Precise Scheduling of Mixed-Criticality Tasks on Varying-Speed Multiprocessors

Tianning She*
Texas State University
USA

Sudharsan Vaidhun*
University of Central Florida
USA

Qijun Gu
Texas State University
USA

Sajal K. Das
sdas@mst.edu
Missouri Univ. of Science and Tech.
USA

Zhishan Guo
zsguo@ucf.edu
University of Central Florida
USA

Kecheng Yang
yangk@txstate.edu
Texas State University
USA

ABSTRACT

In conventional real-time systems analysis, each system parameter is specified by a single estimate, which must pessimistically cover the worst case. Mixed-criticality (MC) design has been proposed to mitigate such pessimism by providing a single system parameter with multiple estimates, which often lead to low-critical and high-critical modes. The majority of the works on MC scheduling is based on the approach that low-critical workloads are (fully or partially) sacrificed at the transition instant from low- to high-critical mode. Recently, another approach called *precise MC scheduling* has been investigated, where no low-critical workload is sacrificed at the mode switch, but instead a processor speed boosting is committed. In this paper, we extend the work on uniprocessor precise MC scheduling to multiprocessor platforms. To tackle this new scheduling problem, we propose two novel algorithms based on the virtual-deadline and fluid-scheduling approaches. For each approach, we present a sufficient schedulability test and prove its correctness. We also evaluate their effectiveness theoretically with speedup bounds and approximation factors as well as experimentally via randomly generated task sets.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Tianning She, Sudharsan Vaidhun, Qijun Gu, Sajal K. Das, Zhishan Guo, and Kecheng Yang. 2021. Precise Scheduling of Mixed-Criticality Tasks on Varying-Speed Multiprocessors. In *29th International Conference on Real-Time Networks and Systems (RTNS'2021), April 7–9, 2021, NANTES, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3453417.3453428

*Both authors contributed equally to this work.

RTNS'2021, April 7–9, 2021, NANTES, France

ACM ISBN 978-1-4503-9001-9/21/04…$15.00
https://doi.org/10.1145/3453417.3453428

1 INTRODUCTION

In the era of multi-core and many-core platforms and commercial-off-the-shelf processors, the gap between average-case and worst-case performance of a task is growing rapidly. Under such a trend, in order to guarantee the worst-case temporal correctness of a system, one approach is to allocate computing resources according to the most pessimistic system assumptions (or models), at the cost of significant resource under-utilization during run-time. To mitigate such pessimism, the mixed-criticality (MC) paradigm has been proposed in real-time systems design. In an MC or Multi-Model System [14], a set of tasks with different criticalities shares the same platform, and various levels of guarantees are provided under different system assumptions. The classical Vestal MC Model [34] separates the timing assumptions by setting multiple worst-case execution time (WCET) estimates for each high-critical task, while under each assumption, it is guaranteed that tasks with the corresponding or higher criticality levels will meet their deadlines.

To provide the desired guarantees under more pessimistic assumptions, a common practice in MC is to degrade the service to tasks of lower criticality levels, and define system-wide execution modes. By providing no or partial guarantee, such as imprecise mixed-criticality that allows graceful degradation of low-criticality tasks in high-criticality mode [3, 15, 31] upon mode switches, the computing capacities can be freed up to handle extra demands from higher criticality tasks. However, current industrial practices require that low-critical tasks should receive full service guarantees, and also no degradation is allowed as they are not "non-critical." As a result, the concept of *precise mixed-criticality* model [13] has been proposed recently, which guarantees the execution of all low-criticality tasks under all modes/assumptions.

Now an important question is: without sacrificing low-critical tasks, how can we provide guarantees to additional requirements from high-critical tasks upon mode switch? Due to the recent advances in hardware and operating systems, techniques like dynamic voltage and frequency scaling (DVFS), make it straightforward to adjust the processing speeds of computing platforms. We believe that by increasing the processor speed, when necessary, one can handle additional computing requirements from high-critical tasks without sacrificing low-critical ones. While most existing MC designs including the ones on varying-speed platforms (*e.g.*, [7, 8, 26]) provide no guarantees to low-critical tasks, a breakthrough work due to Bhuiyan et al. [13] combines the precise scheduling of (sporadic implicit-deadline) MC tasks with varying speed platform.

Under a dual-criticality setting, the authors proposed to minimize the processor speed under less pessimistic WCET assumptions, while guaranteeing that deadlines are met under all circumstances by increasing the processor speed to the maximum possible value upon mode switch triggered by high-critical task's overrun. The works in [13] and [35] proposed virtual deadline based on earliest deadline first (EDF), and fluid-rate based approaches for precise scheduling of MC tasks on uniprocessor platforms, respectively.

The *precise MC* model has two significant benefits. First, it provides full service guarantees to both high- and low-criticalility tasks under all circumstances. Second, with multi-mode settings, when the less pessimistic assumptions are fulfilled, the platform can execute at a lower speed, leading to energy efficiency that often plays an important role in embedded system design. *This paper focuses on precise scheduling of MC tasks upon varying-speed multiprocessors, by combining precise computing and DVFS-based energy conservation on a preemptive identical multiprocessor platform.*

Contributions. To the best of our knowledge, this is the first work that tackles the precise MC scheduling with varying processor speed on a *multiprocessor* platform. Specifically, this work

- presents a new MC system model and formalizes a new MC scheduling problem;
- proposes a new virtual-deadline based scheduler fpEDF-VD and a sufficient schedulability test for it;
- proves a speedup bound $\frac{4m}{m+1} < 4.0$ for fpEDF-VD, where m is the number of processors;
- develops an alternative algorithm based on the fluid-scheduling approach MCF-FR, which offers a closed form schedulability test with a proven approximation ratio;
- conducts empirical schedulability experiments to demonstrate and compare the effectiveness of proposed approaches.

Organization. The rest of the paper is organized as follows. Section 2 describes the system model and our targeted scheduling problem. Section 3 presents fpEDF-VD scheduler and proves its schedulability test and speedup bound. Focusing on an alternative fluid-based scheduling framework, Section 4 presents a polynomial-time algorithm MCF-FR and derives an approximation ratio. Section 5 compares the proposed approaches with state-of-the art methods via randomly generated task sets. Section 6 reviews related work while Section 7 concludes the paper with future research directions.

2 MODEL AND PROBLEM STATEMENT

Let $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a set of n implicit-deadline sporadic MC tasks. Each task, specified by a 3-tuple $\tau_i = (T_i, C_i^L, C_i^H)$, 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 of task τ_i, defined by the worst-case execution time on a unit-speed processor, is estimated a two criticality levels: a low-criticality estimate C_i^L and a high-criticality estimate C_i^H, where it is assumed that $C_i^L \leq C_i^H, \forall i$. Besides, C_i^L (respectively, C_i^H) is also the execution requirement budgets of task τ_i in the L (respectively, H)-*mode*, to be described later. In particular, $C_i^L < C_i^H$ indicates that τ_i is a HI-criticality task that may trigger a system mode switch, whereas $C_i^L = C_i^H$ indicates that τ_i is a LO-criticality task that cannot trigger any system mode switch. Let the j^{th} job of task τ_i be denoted as $J_{i,j}$. In this paper, we assume that the preemption and migration overheads *e.g.*, due to memory interference are negligible. (Equivalently, we assume such overheads are pessimistically taken into account in the execution requirements estimates.)

