F2VD: Fluid Rates to Virtual Deadlines for Precise Mixed-Criticality Scheduling on a Varying-Speed Processor

Kecheng Yang
yangk@txstate.edu
Texas State University

Ashikahmed Bhuiyan
ashik@knights.ucf.edu
University of Central Florida

Zhishan Guo
zsguo@ucf.edu
University of Central Florida

ABSTRACT

Increasingly complex and integrated systems design has led to more timing uncertainty, which may result in pessimism in time-sensitive system design and analysis. To mitigate such pessimism, mixed-criticality (MC) design for real-time systems has been proposed, where highly critical tasks, often with extremely pessimistic execution time estimates, can share the processor with less critical ones in a manner that the latter is sacrificed, completely or partially, to guarantee temporal correctness to the former, when the extremely pessimistic scenario does happen. In contrast to such sacrifice of tasks, the precise MC scheduling model has recently been investigated, where all tasks, including less critical ones, must fully complete their execution in all circumstances. Meanwhile, the processor may operate at a degraded speed when the tasks' runtime behaviors are far from the extreme pessimistic estimates and would recover to the full processing speed once the extremely pessimistic scenario does happen.

This paper presents a generalized fluid-scheduling-based solution to this problem, where feasible fluid-scheduling rates for each task are derived from an optimization problem. Furthermore, this paper proposes a novel algorithm F2VD for setting virtual deadlines from any feasible fluid rates, such that any fluid-scheduling-based solution can be converted to a deadline-based scheduling approach with no schedulability loss, where the latter is generally considered much more practical and easier to implement. Experimental studies based on randomly generated task sets are conducted to verify the theoretical results as well as the effectiveness of the proposed algorithms.

CCS CONCEPTS

• **Computer systems organization** → *Real-time system architecture.*

KEYWORDS

EDF with virtual deadlines, fluid scheduling, precise mixed-criticality, varying-speed platform.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415716

ACM Reference Format:
Kecheng Yang, Ashikahmed Bhuiyan, and Zhishan Guo. 2020. F2VD: Fluid Rates to Virtual Deadlines for Precise Mixed-Criticality Scheduling on a Varying-Speed Processor. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415716

1 INTRODUCTION

There is a trend in the design of computer systems in suiting general-purpose computing and average run-time efficiency. When such advanced design is applied for real-time computing purposes, it often results in more significant variations of temporal behaviors. For example, the worst-case execution time can be of the much larger magnitude of the general average-case performance. However, worst-case timing guarantees must be provided in a real-time system, and pessimistic assumptions on all uncertain behaviors are made during certification processes. As a result, the gap between general performance and the worst-case one is proliferating, and a vast amount of computing resources can be wasted under normal circumstances. This gap can be enormous for functionalities of high criticality (importance level), where extremely pessimistic worst-case execution time (WCET) estimates are often applied.

Mixed-Criticality (MC) design for real-time systems has been proposed, where functionalities of different levels of criticality are facilitated onto a common computing framework to reduce resource costs. Since Vestal's pioneering work [36], the real-time systems community has spent much effort in understanding, expanding, and analyzing MC design (refer to [15] for an up-to-date review). However, criticisms have raised for most of these scheduling strategies. These criticisms are centered around: (i) guaranteeing resources to all the tasks under less pessimistic behaviors of the system and (ii) protecting *only* more important (HI-criticality) tasks under more pessimistic behaviors, e.g., in the event of task overrun.

Recent works proposed degraded services to the less critical tasks, and the released resources are used to guarantee HI-criticality task deadlines [14, 26]. The imprecise mixed-criticality system (IMC) model results in relatively short WCET, and thus allows graceful degradation of LO-criticality tasks in HI-criticality mode [5, 14, 31]. However, they do not fully address the recent criticism [17, 18] regarding how MC models do not reflect current practice with regard to ensuring robustness in safety-critical systems[1].

The criticism mainly arises from the impreciseness of the execution of less critical functionalities, as a result of their execution budget reduction in HI-mode. A recent paper [12] suggested a novel

[1]There is no criticism regarding the suitability of MC theory for verification aspects, which is orthogonal to run-time robustness [2].

precise MC scheduling framework, where *all* tasks receive full execution budget under all circumstances. By following the MC-Fluid framework [6, 28] and leveraging the dynamic voltage and frequency scaling (DVFS) technique, the processor execution speed is minimized under normal circumstances, while resumed to full speed when there is task overrun (characterized as a mode switch). It provides closed-form equations for per-mode (fluid) execution rate assignment for each task by restricting the inter-mode ratio of rates to a common variable across all HI-tasks.

Recent works [7, 8, 27, 32] showed that MC scheduling pairing with varying-speed processors could result in new and challenging problems. However, there is an uncertainty that comes from the varying speed, and LO-tasks must be dropped or reduced to *passively* react to an unpredictable speed degradation during runtime. In contrast, our work (as well as [12]) considers the circumstances where the operating system *actively* adjusts the processor speed. In particular, the processor begins with a degraded speed but will recover to the full speed (to meet all deadlines) when any HI-task shows abnormal worst-case behavior.

Contributions. The main contributions of this work are fourfold.

- We relax the common ratio restriction in [12], and provides an optimal fluid-scheduling rates assignment framework for precise MC scheduling on a varying-speed processor.
- We propose a novel algorithm F2VD, which can transpose any fluid execution rates assignment to a virtual-deadline setting. F2VD converts the theoretical fluid scheduling to the priority-driven scheduling that is practical to implement.
- We also show that such transformation incurs no schedulability loss. This implies that scheduling by virtual deadlines *dominates* that by fluid rates, for the problem of precise MC scheduling on a varying-speed processor. We further demonstrate such dominance is strict, *i.e.*, not an equivalence, by identifying a task set that is schedulable via EDF with virtual deadlines but not feasible with any fluid rates assignment.
- Experimental studies based on randomly generated task sets are conducted to verify the theoretical results as well as the effectiveness of the proposed algorithms.

Organization. In the rest of the paper, we describe the system model and the problem considered herein (Sec. 2), present a fluid scheduling approach to solve the problem (Sec. 3), present algorithm F2VD and prove its correctness (Sec. 4), discuss a dominance relationship implied by F2VD (Sec. 5), experimentally evaluate F2VD (Sec. 6), discuss related work (Sec. 7), and conclude (Sec. 8),

2 SYSTEM MODEL AND THE PROBLEM

We consider a set of n implicit-deadline sporadic MC tasks $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where each task is specified by a 3-tuple as $\tau_i = (C_i^{\text{LO}}, C_i^{\text{HI}}, T_i)$. Each task τ_i releases a (potentially infinite) sequence of *jobs* with a minimum release separation of T_i time units and every job has an absolute deadline T_i time units after its release. The worst-case execution requirement, which is defined by the worst-case execution time on a unit-speed processor, of task τ_i is estimated at two criticality levels: a LO-criticality estimate C_i^{LO} and HI-criticality estimate C_i^{HI}, where it is assumed that $\forall i, C_i^{\text{LO}} \leq C_i^{\text{HI}}$. Besides, C_i^{LO} (C_i^{HI}, respectively) is also the execution requirement budgets of task τ_i in the LO (HI, respectively)-*mode*, which is to be described later. In particular, $C_i^{\text{LO}} < C_i^{\text{HI}}$ indicates that task τ_i is a HI-criticality task (HI-task) that may trigger a system mode switch, whereas $C_i^{\text{LO}} = C_i^{\text{HI}}$ indicates that task τ_i is a LO-criticality task (LO-task) that cannot trigger any system mode switch. Furthermore, the j^{th} job of task τ_i is denoted by $J_{i,j}$, whose release time and absolute deadline are denoted by $r_{i,j}$ and $d_{i,j}$, respectively.

