MC-Fluid: Multi-core Fluid-based Mixed-Criticality Scheduling

Lee, Jaewoo; Ramanathan, Saravanan; Phan, Kieu-My; Easwaran, Arvind; Shin, Insik; Lee, Insup

2017

Lee, J., Ramanathan, S., Phan, K.-M., Easwaran, A., Shin, I., & Lee, I. (2017). MC-Fluid: Multi-core Fluid-based Mixed-Criticality Scheduling. IEEE Transactions on Computers, in press.

https://hdl.handle.net/10356/85847

https://doi.org/10.1109/TC.2017.2759765

MC-Fluid: Multi-core Fluid-based Mixed-Criticality Scheduling

Jaewoo Lee* Saravanan Ramanathan‡ Kieu-My Phan† Arvind Easwaran‡ Insik Shin§ Insup Lee*
Dept. of Computer and Information Science, University of Pennsylvania, USA*
College of Information and Computer Sciences, University of Massachusetts, Amherst, USA†
School of Computer Science and Engineering, Nanyang Technological University, Singapore‡
School of Computing, KAIST, South Korea§
E-mail: insik.shin@cs.kaist.ac.kr

***Abstract*—Owing to growing complexity and scale, safety-critical real-time systems are generally designed using the concept of mixed-criticality, wherein applications with different criticality or importance levels are hosted on the same hardware platform. To guarantee non-interference between these applications, the hardware resources, in particular the processor, are statically partitioned among them. To overcome the inefficiencies in resource utilization of such a static scheme, the concept of mixed-criticality real-time scheduling has emerged as a promising solution. Although there are several studies on such scheduling strategies for uniprocessor platforms, the problem of efficient scheduling for the multiprocessor case has largely remained open. In this work, we design a fluid-model based mixed-criticality scheduling algorithm for multiprocessors, in which multiple tasks are allowed to execute on the same processor simultaneously. We derive an exact schedulability test for this algorithm, and also present an optimal strategy for assigning the fractional execution rates to tasks. Since fluid-model based scheduling is not implementable on real hardware, we also present a transformation algorithm from fluid-schedule to a non-fluid one. We also show through experimental evaluation that the designed algorithms outperform existing scheduling algorithms in terms of their ability to schedule a variety of task systems.**

***Index Terms*—C.3.d Real-time and embedded systems, C.4.g Measurement, evaluation, modeling, simulation of multiple-processor systems**

I. Introduction

Safety-critical real-time systems are becoming increasingly complex systems with several applications of different importance or criticality. Avionics with its Design Assurance Levels A through E (A being the most critical and E the least) [1], and automotive with its Automotive Safety Integrity Levels A through D (D being the most critical and A the least) [2], are two real-world examples of such systems. Recently, there has been a growing attention towards Mixed-Criticality (MC) real-time systems that integrate multiple applications with different criticality levels on a single hardware platform. Integrated Modular Avionics (IMA) [3] is one of the prominent examples of such an integrated system in the industry. A key challenge in MC systems is to eliminate any unintended interference for critical applications from the less critical ones in terms of access to shared hardware resources.

In this paper, we focus on the sharing of a central resource, the processor, specifically a multicore processor (multiprocessor) given their growing popularity in embedded platforms. When assessing the feasibility of using a processor sharing algorithm (scheduling algorithm), real-time systems require assurance of deadline satisfaction even in the worst-case, that is when interference from higher priority applications is at its maximum. As a result, Worst-Case Execution Time (WCET) estimates for applications are used in this assessment. However, due to software and hardware complexities, deriving exact WCET estimates is often difficult and hence pessimistic upper bounds are used. The degree of pessimism depends on the desired safety assurance level, which in turn depends on the criticality level of the application. A more critical application (e.g., level A in avionics or level D in automotive) generally requires a higher safety assurance level because failure to meet deadlines can have serious consequences. Hence, in order to satisfy the certification authorities, it is a common practice to use highly pessimistic WCET estimates when assessing critical applications. However, from a system designers perspective, this level of pessimism leads to severe under-utilization of the processing capacity of the platform. Therefore, it is highly desirable to minimize this WCET pessimism to improve efficiency, while still guaranteeing the required processing capacity for critical applications.