Varying-speed multiprocessor and mode switch. We consider the problem of scheduling the set of tasks τ on m energy-conserving processors that can operate at a *degraded* or *full* speed. All the m processors begin 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 cumulatively executed for a workload of C_i^L under degraded processing speed ρ (thus receiving a cumulative actual execution *time* of C_i^L/ρ units) but still requires further execution, the system is immediately notified, and *all* the m processors start to perform its full speed 1.0 from that instance. We call this moment as the time instant of *mode switch*, from the L-mode (where the processor speed is ρ) to the H-mode (where the processor speed becomes 1.0). The system can recover to the L-mode once *all* processors become idle.

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

We denote the utilization of a task τ_i in L- and H-modes, respectively, by

$$u_i^L = \frac{C_i^L}{T_i} \quad \text{and} \quad u_i^H = \frac{C_i^H}{T_i}.$$

Since $C_i^L = C_i^H$ holds for every LO-criticality task, it also holds $u_i^L = u_i^H$ for such task. We further denote the total utilization of all tasks in L- and H-modes, respectively, by

$$U^L = \sum_i u_i^L \quad \text{and} \quad U^H = \sum_i u_i^H.$$

Let us define $u_{\max}^L = \max_i\{u_i^L\}$ and $u_{\max}^H = \max_i\{u_i^H\}$.

Problem Statement. We address the problem of scheduling the MC tasks on m varying-speed processors to guarantee all deadlines are met in *all* scenarios with the following additional requirements on processor execution speed:

- all processors must only operate at their energy-conserving speed ρ if *all* jobs finish within their C_i^L budget;
- the processors may operate at full speed 1.0 if a *any* HI-criticality job executes beyond its C_i^L budget (yet finishes within its C_i^H budget).

Optimizing the energy-conserving speed (ρ). Given the schedulability problem as stated above, the problem of optimizing the required energy-conserving speed can be addressed by applying

the schedulability tests for $\rho \in [0, 1]$ in a binary-search fashion. However, due to the fact that our schedulability tests are sufficient but not necessary, this method will result in the minimum value of ρ *for passing our schedulability tests* — it might be larger than the minimum feasible value of ρ that can be potentially achieved by a better scheduler and/or better schedulability analysis. To emphasize on the schedulability analysis, in the rest of this paper, we will focus on the problem of determining the schedulability for given constant energy-conserving speed, ρ.

3 SCHEDULING BY VIRTUAL DEADLINES

To solve the aforementioned problem, this section proposes a new scheduler based on the virtual-deadline approach and presents our first algorithm, called fpEDF-VD. We prove a sufficient schedulability test for fpEDF-VD for the precise MC scheduling on m varying-speed processors, and show that fpEDF-VD has a speedup bound $\frac{4m}{m+1} < 4.0$.

3.1 Algorithm fpEDF-VD

The proposed scheduler combines two existing approaches, namely fpEDF and EDF-VD, which are first summarized below.

Algorithm fpEDF**.** Based on the global EDF scheduler, Baruah [10] developed fpEDF by statically prioritizing *high-utilization* tasks for which the utilization exceeds 0.5. It assumes m identical unit-speed processors satisfy the following sufficient schedulability test.

Theorem 3.1 (Theorem 4 in [10]). *Let U_{sum} denote the total utilization of an arbitrary sporadic task set, in which the utilization of task τ_i is denoted by u_i. This task set is schedulable by* fpEDF *on m identical unit-speed processors if*

$$\forall i, u_i \le 1.0 \quad \textit{and} \quad U_{sum} \le \frac{m+1}{2}.$$

Algorithm EDF-VD**.** To address the problem of uniprocessor MC scheduling under Vestal's model [34], EDF-VD algorithm [1, 11] was proposed. In EDF-VD, each HI-critical task is assigned a *virtual deadline* that is smaller than its actual deadline to promote the execution of HI-critical tasks in the L-mode and to leave slack for the potential extra workload upon a mode switch to H-mode. EDF-VD has two phases. In the pre-processing phase, a scaling factor x is calculated and used to determine the virtual deadlines for HI-critical tasks by $\hat{T}_i' = x \cdot T_i$. For LO-critical tasks, their virtual deadlines are set identical to their actual deadlines, *i.e.*, T_i. In the runtime phase, the system starts with the L-mode and the tasks are scheduled according to EDF by their *virtual* deadlines. When a HI-critical task overrun its LO-critical execution time estimate, the system switch to H-mode. At the mode switch, all LO-critical tasks are dropped (or scheduled in the background) and HI-critical tasks are scheduled by their *actual* deadlines henceforth.

Algorithm fpEDF-VD**.** Leveraging fpEDF and EDF-VD, our novel algorithm, called fpEDF-VD, also has two phases. In the *pre-processing phase* (described in Algorithm 1), both the HI-critical and LO-critical tasks are assigned a virtual deadline by the scaling factor. This is because in precise MC scheduling, LO-critical tasks are not dropped at the mode switch and therefore we also need the virtual deadlines to control their carry-over behaviors upon a mode switch. In the *runtime phase* (described in Algorithm 2), the tasks are mapped to a set of non-MC sporadic tasks in the L- and H-mode, respectively, to apply fpEDF in each mode. In other words, we have two mappings from the MC tasks to non-MC sporadic tasks, and upon a mode switch, fpEDF is re-launched with respect to a different set of non-sporadic tasks.

Algorithm 1: Pre-processing Phase of Algorithm fpEDF-VD.

For a dual-criticality task-set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ to be scheduled on m energy-conserving preemptive processors, each of which has energy-conserving speed ρ and full speed 1.0:

- Determine virtual deadline of *all* tasks by computing the scaling factor x:
$$x \leftarrow \max\left(\frac{u^L_{\max}}{\rho}, \frac{U^L}{\frac{m+1}{2}\rho}\right) \tag{1}$$
- If $\max\left(\frac{u^L_{\max}}{\rho}, \frac{U^L}{\frac{m+1}{2}\rho}\right) + \max\left(u^H_{\max}, \frac{U^H}{\frac{m+1}{2}}\right) \le 1$
then set virtual-deadline $\hat{T}_i \leftarrow x \cdot T_i$ for each task τ_i, and return SUCCESS;
Else return FAILURE.

Algorithm 2: Runtime Phase for Algorithm fpEDF-VD.