We consider the problem of scheduling the set of tasks τ on a single processor whose executing speed may vary. The processor begins with a degraded speed $\rho < 1.0$, which indicates that any workload being executed under this speed for t time units is equivalent to that under a unit-speed processor for $\rho \times t$ time units. During runtime, the amount of workload completed for each job is being monitored. If any job $J_{i,j}$ has done C_i^{LO} workload under the degraded processing speed ρ (thus receiving a cumulative actual execution time of C_i^{LO}/ρ units) but still requires further execution, the system is immediately notified, and the processor starts to perform its full speed 1.0 right from that moment. We also call this moment as the time instant of *mode switch*, from the LO-mode (where the processor speed is ρ) to the HI-mode (where the processor speed becomes 1.0). The system can recover to the LO-mode once the processor becomes idle.

Note that, in contrast to a majority of existing work on MC scheduling, no task is *entirely or partially dropped* upon a mode switch, and every job meets its absolute deadline at any system mode. The difference between the two execution requirement budgets upon the mode switch, i.e., $C_i^{\text{HI}} - C_i^{\text{LO}}$, is compensated by the speed upgrade. Furthermore, any job $J_{i,j}$ that has done C_i^{HI} workload but still not completed yet, is considered as *erroneous* and would be terminated then. That is, only HI-tasks, for which $C_i^{\text{LO}} < C_i^{\text{HI}}$, could trigger a mode switch.

3 FLUID SCHEDULING APPROACH

One idealistic approach to solve this scheduling problem is to adapt the fluid scheduling model. In this model, each task receives a fraction of a single processor, so that multiple tasks may progress at specific rates simultaneously even on a single processor, as long as the summation of the allocated rates does not exceed the processor speed at any time instant.

In this paper, we focus on the *dual-rate fluid* scheduling,[2] where each task τ_i is assigned two constant executing rates in LO- and HI-modes, denoted by θ_i^{LO} and θ_i^{HI}, respectively. A set of rates assignment pairs $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ is *feasible* if and only if

$$\sum_i \theta_i^{\text{LO}} \leq \rho, \tag{1}$$

$$\sum_i \theta_i^{\text{HI}} \leq 1.0, \tag{2}$$

where all deadline are guaranteed to meet in both LO- and HI-modes.

Note that, the algorithm proposed in [12] is also based on fluid scheduling, and focused on rates assignments such that $\forall i, \theta_i^{\text{LO}}/\theta_i^{\text{HI}}$ are identical. Therefore, our focus in this paper—the set of task

[2]The conventional fluid scheduling assumes a single constant rate for each task, whereas two rates, i.e., one rate change for each task, have been proposed and considered in the context of MC scheduling [6, 28]. Note that fluid scheduling with no restriction on the number of rate changes can be too general and pointless. For example, *any* actual schedule can be viewed as a fluid schedule where the rate for each task is switching between 0 and 1.0.

systems that are schedulable under dual-rate fluid scheduling with *some* rates assignment—is more general and is a super set of the set of task systems that are schedulable by the algorithm in [12].

We next discuss how to form up the constraints on rates assignment set $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^{n}$ in order to guarantee all deadlines to be met in both HI- and LO-modes. First, for each task τ_i, all its jobs that both are released and have a deadline in LO-mode must meet their deadlines if and only if

$$\frac{C_i^{\text{LO}}}{\theta_i^{\text{LO}}} \leq T_i, \quad \forall i : 1 \leq i \leq n. \tag{3}$$

Second, for each task τ_i, all its jobs that both are released and have a deadline in HI-mode must meet their deadlines if and only if

$$\frac{C_i^{\text{HI}}}{\theta_i^{\text{HI}}} \leq T_i, \quad \forall i : 1 \leq i \leq n. \tag{4}$$

Furthermore, the general idea of dual-rate fluid scheduling does not necessarily dictate the relationship between θ_i^{LO} and θ_i^{HI}. Nonetheless, Theorem 3.1 shows that we can restrict out attention to *non-decreasing* dual-rate fluid scheduling only, where it is required that

$$\theta_i^{\text{LO}} \leq \theta_i^{\text{HI}}, \quad \forall i : 1 \leq i \leq n. \tag{5}$$

THEOREM 3.1. *Any task system that is schedulable under dual-rate fluid scheduling must also be schedulable under* ***non-decreasing*** *dual-rate fluid scheduling.*

PROOF. To see this theorem is true, we need to notice that if a task system is schedulable under some dual-rate fluid scheduling where $\theta_i^{\text{LO}} > \theta_i^{\text{HI}}$ for some i, then this system must still be schedulable when we assign $\theta_i^{\text{LO}} \leftarrow \theta_i^{\text{HI}}$. This is because the total rate constraints (1) and (2) cannot be violated by this assignment that reduces θ_i^{LO}, and Constraint (4) implies that a single constant execution rate of θ_i^{HI} in both LO- and HI-modes for task τ_i (which is the scenario for τ_i after the reducing) guarantees all its deadlines met regardless whether and where the mode switches, because $C_i^{\text{LO}} \leq C_i^{\text{HI}}$. ■

Therefore, under the **non-decreasing** dual-rate assumption, for each task τ_i, its job (if any) that is released in LO-mode but also executes in HI-mode must meet its deadline (which is in HI-mode) if and only if

$$\frac{C_i^{\text{LO}}}{\theta_i^{\text{LO}}} + \frac{C_i^{\text{HI}} - C_i^{\text{LO}}}{\theta_i^{\text{HI}}} \leq T_i, \quad \forall i : 1 \leq i \leq n. \tag{6}$$

This is sufficient because $C_i^{\text{LO}}/\theta_i^{\text{LO}}$ time units after its release is the latest time for the mode switch to be triggered and the mode switch is triggered earlier by any other job, the deadline must also be met by (5). This is necessary because any HI-task can be the one that triggers the mode switch and executes up to exact C_i^{HI} budget. We can generally claim $\forall i$ in (6) because if τ_i is a LO-task (*i.e.*, $C_i^{\text{LO}} = C_i^{\text{HI}}$), then (6) reduces to (3).