Several MC scheduling algorithms have been proposed that enable the use of less pessimistic (or even optimistic) execution estimates by operating the system in two modes (see [4] for a detailed review). In the normal mode of operation, processing capacity is reserved for application tasks based on the less pessimistic or optimistic estimates, and all task deadlines are guaranteed to be met. In the eventuality that a critical application task requires additional processing capacity, it is given preference in lieu of a less critical application. That is, the mode of operation changes to critical, and deadline guarantees in this mode depend on application criticality. Thus, system efficiency is improved by using less stringent execution requirements in the normal mode, while the mode switch guarantees the required processing capacity for critical applications. Although many results are published for single processor MC scheduling including algorithms with optimal speedup factors[1], multiprocessor MC scheduling has received much less attention.

Existing multiprocessor MC scheduling algorithms can be broadly classified into two categories, *global* in which application tasks can migrate between the processors at runtime, and *partitioned* in which tasks are statically assigned to individual processors. In this paper, we focus on global scheduling, and in particular consider the *fluid scheduling model* [6]–[8]. Under fluid scheduling, a task can execute using fractional (< 1) execution rates on a hypothetical processor, thus enabling multiple tasks to make progress simultaneously on a single processor.

[1]Speedup factor of an algorithm is defined as the additional processor speed required to schedule all feasible task sets using that algorithm [5].

This execution flexibility often leads to scheduling algorithms with vastly improved schedulability performance, as is evident in the domain of non mixed-criticality multiprocessor real-time scheduling, where several optimal fluid scheduling algorithms have been developed [7], [9]–[11]. Inspired by these results, we develop a new fluid scheduling algorithm, called *MC-Fluid*, for multiprocessor MC real-time systems in this work. Taking into account the fact that MC systems operate in two modes with different execution requirements, the proposed algorithm allows two different fractional execution rates for each task. Since fluid scheduling cannot be implemented on a real processor, we also derive a mapping from the fluid algorithm to a non-fluid one (execution rate exactly 1).

There have been different rate assignment algorithms for MC-Fluid [12]–[14]. In [12], we presented the OERA (Optimal Execution Rate Assignment) algorithm to determine the optimal rates by solving a convex optimization problem in polynomial time. Recently, Baruah et al. [13] presented a simplified rate assignment algorithm called MCF with linear-time complexity, and also showed that MCF has an optimal speedup factor of 4/3 [15]. Since OERA dominates MCF in terms of schedulability, OERA is also speedup-optimal. In [14], we presented the MC-Slope rate assignment algorithm with linearithmic ($O(n \log n)$) complexity.

This paper is an extended version of our earlier work [12], [14]. In [12], we presented an optimal rate assignment algorithm under two rate scheduling. In this paper, we present another optimal rate assignment algorithm, called MC-Derivative, which is extended from MC-Slope in [14]. While OERA relies on convex optimization and has $O(n^2)$ time complexity, MC-Derivative relies on the first derivative principles to determine the execution rates and has $O(n \log n)$ time complexity. MC-Derivative is motivated by the fact that OERA has little flexibility in scalability (e.g., the extension to multi-criticality systems). In addition, MC-Derivative with a smaller time complexity than OERA is more suited for online admission control wherein tasks arrive at runtime and thus the execution rates need to be computed dynamically. The comparison among MC-Fluid, MCF, and MC-Derivative is summarized in Table I.

In spite of the advantages of fluid-based scheduling, it is not directly implementable on real discrete-time platforms due to its unrealistic assumption that one processor can execute multiple tasks at the same time. To tackle this, we present the MC-Discrete algorithm, which maps the MC-Fluid schedule to a discrete-time schedule for real hardware platforms. While the earlier MC-DP-Fair [12] constructs a non-fluid schedule with real-number execution times and deadlines, MC-Discrete constructs a non-fluid schedule with discrete execution times and deadlines, which is preferred in practice. **Contributions.** Our contributions are summarized as follows:

- We present a fluid model-based multiprocessor MC scheduling algorithm, called MC-Fluid, with criticality-dependent execution rates per task (Sec. III) and analyze its exact schedulability (Sec. IV).
- We propose MC-Derivative rate assignment algorithm that optimally assigns execution rates based on the first derivative principles and has $O(n \log n)$ complexity (Sec. VI).
- We present MC-Discrete algorithm, which constructs a discrete-time schedule from MC-Fluid schedule (Sec. VII).
- Simulation results show that MC-Fluid and MC-Discrete outperform other existing approaches (Sec. VIII).