If the pre-processing phase returns SUCCESS, then fpEDF-VD schedules tasks during runtime as follows:

- In the L-mode, fpEDF-VD schedules tasks by their virtual deadlines on m **speed-ρ** processors.
It is equivalent to scheduling the task set $\{(T_i', C_i')\}_{i=1}^n$ on m **unit-speed** processors, where $T_i' = xT_i$ and $C_i' = C_i^L/\rho$ for every task τ_i,
Then, task set $\{(T_i', C_i')\}_{i=1}^n$ is scheduled by fpEDF where tasks for which $C_i'/T_i' > 0.5$ are considered *high-utilization*.
- In the H-mode, fpEDF-VD schedules the tasks' actual deadlines on m unit-speed processors.
It is equivalent to scheduling the task set $\{(T_i'', C_i'')\}_{i=1}^n$ on m unit-speed processors, where $T_i'' = (1-x)T_i$ and $C_i'' = C_i^H$ for every task τ_i,
Then, task set $\{(T_i'', C_i'')\}_{i=1}^n$ is scheduled by fpEDF where tasks for which $C_i''/T_i'' > 0.5$ are considered *high-utilization*.

3.2 Schedulability Test

We now derive a sufficient schedulability test for fpEDF-VD in Theorem 3.4, which is based on two lemmas. First, a sufficient schedulability test and its correctness in the L-mode is established by the following lemma.

Lemma 3.2. *All tasks meet their virtual deadlines in the L-mode under* fpEDF-VD *if*

$$x \ge \frac{u^L_{\max}}{\rho} \quad \textit{and} \quad x \ge \frac{U^L}{\frac{m+1}{2}\rho}.$$

Proof. As shown in Algorithm 2, all tasks in the L-mode are mapped to $\{(T_i', C_i')\}$, which are scheduled by fpEDF. By Theorem 3.1, if $\forall i, \frac{C_i'}{T_i'} \leq 1$ and $\sum_i \frac{C_i'}{T_i'} \leq \frac{m+1}{2}$, then all tasks in $\{(T_i', C_i')\}$ must meet their deadlines, *i.e.*, all MC tasks in τ must meet their *virtual* deadlines in the L-mode. Additionally, we have

$$\forall i, \frac{C_i'}{T_i'} \leq 1 \Leftrightarrow \forall i, \frac{C_i^L/\rho}{xT_i} \leq 1 \Leftrightarrow \forall i, u_i^L \leq x\rho \Leftrightarrow x \geq \frac{u_{\max}^L}{\rho},$$

$$\text{and } \sum_i \frac{C_i'}{T_i'} \leq \frac{m+1}{2} \Leftrightarrow \sum_i \frac{C_i^L/\rho}{xT_i} \leq \frac{m+1}{2} \Leftrightarrow \frac{U_i^L}{x\rho} \leq \frac{m+1}{2} \Leftrightarrow x \geq \frac{U^L}{\frac{m+1}{2}\rho}.$$

Thus, the lemma follows. ∎

Next, a sufficient schedulability test and its correctness in the H-mode is shown by the following lemma.

Lemma 3.3. *All tasks meet their actual deadlines in the H-mode under* fpEDF-VD *if*

$$x \leq 1 - u_{\max}^H \quad \textit{and} \quad x \leq 1 - \frac{U^H}{\frac{m+1}{2}}.$$

Proof. We first prove that it is *safe* to map the task set τ in the H-mode to the task set $\{(T_i'', C_i'')\}$ by treating the time instant of mode switch, t^*, as the typical "last-idle instant" in EDF schedulability analysis. To this end, note that for the jobs released in the L-mode, any job with a virtual deadline before t^* must have completed by t^* (by Lemma 3.2) while any job with a virtual deadline at or after t^* must have its deadline at least $(1-x)T_i = T_i''$ time units after t^* (safely assuming that it has not received any execution during L-mode, which is the worst-case from H-mode perspective). Whereas, for jobs released in the H-mode, its deadline is at T_i time units after its release (even further apart). Meanwhile, every job of task τ_i cannot execute for more than $C_i^H = C_i''$ in any scenario. Thus, it is sufficient to model the behavior of the MC task set τ in the H-mode as a simple sporadic task set $\{(T_i'', C_i'')\}$, and therefore this mapping is safe.

Given this mapping and $\{(T_i'', C_i'')\}$ is scheduled by fpEDF (Theorem 3.1), we conclude that if $\forall i, \frac{C_i''}{T_i''} \leq 1$ and $\sum_i \frac{C_i''}{T_i''} \leq \frac{m+1}{2}$, then all tasks in $\{(T_i'', C_i'')\}$ must meet their deadlines, *i.e.*, all MC tasks in τ must meet their *actual* deadlines in the H-mode. We also have

$$\forall i, \frac{C_i''}{T_i''} \leq 1 \Leftrightarrow \forall i, \frac{C_i^H}{(1-x)T_i} \leq 1 \Leftrightarrow \forall i, u_i^H \leq 1-x \Leftrightarrow x \leq 1-u_{\max}^H,$$

$$\text{and } \sum_i \frac{C_i''}{T_i''} \leq \frac{m+1}{2} \Leftrightarrow \sum_i \frac{C_i^H}{(1-x)T_i} \leq \frac{m+1}{2} \Leftrightarrow \frac{U_i^H}{1-x} \leq \frac{m+1}{2} \Leftrightarrow x \leq 1 - \frac{U^H}{\frac{m+1}{2}}.$$

Thus, the lemma follows. ∎

Lemmas 3.2 and 3.3 lead to a sufficient schedulability test for fpEDF-VD as follows.

Theorem 3.4. fpEDF-VD *correctly schedules a dual-criticality task-set* $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ *on* m *energy-conserving preemptive processors, each with energy-conserving speed* ρ *and max speed* 1.0*, if*

$$\max\left(\frac{u_{\max}^L}{\rho}, \frac{U^L}{\frac{m+1}{2}\rho}\right) + \max\left(u_{\max}^H, \frac{U^H}{\frac{m+1}{2}}\right) \leq 1. \tag{2}$$

Proof. By the assignment of x in expression (1), we have

$$x = \max\left(\frac{u_{\max}^L}{\rho}, \frac{U^L}{\frac{m+1}{2}\rho}\right).$$

Therefore, $x \geq \frac{u_{\max}^L}{\rho}$ and $x \geq \frac{U^L}{\frac{m+1}{2}\rho}$. Also, by inequality (2), $x < 1$. Thus, in the L-mode, by Lemma 3.2, all virtual deadline and hence all actual deadlines are met under fpEDF-VD.

Again, by inequality (2), we have

$$x \leq 1 - \max\left(u_{\max}^H, \frac{U^H}{\frac{m+1}{2}}\right),$$

which implies

$$x \leq 1 - u_{\max}^H \quad \text{and} \quad x \leq 1 - \frac{U^H}{\frac{m+1}{2}}.$$

Thus, in the H-mode, by Lemma 3.3, all actual deadlines are met under fpEDF-VD. Hence the theorem. ∎

3.3 Speedup Bound

This section derives a *speedup bound* (defined below) for fpEDF-VD.

