Precise Mixed-Criticality Scheduling on Varying-Speed Multiprocessors

Sudharsan Vaidhun, *Student Member, IEEE,* Tianning She, *Student Member, IEEE,* Qijun Gu, *Member, IEEE,* Sajal K. Das, *Fellow, IEEE,* Kecheng Yang, *Member, IEEE,* and Zhishan Guo, *Senior Member, IEEE,*

Abstract— While traditional real-time systems analysis requires single pessimistic estimates to represent system parameters, the mixed-criticality (MC) design proposes to use multiple estimates of system parameters with different levels of pessimism, resulting in low critical workloads sacrificed at run-time in order to provide guarantees to high critical workloads. Shortcomings of the MC design were improved recently by the precise MC scheduling technique in which the processor speed is increased at run-time to provide guarantees to both low and high critical workloads. Aiming to extend the precise MC scheduling to multiprocessor computing platforms, this paper proposes three novel scheduling algorithms that are based on virtual-deadline and fluid-scheduling approaches. We prove the correctness of our proposed algorithms through schedulability analysis and also present their theoretical effectiveness via speedup bounds and approximation factor calculations. Finally, we evaluate their performance experimentally via randomly generated task sets and demonstrate that the fluid-scheduling algorithms outperform the virtual-deadline algorithm.

Index Terms—precise scheduling, mixed-criticality real-time systems, varying-speed platform, fluid scheduling, multiprocessors

1 Introduction

For task models in traditional real-time systems, each system parameter is represented by a single value estimate, which results in a large gap between the worst-case and average-case performance on modern computing platforms with complex instruction sets. The adoption of commercial-off-the-shelf multi-core and many-core platforms has further worsened this problem, leading to poor resource utilization and reducing the benefits provided by additional cores. The *mixed-criticality* (MC) design paradigm addresses the resource under-utilization issue by allowing tasks with different levels of criticality with varying levels of guarantees. The MC paradigm uses the classical Vestal's model [43] that introduced tasks of different criticality levels with worst-case execution time estimates calculated from techniques requiring different levels of pessimism. Precisely, higher critical tasks utilize more pessimistic estimates and vice-versa. Such a mechanism prevents under-utilization by under-allocating resources until more resources are necessary. On the other hand, the approaches in [2], [7] proposed multiple estimates for task periods, deadlines, and processor speeds. Recently, the Multi-Model system [18] considered multiple task models with varying levels of guarantees.

- *S. Vaidhun and Z. Guo are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA. E-mail: sudharsan.vaidhun@knights.ucf.edu, zsguo@ucf.edu.*
- *T. She, Q. Gu, and K. Yang are with the Department of Computer Science, Texas State University, San Marcos, TX, USA. E-mail: {t_s374, qijun, yangk}@txstate.edu.*
- *S. K. Das is with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA. E-mail: sdas@mst.edu.*
- *Corresponding Author: Zhishan Guo*

Manuscript received XXXXX XX, 20XX; revised XXXXX XX, 20XX.

In the MC scheduling, the service provided to the lower criticality workloads will be degraded in the higher criticality mode in order to provide guarantees under all modes. Service degradation occurs by either dropping the lower criticality workload altogether or providing only partial service. Such imprecise mixed-criticality scheduling [5], [19], [38] allows the higher critical workloads in higher critical modes of operation to meet more pessimistic resource requirements while gracefully degrading lower critical workloads. However, providing degraded service to non-critical tasks is not applicable to certain safety-critical applications. In such cases, current industrial practices require that guarantees be provided to all tasks despite their level of criticality. The recently proposed *precise mixed-criticality* model [16] aims to provide full computation with no degradation to the tasks in the system under all operating modes and models.

To this end, an important question arises: without sacrificing the low-critical tasks, is it possible to provide guarantees to the high-critical tasks upon mode switch with the additional workload requirements introduced in high-critical mode? A possible solution is to utilize the dynamic voltage and frequency scaling (DVFS) feature in modern hardware. The ability to change the processor frequency during run-time means that the additional workload required to be guaranteed in high-critical mode can be accommodated by increasing the processor speed sufficiently. Since the processor speed can be increased, it will no longer be necessary to drop low-critical tasks thereby addressing the industry needs. However, the processor speed cannot be increased indefinitely and therefore the full (or maximum) speed of the processors must be taken into account in the schedulability test. As an added benefit, Huang et al. [34] with their adoption of DVFS techniques to mixed-criticality scheduling problem have shown the benefits of energy savings. Since overruns are expected to be rare events, it is

beneficial to operate the system at a lower frequency unless deadlines cannot be met. While most of the existing MC designs (e.g., [7], [8], [34]) on varying-speed platforms provide no guarantees to low-critical tasks, the work in [16] combines the precise scheduling of (sporadic implicit-deadline) MC tasks with varying speed platform. For dual-criticality settings, this approach aims to minimize the processor speed under less pessimistic worst-case execution time (WCET) assumptions or models. The deadlines are guaranteed via increased processor speed to the maximum value upon mode switch owing to the overrun of the high-critical task. In [16] and [45], the authors presented virtual-deadlines based on the earliest deadline first (EDF) algorithm and fluid-rate based approaches for precise scheduling of MC tasks on the uniprocessor platform.

To summarize, the *precise MC* model introduced two important advantages over existing approaches: (i) irrespective of the mode of operation, both low- and high-criticality tasks are guaranteed full service throughout operation; and (ii) with the multi-model formulation, the processors can execute at a slower speed if the requirements are guaranteed under less pessimistic assumptions. The reduced speed of operation directly improves energy efficiency, which is an important design consideration in embedded systems.

This work targets integrating DVFS-based energy conservation and precise computing on an (identical) multiprocessor platform for MC tasks. Energy efficiency is achieved by operating the processors at a degraded speed during normal operation and switching to their full speed only in the high-criticality mode. A preliminary version of this work was recently published in a conference [41] where we proposed two independent scheduling techniques along with their theoretical analysis. This work extends the conference version by formulating a mathematical programming problem (MCF-MP) to identify the optimal fluid rates, and comparing the performance of our heuristic algorithms with state-of-the-art methods using randomized workloads. This work also includes additional performance evaluations and a plot to visualize the approximation ratio for MCF-FR.

Contributions. Our work targets varying-speed multiprocessors and MC tasks under preemptive and precise scheduling requirements. Specifically, we

- present a novel mixed-criticality system model and formalize a new MC scheduling problem;
- propose a novel virtual-deadline based scheduler fpEDF-VD and prove a schedulability test that is sufficient only;
- develop two alternative algorithms, namely MCF-MP and MCF-FR, that are both fluid-scheduling based approaches, where MCF-MP leverages nonlinear optimization solvers to achieve the best schedulability while MCF-FR is a sub-optimal approach with a closed form solution and bounded approximation ratio;
- design schedulability experiments to demonstrate (empirically) the effectiveness of our proposed methods and compare their performances.