TABLE I
THE SPEEDUP-OPTIMAL RATE ASSIGNMENT ALGORITHMS

Algorithm	**Rate Assignment**	**Complexity**
OERA [12]	optimal	$O(n^2)$
MCF [13]	suboptimal	$O(n)$
MC-Derivative (this paper)	optimal	$O(n \log n)$

Related Work. Since Vestal [16] introduced a fixed-priority scheduling algorithm with a MC task model (different WCET estimates for each task). The MC scheduling problem has received growing attention, in particular, on uniprocessor (see [4] for a survey).

Unlike the uniprocessor case, the multiprocessor case has not been studied much [17]–[22]. Anderson et al. [17] first considered multiprocessor MC scheduling with a two-level hierarchical scheduler. Pathan [19] proposed a global fixed priority multiprocessor scheduling algorithm for MC task systems. Li et al. [18] introduced a global scheduling algorithm with a speedup factor of $1+\sqrt{5}$ (≈ 3.236). Baruah et al. [20] presented a partitioned scheduling algorithm with a speedup factor of $8/3$ (≈ 2.666). Gu et al. [21] proposed a partitioned scheduling approach considering task-level virtual deadline assignment based on Ekberg and Yi [23]. Ren and Phan [22] proposed another partitioned scheduling approach considering multiple parameters (period and criticality-dependent utilizations), which reduces resource overbooking in MC scheduling. However, these multiprocessor MC approaches suffers from low schedulability.

Based on the fluid scheduling model for non-MC systems, MC-Fluid [12]–[14] has overcome the limitation of multiprocessor MC scheduling and achieved high schedulability. In this work, we extend our previous work [12], [14] with different rate assignment and different schedule generation strategies for non-fluid platforms.

II. PRELIMINARIES

We study the Mixed-Criticality (MC) scheduling problem on a hard real-time system with m identical processors. In this paper, we consider *dual-criticality* systems with two distinct criticality levels: HI (high) and LO (low).

Task Model. Each MC task is either a LO-criticality task (LO-task) or a HI-criticality task (HI-task). Each MC task τ_i is characterized by $(T_i, C_i^L, C_i^H, \chi_i)$, where $T_i \in \mathbb{N}$ is minimum inter-job separation time, $C_i^L \in \mathbb{R}^+$ is LO-criticality WCET (LO-WCET), $C_i^H \in \mathbb{R}^+$ is HI-criticality WCET (HI-WCET), and $\chi_i \in \{HI, LO\}$ is a task criticality level. Since HI-WCETs are based on conservative assumptions, we assume that $0 < C_i^L \leq C_i^H \leq T_i$. A task τ_i has a relative deadline equal to T_i. Any task can be executed on at most one processor at any time instant.

We consider a MC sporadic task set $\tau = \{\tau_i\}$, where a task τ_i represents a potentially infinite job release sequence. LO-task set (τ_L) and HI-task set (τ_H) are defined as $\tau_L \overset{\text{def}}{=} \{\tau_i \in \tau | \chi_i = LO\}$ and $\tau_H \overset{\text{def}}{=} \{\tau_i \in \tau | \chi_i = HI\}$.

The Correctness of MC Systems. The *system mode* is a system-wide variable representing the system criticality level

(LO or HI). In LO-mode (the system mode is LO), we assume that no job executes for more than its LO-WCET. In HI-mode, we assume that no job executes for more than its HI-WCET.

We consider *MC-schedulability* of a MC system, which consists of both LO-mode schedulability and HI-mode schedulability: LO-mode schedulability implies that jobs of all LO- and HI-tasks can complete to execute for their LO-WCETs before their deadlines in LO-mode; and HI-mode schedulability implies that jobs of all HI-tasks can complete to execute for their HI-WCETs before their deadlines in HI-mode.