Speedup bound. An algorithm $\mathcal{A}$ having a speedup bound s means that any system that is schedulable by a potentially optimal algorithm on m energy-conserving processors with degraded speed ρ and full speed 1.0 for each processor, must also be schedulable by algorithm $\mathcal{A}$ on m energy-conserving processors with degraded speed $\rho \cdot s$ and full speed s for each processor.

Lemma 3.5. *Any system that is schedulable by a potentially optimal algorithm on* m *energy-conserving processors with degraded speed* $\frac{m+1}{4m}\rho$ *and full speed* $\frac{m+1}{4m}$ *for each processor must also be schedulable by algorithm* fpEDF-VD *on* m *energy-conserving processors with degraded speed* ρ *and full speed* 1.0 *for each processor.*

Proof. To be schedulable even by an optimal scheduler, any individual task's utilization must be at most the speed of a single processor in both L- and H-mode, *i.e.*,

$$\forall i, u_i^L \leq \frac{m+1}{4m}\rho \Rightarrow u_{\max}^L \leq \frac{m+1}{4m}\rho,$$

$$\text{and } \forall i, u_i^H \leq \frac{m+1}{4m} \Rightarrow u_{\max}^H \leq \frac{m+1}{4m}.$$

Furthermore, to be schedulable even by an optimal scheduler, the sum of utilization of all tasks must be at most the sum of all processors' speed in both L- and H-modes, *i.e.*,

$$U^L \leq \frac{m+1}{4m}\rho \cdot m,$$

$$\text{and } U^H \leq \frac{m+1}{4m} \cdot m.$$

Therefore,

$$\begin{aligned}
&\max\left(\frac{u^L_{\max}}{\rho}, \frac{U^L}{\frac{m+1}{2}\rho}\right) + \max\left(u^H_{\max}, \frac{U^H}{\frac{m+1}{2}}\right) \\
&\leq \max\left(\frac{\frac{m+1}{4m}\rho}{\rho}, \frac{\frac{m+1}{4m}\rho \cdot m}{\frac{m+1}{2}\rho}\right) + \max\left(\frac{m+1}{4m}, \frac{\frac{m+1}{4m} \cdot m}{\frac{m+1}{2}}\right) \\
&= \max\left(\frac{m+1}{4m}, \frac{1}{2}\right) + \max\left(\frac{m+1}{4m}, \frac{1}{2}\right) \\
&= \frac{1}{2} + \frac{1}{2}, \quad \text{because } m \geq 1 \\
&= 1 \qquad (3)
\end{aligned}$$

From expression (3) and Theorem 3.4, we conclude that this task set is schedulable by fpEDF-VD on m energy-conserving processors with degraded speed ρ and full speed 1.0 for each processor. Hence the lemma follows. ■

By re-defining a unit of speed (scaling up the value of all speed parameters by multiplying $\frac{4m}{m+1}$), Lemma 3.5 can be re-written as

> Any system schedulable by some (potentially optimal) algorithm on m energy-conserving processors with degraded speed ρ and full speed 1.0 for each processor, must also be schedulable by algorithm fpEDF-VD on m energy-conserving processors with degraded speed $\frac{4m}{m+1}\rho$ and full speed $\frac{4m}{m+1}$ for each processor.

This, by definition, yields a speedup bound for fpEDF-VD as:

Theorem 3.6. *Algorithm* fpEDF-VD *has a speedup bound* $\frac{4m}{m+1}$.

Since $\forall m$, $\frac{4m}{m+1} < 4.0$, the following corollary is immediate.

Corollary 3.7. *Algorithm* fpEDF-VD *has a speedup bound* 4.0.

4 FLUID SCHEDULING

In this section, we focus on an alternative approach based on the concept of fluid scheduling. In fluid scheduling, each task may receive a fraction of a processor, so that *all* tasks may progress at specific rates simultaneously even if the number of tasks exceeds the number of available processors. It is required for such rates that **(i)** the summation of the rates of all tasks does not exceed the platform capacity (*i.e.*, the product of the number of processors and the individual processor speed), and **(ii)** the rate of each individual task does not exceed the speed of an individual processor. Admittedly, the notion of all tasks progressing simultaneously at constant executing rates is theoretical and idealistic. However, such simultaneous progression can be implemented by slicing the timeline to smaller pieces or by certain fairness based scheduling algorithms (e.g., DP-Fair [30]), which has been successfully adapted to implement fluid scheduling for MC tasks [29].

When it comes to MC scheduling, the *dual-rate fluid* scheduling[1] is often considered, where each task τ_i is assigned two constant execution rates in L- and H-modes, denoted by θ^L_i and θ^H_i, respectively. Specifically, for each LO-criticality task, a constant execution speed of $\theta_i = u_i$ would be sufficient under both modes. By contrast, for a HI-criticality task, it would require a speed larger than its LO-utilization in the L-mode (to create sufficient gap after the mode switch to accommodate the additional execution requirement) and an even larger speed after the mode switch. Such a relationship is illustrated in Figure 1, where the blue line indicates LO-criticality task setting and the red line represents HI-criticality task settings.

[1]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 in the context of MC scheduling [6, 29]. Fluid scheduling with no restriction on the number of rate changes can be too general. For example, *any* actual schedule can be viewed as a fluid schedule where the rate for each task switches between 0 and 1.0.

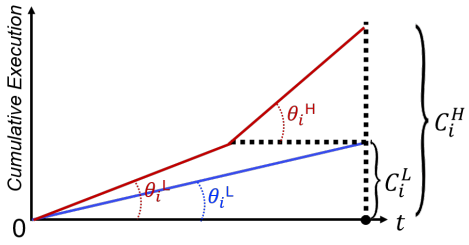

Figure 1: Relation between fluid execution rate and cumulative execution over time of a task under MCF framework.

4.1 A Fluid Assignment and Execution Scheme

Let us now present an approach to find a feasible execution rate assignment $\{(\theta^L_i, \theta^H_i)\}_{i=1}^n$ by restricting the ratio between the L-mode and the H-mode to be the same among all HI-criticality tasks. That is, each task τ_i is assigned an execution speed of $\theta^H_i = \theta_i$ in the H-mode and a speed of $\theta^L_i = \lambda \cdot \theta_i$ in the L-mode, where $\lambda \geq 0$. The ratio $\theta^L_i / \theta^H_i = \lambda$ for all i; while for each LO-criticality task, we assign a constant execution speed of $\theta_i = u_i$ under both modes.