Note that, each of the above equations is "if an only if" and all three possible situations of a job (entirely in LO or HI-mode, and across the mode switch time instant) have been exhausted. Therefore, Constraints (1)—(6) are a necessary and sufficient condition for the MC task system on the varying-speed processor to be schedulable by *any* two-rate fluid scheduling.

With Constraints (1)—(6) and the additional set of non-negative rate assignment constraints ($\theta_i^{\text{LO}} \geq 0, \forall i$), we have an optimization problem with linear and linear fractional inequality constraints, where θ_i^{LO} and θ_i^{HI} are variables and all others are problem input constants. Thus, in total, there are $O(n)$ variables, $O(n)$ linear constraints, and $O(n)$ linear fractional constraints, where n is the number of tasks. Efficient numerical solvers (such as `fmincon` [19] or `CVX` [20] in Matlab) can be used to find a feasible solution.

Objective function. We have shown that the MC task system on the varying-speed processor to be schedulable under two-rate fluid scheduling if and only if a feasible solution exists for Constraints (1)—(6). Therefore, an objective function is not necessary for determining schedulability for a given degraded speed ρ. Thus, we can simply choose a trivial objective function "minimize 1" when applying the solver. On the other hand, if the degraded speed is not given as part of the system setting, then we can replace Constraint (1) by the following optimization objective:

$$\min \sum_i \theta_i^{\text{LO}}. \tag{7}$$

4 ALGORITHM F2VD

Although the considered problem can be solved by the dual-rate fluid scheduling as presented in Sec. 3, fluid scheduling is often considered as an idealistic model for analysis but not practical to implement due to potentially overwhelming runtime overheads for frequent context switches.

In this section, we present our proposed algorithm F2VD [3], and prove its correctness. F2VD is a deadline-based priority-driven scheduling algorithm, which can be implemented in a similar manner to the conventional EDF. In particular, inspired by the widely-received MC scheduling algorithm EDF-VD [4], we focus on the virtual-deadline based scheduling with potentially different virtual-deadline settings [16] than the original EDF-VD [3]. Under virtual-deadline based scheduling, in addition to the actual relative deadline D_i, each task τ_i is assigned a relative virtual deadline D_i', *i.e.*, every job of task τ_i has an absolute actual (virtual, respectively) deadline D_i (D_i', respectively) time units after its release. In LO-mode, jobs are scheduled by their *virtual* deadlines, whereas they switch to be scheduled by their *actual* deadlines upon a mode switch to HI-mode. In LO-mode (HI-mode, respectively), the earlier the virtual (actual, respectively) deadline, the higher the priority.

Specifically, algorithm F2VD can be described as follows.

- Obtain a set fluid rate pairs $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^{n}$ that is a *feasible solution* for Constraints (1)—(6) as described in Sec. 3.
- Calculate the relative virtual deadline D_i' for each task τ_i by

$$D_i' = \frac{C_i^{\text{LO}}}{\theta_i^{\text{LO}}}. \tag{8}$$

- The system begins with LO-mode, where the processor is operating at the degraded speed ρ and jobs are scheduled by EDF according to their *virtual* deadlines.
- Once any job $J_{i,j}$ has received C_i^{LO} execution budget (*i.e.*, has executed for C_i^{LO}/ρ time units in LO-mode) but has not completed yet, the system immediately switch to HI-mode, where the processor recovers to its full speed, 1.0, and jobs are scheduled by EDF according to their *actual* deadlines.

[3]"F2VD" stands for "fluid to virtual deadlines"

- If the processor becomes idle (*i.e.*, all released workload has been completed) in HI-mode, then the system switches back to LO-mode.

Finally, the correctness of F2VD, *i.e.*, all deadlines are guaranteed to meet under F2VD, is proven by Theorem 4.1.

THEOREM 4.1. *If a feasible solution* $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ *that satisfies Constraints* (1)—(6) *can be found, then F2VD guarantees all deadlines met in both* LO- *and* HI-*modes.*

PROOF. Because $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ is a feasible solution, by (1) and (8), it is clear that all *virtual* deadlines in LO-mode must be met by the classic utilization based schedulability test of preemptive EDF [30]. Furthermore, by (3) and (8), we have $D_i' \leq T_i$ for all i, *i.e.*, every *virtual* deadline is no later than its corresponding *actual* deadline. Thus, all *actual* deadlines in LO-mode must be met as well.

In the rest of the proof, we focus on the actual deadlines in HI-mode. We suppose that the system does switch to HI-mode, and some actual deadlines in HI-mode are missed and then will show a contradiction at the end. With this supposition, we let t^* denote the mode-switch time instant and let t_d denote the first missed deadline, which must be in HI-mode since all deadlines in LO-mode are met (as shown above). We then restrict our attention to a corresponding *minimal job set instance* where we remove any workload that does not impact, directly or indirectly, the scheduling of the job with its deadline at t_d. In other words,

> Any further dropping or execution-requirement reducing for any job in the minimal job set instance must have an impact on the scheduling of the job with its deadline at t_d.

Without loss of generality, we define the earliest release time of any job in the minimal job set instance as time 0, which is also as known as *the latest idle time instant before* t_d in the literature. Therefore, in the minimal job set instance, we have

(M1) time interval $[0, t_d)$ must be a busy time interval;
(M2) all jobs that are released after t^* and have actual deadlines after t_d must have been removed; and
(M3) all jobs that have virtual deadlines (and therefore also actual deadlines) after t_d must have been removed.

If $t^* \leq 0$, *i.e.*, the latest idle time instant before t_d is in HI-mode, it is straightforward that the deadline at t_d cannot be missed, given that both (4) and (2) hold and $[0, t_d)$ is busy by **(M1)**. Therefore, in the rest of this proof and also the rest of this section, we focus on the case that

$$0 < t^* < t_d. \tag{9}$$

By **(M1)**, missing the deadline at t_d implies that the total amount of workload in this minimal job set instance $W > \rho \times t^* + 1.0 \times (t_d - t^*)$. However, as proven in the following Lemma 4.2, for each task τ_i, such workload $W_i \leq \theta_i^{\text{LO}} \times t^* + \theta_i^{\text{HI}} \times (t_d - t^*)$. Therefore,

$$\begin{aligned} W = \sum_i W_i &\leq \sum_i \theta_i^{\text{LO}} \times t^* + \sum_i \theta_i^{\text{HI}} \times (t_d - t^*) \\ &\leq \{\text{by (3) and (4)}\} \\ &\quad \rho \times t^* + 1.0 \times (t_d - t^*), \end{aligned}$$

which contradicts $W > \rho \times t^* + 1.0 \times (t_d - t^*)$. Therefore, the supposition of missing deadlines is not true and the theorem follows. ∎

The following lemmas are for supporting the last part in the proof of Theorem 4.1, in a top-down structuring fashion. Therefore, please note that they are all reasoning with respect to the *minimal job set instance* in the proof of Theorem 4.1 and (9) holds. Also, as assumed in Theorem 4.1 that $\{(\theta_i^{\text{LO}}, \theta_i^{\text{HI}})\}_{i=1}^n$ is a feasible solution, Constraints (1)—(6) must hold.