The MC System Scenario. We assume the following scenario:

- The initial system mode is LO. In LO-mode, jobs of all LO- and HI-tasks are released.
- If a job of any HI-task $\tau_i \in \tau_H$ executes for more than its LO-WCET (C_i^L), the system mode switches from LO to HI (called *mode switch*). At mode switch, the system immediately discards all the jobs of LO-tasks.
- After mode switch, only the jobs of HI-tasks are released.

If a job of any LO-task $\tau_i \in \tau_L$ (likewise HI-task $\tau_i \in \tau_H$) executes for more than C_i^L in LO-mode (likewise C_i^H in HI-mode), we regard that the system has a fault and do not consider the case.

The problem to determine the time instant of switch-back from HI-mode to LO-mode is beyond the scope of this paper because it is irrelevant to the MC schedulability problem[2].

Utilization. A task or a task set can be represented as its utilization. LO- and HI-task utilizations of a task τ_i are defined as $u_i^L \overset{\text{def}}{=} C_i^L/T_i$ and $u_i^H \overset{\text{def}}{=} C_i^H/T_i$, respectively. System-level utilizations of a task set τ are defined as $U_L^L \overset{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L$, $U_H^L \overset{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L$, and $U_H^H \overset{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H$.

III. MC-Fluid Scheduling Framework

In this section, we review the existing fluid scheduling platform and present a new MC-Fluid scheduling algorithm based on the fluid platform.

A. The Fluid Scheduling Platform

Consider a platform where each processor can be allocated to one or more jobs simultaneously. Each job can be regraded as executing on a dedicated fractional processor with a speed smaller than or equal to one. This scheduling platform is referred to *fluid scheduling platform* [6]–[8].

Definition 1 (Fluid scheduling platform [8])**.** *The fluid scheduling platform is a scheduling platform where a job of a task is executed on a fractional processor at all time instants.*

The fluid scheduling platform continuously executes all tasks with their *execution rates.*

Definition 2 (Execution rate)**.** *A task τ_i is said to be executed with* ***execution rate*** *$\theta_i(t_1, t_2) \in \mathbb{R}^+$, s.t. $0 < \theta_i(t_1, t_2) \leq 1$, if every job of the task is executed on a fractional processor*[3] *with a speed of $\theta_i(t_1, t_2)$ over a time interval $[t_1, t_2]$, where t_1 and t_2 are time instants s.t. $t_1 \leq t_2$.*

Schedulability of a fluid platform requires two conditions:

1) Task-schedulability: each task has an execution rate that ensures to meet its deadline.
2) Platform feasibility: execution rates of all tasks are feasible on the multiprocessor platform.

In non-MC multiprocessor systems, many optimal scheduling algorithms [7], [9]–[11] have been proposed based on the fluid platform. These algorithms employ a single static rate for each job of a task $\tau_i \in \tau$ from its release to its deadline: $\forall k,\ \theta_i(r_i^k, d_i^k) = \theta_i$ where r_i^k and d_i^k are the release time and deadline of a job J_i^k (the k-th job of task τ_i), respectively. They satisfy task-schedulability by assigning C_i/T_i to θ_i, which is the task utilization of a non-MC task $\tau_i = (T_i, C_i)$ where C_i is its WCET. They satisfy platform feasibility if $\sum_{\tau_i \in \tau} \theta_i \leq m$.

In Lemma 1, we present platform feasibility of fluid model, applicable for both MC and non-MC systems. We will discuss task-schedulability for MC systems later in Sec. IV.

Lemma 1 (Platform feasibility, from [7])**.** *Given a task set τ, all tasks can be executed with their execution rates iff $\sum_{\tau_i \in \tau} \theta_i \leq m$.*

B. MC-Fluid Scheduling Algorithm

The fluid scheduling algorithm with a single static execution rate per task is inefficient in resource utilization of MC systems. For example, we consider the worst-case reservation approach by assigning $\theta_i := u_i^H$ for each HI-task and $\theta_i := u_i^L$ for each LO-task. Then, the result of rate assignment is overly pessimistic because task characteristics of MC systems are changed at mode switch. According to a typical dual-criticality system behaviors, the system changes task characteristic at mode switch, from executing all LO- and HI-tasks to executing only HI-tasks. Thus, if a scheduling algorithm is allowed to adjust the execution rate of tasks at mode switch, it can reduce the pessimism of the single rate assignment, considering the dynamics of MC systems. We propose a fluid scheduling algorithm, called *MC-Fluid*, with two static per-task execution rates. Informally, MC-Fluid executes each task $\tau_i \in \tau$ with θ_i^L in LO-mode and with θ_i^H in HI-mode.