We now describe the organization of the rest of this paper. Section 2 discusses the system model and formally states the targeted problem. Section 3 presents a virtual-deadline based fpEDF-VD scheduler. Focusing on an alternative fluid-based scheduling framework, Section 4 presents two algorithms, namely a mathematical-programming based approach MCF-MP leveraging known nonlinear optimization solvers, and a polynomial-time algorithm MCF-FR proving an approximation ratio. Section 5 compares the proposed approaches with state-of-the-art methods with a large numbers of task sets that are randomly generated. Section 6 summarizes the related work, and Section 7 draws conclusions with directions of future research.

2 System Model and Problem Statement

We consider a set of n implicit-deadline sporadic MC tasks denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task $\tau_i = (T_i; C_i^L; C_i^H)$ is specified by a 3-tuple as explained below. Each task τ_i releases a sequence of *jobs* with a minimum inter-release separation of T_i time units and every job has an absolute deadline of T_i after its release. The worst-case execution time (WCET) of task τ_i is estimated at two criticality levels: a low-criticality estimate (C_i^L) and a high-criticality estimate ($C_i^H \geq C_i^L$), both on a unit-speed processor. Besides, C_i^L and C_i^H are also the execution requirement budgets of task τ_i in the L- and H-*modes*, respectively. We use $J_{i,j}$ to denote the j^{th} job of task τ_i.

System Model. The MC tasks are to be scheduled on m energy-conserving processors which can operate at either a *degraded* or *full* speed. Given the set τ of n mixed-criticality tasks we consider the problem of scheduling on the varying speed multiprocessor platform. Initially, all m processors operate at a degraded speed $s < 1$. Under this degraded energy-conserving speed s, any workload executed for t time units has consumed a processor budget equivalent to $s \cdot t$ time units under a unit-speed processor. During runtime, the amount of budget consumed by each job of each task is monitored by the scheduler. Each job $J_{i,j}$ is expected to signal completion before it exhausts its low-criticality budget estimate C_i^L (which corresponds to an actual execution time of C_i^L/s time units). If a HI-criticality task does not signal completion, *all* m processors begin to execute at their full speed 1.0. We call this time instant a *mode switch*, when the system increases its processor speed from s (in L-mode) to 1.0 (in H-mode). The system can resume operation in L-mode once *all* processors idle. Note that, no task can exceed their C_i^H budget and doing so is considered *erroneous* behavior and such jobs are terminated. As a consequence, LO-criticality tasks do not trigger a mode change, since $C_i^L = C_i^H$ for LO-criticality tasks.

It is also worth noting that, under the proposed model, no task will suffer from any 'drop' upon a mode switch, as we provide full service guarantee to all jobs under all circumstances. The increased processor speed upon mode switch compensates the additional execution requirement budget $C_i^H - C_i^L$. This differs from most existing works on MC scheduling.

We use u_i^L and u_i^L to denote the utilization of task τ_i in L- and H-modes respectively, where

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

For any LO-criticality task, $u_i^L = u_i^H$ since $C_i^L = C_i^H$.

The overall per-mode system utilizations (of all tasks) are denoted as

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

We use $u_{\max}^L = \max_i \{u_i^L\}$ and $u_{\max}^H = \max_i \{u_i^H\}$ to denote the dominating per-mode utilizations (per task).

Problem Statement. Given a set of MC tasks and m identical processors with speed that may vary, we hope to find a correct scheduling mechanism, such that all timely execution requirements are guaranteed in all circumstances. Specifically,

- all processors must only operate at their (known) energy-conserving speed < 1 if *all* jobs finish within C_i^L budget;
- all processors may operate at their full speed 1.0 if *any* HI-criticality job executes beyond its C_i^L budget, and yet finishes within its C_i^H budget.

3 Scheduling by Virtual Deadlines

This section solves the aforementioned problem by proposing a new scheduler based on the virtual-deadline approach. Specifically, we present fpEDF-VD algorithm, and prove the correctness of precise MC scheduling on m varying-speed processors by developing a sufficient schedulability test.

3.1 Algorithm fpEDF-VD

The proposed fpEDF-VD scheduler combines two existing approaches, namely fpEDF and EDF-VD. Based on the global EDF scheduler, the fpEDF scheduler was developed in [9] by statically prioritizing *heavy* tasks with utilization exceeding 0.5. fpEDF assumes that m identical unit-speed processors satisfy the following sufficient schedulability test.

Theorem 1 (Theorem 4 in [9])**.** *Let U_{sum} denote the total utilization of an arbitrary sporadic task set, in which u_i denoted the utilization of task τ_i. This task set is schedulable by* fpEDF *on m identical unit-speed processors if*

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

As proposed in the EDF-VD scheduler [3], [10], the technique of setting virtual deadlines is to promote the execution of HI-critical tasks in the L-mode and to leave slacks for the potential extra workload at a mode switch from L to H. A scaling factor x determines the virtual deadlines for HI-critical tasks by $\hat{T}_i^0 = x \cdot T_i$. For LO-critical tasks, their virtual deadlines are set identical to their actual deadlines, T_i. Then, the tasks are scheduled according to EDF by their *virtual* deadlines in the L-mode and by their *actual* deadlines once the system is switched to the H-mode.

Let us now present our novel algorithm, called fpEDF-VD that consists of two phases – a pre-processing phase as described in Algorithm 1, and a runtime phase as described in Algorithm 2. In fpEDF-VD, both HI-critical tasks as well as the LO-critical tasks are assigned virtual deadlines calculated 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, the tasks are mapped to a set of non-MC sporadic tasks in the L- and H-modes, 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 for Algorithm fpEDF-VD.

Data: Dual-criticality task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, number of processors m, energy-conserving speed

Calculate scaling factor x

$$x \geq \max\left\{ \frac{u_{\max}^L}{1 - u_{\max}^H}, \frac{U^L}{\frac{m+1}{2} - u_{\max}^H} \right\} \tag{1}$$

if $\max\left\{ \frac{u_{\max}^L}{1 - u_{\max}^H}, \frac{U^L}{\frac{m+1}{2} - u_{\max}^H} \right\} + \max\left\{ u_{\max}^H, \frac{U^H}{\frac{m+1}{2}} \right\} \leq 1$ **then**
  foreach $\tau_i \in \tau$ **do**
    $\hat{T}_i^0 \leftarrow x \cdot T_i$
  end
  return SUCCESS
else
  return FAILURE
end

3.2 Schedulability Test

We derive a sufficient schedulability test for fpEDF-VD in Theorem 2. First, the correctness of our proposed scheduler in the L-mode is established by the following lemma.

Lemma 1. *All tasks must meet their virtual deadlines in the L-mode under* fpEDF-VD*, if*

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

Proof. As shown in Algorithm 2, all tasks in the L-mode is mapped to $\{(T_i^0, C_i^0)\}$, which is scheduled by fpEDF. By Theorem 1, if $\forall i, \frac{C_i^0}{T_i^0} \leq 1$ and $\sum_i \frac{C_i^0}{T_i^0} \leq \frac{m+1}{2}$, then all tasks in $\{(T_i^0, C_i^0)\}$ must meet their deadlines, i.e., all MC tasks in must meet their *virtual* deadlines in the L-mode. Meanwhile, we also have

$$\forall i, \frac{C_i^0}{T_i^0} \leq 1, \quad \forall i, \frac{C_i^L}{xT_i} \leq 1, \quad \forall i, u_i^L \leq x \leq u_{\max}^L,$$

$$\text{and} \quad \sum_i \frac{C_i^0}{T_i^0} \leq \frac{m+1}{2}, \quad \sum_i \frac{C_i^L}{xT_i} \leq \frac{m+1}{2}, \quad \frac{U_i^L}{x} \leq \frac{m+1}{2}, \quad x \geq \frac{U^L}{\frac{m+1}{2}}.$$

Thus, the lemma follows. □

Next, the correctness of fpEDF-VD in the H-mode is established by the following lemma.