LEMMA 4.2. *For each task* τ_i*, the total amount of workload in the minimal job set instance is upper bounded by*

$$W_i \leq \theta_i^{\text{LO}} \times t^* + \theta_i^{\text{HI}} \times (t_d - t^*),$$

PROOF. This lemma is proven by discussing the following three exhaustive cases about the job of τ_i that is released before t^* and has an absolute deadline after t^*.

- If τ_i does not have a job $J_{i,j}$ such that $r_{i,j} < t^*$ and $d_{i,j} > t^*$, then this lemma is established by Lemma 4.3 proven later.
- If τ_i has a job $J_{i,j}$ such that $r_{i,j} < t^*$, $d_{i,j} > t^*$, and $d_{i,j} \leq t_d$, then this lemma is established by Lemma 4.4 proven later.
- If τ_i has a job $J_{i,j}$ such that $r_{i,j} < t^*$, $d_{i,j} > t^*$, and $d_{i,j} > t_d$, then this lemma is established by Lemma 4.5 proven later.

∎

LEMMA 4.3. *If* τ_i *does not have a job* $J_{i,j}$ *such that* $r_{i,j} < t^*$ *and* $d_{i,j} > t^*$*, then* $W_i \leq \theta_i^{\text{LO}} \times t^* + \theta_i^{\text{HI}} \times (t_d - t^*)$,

PROOF. In this lemma, all released jobs (of τ_i) before t^* must have their actual deadlines (and therefore their virtual deadlines) before t^*. Also, none of them can have workload beyond their LO-criticality WCET, as t^* is the mode switch time instant. Therefore, the workload from jobs released before t^* is at most

$$\begin{aligned} \left\lfloor \frac{t^*}{T_i} \right\rfloor \times C_i^{\text{LO}} &\leq \frac{t^*}{T_i} \times C_i^{\text{LO}} \\ &\leq \{\text{by (3)}\} \\ &\quad \theta_i^{\text{LO}} \times t^*. \end{aligned}$$

On the other hand, by **(M2)**, the workload from jobs released at or after t^* is at most

$$\begin{aligned} \left\lceil \frac{t_d - t^*}{T_i} \right\rceil \times C_i^{\text{HI}} &\leq \frac{t_d - t^*}{T_i} \times C_i^{\text{HI}} \\ &\leq \{\text{by (4)}\} \\ &\quad \theta_i^{\text{HI}} \times (t_d - t^*). \end{aligned}$$

Thus, the lemma follows. ∎

LEMMA 4.4. *If* τ_i *has a job* $J_{i,j}$ *such that* $r_{i,j} < t^*$, $d_{i,j} > t^*$*, and* $d_{i,j} \leq t_d$*, then* $W_i \leq \theta_i^{\text{LO}} \times t^* + \theta_i^{\text{HI}} \times (t_d - t^*)$,

PROOF. Because $r_{i,j} < t^* < d_{i,j} < t_d$ in this lemma, the workload by τ_i from its jobs released before $r_{i,j}$ is at most

$$\begin{aligned} \left\lfloor \frac{r_{i,j}}{T_i} \right\rfloor \times C_i^{\text{LO}} &\leq \frac{r_{i,j}}{T_i} \times C_i^{\text{LO}} \\ &\leq \{\text{by (3)}\} \\ &\quad \theta_i^{\text{LO}} \times r_{i,j}, \end{aligned}$$

and the workload by τ_i from its jobs released after $d_{i,j}$ is at most

$$\begin{aligned}\left\lfloor \frac{t_d - d_{i,j}}{T_i} \right\rfloor \times C_i^{\text{HI}} &\le \frac{t_d - d_{i,j}}{T_i} \times C_i^{\text{HI}} \\ &\le \{\text{by (4)}\} \\ &\quad \theta_i^{\text{HI}} \times (t_d - d_{i,j}).\end{aligned}$$

In the rest of the proof, we focus on proving that the workload by $J_{i,j}$, denoted by $W_{i,j}$, must satisfy that

$$W_{i,j} \le \theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - t^*), \tag{10}$$

by which the lemma follows. We prove (10) by discussing two cases for the absolute virtual deadline of $J_{i,j}$, denoted by $d'_{i,j}$.

Case1: $d'_{i,j} < t^*$**.** In this case, $J_{i,j}$ must have completed by $d'_{i,j}$ by having done no more than C_i^{LO} workload; otherwise the mode switch must be triggered at or before $d'_{i,j} < t^*$, which contradicts the definition of t^*. That is,

$$W_{i,j} \le C_i^{\text{LO}}.$$

On the other hand,

$$\begin{aligned}&\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - t^*) \\ \ge\ &\{\text{because } \theta_i^{\text{LO}} \le \theta_i^{\text{HI}}\} \\ &\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{LO}} \times (d_{i,j} - t^*) \\ =\ &\theta_i^{\text{LO}} \times (d_{i,j} - r_{i,j}) = \theta_i^{\text{LO}} \times T_i \\ \ge\ &\{\text{by (3)}\} \\ &C_i^{\text{LO}}.\end{aligned}$$

Thus, (10) holds in this case.

Case2: $d'_{i,j} \ge t^*$**.** Because this system is schedulable under the two-rate fluid scheduling, it must hold that

$$\theta_i^{\text{LO}} \times (d'_{i,j} - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - d'_{i,j}) \ge C_i^{\text{HI}}. \tag{11}$$

Intuitively, it reflects the scenario that $J_{i,j}$ triggers a mode switch in the fluid schedule. Alternatively, it is also implied by the necessary condition (6) as

$$\begin{aligned}&\theta_i^{\text{LO}} \times (d'_{i,j} - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - d'_{i,j}) \\ =\ &\theta_i^{\text{LO}} \times D'_i + \theta_i^{\text{HI}} \times (T_i - D'_i) \\ \ge\ &\{\text{by (8)}\} \\ &C_i^{\text{LO}} + \theta_i^{\text{HI}} \times T_i - \frac{\theta_i^{\text{HI}} \times C_i^{\text{LO}}}{\theta_i^{\text{LO}}} \\ \ge\ &\{\text{by (6)}\} \\ &C_i^{\text{HI}},\end{aligned}$$

Furthermore, because $\theta_i^{\text{LO}} \le \theta_i^{\text{HI}}$ and $d'_{i,j} \ge t^*$ in this case, we have

$$\theta_i^{\text{LO}} \times (d'_{i,j} - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - d'_{i,j}) \le \theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - t^*). \tag{12}$$

By (11) and (12),

$$\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (d_{i,j} - t^*) \ge C_i^{\text{HI}}.$$

Meanwhile, because $J_{i,j}$ is a single job of τ_i, $W_{i,j} \le C_i^{\text{HI}}$ must always hold. Thus, (10) holds in this case as well. ■