Recall that we have simplified notations for per-mode utilization:

$$U^L = \sum_i u^L_i \quad \text{and} \quad U^H = \sum_i u^H_i.$$

Similarly, letting $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ denote the set of LO-criticality and HI-criticality tasks, respectively, we define per-mode total utilization for $\mathcal{T}_{\text{LO}}$ and $\mathcal{T}_{\text{HI}}$ as follows:

$$\begin{aligned}
U^L_{\text{LO}} &= \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u^L_i = \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u^H_i = U^H_{\text{LO}}, \\
U^L_{\text{HI}} &= \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u^L_i \quad \text{and} \quad U^H_{\text{HI}} = \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u^H_i.
\end{aligned}$$

Algorithm 3 describes the proposed algorithm MCF-FR to select the proper λ and θ_i values for any given precise MC task system. The schedulability directly depends on whether the resulting λ can be upper bounded by the degraded speed ρ. Clearly, MCF-FR and its schedulability test runs in linear (polynomial) time with respect to the number of tasks. Note that it is a *sufficient only* algorithm to solve the dual-rate fluid scheduling problem, *i.e.*, there exists systems for which MCF-FR returns FAILURE while feasible dual-rate assignments may still exist.

Note that, the assignment of λ by Eq. (4) in Algorithm 3 sufficiently guarantees that $0 < \lambda \leq 1$ because $U^H \leq m$ and $\forall i$, $u^H_i \leq 1$ must hold; otherwise, the deadlines cannot be guaranteed by any algorithm due to over-utilization.

Algorithm 3: Algorithm MCF-FR.

For a dual-criticality task-set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ to be scheduled on m energy-conserving preemptive processors, each having energy-conserving speed ρ and max speed 1.0:

- A system-wide parameter λ and per-task parameters θ_i are computed as:

$$\lambda = \max\left\{\frac{U^L}{m + U^L - U^H}, \max_i\left\{\frac{u_i^L}{1 + u_i^L - u_i^H}\right\}\right\} \tag{4}$$

$$\forall \tau_i \in \mathcal{T}_{\text{HI}}, \quad \theta_i = \frac{u_i^L}{\lambda} + u_i^H - u_i^L \tag{5}$$

- If the energy-conserving speed $\lambda \le \rho$
  then each HI-criticality task τ_i is to be executed at fluid rate $\theta_i^L = \lambda \cdot \theta_i$ in the L-mode and at fluid rate $\theta_i^H = \theta_i$ in the H-mode, while each LO-criticality task is to be executed at fluid rate $\theta_i^L = \theta_i^H = u_i^L$ in both modes, and return SUCCESS;
  Else return FAILURE.

4.2 Correctness Guarantee

In the following, we show that by selecting λ and θ_i according to MCF-FR in Algorithm 3, if $\lambda \le \rho$, then the system is schedulable under MCF-FR.

According to Lemmas 4.1, 4.2, and 4.3, we first show that the resulted fluid rates by MCF-FR must be valid. That is, in either L- or H-mode, the assigned rate to every task must not exceed the speed of an individual processor, and the total assigned rates to all tasks must not exceed the sum of all processors' speed.

Lemma 4.1. *If* MCF-FR *returns* SUCCESS, *then*

$$\forall i : 1 \le i \le n,\ \theta_i^L \le \rho \ \textit{and}\ \theta_i^H \le 1.0.$$

Proof. For LO-criticality tasks, $\theta_i^L = \theta_i^H = u_i^L$. By Eq. (4), it is clear that $\lambda \ge u_i^L, \forall i$. Also, MCF-FR as described in Algorithm 3 returning SUCCESS implies that $\lambda \le \rho$. Therefore, this lemma holds for any LO-criticality task.

In the rest of this proof, we focus on an arbitrary HI-criticality task τ_i, and note that Eq. (4) in Algorithm 3 implies that

$$\forall \tau_i \in \mathcal{T}_{\text{HI}},\ \lambda \ge \frac{u_i^L}{1 + u_i^L - u_i^H}. \tag{6}$$

Then, by Eq. (5) we have

$$\begin{aligned}
\theta_i^H &= \frac{u_i^L}{\lambda} + u_i^H - u_i^L \\
&\le \frac{u_i^L}{\frac{u_i^L}{1+u_i^L-u_i^H}} + u_i^H - u_i^L \qquad \{\text{by Eq. (6)}\} \\
&= 1 + u_i^L - u_i^H + u_i^H - u_i^L \\
&= 1,
\end{aligned}$$

and also have

$$\begin{aligned}
\theta_i^L &= \lambda \cdot (\frac{u_i^L}{\lambda} + u_i^H - u_i^L) \\
&= u_i^L + \lambda \cdot u_i^H - \lambda \cdot u_i^L \\
&= \frac{u_i^L}{1 + u_i^L - u_i^H} \cdot (1 + u_i^L - u_i^H) + \lambda u_i^H - \lambda u_i^L \\
&\le \lambda \cdot (1 + u_i^L - u_i^H) + \lambda u_i^H - \lambda u_i^L \qquad \{\text{by Eq. (6)}\} \\
&= \lambda \\
&\le \rho. \qquad \{\text{because MCF-FR returns SUCCESS}\}
\end{aligned}$$

Thus, the lemma follows. ∎

Lemma 4.2. *If* MCF-FR *returns* SUCCESS, *then* $\sum_i \theta_i^L \le \rho \cdot m$.

Proof. First of all, we have the following equality.

$$\begin{aligned}
\sum_i \theta_i^L &= \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} \theta_i^L + \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \theta_i^L \\
&= \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u_i^L + \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \left(\lambda \cdot (\frac{u_i^L}{\lambda} + u_i^H - u_i^L)\right) \\
&= U_{\text{LO}}^L + U_{\text{HI}}^L + \lambda \cdot (U_{\text{HI}}^H - U_{\text{HI}}^L) \\
&= U^L + \lambda \cdot (U_{\text{HI}}^H - U_{\text{HI}}^L)
\end{aligned}$$

Then, by Eq. (4) in Algorithm 3, we have

$$\lambda \ge \frac{U^L}{m + U^L - U^H} \Rightarrow U^L \le \lambda \cdot (m + U^L - U^H).$$

Therefore, we have the following inequality.

$$\begin{aligned}
\sum_i \theta_i^L &\le \lambda \cdot (m + U^L - U^H) + \lambda \cdot (U_{\text{HI}}^H - U_{\text{HI}}^L) \\
&= \lambda \cdot (m + U^L - U^H + U_{\text{HI}}^H - U_{\text{HI}}^L) \\
&= \lambda \cdot (m + (U^L - U_{\text{HI}}^L) - (U^H - U_{\text{HI}}^H)) \\
&= \lambda \cdot (m + U_{\text{LO}}^L - U_{\text{LO}}^H) \\
&= \lambda \cdot m \qquad \{\text{because } U_{\text{LO}}^L = U_{\text{LO}}^H\} \\
&\le \rho \cdot m \qquad \{\text{because MCF-FR returns SUCCESS}\}
\end{aligned}$$

Thus, the lemma follows. ∎

Lemma 4.3. *If* MCF-FR *returns* SUCCESS, *then* $\sum_i \theta_i^H \le m$.