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-** processors.

It is equivalent to scheduling the task set $\{(T_i^0; C_i^0)\}_{i=1}^n$ on m **unit-speed** processors, where $T_i^0 = x \ T_i$ and $C_i^0 = C_i^L =$ for every task $_i$,

Then the task set $\{(T_i^0; C_i^0)\}_{i=1}^n$ is scheduled by fpEDF where tasks with $C_i^0/T_i^0 > 0.5$ are considered as heavy tasks.

In the H-mode, fpEDF-VD schedules tasks by actual deadlines on m unit-speed processors.

It is equivalent to scheduling the task set $\{(T_i^{00}; C_i^{00})\}_{i=1}^n$ on m unit-speed processors, where $T_i^{00} = (1 \quad x)T_i$ and $C_i^{00} = C_i^H$ for every task $_i$,

Then, task set $\{(T_i^{00}; C_i^{00})\}_{i=1}^n$ is scheduled by fpEDF where tasks with $C_i^{00}/T_i^{00} > 0.5$ are considered as heavy tasks.

Lemma 2. *A task with either* LO- *or* HI-*criticality will meet its actual deadline in the H-mode under* fpEDF-VD *as far as the following conditions hold:*

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

Proof. In this paragraph, we demonstrate that one can *safely* map the task set under H-mode to another corresponding implicit sporadic one $\{(T_i^{00}; C_i^{00})\}$. This is done by treating the time instant of mode switch (t) as the "last-idle instant" under the classical EDF schedulability analysis. Note that with Lemma 1, jobs released during L-mode must have finished their executions before t if their virtual deadlines are on or before t . In contrast, for jobs with virtual deadlines after t , their actual deadlines must be at least $(1 \quad x)T_i = T_i^{00}$ time units after t — for analysis purposes, we safely assume that no execution to those jobs have begun before the mode switch, such that the analysis of H-mode covers the worst case. While if a job is released in the H-mode, it is sufficient (and safe) to model it as a simple sporadic task set $\{(T_i^{00}; C_i^{00})\}$ due to the following two facts: (i) the job must have a deadline at least T_i time units after its release; (ii) the job cannot execute for more than $C_i^H = C_i^{00}$ in any scenario.

Consider two facts: (i) we apply algorithm fpEDF to schedule the mapped task set $\{(T_i^{00}; C_i^{00})\}$, and (ii) Theorem 1. We can conclude that if $\forall i; \frac{C_i^{00}}{T_i^{00}}$ 1 and $\sum_i \frac{C_i^{00}}{T_i^{00}}$ $\frac{m+1}{2}$, then all tasks in $\{(T_i^{00}; C_i^{00})\}$ must meet their deadlines. That is, all MC tasks in must meet their *actual* deadlines in the H-mode. Meanwhile, we also have

$$\forall i; \frac{C_i^{00}}{T_i^{00}} \quad 1 \ , \ \forall i; \frac{C_i^H}{(1 \quad x)T_i} \quad 1 \ , \ \forall i; u_i^H \quad 1 \quad x \ , \ x \quad 1 \quad u_{\max}^H;$$

and $\sum_i \frac{C_i^{00}}{T_i^{00}} \quad \frac{m+1}{2} \ , \ \sum_i \frac{C_i^H}{(1 \quad x)T_i} \quad \frac{m+1}{2}$

$$, \ \frac{U_i^H}{1 \quad x} \quad \frac{m+1}{2} \ , \ x \quad 1 \quad \frac{U^H}{\frac{m+1}{2}}.$$

Thus, the lemma follows. □

In the following we will prove that Lemmas 1 and 2 yield a sufficient schedulability test for our proposed scheduler.

Theorem 2. fpEDF-VD *correctly schedule a dual-criticality task set* $= \{ _1; \ _2; \ldots; \ _n\}$ *on* m *energy-conserving preemptive processors, each having energy-conserving speed* *and max speed* 1.0, *if*

$$\max \quad \frac{u_{\max}^L}{}; \frac{U^L}{\frac{m+1}{2}} \quad + \max \quad u_{\max}^H; \frac{U^H}{\frac{m+1}{2}} \quad 1. \quad (2)$$

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

$$x = \max \quad \frac{u_{\max}^L}{}; \frac{U^L}{\frac{m+1}{2}} \quad .$$

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

Again, inequality (2) yields

$$x \quad 1 \quad \max \quad u_{\max}^H; \frac{U^H}{\frac{m+1}{2}} \quad ;$$

which implies

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

Thus, in the H-mode, by Lemma 2, all actual deadlines are met under fpEDF-VD. Hence the theorem. □

4 Dual-Rate Fluid Scheduling

Despite being a simple algorithm to implement, in fpEDF-VD, the virtual deadline assignment is a coarse-grained approach. However, pFair class of algorithms are shown to be optimal for multiprocessors [11] for non-MC task models and has the best speedup factor of $(1 + \sqrt{5})/2$ for MC task models. This section focuses on fluid-rate-based scheduling techniques as they belong to pFair class of algorithms and have potential to outperform virtual deadline based approaches. Specifically, we restrict our attention to the so-called *dual-rate fluid* scheduling [6], [37], where each task $_i$ is assigned two constant execution rates: $_i^L$ in L-mode and $_i^H$ in H-mode. Under fluid scheduling, all tasks conceptually progress simultaneously by "fractions" of processor at their constant executing rates (per mode, in our particular context). Such simultaneous progression can be implemented by slicing the timeline into small pieces. Lee et al. has successfully demonstrated such implementation feasibility with MC-DP-Fair [37].

Note that for each LO-criticality task, an execution speed of $_i = u_i$ would be ideal. However, a HI-criticality task will need a faster speed (comparing to its LO-utilization) — this is to provide enough time window beyond the mode switch

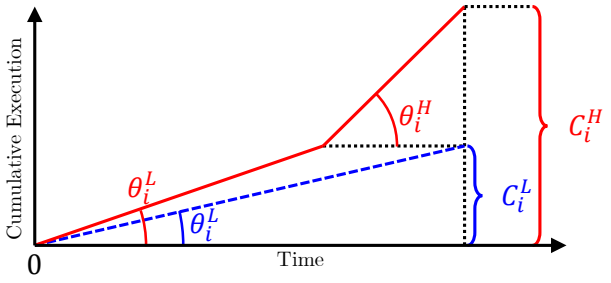

Fig. 1: Relationship between fluid execution rate and cumulative execution over time of a task under MCF framework.

for the additional execution to finish. It should be typical that the execution speed for a HI-criticality task should be larger after the mode switch. Figure 1 demonstrates such a relationship between the execution rates, where the blue dashed parts indicates LO-criticality task setting and the red dashed parts represents HI-criticality task settings under the MC-Fluid (MCF) framework.