Lemma 4.5. *If τ_i has a job $J_{i,j}$ such that $r_{i,j} < t^*$, $d_{i,j} > t^*$, and $d_{i,j} > t_d$, then $W_i \le \theta_i^{\text{LO}} \times t^* + \theta_i^{\text{HI}} \times (t_d - t^*)$,*

Proof. Because $r_{i,j} < t^*$ in this lemma, the workload by τ_i from its jobs released before $r_{i,j}$ is at most

$$\begin{aligned}\left\lfloor \frac{r_{i,j}}{T_i} \right\rfloor \times C_i^{\text{LO}} &\le \frac{r_{i,j}}{T_i} \times C_i^{\text{LO}} \\ &\le \{\text{by (3)}\} \\ &\quad \theta_i^{\text{LO}} \times r_{i,j},\end{aligned}$$

Because $d_{i,j} > t_d$ and by **(M2)**, any workload by τ_i from its jobs released after $d_{i,j}$ must have been removed in the minimal job set instance. Therefore,

$$W_i \le \theta_i^{\text{LO}} \times r_{i,j} + W_{i,j}, \tag{13}$$

where $W_{i,j}$ denotes the workload from $J_{i,j}$ in the minimal job set instance (*i.e.*, the amount of workload by $J_{i,j}$ that can have an impact on the scheduling of the job with its deadline at t_d).

Because $d_{i,j} > t_d$, $J_{i,j}$ cannot have any workload in HI-mode; otherwise, such workload should have been removed in the minimal job set instance. On the other hand, the workload from $J_{i,j}$ in LO-mode cannot exceed C_i^{LO}, *i.e.*,

$$W_{i,j} \le C_i^{\text{LO}}; \tag{14}$$

otherwise, it should have been an earlier mode switch before t^*, which contradicts the definitions of t^*.

Furthermore, by **(M3)**, the virtual deadline of $J_{i,j}$, denoted by $d'_{i,j}$, must be no later than t_d, *i.e.*, $d'_{i,j} \le t_d$. Therefore,

$$\begin{aligned}&\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (t_d - t^*) \\ \ge\ &\{\text{because } d'_{i,j} \le t_d \text{ and } \theta_i^{\text{HI}} \ge \theta_i^{\text{LO}}\} \\ &\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{LO}} \times (d'_{i,j} - t^*) \\ =\ &\theta_i^{\text{LO}} \times (d'_{i,j} - r_{i,j}) = \theta_i^{\text{LO}} \times D'_i,\end{aligned}$$

which, by (8), implies

$$\theta_i^{\text{LO}} \times (t^* - r_{i,j}) + \theta_i^{\text{HI}} \times (t_d - t^*) \ge C_i^{\text{LO}}. \tag{15}$$

By (13), (14), and (15), the lemma follows. ■

An interesting observation. For simplicity, we denote an arbitrary job by J_k and denote its virtual and actual absolute deadlines by d'_k and d_k. Then, **(M3)** (that jobs with virtual deadlines after t_d is removed) is based on the following fact **(C1)**.

(C1) For any two jobs J_1 and J_2, if $d'_1 \le d_1 < d'_2 \le d_2$, then J_2 cannot have any impact on the execution of J_1.

Although the **(M3)** resulted from the above fact is already sufficient for completing our proofs, one might wonder that: could we make the following stronger claim?

(C2) For any two jobs J_1 and J_2, if $d'_1 < d'_2$ and $d_1 < d_2$, then J_2 cannot have any impact on the execution of J_1.

At a glance, **(C2)** seems to make sense, because J_1 would have strictly higher priority than J_2 in both LO- and HI-modes. However, **(C2)** is not true due to possible *indirect* impacts under the MC (with potential mode switch) setting. Let us consider the potential existence of another job J_3 such that $d'_1 < d'_2 < d'_3 < d_3 < d_1 < d_2$, which satisfies that $d'_1 < d'_2$ and $d_1 < d_2$. Then, because $d'_2 < d'_3$, J_2 may have an impact on the scheduling of J_3 in LO-mode, which inherently may have an impact on the scheduling of J_3 in HI-mode. Furthermore, in HI-mode, the scheduling of J_3 clearly may have an impact on the scheduling of J_1 due to $d_3 < d_1$. That is, J_2 might indirectly impact the scheduling of J_1, even if $d'_1 < d'_2$ and $d_1 < d_2$. Thus, **(C2)** is not true. In contrast, the condition in **(C1)** is sufficient to eliminate such indirect impacts.

5 DISCUSSIONS ABOUT DOMINANCE

Given algorithm F2VD and its proven correctness, it, in fact, leads to a dominance relationship between two *families* of scheduling algorithms on the precise MC scheduling problem on a varying-speed processor, namely the dual-rate fluid scheduling family and the EDF-VD family.

A system is deemed schedulable by the dual-rate fluid scheduling family if and only if there exists a dual-rate assignment so that all deadlines in both LO- and HI-modes are guaranteed met in all circumstances — this is exactly what has been presented in Sec. 3. On the other hand, a system is deemed schedulable by the EDF-VD family if and only if there exists a virtual-deadline assignment so that all deadlines in both LO- and HI-modes are guaranteed met in all circumstances — this is more general than being schedulable by F2VD because the virtual-deadline assignment derived by F2VD is just one such assignment and EDF-VD family includes all other possible virtual-deadline assignments as well. Therefore, F2VD and its correctness directly implies that the EDF-VD family dominates the dual-rate fluid scheduling family with respect to schedulability, as summarized in the following corollary.

Corollary 5.1. *Any task system that is schedulable by the two-rate fluid scheduling family is also schedulable by the EDF-VD family.*

Proof. It directly follows from Theorem 4.1 and the facts that Constraints (1)—(6) are a necessary and sufficient condition for the schedulability by *any* two-rate fluid scheduling as discussed in Sec. 3 and F2VD always results in a subset of the EDF-VD family. ■

Furthermore, the following lemma shows that this is a strict dominance, instead of a potential equivalence, relationship between the two families.

Lemma 5.2. *There exists a system that is schedulable by the EDF-VD family under* ***some*** *virtual-deadline setting but is not schedulable by dual-rate fluid scheduling family under* ***any*** *rates assignment.*

Proof. This lemma is a “there exists” statement and there can be proven by the following specific example.

We consider a varying-speed processor of degraded speed 0.5 and full speed 1.0 and an MC task set consisting of just two implicit-deadline HI-tasks τ_1 and τ_2 that have the parameters summarized in the table in Fig. 1. We also assume the tasks are synchronous (*i.e.*, both tasks starting job releases at time 0) and periodic (*i.e.*, job release separation is always *exact* T_i time units).