Proof. First of all, we have the following equality.

$$\begin{aligned}
\sum_i \theta_i^H &= \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} \theta_i^H + \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} \theta_i^H \\
&= \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u_i^L + \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} (\frac{u_i^L}{\lambda} + u_i^H - u_i^L) \\
&= U_{\text{LO}}^L + \frac{U_{\text{HI}}^L}{\lambda} + U_{\text{HI}}^H - U_{\text{HI}}^L \\
&\le \frac{U_{\text{LO}}^L}{\lambda} + \frac{U_{\text{HI}}^L}{\lambda} + U_{\text{HI}}^H - U_{\text{HI}}^L \quad \{\text{because } 0 < \lambda \le 1\} \\
&= \frac{U^L}{\lambda} + U_{\text{HI}}^H - U_{\text{HI}}^L
\end{aligned}$$

Then, by Eq. (4) we have

$$\lambda \geq \frac{U^L}{m + U^L - U^H}.$$

Therefore,

$$\begin{aligned}\sum_i \theta_i^H &\leq \frac{U^L}{\frac{U^L}{m+U^L-U^H}} + U_{\text{HI}}^H - U_{\text{HI}}^L \\ &= m + U^L - U^H + U_{\text{HI}}^H - U_{\text{HI}}^L \\ &= m + (U^L - U_{\text{HI}}^L) - (U^H - U_{\text{HI}}^H) \\ &= m + U_{\text{LO}}^L - U_{\text{LO}}^H \\ &= m,\end{aligned}$$

where the last step is because $U_{\text{LO}}^L = U_{\text{LO}}^H$. Thus, the lemma follows. ■

The following lemma shows that MCF-FR correctly schedules all HI-criticality tasks.

Lemma 4.4. *In* MCF-FR*, by assigning execution rate via Eq.* (5)*, each* HI*-criticality task will receive enough execution by its deadline.*

Proof. For any HI-criticality task τ_i, any of its jobs must be in one of three following cases.

Case A: the system stays in L-mode during its scheduling window.[2]
In this case, it suffices to show that $\theta_i^L \geq u_i^L$:

$$\begin{aligned}\theta_i^L &= \lambda\theta_i \\ &= \lambda(\frac{u_i^L}{\lambda} + u_i^H - u_i^L) \qquad \{\text{by Eq. (5)}\} \\ &= u_i^L + \lambda(u_i^H - u_i^L) \\ &\geq u_i^L \qquad \{\text{since} \quad u_i^H - u_i^L \geq 0\}.\end{aligned}$$

Case B: the system stays in H-mode during its scheduling window.
In this case, it suffices to show that $\theta_i^H \geq u_i^H$:

$$\begin{aligned}\theta_i^H &= \theta_i \\ &= \frac{u_i^L}{\lambda} + u_i^H - u_i^L \qquad \{\text{by Eq. (5)}\} \\ &= u_i^H + \frac{1-\lambda}{\lambda}u_i^L \\ &\geq u_i^H \qquad \{\text{since} \quad 0 \leq \lambda \leq 1\}.\end{aligned}$$

Case C: there is a mode switch (L to H) in its scheduling window.
In this case, the task is executed at a degraded speed θ_i^L before the mode switch and then at a faster speed θ_i^H at and after the mode switch time instant. Let r denote the release time of this job of interest and let $t = r + (C_i^L/\theta_i^L)$. If the mode switch happens after time t, then this job must have finished by the mode switch (thus by its deadline). Otherwise, the mode switch must be triggered earlier at time t when it has cumulatively executed for C_i^L but not finished.

Therefore, in the rest of this proof, we focus on the scenario that the mode switch happens at or before time t. Since $\theta_i^H \geq \theta_i^L$, the later mode switch occurs, the less commutative execution this job can receive within its scheduling window (of fixed total length of T_i). Thus, it suffices to consider just the worst-case situation when mode switch happens exactly at time t. In this worst-case scenario, the deadline of the job of interest must be met if

$$t + \frac{C_i^H - C_i^L}{\theta_i^H} \leq r + T_i \Leftrightarrow \frac{C_i^L}{\theta_i^L} + \frac{C_i^H - C_i^L}{\theta_i^H} \leq T_i.$$

This is mathematically implied by the fluid rate assignment according to Eq. (5) in Algorithm 3 as follows:

$$\begin{aligned}\theta_i = \frac{u_i^L}{\lambda} + u_i^H - u_i^L \Rightarrow \theta_i &= \frac{C_i^L}{\lambda \cdot T_i} + \frac{C_i^H - C_i^L}{T_i} \\ \Rightarrow T_i &= \frac{C_i^L}{\lambda \cdot \theta_i} + \frac{C_i^H - C_i^L}{\theta_i} \\ \Rightarrow T_i &= \frac{C_i^L}{\theta_i^L} + \frac{C_i^H - C_i^L}{\theta_i^H}.\end{aligned}$$

Combining all three cases, the lemma follows. ■

The following theorem establishes the correctness of our schedulablility test for algorithm MCF-FR.

Theorem 4.5. *If* MCF-FR *returns* SUCCESS*, then the fluid rate assignment by* MCF-FR *must be valid and all deadline must be met under* MCF-FR *in both L- and H-modes.*

Proof. By Lemmas 4.1, 4.2, and 4.3, the fluid rate assignment by MCF-FR must be valid. Furthermore, Lemma 4.4 shows that all deadlines of HI-criticality tasks must be met under MCF-FR in both L- and H-modes.

Moreover, all deadlines of LO-criticality tasks must be met under MCF-FR in both L- and H-modes since each LO-criticality task does not change behaviors between the two modes and is assigned a sufficient constant execution rate $\theta_i^L = \theta_i^H = u_i$.

Thus, the theorem follows. ■

4.3 Approximation Ratio Bound

The following theorem establishes an *approximation ratio* for MCF-FR, where an approximation ratio α is defined as: any system that is schedulable under some (potential optimal) algorithm with degraded speed ρ^* must also be schedulable by MCF-FR with degraded speed ρ such that $\frac{\rho}{\rho^*} \leq \alpha$. Clearly $\alpha \geq 1$; the smaller the α, the closer is the approximation to the optimality. Note that the difference between this approximation ratio and a *speedup bound* is that, the full speed remains the same (1.0) with respect to both the approximate algorithm and the optimal algorithm.

Theorem 4.6. *Algorithm* MCF-FR *has an approximation ratio no greater than*

$$\max\left\{\frac{m}{m + U^L - U^H}, \max_i\left\{\frac{1}{1 + u_i^L - u_i^H}\right\}\right\}.$$

Proof. By Eq. (4), a system will be schedulable under MCF-FR given the degraded speed

$$\rho = \lambda = \max\left\{\frac{U^L}{m + U^L - U^H}, \max_i\left\{\frac{u_i^L}{1 + u_i^L - u_i^H}\right\}\right\}. \quad (7)$$

[2]The scheduling window of a job is defined by the time interval from its release time to its absolute deadline.