4.1 Optimization: Mathematical Programming

We propose MCF-MP algorithm which uses mathematical programming to optimally obtain the fluid-rate pair $(\ _i^L;\ _i^H)$ for each task $\ _i$. Specifically, we investigate the necessary and sufficient schedulability condition for precise MC scheduling on m varying-speed processors under this dual-rate fluid scheduling model. In particular, the pairs $f(\ _i^L;\ _i^H)g_{i=1}^n$ are considered as variables in the optimization formulation, whereas the number of processors, the degraded speed , and the task parameters – execution budget estimates and period are considered as inputs to the problem.

Our goal is to ensure that the sum of assigned rates does not exceed the capacity of the computing platform, and each task, as it is *sequential*, is never assigned a rate greater than 1:0. The temporal correctness of every task can be guaranteed in the precise MC scheduling sense. That is, a set of rate assignment pairs $f(\ _i^L;\ _i^H)g_{i=1}^n$ is *feasible* if and only if

$$\sum_{i=1}^{n} \ _i^L \quad ; \quad 8i : 1 \quad i \quad n \tag{3}$$

$$\ _i^L \quad m; \tag{4}$$

$$\sum_{i=1}^{n} \ _i^H \quad 1{:}0; \quad 8i : 1 \quad i \quad n \tag{5}$$

$$\ _i^H \quad m; \tag{6}$$

and all deadlines are guaranteed to meet in both H- and L-modes.

Let us now discuss how to obtain the constraints on the rate assignment set $f(\ _i^{\text{LO}};\ _i^{\text{HI}})g_{i=1}^n$ to guarantee all deadlines in both H- and L-modes. First, for each task $\ _i$, all of its jobs that are released and have a deadline in L-mode, must meet their deadlines if and only if $C_i^L = \ _i^L \quad T_i; 8i : 1 \quad i \quad n$. That is,

$$\ _i^L \quad u_i^L; \quad 8i : 1 \quad i \quad n: \tag{7}$$

Second, for each task $\ _i$, all of its jobs that are released and have a deadline in H-mode must meet their deadlines if and only if $C_i^H = \ _i^{\text{HI}} \quad T_i; 8i : 1 \quad i \quad n:$ In other words,

$$\ _i^H \quad u_i^H; \quad 8i : 1 \quad i \quad n: \tag{8}$$

Furthermore, while the relationship between $\ _i^L$ and $\ _i^H$ is not restricted to the general idea of dual-rate fluid scheduling (see Footnote 1), Theorem 3 below shows that we can restrict our attention to the *non-decreasing* dual-rate fluid scheduling only, where it is required that

$$\ _i^L \quad \ _i^H; \quad 8i : 1 \quad i \quad n: \tag{9}$$

Theorem 3. *For the precise MC scheduling problem considered herein, any task system that is schedulable under dual-rate fluid scheduling must also be schedulable under* ***non-decreasing*** *dual-rate fluid scheduling.*

Proof. Suppose a system is schedulable under some rate assignment of dual-rate fluid scheduling that does not satisfy expression (9), i.e., $\ _i^L > \ _i^H$ for some i. Then, this system must still be schedulable if we make a rate assignment change by assigning $\ _i^L \quad \ _i^H$. This is because the total rate Constraints (4) and (6) cannot be violated by this assignment that reduces $\ _i^L$; while Constraint (8) implies that a single constant execution rate of $\ _i^H$ in both L- and H-modes for task $\ _i$ (this is the scenario for $\ _i$ after reduction) guarantees that all of its deadlines are met regardless of whether and where the mode switches, because $C_i^L \quad C_i^H$. Thus, applying such rate re-assignment multiple times results in a fluid rate assignment satisfying (9) while not compromising schedulability. □

Therefore, under **non-decreasing** dual-rate assumption, for each task $\ _i$, its job (if any) released in L-mode but executed in H-mode must meet its deadline (in H-mode) if and only if

$$\frac{C_i^L}{\ _i^L} + \frac{C_i^H \quad C_i^L}{\ _i^H} \quad T_i; \quad 8i : 1 \quad i \quad n: \tag{10}$$

This is sufficient because $C_i^L = \ _i^L$ time units after its release is the latest time for the mode switch to be triggered and if the mode switch is triggered earlier by any other job, then the deadline must also be met by (9). This is necessary because any HI-task can be the one that triggers the mode switch and needs to execute exact C_i^H budget. If $\ _i$ is a LO-task (i.e., $C_i^L = C_i^H$), then (10) reduces to (7). Therefore, we can claim $8i$ in inequality (10).

Note that, each of the above Constraints (3)-(10) is "if and only if" and all three possible situations of a job (entirely in L- or H-mode, and across the mode switch time instant) have been exhausted. Therefore, Constraints (3)-(10) are a necessary and sufficient condition for the MC task system on the m varying-speed processors to be schedulable by *any* two-rate fluid scheduling.

With Constraints (3)-(10), we have an optimization problem with linear and linear fractional inequality constraints, where $\ _i^L$ and $\ _i^H$ are variables and all others are problem input constants[1]. Thus, in total, there are $O(n)$ variables,

1. Note that the non-negative rate assignment constraints ($\ _i^L \geq 0 \wedge \ _i^H \geq 0; \forall i$) are implied by inequalities (7) and (8)

Algorithm 3: Algorithm MCF-MP.

Data: Dual-criticality task set $= f_1; _2; \ldots; _n g$, number of processors m, energy-conserving speed , maximum speed 1:0

if *a feasible solution* $f(_i^L; _i^H)g_{i=1}^n$ *subject to Constraints* (3)-(10) *is found* **then**
  in the L-mode, execute each task $_i$ at fluid rate $_i^L$
  in the H-mode, execute each task $_i$ at fluid rate $_i^H$
  return SUCCESS
else
  return FAILURE
end

$O(n)$ linear constraints, and $O(n)$ linear fractional constraints, where n is the total number of tasks. This results in our mathematical programming based algorithm, called MCF-MP, which is summarized in Algorithm 3. Efficient numerical solvers (such as `fmincon` [28] or `CVX` [30] in Matlab) can be used to find a feasible solution.

4.2 Fixed-Ratio Fluid Rates

Although MCF-MP optimally solves the dual-rate fluid scheduling problem, the fractional constraints lead it to be a non-linear optimization, which can take excessive amount of time for the solver to return. Therefore, we present an alternative approach to find a feasible solution $f(_i^L; _i^H)g_{i=1}^n$ by restricting the ratio between the L- and H-modes to be the same for all tasks. That is, each task $_i$ is assigned a execution speed $_i^H = _i$ in the H-mode and a execution speed $_i^L = _i$ in the L-mode where $0 < 1$. The ratio $_i^L = _i^H = 8i$.