For this system, a virtual-deadline assignment of $D'_1 = 2, D'_2 = 6$ would guarantee all deadlines to be met in any circumstance. Fig. 2 shows the two edge cases. Namely, Fig. 2(a) shows the scenario where τ_1 does not trigger the mode switch but τ_2 does later, and Fig. 2(b) shows the scenario where τ_1 triggers the mode switch. Moreover, if neither task triggers the mode switch, it is clear that both of them must complete by their respective virtual deadlines — we omit a figure for this case. Note that a similar pattern of one of the three cases above would happen every 8-time units, i.e., the typical period shared by the two synchronous, periodic tasks.

Task	C_i^{LO}	C_i^{HI}	T_i
τ_1	1	3	8
τ_2	2	4	8

Figure 1: Example task system in Lemma 5.2.

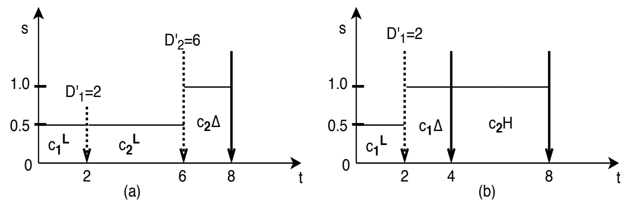

Figure 2: Illustration for different scenarios of the schedule under the virtual-deadline based scheduling.

On the other hand, we would argue that this system cannot be schedulable by dual-rate fluid scheduling under *any* rates assignment. We consider the scenario that the first jobs of both tasks will need to execute up to their HI-criticality budgets, *i.e.*, $C_1/^{\text{HI}} = 3$ and $C_2/^{\text{HI}} = 4$. Note that, the system cannot be clairvoyant to know this from the beginning but still needs to start with the degraded speed and only commit a mode switch when either of the two tasks exhausts their LO-criticality execution budget. We discuss the following two exhaustive cases under any dual-rate fluid scheduling.

- **(i)** τ_1 exhausts the amount of C_1^{LO} execution budget **no later** than τ_2 exhausts the amount of C_2^{LO} execution budget.
- **(ii)** τ_1 exhausts the amount of C_1^{LO} execution budget **later** than τ_2 exhausts the amount of C_2^{LO} execution budget.

In Case **(i)**, it is clear that τ_2 must receive positive (non-zero) execution rate in LO-mode, *i.e.*, $\theta_2^L > 0$; otherwise, it will surely miss its deadlines if no mode switch ever happens. Therefore, $\theta_1^L < 0.5$ because $\theta_1^L + \theta_2^L \leq 0.5$, which is the degraded processor speed in LO-mode. This means that the time instant t^* for the mode switch by task τ_1 must be $t^* \geq C_1^{\text{LO}}/\theta_1^L > 2$. As a result, the total execution budget received by both tasks in the first period is $0.5t^* + (8 - t^*) < 7$. Because total execution requirement by the two tasks can be $C_1^{\text{HI}} + C_2^{\text{HI}} = 7$ in the window $[0, 8)$, at least one of the tasks must miss a deadline at time 8, no matter how the fluid rates are assigned.

In Case **(ii)**, because the degraded speed in LO-mode is 0.5. The time instant t^* for the mode switch by task τ_2 must be $t^* \geq C_2^{\text{LO}}/0.5 = 4$. As a result, the total execution budget received by both tasks in the first period is $0.5t^* + (8 - t^*) \leq 6 < 7$. For the same reason as in Case **(i)**, at least one of the tasks must miss a deadline at time 8, no matter how the fluid rates are assigned. ■

Thus, we can conclude the following theorem.

Theorem 5.3. *For the precise MC scheduling problem on a varying-speed processor, the EDF-VD family strictly dominates the two-rate fluid scheduling family with respect to schedulability.*

Proof. It follows from Corollary 5.1 and Lemma 5.2. ■

This result is very inspiring and quite counter-intuitive. In many scheduling problems, especially when implicit-deadline tasks are considered, fluid scheduling often results in the best schedulability in theory, though it can be impractical to implement. For example,

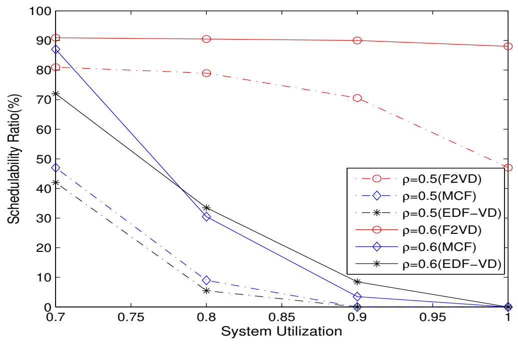

Figure 3: Comparison of schedulability ratio between F2VD, EDF-VD and the MCF. We vary the ρ values with other fixed parameters (i.e., $[U_{down}, U_{up}] = [0.02, 0.2]; [T_{down}, Tup] = [5, 50]; [Z_{down}, Z_{up}] = [1, 4]; P = 0.5$).

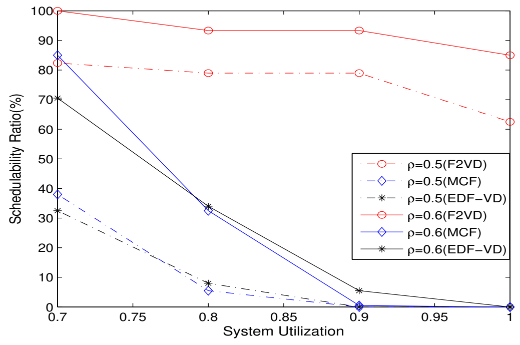

Figure 4: Comparison of schedulability ratio between F2VD, EDF-VD and the MCF. We vary the ρ values with other fixed parameters (i.e., $[U_{down}, U_{up}] = [0.02, 0.2]; [T_{down}, Tup] = [5, 50]; [Z_{down}, Z_{up}] = [1, 8]; P = 0.5$).

the optimal speedup factor of 4/3 for multiprocessor MC scheduling is achieved via fluid scheduling [6], while EDF-VD can only achieve a much worse speedup factor of 8/3. It is no longer the case for the problem considered in this work (and [12]). In future work, we plan to further explore the solution space of the entire EDF-VD family to achieve an even better understanding of the precise MC scheduling problem on a varying-speed processor.

6 EVALUATION

In this section, we report the performance of F2VD through experimental results, conducted on a randomly generated task-set. We also compare our algorithm with the approaches studied in [12]. To generate a random task-set, we use the workload generation model proposed by Guan et al. [21]. We describe the input specifications to generate the workload (used in this experiment) as follows:

- U_{bound}: the upper bound of the system utilization.
- $[T_{down}, T_{up}]$: the range of the minimum inter-arrival period of a task ,i.e., $0 \le T_{down} \le T_i \le T_{up}$.
- $[U_{down}, U_{up}]$: the range of the utilization of a task. This value (let us denote it by u_i) is used to to obtain execution time of a task in the LO-mode, i.e., $\forall \tau_i \in \tau : c_i = u_i \times T_i$, where, $0 \le U_{down} \le u_i \le U_{up} \le 1$.
- $[Z_{down}, Z_{up}]$: the range of the ratio of HI and LO-criticality WCET, here $1 \le Z_{down} \le Z_{up}$.
- P: the probability that a task is a HI-task. Here, $0 \le P \le 1$

Finally, we compare our algorithm with the following baselines:

- *EDF-VD* [12] and *MCF* [12], respectively denoted by $\rho = X(EDF - VD)$ and $\rho = X(MCF)$, where $X = \rho$ values.