On the other hand, for a system to be schedulable even under an optimal scheduling algorithm, the degraded speed ρ^* must satisfy

$$m \cdot \rho^* \geq U^L \tag{8}$$

for the system not being overutilized in the L-mode.

Furthermore, it is also necessary to have a sufficient degraded speed that is at least any individual task's LO-utilization for being schedulable even under an optimal scheduling algorithm. This is because any individual sequential task cannot be simultaneously executed on multiple processors. That is,

$$\rho^* \geq u_i^L \tag{9}$$

Therefore, by (7), (8), and (9), we have

$$\begin{aligned}\frac{\rho}{\rho^*} &= \max\left\{\frac{U^L}{\rho^* \cdot (m + U^L - U^H)}, \max_i\left\{\frac{u_i^L}{\rho^* \cdot (1 + u_i^L - u_i^H)}\right\}\right\} \\ &\leq \max\left\{\frac{m}{m + U^L - U^H}, \max_i\left\{\frac{1}{1 + u_i^L - u_i^H}\right\}\right\}.\end{aligned}$$

The theorem follows. ∎

5 EXPERIMENTAL EVALUATION

This section evaluates the proposed virtual-deadline based algorithm fpEDF-VD as well as the dual-rate fluid scheduling algorithm MCF-FR via schedulability ratio on randomly generated task sets.

Workload generation. The implicit deadline sporadic MC task sets are generated randomly using a procedure adopted from [21]. The parameters controlling the workload generation are as follows:

- $m \in [2, 4, 8]$ is the number of cores in the system.
- $\rho \in [0.3, 0.5, 0.7, 0.9]$ is processor's energy-conserving speed.
- U_{bound} is the relative utilization given by U^H/m, where U^H is the total utilization of the system in HI-criticality mode.
- The LO-criticality execution time for each task is randomly chosen in the range $[C_{down}, C_{up}]$.
- The LO-criticality utilization for each task is randomly chosen in the range $[u_i^H/R, u_i^H]$ where R is the upper bound on the ratio of HI-criticality to LO-criticality utilization.
- P: Probability that the chosen task is HI-critical; $0 \leq P \leq 1$

The values of workload generation parameters are $C_{down} = 1$, $C_{up} = 100$, $R = 4$, and $P = 0.5$. The time period and HI-criticality execution time for each task is obtained from LO-criticality execution time and LO-criticality utilization of the respective tasks. The task sets are generated by iteratively adding tasks to the system whose parameters are randomly sampled from their ranges, until the task set reaches the desired relative utilization, U_{bound}. Although the workload generation procedure is randomized, due to the dependent nature of some parameters, the workload might not be completely random so as to meet the workload generation requirements. For evaluation, the total utilization of the workload is normalized to the number (m) of cores and represented as U_{bound}, which represents the per-core utilization. The evaluation metric is the ratio of task sets that meet the schedulability requirements of each algorithm. Each data point in each sub-figure in Figure 2 is based on 1,000 randomly generated task sets.

Results. Figure 2 shows the schedulability ratio for varying relative system utilizations under each combination of m and ρ values. For lower relative system utilization, both approaches have a high schedulability ratio. However, as the per-core utilization increases, the schedulability degrades. It can also be observed that the virtual deadline based fpEDF-VD vastly underperforms compared to the fluid scheduling based MCF-FR. For both algorithms, as ρ increases, the schedulability increases for a fixed value of m. Although the schedulability is improved at higher values of ρ, the benefits associated with reduced processor speed in LO-criticality mode is reduced. For fpEDF-VD, the decreasing performance for a given per-core workload with increasing cores matches the speedup bound $4m/(m + 1)$ stated in Theorem 3.6.

6 RELATED WORK

Several variants of the MC model have been proposed since it was originally introduced by Vestal [34]. A detailed survey of the updated models and results can be found in [16]. Traditionally, LO-criticality jobs were droppped in favor of guaranteeing correctness for HI-criticality jobs [2, 8, 17, 18]. However, recent works have proposed various techniques to avoid dropping LO-criticality tasks but rather gradually degrade the performance or even provide full service guarantees in HI-criticality mode. The first approach of providing degraded service, called imprecise scheduling, was first addressed by the imprecise mixed-criticality (IMC) model [15]; it allocates time budgets to LO-criticality tasks when there is a mode-switch to HI-criticality mode. Other approaches to provide imprecise scheduling propose to reduce the utilization budgets [15, 25, 27, 28] either in the form of reduced execution window, increased period, or dropping some jobs.

Degraded guarantees, although better than no guarantees, are not acceptable for certain applications as pointed out in [19]. To address the shortcomings of service degradation, precise scheduling techniques where full service is guaranteed to LO-criticality tasks has been gaining traction. The schedulability analysis of the IMC model has been studied for both fixed-priority scheduling and EDF-VD [15] and [31], respectively. The authors in [4] considered a generalization of the Vestal model where the less critical functionalities are not entirely discarded even in the HI-criticality mode. In [29] is proposed MC-Fluid, a fluid model-based scheduling algorithm, for the MC tasks in a multiprocessor platform. In the MC-Fluid model, for each task, a criticality dependent execution rate is determined. The authors also proposed MC-DP-Fair, which is an implementable version (on a real-hardware) of MC-Fluid. The authors in [6] derived MCF, a simplified variant of MC-Fluid, which for a dual-criticality system, has a speedup bound no worse than 1.33, improved from 1.618. Considering the adaptive MC- Weakly Hard model, a response time-based schedulability analysis was proposed in [20] that guarantees a minimum service for LO-criticality tasks in case of a mode switch.