Recall that we have simplified notations for per-mode utilization of the whole task set :

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

Let us now present an algorithm MCF-FR (see Algorithm 4) to choose a system-wide parameter and the fluid rates $_i$ for any given MC task system; the schedulability directly depends on whether the resulting can be upper bounded by the degraded speed . Due to these simple condition checks, MCF-FR algorithm and the corresponding schedulability test run in polynomial time. Note that the MCF-FR is only a *sufficient* algorithm for the dual-rate fluid scheduling problem, since there exists a task set for which MCF-MP returns SUCCESS, whereas MCF-FR does not as shown by the following example.

Example 4.1. *Consider a task set with* 5 *tasks with utilizations presented in Table 1. When* $= 0:3$ *and* $m = 2$*, according to* MCF-FR*, Equation* (11) *results in* $= 0:31676 >$ *and therefore Algorithm 4 returns* FAILURE*. However,* MCF-MP *successfully returns the fluid rates presented in Table 1 that satisfy the Constraints* (3)-(10).

In the rest of this section, we show that by selecting and $_i$ according to MCF-FR in Algorithm 4, if , then the system is schedulable under MCF-FR, i.e., the resulting $f(_i^L; _i^H)g_{i=1}^n$ is guaranteed to satisfy constraints (3)-(10).

Algorithm 4: Algorithm MCF-FR.

Data: Dual-criticality task set $= f_1; _2; \ldots; _n g$, number of processors m, energy-conserving speed , maximum speed 1:0

A system-wide parameter and per-task parameters $_i$ are computed as:

$$\max \left(\frac{U^L}{m + U^L \quad U^H}; \max_i \frac{u_i^L}{1 + u_i^L \quad u_i^H} \right) \tag{11}$$

$$8i; \quad _i = \frac{u_i^L}{} + u_i^H \quad u_i^L \tag{12}$$

if **then**
  in the L-mode, execute each task $_i$ at fluid rate $_i^L$ $_i$
  in the H-mode, execute each task $_i$ at fluid rate $_i^H = _i$
  return SUCCESS;
else
  return FAILURE
end

i	Criticality	u_i^L	u_i^H	$_i^L$	$_i^H$
1	HI	0:128057	0:287319	0:1537	0:9789
2	HI	0:089914	0:144498	0:1009	0:5309
3	LO	0:111853	0:111853	0:1125	0:1420
4	HI	0:006206	0:036006	0:0113	0:0921
5	LO	0:220324	0:220324	0:2210	0:2445
			Summation	0:5994	1:9884

TABLE 1: A feasible set of fluid rates using MCF-MP where $n = 5$, $= 0:3$, $m = 2$

Note that, by Eq. (11), we have

$$0 < \quad 1 \tag{13}$$

because $U^H \quad m$ and $u_i^H \quad 1$, must hold for all i; otherwise, deadlines cannot be guaranteed by any algorithm due to overutilization.

Lemma 3. *If* MCF-FR *returns* SUCCESS*, then Constraints* (3) *and* (5) *must be true.*

Proof. By Eq. (12),

$$_i = \frac{u_i^L}{} + u_i^H \quad u_i^L$$

Eq. (11) implies $\frac{u_i^L}{1+u_i^L \quad u_i^H}$, so

$$\begin{aligned} _i & \frac{u_i^L}{\frac{u_i^L}{1+u_i^L \quad u_i^H}} + u_i^H \quad u_i^L \\ & = 1 + u_i^L \quad u_i^H + u_i^H \quad u_i^L \\ & = 1 \end{aligned}$$

Furthermore, MCF-FR returning SUCCESS means that . Therefore, $_i$. Since MCF-FR assigns $_i^L = _i$ and $_i^H = _i$, Constraints (3) and (5) clearly hold. □

Lemma 4. *If* MCF-FR *returns* SUCCESS*, then Constraints* (4) *and* (6) *must be true.*

Proof. According to Eq. (12),

$$\sum_i \alpha_i = \frac{\sum_i u_i^L}{} + \sum_i u_i^H \quad \sum_i u_i^L$$
$$= \frac{U^L}{} + U^H \quad U^L:$$

Whereas, Eq. (11) implies $\frac{U^L}{m+U^L \quad U^H}$, so

$$\sum_i \frac{U^L}{\frac{U^L}{m+U^L \quad U^H}} + U^H \quad U^L$$
$$= m + U^L \quad U^H + U^H \quad U^L$$
$$= m:$$

Since MCF-FR returning SUCCESS means , we have

$$\sum_i (\quad _i) \qquad m:$$

Moreover, MCF-FR assigns $_i^L =$ $_i$ and $_i^H =$ $_i$. Hence the lemma. □

Lemma 5. *In* MCF-FR*, Inequalities* (7), (8) *and* (9) *must be true.*

Proof. From Eq. (12), we obtain

$$_i^H = \ _i = u_i^H + \ \frac{1}{} \quad 1 \quad u_i^L;$$

$$_i^L = \quad _i = u_i^L + \quad (u_i^H \quad u_i^L):$$

With the help of Inequality (13) and the fact $u_i^H \quad u_i^L > 0$, we have $_i^H \quad u_i^H$ and $_i^L \quad u_i^L$, which are Inequalities (7) and (8). Moreover, since $_i^L = \ _i^H$, Inequality (13) implies $_i^L \quad _i^H$ which is (9). Thus, the lemma follows. □

Lemma 6. *In* MCF-FR*, Inequality* (10) *must be true.*

Proof.

$$\text{Inequality (10)} \ , \ \frac{C_i^L}{_i^L} + \frac{C_i^H \quad C_i^L}{_i^H} \quad T_i$$
$$, \ \frac{C_i^L}{_i} + \frac{C_i^H \quad C_i^L}{_i} \quad T_i$$
$$, \ _i \quad \frac{C_i^L}{T_i} + \frac{C_i^H \quad C_i^L}{T_i}$$
$$, \ _i \quad \frac{u_i^L}{} + u_i^H \quad u_i^L;$$

which must hold because of Eq. (12). Hence the lemma. □

The next theorem establishes the correctness of algorithm MCF-FR.

Theorem 4. *If* MCF-FR *returns* SUCCESS*, then the resulting fluid rates* $f(_i^L; _i^H)g_{i=1}^n$ *must be a feasible solution for the precise MC scheduling problem.*

Proof. In Section 4.1 we established Constraints (3)-(10) as a necessary and sufficient condition for for the precise MC scheduling problem. from Combining Lemmas 3, 4, 5, and 6, this theorem follows. □

Now we calculate an *approximation ratio* for MCF-FR which is defined as follows. Any system that is schedulable under some (potentially optimal) algorithm with degraded speed must also be schedulable by MCF-FR with degraded speed such that — . Here 1 and therefore, the lower the , the closer the approximation is to optimality. The difference between the approximation ratio and a *speedup bound* is that, the full speed remains the same (1.0) with respect to both the approximate and optimal algorithms.