Definition 3 (MC-Fluid scheduling algorithm)**.** *MC-Fluid is defined with LO- and HI-execution rates (θ_i^L and θ_i^H) for each task $\tau_i \in \tau$. For a job J_i^k of a task τ_i, MC-Fluid assigns θ_i^L to $\theta_i(r_i^k, \min(t_M, d_i^k))$ and θ_i^H to $\theta_i(\max(t_M, r_i^k), d_i^k)$ where r_i^k is its release time, d_i^k is its deadline, and t_M is the time instant of mode switch. Since all LO-tasks are dropped at mode switch, θ_i^H is not specified for any LO-task $\forall \tau_i \in \tau_L$.*

The *execution amount* of a job indicates the processor resources consumed by the job within a time interval.

Definition 4. *For a task $\tau_i \in \tau$, the* ***execution amount*** *of a job in a time interval of length t in LO- and HI-mode, denoted by $E_i^L(t)$ and $E_i^H(t)$, are the total amount of processor resources that the job has consumed during this time interval in LO- and HI-mode, respectively: $E_i^L(t) \overset{\text{def}}{=} \theta_i^L \cdot t$ and $E_i^H(t) \overset{\text{def}}{=} \theta_i^H \cdot t$.*

IV. Schedulability Analysis

In this section, we analyze the schedulability of MC-Fluid. The following theorem shows the MC-schedulability of MC-Fluid, which consists of LO-mode schedulability and HI-mode schedulability. In each mode, we need task-schedulability (Eq. (1) in LO-mode and Eq. (2) in HI-mode) and platform feasibility (Eqs. (3) and (4)).

[2]There are available switch-back protocols [15], [24].

[3]Since the task cannot be executed on more than one processor, $\theta_i \leq 1$.

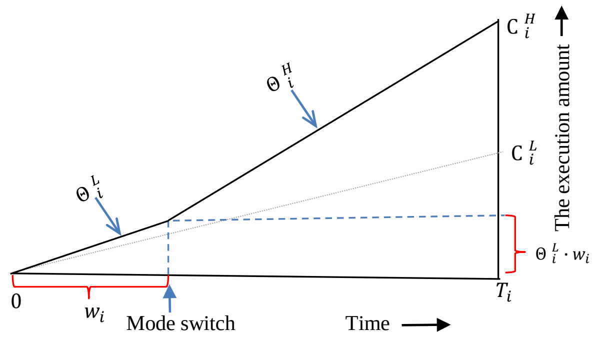


Fig. 1. The model of a carry-over job of a task $\tau_i \in \tau_H$ where the system mode is switched at w_i and the job is executed with θ_i^L before the mode switch and with θ_i^H after the mode switch

Theorem 1 (MC-schedulability)**.** *A task set τ, where each task $\tau_i \in \tau$ has LO- and HI-execution rates (θ_i^L and θ_i^H), is MC-schedulable under* MC-Fluid *iff*

$$\forall \tau_i \in \tau,\ \theta_i^L \geq u_i^L, \tag{1}$$

$$\forall \tau_i \in \tau_H,\ \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} \leq 1, \tag{2}$$

$$\sum_{\tau_i \in \tau} \theta_i^L \leq m, \tag{3}$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m. \tag{4}$$

To prove Theorem 1, we need to derive task-schedulability condition and platform feasibility condition in each mode.

Task-schedulability. We first consider task-schedulability in LO-mode in the following lemma.

