence schedulablity of $\mathsf{TL}(\tau,\ \mathsf{Fluid},\ \mathsf{Fluid\text{-}FP})$. Among four schedulability tests employing DM, LD and combinations thereof, $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$ shows the best performance due to their synergy as we discussed in Section V-E. In particular, $\mathsf{TL}^{\mathsf{LD/DM}}_{\mathsf{Fluid}}$ covers from 56.3% to 25.8% of task sets satisfying $\delta_{sum}(\tau) > m$ as m increases from 2 to 16.

Note that while we discussed constrained-deadline task sets only, the schedulability results for implicit-deadline task sets are straightforward. That is, all the five schedulability tests for the proposed framework cover all task sets with $\sum_{\tau_i \in \tau} C_i/T_i \leq m$, which is the same as any implicit-deadline optimal scheduling algorithms.

VII. Conclusion and Discussion

In this paper, we proposed a two-level scheduling framework which takes advantages of both implicit-deadline optimal and heuristic scheduling algorithms. We first presented a general case how to address three technical issues I1–I3. We then presented a specific case how to further improve schedulability by utilizing the characteristics of the specific case. We demonstrated that the proposed framework not only outperforms all existing scheduling algorithms in covering schedulable task sets, but also finds a number of additional schedulable task sets that have not been proven schedulable by any existing scheduling algorithm.

While we showed two cases how to exploit the proposed framework, the potential of the framework is not restricted to the two cases. The first direction of future work is to improve schedulability of the framework, entailing not only selection/development of more proper implicit-deadline optimal and heuristic scheduling algorithms that minimize the interference to tasks in τ^{LO}, but also development of a tight schedulability test for the framework with those scheduling algorithms. The second direction is to apply the framework to other settings. For example, while $\mathsf{TL}(\tau,\ \mathsf{Fluid},\ \mathsf{Fluid\text{-}FP})$ and its wrapping algorithm as of now require limited online information of the upcoming time instant at which any job has its deadline or release time, we may relax such a constraint by applying other scheduling algorithms (e.g., P-Fair), which needs development of its tight schedulability test.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016R1D1A1B03930580, 2016R1A6A3A11930688) and the Ministry of Science, ICT & Future Planning (2017R1A2B2002458, 2017H1D8A2031628). Jinkyu Lee is the corresponding author.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS*, 1990, pp. 182–190.

[3] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.

[4] J. H. Anderson and A. Srinivasan, "Early-release fair scheduling," in *ECRTS*, 2000, pp. 35–43.

[5] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS*, 2006, pp. 101–110.

[6] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *RTCSA*, 2006, pp. 322–334.

[7] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS*, 2010, pp. 3–13.

[8] G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic, "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks," in *ECRTS*, 2012, pp. 13–23.

[9] E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt, "Quasi-partitioned scheduling: optimality and adaptation in multiprocessor real-time systems," *Real-Time Systems*, vol. 52, no. 5, pp. 566–597, 2016.

[10] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *RTSS*, 2011, pp. 104–115.

[11] M. Cirinei and T. P. Baker, "EDZL scheduling analysis," in *ECRTS*, 2007, pp. 9–18.

[12] R. I. Davis and A. Burns, "FPZL schedulability analysis," in *RTAS*, 2011, pp. 245–256.

[13] J. Lee, A. Easwaran, and I. Shin, "LLF schedulability analysis on multiprocessor platforms," in *RTSS*, 2010, pp. 25–36.

[14] ——, "Maximizing contention-free executions in multiprocessor scheduling," in *RTAS*, 2011, pp. 235–244.

[15] H. S. Chwa, H. Back, S. Chen, J. Lee, A. Easwaran, I. Shin, and I. Lee, "Extending task-level to job-level fixed priority assignment and schedulability analysis using pseudo-deadlines," in *RTSS*, 2012, pp. 51–62.

[16] H. Back, H. S. Chwa, and I. Shin, "Schedulability analysis and priority assignment for global job-level fixed-priority multiprocessro scheduling," in *RTAS*, 2012, pp. 297–306.

[17] H. S. Chwa, H. Back, J. Lee, K. M. Phan, and I. Shin, "Capturing urgency and parallelism using quasi-deadlines for real-time multiprocessor scheduling," *Journal of Systems and Software*, vol. 101, pp. 15–29, 2015.

[18] P. Holman and J. H. Anderson, "Adapting pfair scheduling for symmetric multiprocessors," *Journal of Embedded Computing*, vol. 1, pp. 543–564, 2005.

[19] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science*, vol. 6, no. 1, pp. 1–12, 1959.

[20] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[21] W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, pp. 177–185, 1974.

[22] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS*, 2007, pp. 149–160.

[23] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.

[24] R. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *RTSS*, 2009, pp. 398–409.

[25] H. Baek, H. S. Chwa, and J. Lee, "Supplement of "Beyond Implicit-Deadline Optimality: A Multiprocessor Scheduling Framework for Constrained-Deadline Tasks"." [Online]. Available: http://rtcl.skku.edu/papers/RTSS17-supplement.pdf

[26] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[27] T. P. Baker, "Comparison of empirical success rates of global vs. partitioned fixed-priority EDF scheduling for hard real-time," Department of Computer Science, Florida State University, Tallahassee, Tech. Rep. TR–050601, 2005.

[28] T. P. Baker and M. Cirinei, "A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks," in *RTSS*, 2006, pp. 178–190.