Theorem 5. *Algorithm* MCF-FR *has an approximation ratio no greater than*

$$\max \ \frac{m}{m + U^L \quad U^H}; \max_i \ \frac{1}{1 + u_i^L \quad u_i^H} \quad :$$

Proof. By Eq. (11), a system will be schedulable under MCF-FR given the degraded speed

$$= \quad = \max \ \frac{U^L}{m + U^L \quad U^H}; \max_i \ \frac{u_i^L}{1 + u_i^L \quad u_i^H} \quad : \tag{14}$$

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

$$m \qquad U^L \tag{15}$$

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

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

$$u_i^L \tag{16}$$

Therefore, by Expressions (14), (15), and (16), we have

$$— = \max \ \frac{U^L}{(m + U^L \quad U^H)}; \max_i \ \frac{u_i^L}{(1 + u_i^L \quad u_i^H)}$$
$$\max \ \frac{m}{m + U^L \quad U^H}; \max_i \ \frac{1}{1 + u_i^L \quad u_i^H} \quad :$$

Therefore, the theorem follows. □

5 Performance Evaluation

In this section, we compare and contrast the performance of the virtual-deadline based algorithm fpEDF-VD and the dual-rate fluid scheduling algorithms MCF-MP and MCF-FR under randomly generated workloads.

Workload Generation. The task sets were generated by replacing the UUnifast algorithm [17] with its multiprocessor version, the UUnifast-discard algorithm [22]. For a target utilization U and n tasks, UUnifast returns a set of n utilizations. We adapt this procedure by providing the per-core utilization as input and then scale the resulting utilizations by the number of cores m. If the resulting set of utilizations are all 1, then we return the utilizations, else discard and repeat the process until we return.

$n \ 2 \ f20; 40; 60; 80; 100g$ is the number of cores in the system.

$m \ 2 \ f2; 4; 8g$ is the number of cores in the system.

$\ 2 \ f0:3; 0:5; 0:7; 0:9g$ is the energy-conserving speed of the processors.

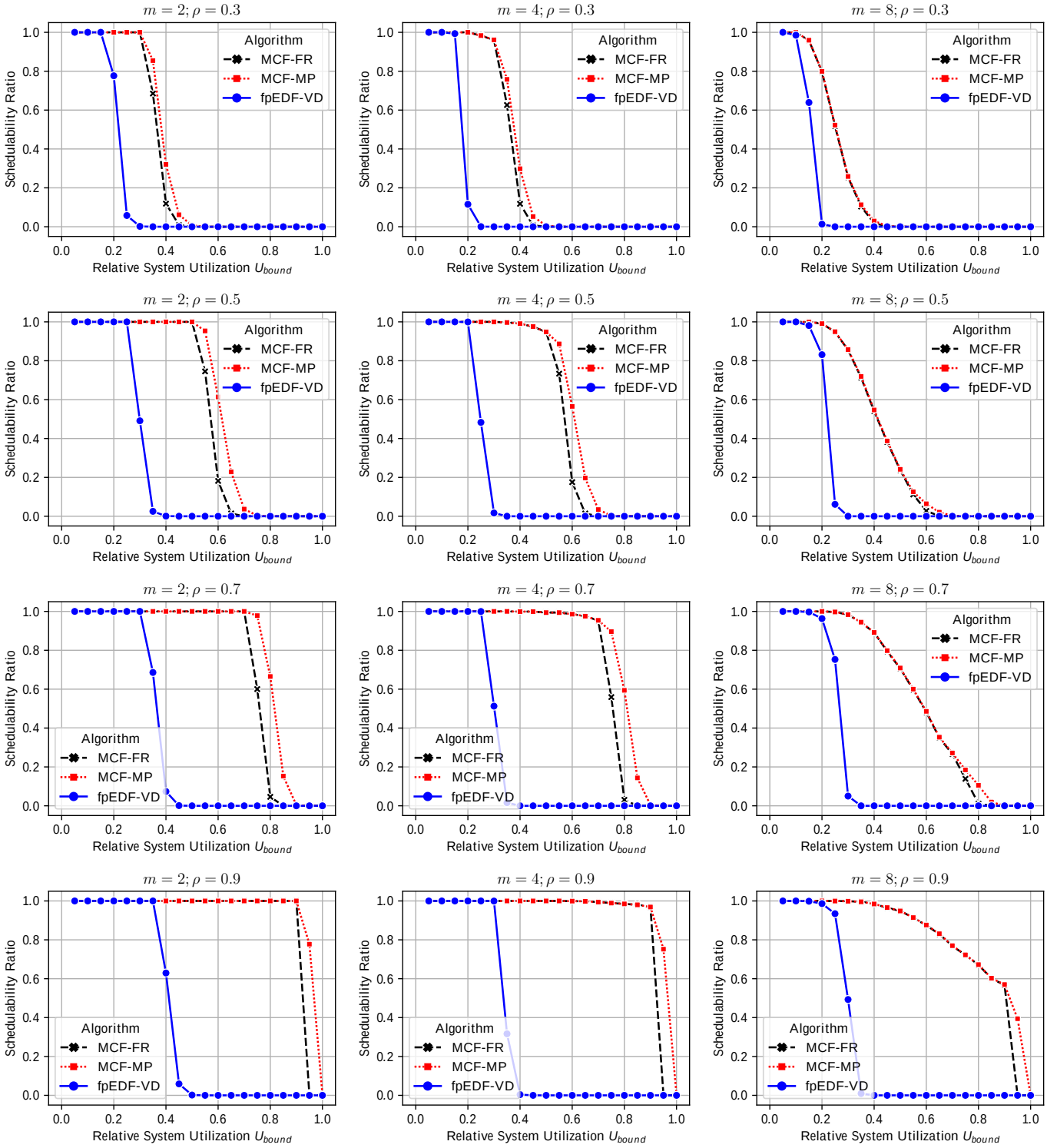

Fig. 2: A comparison of schedulability ratios of the three scheduling algorithms under different values of m and , where $n = 20$.

$U_{bound} = U^H/m$ is the relative utilization which denotes the per-core utilization, and $U^H = \sum_i u_i^H$ is the total utilization of the system in HI-criticality mode.

The LO-criticality execution time C_i^L for each task is randomly chosen in the range $[C_{down}; C_{up}]$.

R: The LO-criticality utilization for each task is randomly chosen in the range $[u_i^H/R; u_i^H]$, where $R \geq 1$.

P: the probability that the chosen task is HI-critical such that $0 < P < 1$.

The specific parameters chosen for workload generation are as follows $C_{down} = 1$, $C_{up} = 100$, $R = 4$, and $P = 0.5$. With these range limits and constants, we generate 2000 randomly generated task sets. The other workload parameters are derived from the parameters as follows. The LO-criticality utilization is calculated based on the R parameter described above. The LO-criticality execution time