In our experiment, we change the value of ρ (ranging from [0.5,0.6]) and the system utilization U_{bound} (ranging from [0.7,1.0]), and report the *schedulability ratio*, which is defined as the ratio of scheduled task-sets over the total number of task-sets. In Fig. 3, we show the comparison (w.r.t schedulability ratio) between our algorithm, EDF-VD, and the MCF for different utilization and ρ values. Fig. 3 reports that our approach outperforms both the EDF-VD and MCF approaches by large margin. All these approaches follow the same trend, i.e., for any ρ values, schedulability ratio decreases when system utilization increases. While, for the same system utilization and ρ value, the schedulability ratio of our approach is almost twice as large compared to the other approaches. Similar to the Fig. 3, we report the performance comparison (w.r.t. schedulability ratio) for these three approaches in Fig. 4 with a different $[Z_{down}, Z_{up}]$ value. Compared to the previous evaluation setup (i.e., $[Z_{down}, Z_{up}] = 4$), we observe a similar improvement in the schedulability ratio for our approach.

In Fig. 5, we report the time needed (with and without an objective function, refer to Sec. 3 for details) by F2VD to return the solution for a different size of task-set. We use the CVX solver [20] (in Matlab) to achieve a feasible solution. As expected, we observe a proportional relationship between the task-set size and the time needed (to return a feasible solution, if any) by the CVX solver. For a small task-set, we also see an improvement in the time when we use the solver without an objective function. Finally, these results do not demonstrate a significant variation in the time required to return a solution (with and without the objective function) with the changes in system utilization.

In Fig. 6, we report the *percentage of the feasible solution*, i.e., (the ratio of the number of the task sets for which F2VD returns a feasible solution over the total number of task sets), under different utilization and ρ values. From this figure, we observe that our algorithm finds a solution on average 70% cases. Also, we observe that the success percentage drops when the system utilization increases. This is because higher system utilization often leads to a system that is more difficult to schedule, which results in the optimization problem becoming harder or even infeasible to solve.

7 RELATED WORK

Since the seminal work by Vestal et al. [36], there has been extensive research on real-time scheduling of the sequential and parallel workload model in an MC platform, few to mention [1, 4, 7, 9, 13, 16, 29]. Most of these works considered that the system starts in the normal (LO-) mode, and unless there is a task overrun, all the tasks are guaranteed execution w.r.t their LO-WCET. However, if the system switches its criticality level (from LO to HI), e.g., due to a task overrun, the HI-tasks receive full service guarantee, while the LO-tasks receive no service guarantee [4, 7, 8, 16, 22, 25]. Most of these works

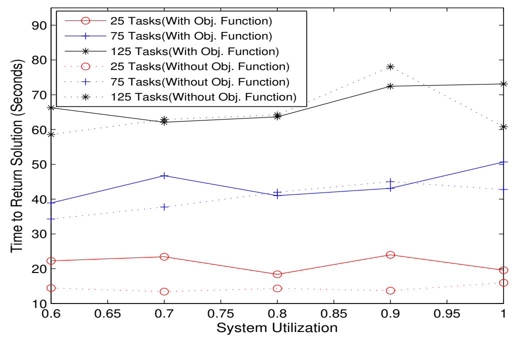

Figure 5: Time needed by F2VD to return the solution for a different size of task-set. We set the value of ρ to 0.5 with other fixed parameters as in Fig. 3.

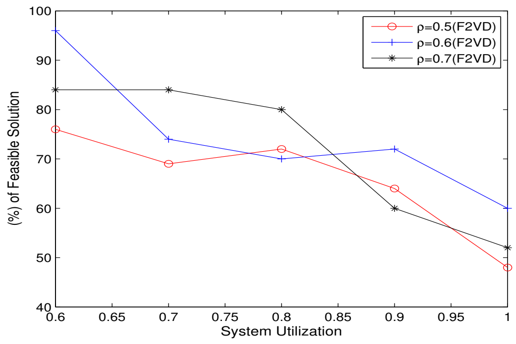

Figure 6: Percentage of solution returned by the F2VD algorithm under different utilization and ρ values with other fixed parameters as in Fig. 3.

have provided the schedulability analysis for this model considering the EDF-VD scheduling policy. A fluid model-based scheduling policy (each task executes at a speed dependant to its criticality level), MC-Fluid, was proposed by Lee et al. [28]. As the real hardware platforms can not directly implement the MC-Fluid scheduling policy, Lee et al. [28] also proposed MCDP-Fair, which can be implemented on a real platform while preserving the properties of the MC-Fluid. Based on the concept of the MC-Fluid, Baruah et al. [6] proposed MCF, a simplified yet improved algorithm (w.r.t.speedup bound). In a dual-criticality platform, Baruah et al. [6] proved an improved speedup bound of at most 1.33 for MCF. However, none of them considered energy-awareness in an MC platform.

Although a significant amount of works studied the energy-aware real-time task scheduling in the non-MC platform (few to mention [10, 11, 23, 24, 33, 34]), little progress has been made on energy-aware scheduling in the MC platform. Huang et al. [27] proposed an energy-aware scheduling by increasing the processor speed after a mode-switch. Narayana et al. [32] extended this work to the multi-core platform. All these works assumed that none of the LO-tasks receive service in the HI-mode.

Ernst et al. [17] and Esper et al. [18] criticized the MC model discussed above of being impractical as this model provides no service guarantee to any of the LO-tasks during HI-mode. The work proposed by Burns et al. [14] and Su et al. [35] considered executing the LO-tasks with an extended time. Some recent works proposed the IMC model [5, 14, 31]. In this model, degraded service are provided to the LO-tasks even in the HI-mode. Unlike these works, Bhuiyan et al. [12] proposed an energy-aware scheduling policy (based on EDF-VD and MCF) where all the LO-tasks are guaranteed to receive full service even in the HI-mode. However, (for the MCF algorithm), they restricted the inter-mode processor share ratio to a common variable across all HI-tasks.

8 CONCLUSION

Conventional MC scheduling design in real-time systems sacrifices the less critical tasks, entirely or partially, to guarantee temporal correctness to the more critical ones, when the extremely pessimistic scenario does happen. As a result, the computation of less critical (but not non-critical) tasks can be rendered imprecise if not completely dropped. In this paper, we have studied the problem of precise MC scheduling on a varying-speed processor, where all tasks, including less critical ones, must fully complete their execution in all circumstances. Meanwhile, the processor may begin with operating at a degraded speed and will recover to the full processing speed to cover the additional computation requirement once extremely pessimistic scenarios do happen during runtime.