Non-functional requirements such as energy consumption and its relationship to the operating frequency of the processors has been a growing concern in non-mixed-crticality systems [12, 23, 24, 33] as well as mixed-criticality systems [9, 22, 26]. One of the earlier works to exploit the DVFS technique for energy minimization by reducing the operating frequency is due to [26]. Using the DVFS technique, the processor is later changed to a higher frequency when needed, such as in HI-criticality mode. The benefit of minimizing overall energy consumption by throttling speeds during

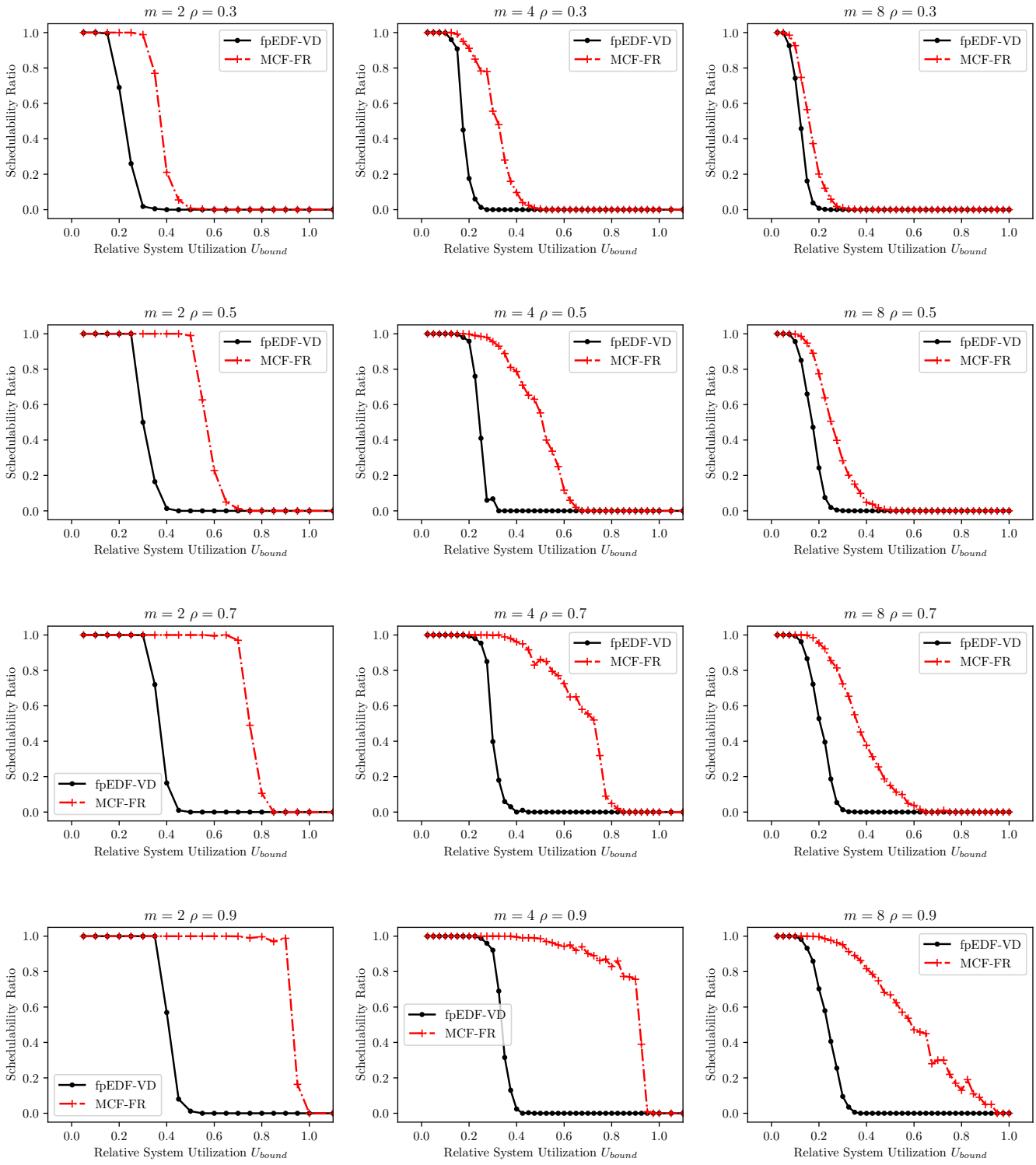

Figure 2: Schedulability ratio comparisons for different values of m and ρ.

runtime has also been established in [26]. A drawback of the approach is that all LO-criticality tasks are degraded in HI-criticality mode. A natural extension to multi-core processors was explored in [5, 32]. The combination of providing precise scheduling on varying speed uniprocessor system has been explored in [13, 35].

7 CONCLUSION

In this paper, we have presented our work on precise MC scheduling on varying-speed multiprocessor platforms. In particular, we have developed two algorithms called fpEDF-VD and MCF-FR. For each algorithm, we have presented a sufficient schedulability test and have shown its correctness. Furthermore, we have proved a speedup bound for fpEDF-VD and an approximation ratio for MCF-FR. To compare these algorithms and demonstrate their effectiveness, we have conducted empirical schedulability studies on randomly generated task systems.

In future, we plan to investigate a tighter speedup bound and approximation ratio. Although we have limited our attention to the scenarios where all processors must operate at the same speed (*i.e.*, all processors at the energy-conserving speed or all processors at the full speed), it would be interesting to explore scenarios where individual processors are able to switch speeds independently. Finally, we plan to conduct further quantitative study on actual energy savings with on-board implementations.

ACKNOWLEDGMENTS

This work was partially supported by NSF grants CNS-1850851, PPoSS-2028481, OAC-1725755, CCF-1659807, CNS-1156712, CNS-1545050, a start-up grant from the University of Central Florida, and start-up and REP grants from Texas State University.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] Sanjoy Baruah, Bipasa Chattopadhyay, Haohan Li, and Insik Shin. 2014. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems* 50, 1 (2014), 142–177.

[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 Baruah and Zhishan Guo. 2014. Scheduling Mixed-Criticality Implicit-Deadline Sporadic Task Systems upon a Varying-Speed Processor. In *2014 IEEE Real-Time Systems Symposium*. 31–40.

[10] Sanjoy K Baruah. 2004. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Trans. Comput.* 53, 6 (2004), 781–784.

[11] Sanjoy K Baruah, Vincenzo Bonifaci, Gianlorenzo D-Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. 2011. Mixed-criticality scheduling of sporadic task systems. In *European Symposium on Algorithms*. Springer, 555–566.

[12] 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.

[13] 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*. 123–132.

[14] Alan Burns. 2019. Multi-Model Systems – an MCS by Any Other Name. In *Proceedings of the 7th International Workshop on Mixed Criticality Systems (WMC)*.

[15] Alan Burns and Sanjoy Baruah. 2013. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems*.

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

[17] Arvind Easwaran. 2013. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Proceedings of the 34th Real-Time Systems Symposium (RTSS), IEEE*. IEEE, 78–87.

[18] 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.

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

[20] Oliver Gettings, Sophie Quinton, and Robert I Davis. 2015. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. ACM, 237–246.

[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*. 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, 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.

[26] 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.

[27] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. 2015. Run and be safe: Mixed-criticality scheduling with temporary processor speedup. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 1329–1334.

[28] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. 2013. Maximizing the execution rate of low criticality tasks in mixed criticality system. *Proc. WMC, RTSS* (2013), 43–48.

[29] 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.

[30] Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott Brandt. 2010. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In *2010 22nd Euromicro Conference on Real-Time Systems*. IEEE, 3–13.

[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] Saad Zia Sheikh and Muhammad Adeel Pasha. 2018. Energy-Efficient Multi-core Scheduling for Hard Real-Time Systems: A Survey. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 6 (2018), 94.

[34] 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)*.

[35] 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 *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.