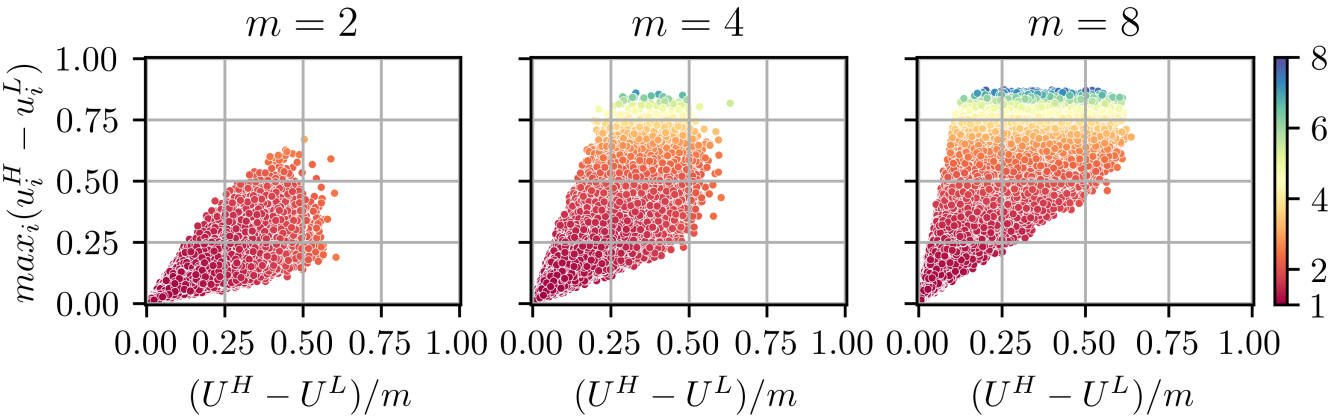

Fig. 3: Scatter plot showing the approximation ratio for the generated task sets, where $n = 20$

	Number of tasks (n)				
Number of cores (m)	20	40	60	80	100
2	0.210	0.344	0.705	0.907	1.310
4	0.193	0.405	0.628	0.931	1.187
8	0.177	0.410	0.627	0.838	1.223

TABLE 2: Average computation time using MCF-MP

is randomly chosen in the range $[C_{down}; C_{up}]$. The time period and HI-criticality execution time for each task are calculated from the LO-criticality execution time and LO-criticality utilization of the corresponding tasks. We note that even though we generate the workload in a randomized manner, the resulting task model parameters are not truly random due to the inherent dependency between individual task parameters and the need to satisfy the task generation conditions mentioned earlier. For presenting the evaluation results, the total utilization of the workload is normalized to the number of cores m denoted by U_{bound}. In other words, U_{bound} represents the per-core utilization. Then, we evaluate the performance in terms of the ratio of task sets that meet the schedulability requirements for each proposed algorithm. The mathematical programming formulation used in MCF-MP is implemented using the CVXPY tool [1], [24] written in Python. The source code for the experiments have been published for reproducibility. [2]

Experimental Results. Figure 2 shows the schedulability ratio for varying relative system utilizations under each combination of m and values. Among the two fluid-scheduling algorithms, we observe that MCF-MP upper-bounds the MCF-FR for all schedulability experiment settings, which can be attributed to the optimality of the MCF-MP algorithm. Although sufficient, it is observed that MCF-FR closely follows MCF-MP despite being a polynomial-time schedulability test. On the other hand, the virtual-deadline based algorithm fpEDF-VD performs considerably worse with poor schedulability ratios. For lower relative system utilization, all three approaches have high schedulability ratios. However, as the number m of processors increases along with a proportional workload increase, the schedulability degrades. Conversely, as increases, the schedulability increases for a fixed value of m.

2. Source Code available at https://doi.org/10.24433/CO.6807714.v2

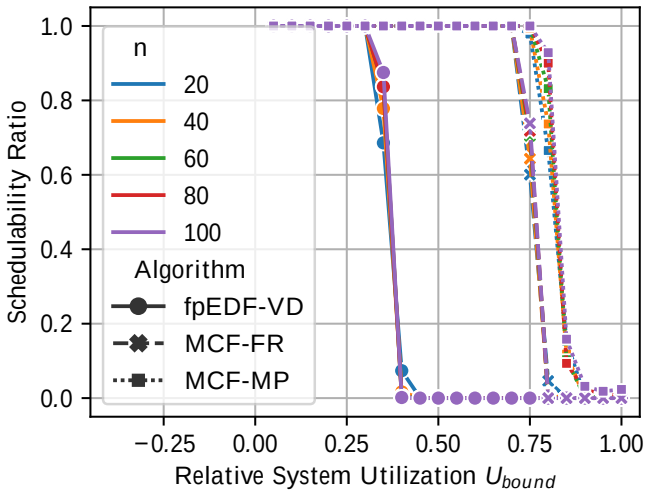

Fig. 4: Varying the number of tasks for $m = 2$, $= 0.7$

Figure 4 shows the effect of number of tasks on the schedulability ratio for all three algorithms. We notice that the number of tasks does not have a significant influence on the overall schedulability ratio. Although we have only presented the case for $m = 2$ and $= 0.7$, the other cases have a similar pattern and we omitted those for brevity.

Furthermore, to visualize the theoretical approximation ratio derived in Theorem 5, in Figure 3 we plot the approximation ratio using a scatter plot. The x- and y-axis of Figure 3 are chosen to meaningfully capture the two terms that determine the approximation ratio. For each generated task set, the color of the marker in the scatter plot is used to show the approximation ratio and can be interpreted through the color bar included in the figure.

In Table 2, we also present the average computation times when running MCF-MP on the Stokes high performance computing cluster at UCF. While the experiments were run on multiple compute nodes simultaneously, each instance of MCF-MP ran as a single task on the cluster with no parallelization. We notice that increasing the number of cores m does not drastically affect the computation, but

surprisingly reduces the computation time. We believe this is due to Conditions (4) and (6) being relaxed as m increases. On the other hand, increasing n increases the computation time. Regardless, with $m = 8$ and $n = 100$, the computation takes less than 1:5 seconds.

Overall Performance. From our experiments, we observe the following.

MCF-MP *outperforms* MCF-FR. Since both the algorithms follow fluid scheduling, this can be explained by the optimality of the mathematical programming approach.

MCF-MP *outperforms* fpEDF-VD. In our experiments, every task set schedulable by fpEDF-VD is also schedulable by MCF-MP; however, there exist some task sets that are schedulable by MCF-MP but are not schedulable by fpEDF-VD. Due to the complexity of the MCF-MP algorithm, we were unable to search for a case where fpEDF-VD can schedule a task set that is deemed unschedulable by MCF-MP.

While MCF-MP outperforms the other two approaches at the cost of significant computations, experiments show that MCF-FR has a closer-to-optimal performance among the fluid scheduling approaches, most notable at higher m values. Since fpEDF-VD and MCF-FR are both polynomial-time algorithms, we generated 1 million task sets to identify a task set where fpEDF-VD can schedule a task set that fails the MCF-FR test, and found no such case.

Although we have not provided theoretical guarantees for the above observations, the schedulability experiments are based on a randomized task set and suggests a general trend.

6 Related Work

In the literature, several variations of MC models have been proposed since its original introduction by Vestal [43]. The survey by Burns and Davis [20], [21] provides a detailed summary of the models and their results. Since the introduction by Vestal, most works on mixed-criticality scheduling provide no guarantees to LO-criticality tasks in H-mode [4], [8], [25], [26]. Burns and Baruah [19] proposed IMC model, an alternative to traditional MC model that reduces the priority of LO-criticality tasks while still allowing their execution in H-mode. Both fixed-priority [19] and EDF-VD [38] scheduling approaches have been studied for the IMC model. Baruah et al. generalized the Vestal model such that the lower criticality tasks are not completely discarded in the higher critical modes.