We have investigated solving this problem by dual-rate fluid scheduling. We formulated the problem of finding feasible execution rates for each task as an optimization problem, which can be solved by numerical solvers that are readily available in Matlab. To overcome potential difficulties in implementing fluid scheduling, we have proposed algorithm F2VD, which is a virtual-deadline based scheduling algorithm and, therefore, is considered much more practical to implement than fluid scheduling. In particular, F2VD sets the virtual deadlines for the tasks by any feasible execution rates assignment if there exists one, and can guarantee all deadlines are met. Consequently, the existence and correctness of F2VD lead to the fact that the EDF-VD family strictly dominates dual-rate fluid scheduling in schedulability for the precise MC scheduling on a varying-speed processor. We have also conducted experiments to demonstrate the presented results and to evaluate the effectiveness of the proposed algorithm F2VD.

Future work. The direct next step of this work would be extending the underlying uniprocessor platform to a multiprocessor one. A key system configuration would be whether each of the multiple processors is able to adjust their operating speed independently or all (or a group of) processors must always be at the same speed. Each assumption may correspond to a certain set of hardware platforms and may result in a distinct variant of the scheduling problem. Moreover, as discussed in Sec. 5, exploring the solution space of the entire EDF-VD family that may not relate any fluid execution rates can be another interesting problem to study. This investigation can be directed in either a single or multiple varying-speed processors.

ACKNOWLEDGMENTS

We would like to thank Prof. Sanjoy Baruah from Washington University in St Louis for sharing his valuable insights. This work is supported by NSF grant CNS-1850851, a start-up grant from the University of Central Florida, and start-up and REP grants from Texas State University.

REFERENCES

[1] Sanjoy Baruah. 2016. The federated scheduling of systems of mixed-criticality sporadic DAG tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 227–236.

[2] S. Baruah. 2018. Mixed-Criticality Scheduling Theory: Scope, Promise, and Limitations. *IEEE Design Test* 35, 2 (2018), 31–37.

[3] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo DAngelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2012. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS), IEEE*. IEEE, 145–154.

[4] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'angelo, Haohan Li, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2015. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM (JACM)* 62, 2 (2015), 14.

[5] Sanjoy Baruah, Alan Burns, and Zhishan Guo. 2016. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS), IEEE*. IEEE, 131–138.

[6] Sanjoy Baruah, Arvind Easwaran, and Zhishan Guo. 2015. MC-Fluid: simplified and optimally quantified. In *2015 IEEE Real-Time Systems Symposium*. IEEE, 327–337.

[7] Sanjoy Baruah and Zhishan Guo. 2013. Mixed-criticality scheduling upon varying-speed processors. In *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 68–77.

[8] Sanjoy Baruah and Zhishan Guo. 2014. Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor. In *Proceedings of the 35th Real-Time Systems Symposium (RTSS), IEEE*. IEEE, 31–40.

[9] Sanjoy K Baruah, Alan Burns, and Robert I Davis. 2011. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 34–43.

[10] Ashikahmed Bhuiyan, Zhishan Guo, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. 2018. Energy-efficient real-time scheduling of DAG tasks. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 5 (2018), 84.

[11] Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2020. Energy-Efficient Parallel Real-Time Scheduling on Clustered Multi-Core. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2097–2111.

[12] Ashikahmed Bhuiyan, Sai Sruti, Zhishan Guo, and Kecheng Yang. 2019. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. ACM, 123–132.

[13] Ashikahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2019. Mixed-Criticality Multicore Scheduling of Real-Time Gang Task Systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 469–480.

[14] Alan Burns and Sanjoy Baruah. 2013. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS)*.

[15] Alan Burns and Robert I Davis. 2017. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)* 50, 6 (2017), 82.

[16] Pontus Ekberg and Wang Yi. 2014. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems* 50, 1 (2014), 48–86.

[17] Rolf Ernst and Marco Di Natale. 2016. Mixed Criticality Systems - A History of Misconceptions? *IEEE Design & Test* 33, 5 (2016), 65–74.

[18] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. 2015. How realistic is the mixed-criticality real-time system model?. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. ACM, 139–148.

[19] Fmincon 2018. https://www.mathworks.com/help/optim/ug/fmincon.html.

[20] Michael Grant and Stephen Boyd. 2014. CVX: Matlab Software for Disciplined Convex Programming, version 2.1. http://cvxr.com/cvx.

[21] Nan Guan, Pontus Ekberg, Martin Stigge, and Wang Yi. 2013. Improving the scheduling of certifiable mixed-criticality sporadic task systems. *Technical Report 2013–008* (2013).

[22] Zhishan Guo and Sanjoy Baruah. 2015. The concurrent consideration of uncertainty in WCETs and processor speeds in mixed-criticality systems. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. ACM, 247–256.

[23] Zhishan Guo, Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, and Nan Guan. 2019. Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 156–168.

[24] Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. 2017. Energy-efficient multi-core scheduling for real-time DAG tasks. (2017).

[25] Zhishan Guo, Luca Santinelli, and Kecheng Yang. 2015. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 187–196.

[26] Zhishan Guo, Kecheng Yang, Sudharsan Vaidhun, Samsil Arefin, Sajal K Das, and Haoyi Xiong. 2018. Uniprocessor Mixed-Criticality Scheduling with Graceful Degradation by Completion Rate. In *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 373–383.

[27] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. 2014. Energy efficient dvfs scheduling for mixed-criticality systems. In *Proceedings of the 14th International Conference on Embedded Software, ACM*. ACM, 11.

[28] Jaewoo Lee, Kieu-My Phan, Xiaozhe Gu, Jiyeon Lee, Arvind Easwaran, Insik Shin, and Insup Lee. 2014. Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *2014 IEEE Real-Time Systems Symposium*. IEEE, 41–52.

[29] Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. 2017. Mixed-criticality federated scheduling for parallel real-time tasks. *Real-time systems* 53, 5 (2017), 760–811.

[30] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[31] Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor Stefanov, and Wang Yi. 2016. EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees. In *Proceedings of the 37th Real-Time Systems Symposium (RTSS), 2016 IEEE*. IEEE, 35–46.

[32] Sujay Narayana, Pengcheng Huang, Georgia Giannopoulou, Lothar Thiele, and R Venkatesha Prasad. 2016. Exploring energy saving for mixed-criticality systems on multi-cores. In *Proceedings of the 22nd Real-Time and Embedded Technology and Applications Symposium (RTAS), IEEE*. IEEE, 1–12.

[33] Antonio Paolillo, Joël Goossens, Pradeep M Hettiarachchi, and Nathan Fisher. 2014. Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 1–10.

[34] Antonio Paolillo, Paul Rodriguez, Nikita Veshchikov, Joël Goossens, and Ben Rodriguez. 2016. Quantifying energy consumption for practical fork-join parallelism on an embedded real-time operating system. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM, 329–338.

[35] Hang Su and Dakai Zhu. 2013. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 147–152.

[36] S. Vestal. 2007. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*.