Moving away from providing no guarantees, recent works have attempted to provide degraded guarantees to LO-criticality tasks in H-mode while still providing full guarantees in L-mode. We categorize such works where at least degraded guarantees are provided as imprecise MC. Reducing the utilization budgets [19], [23], [33], [35], [36] in H-mode by means of reducing execution time, increasing task periods, or by dropping selected jobs. The adaptive MC Weakly hard model [29] proposed by Gettings et al. imposes weakly-hard constraints on LO-criticality tasks in H-mode. For multiprocessor systems, Xu and Burns propose a technique to move LO-criticality tasks to a processor not experiencing a mode change [44]. Santy et al. [40] propose a technique to suspend LO-criticality tasks only if not suspending can cause a HI-criticality task to miss its deadline. Along the same vein, Bate et al. propose a bailout protocol [12], [13] which aims to return the system to L-mode as soon as possible after a switch to H-mode. This is then further extended in [14] to delay the switch to H-mode. Although better than providing no guarantees, degraded guarantees are not applicable to some applications and the criticisms against such a method was presented in [27]. In contrast to service degradation, precise scheduling techniques provide full service guarantees in all modes of operation to all tasks in the system. We categorize such works as precise MC. Recently, a precise MC model was proposed where full guarantees were provided in both criticality modes of operation on a varying speed uniprocessor [16].

To solve MC problems, Lee et al proposed the MC-Fluid scheduling algorithm for multiprocessor systems, where the rate of execution of a task depends on its criticality [37]. A more practical version of MC-Fluid, the MC-DP-Fair algorithm was also proposed so that fluid model-based scheduling techniques can be implemented on hardware. Baruah et al. simplified the fluid scheduling algorithm with their MCF algorithm which has a speedup bound no more than 1.33 [6] for the dual-criticality case. Finally, the speedup bound is improved for MC-Fluid from 1.618 to 1.33.

Energy consumption in embedded systems is a growing concern and several works [15], [31], [32], [42] have proposed scheduling techniques for non-mixed-criticality systems. The DVFS feature of processors allows runtime changes to processor speed and can be exploited to reduce energy consumption [34]. Extensions to multiprocessor platforms was proposed in [39]. Using DVFS to provide precise scheduling to MC tasks was recently explored in [16], [45].

To summarize, (1) existing works on MC either allow partial or full degradation of LO-criticality tasks in H-mode; (2) the DVFS capability has been exploited for purposes of energy minimization in MC systems. Our work allows precise computation for LO-criticality tasks in both modes on multiprocessor systems, while minimizing energy during their normal (and expected) operation in L-mode. This work is a stepping stone towards maximizing the potential of energy-saving architectures such as big.LITTLE for real-time applications.

7 Conclusion

This paper proposed a framework for precise mixed-criticality (MC) scheduling on varying-speed multiprocessor platforms. Specifically, we presented three novel algorithms. The first algorithm fpEDF-VD is based on virtual-deadlines and we proved a speedup bound for the algorithm. Next, we proposed MCF-MP, a mathematical programming formulation to calculate the optimal fluid rate for a fluid-rate based scheduling approach. With the optimal rate known, we proposed MCF-FR which is a polynomial time algorithm and we calculated its approximation ratio. For each algorithm, we provided its correctness and a sufficient schedulability test. In our empirical schedulability study using randomly generated tasks, we observe that MCF-FR despite being a

polynomial time algorithm has a closer-to-optimal performance among the fluid scheduling approaches and even outperforms the fpEDF-VD algorithm in terms of schedulability ratio.

In future, we plan to investigate if the proposed speedup bound and approximation ratio are tight; else how to improve them. While we have considered that the processors operate at the same speed (energy-conserving or full speed), we will extend to the case where the processors can switch their speeds independently. Further, we will formally prove the dominance relationships among these algorithms or identify counter-examples to show non-dominance. Finally, using on-board implementations, we plan to evaluate energy savings on real-world conditions.

ACKNOWLEDGMENTS

This work was partially supported by the NSF grants under award numbers CNS-1156712, CCF-1659807, OAC-1725755, CNS-1850851, PPoSS-2028481, CNS-2104181, and OAC-2104078 and start-up grants from Texas State University and University of Central Florida.

REFERENCES

[1] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[2] S. Baruah. Schedulability analysis for a general model of mixed-criticality recurrent real-time tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 25–34. IEEE, 2016.

[3] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 145–154. IEEE, 2012.

[4] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM (JACM)*, 62(2):14, 2015.

[5] S. Baruah, A. Burns, and Z. Guo. 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*, pages 131–138. IEEE, 2016.

[6] S. Baruah, A. Easwaran, and Z. Guo. Mc-fluid: simplified and optimally quantified. In *2015 IEEE Real-Time Systems Symposium*, pages 327–337. IEEE, 2015.

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

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

[9] S. K. Baruah. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, 2004.

[10] S. K. Baruah, V. Bonifaci, G. D-Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *European Symposium on Algorithms*, pages 555–566. Springer, 2011.

[11] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[12] I. Bate, A. Burns, and R. I. Davis. A bailout protocol for mixed criticality systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 259–268. IEEE, 2015.

[13] I. Bate, A. Burns, and R. I. Davis. An enhanced bailout protocol for mixed criticality embedded software. *IEEE Transactions on Software Engineering*, 43(4):298–320, 2016.

[14] I. Bate, A. Burns, and R. I. Davis. Analysis-runtime co-design for adaptive mixed criticality scheduling. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 187–200. IEEE, 2022.

[15] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong. Energy-efficient real-time scheduling of dag tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(5):84, 2018.

[16] A. Bhuiyan, S. Sruti, Z. Guo, and K. Yang. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, pages 123–132, 2019.

[17] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154, 2005.

[18] A. Burns. Multi-model systems – an mcs by any other name. In *Proceedings of the 7th International Workshop on Mixed Criticality Systems (WMC)*, 2019.

[19] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems*, 2013.

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

[21] A. Burns and R. I. Davis. *Mixed Criticality Systems - A Review: (13th Edition, February 2022)*. Feb. 2022. This is the 13th version of this review now updated to cover research published up to the end of 2021.

[22] R. I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[23] R. I. Davis, A. Burns, and I. Bate. Compensating adaptive mixed criticality scheduling. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, pages 81–93, 2022.

[24] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

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

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

[27] R. Ernst and M. Di Natale. Mixed criticality systems - A history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[28] https://www.mathworks.com/help/optim/ug/fmincon.html.

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

[30] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, Mar. 2014.

[31] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan. Energy-efficient real-time scheduling of dags on clustered multi-core platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 156–168. IEEE, 2019.

[32] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong. Energy-efficient multi-core scheduling for real-time dag tasks. In *29th Euromicro conference on real-time systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[33] Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 373–383. IEEE, 2018.

[34] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. Energy efficient dvfs scheduling for mixed-criticality systems. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2014.

[35] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. Run and be safe: Mixed-criticality scheduling with temporary processor speedup. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1329–1334. IEEE, 2015.

[36] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. *Proc. WMC, RTSS*, pages 43–48, 2013.

[37] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *2014 IEEE Real-Time Systems Symposium*, pages 41–52. IEEE, 2014.