Lemma 2. *A task $\tau_i \in \tau$ can meet its deadline in LO-mode iff $\theta_i^L \geq u_i^L$.*

Proof: ($\Leftarrow$) Consider a job of the task which is finished in LO-mode. We need to show that the execution amount of the job from its release time (time 0) to its deadline (time T_i) is greater than or equal to LO-WCET (C_i^L). From $\theta_i^L \geq u_i^L$,

$$\theta_i^L \cdot T_i \geq u_i^L \cdot T_i \ \Rightarrow\ E_i^L(T_i) \geq C_i^L. \qquad \text{(by Def. 4)}$$

($\Rightarrow$) We prove the contrapositive: if $\theta_i^L < u_i^L$, then the task cannot meet its deadline in LO-mode. It is true because $E_i^L(T_i) = \theta_i^L \cdot T_i < u_i^L \cdot T_i = C_i^L$. ■

Next, we consider task-schedulability in HI-mode. In HI-mode, we only consider task-schedulability of HI-tasks because LO-tasks are dropped in HI-mode. Consider a job of a HI-task τ_i as shown in Fig. 1. We define a *carry-over job* as a job that is released in LO-mode and finished in HI-mode as shown in Fig. 1. For the schedulability of HI-task, we only need to consider the carry-over job because it includes all possible cases (the job released in LO-mode and finished in HI-mode and the job released in HI-mode and finished in HI-mode). Let $w_i \in \mathbb{R}^+$ be the length of a time interval from the release time of the job to mode switch (or an executed time of the job in LO-mode). We can express the second case by setting $w_i = 0$ (zero executed time in LO-mode).

To derive task-schedulability for a carry-over job of τ_i, we need to know the relative time of the mode switch (w_i). We first derive task-schedulability condition with a given w_i. Since the mode switch triggers in the middle of its execution, the job is executed with θ_i^L before the mode switch and with θ_i^H after the mode switch. A cumulative execution amount of the job from its release time to its deadline (T_i) consists of its execution amount from its release time to mode switch with θ_i^L and its execution amount from mode switch to its deadline with θ_i^H:

$$E_i^L(w_i) + E_i^H(T_i - w_i) = \theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i)$$

by Def. 4. Task-schedulability condition with w_i is that the cumulative execution amount of the job is greater than or equal to its HI-WCET (C_i^H):

$$\theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H. \tag{5}$$

Since the MC system model assumes that the time instant of mode switch is unknown until runtime scheduling, we should consider all possible mode switch scenarios (any valid w_i). Note that $0 \leq w_i \leq T_i$ because mode switch can happen at any time instant between release time and deadline of the job. Therefore, task-schedulability is Eq. (5) for any w_i in $[0, T_i]$:

$$\forall w_i,\ \theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H. \tag{6}$$

To sum up, task-schedulability in HI-mode is Eq. (6).

Now, we will derive Eq. (6) independent of w_i. Its derivation is different depending on whether $\theta_i^L > \theta_i^H$ or $\theta_i^L \leq \theta_i^H$. Lemma 3 considers the first case ($\theta_i^L > \theta_i^H$).

Lemma 3. *For a given θ_i^H, a HI-task τ_i when θ_i^L is larger than θ_i^H can meet its deadline in HI-mode iff it meets its deadline in HI-mode when θ_i^L is equal to θ_i^H.*

Proof: ($\Leftarrow$) Suppose that θ_i^L is set to θ_i^H and the task meets its deadline in HI-mode. From Eq. (6), we have

$$\forall w_i,\ \theta_i^H \cdot w_i + \theta_i^H \cdot (T_i - w_i) = \theta_i^H \cdot T_i \geq C_i^H. \tag{7}$$

Let $\theta_i^{L\prime}$ be the changed value of θ_i^L where $\theta_i^{L\prime} > \theta_i^H$. To show that the task can meet its deadline in HI-mode with $\theta_i^{L\prime}$, we need to show that Eq. (6) holds: $\forall w_i$,

$$\begin{aligned} \theta_i^{L\prime} \cdot w_i + \theta_i^H \cdot (T_i - w_i) &> \theta_i^H \cdot w_i + \theta_i^H \cdot (T_i - w_i) \\ &= \theta_i^H \cdot T_i, \end{aligned}$$

which is greater than or equal to C_i^H by Eq. (7).

($\Rightarrow$) Suppose that θ_i^L is set to a value larger than θ_i^H and the task can meet its deadline in HI-mode.

We claim that $\theta_i^H \geq u_i^H$. We prove it by contradiction: suppose that $\theta_i^H < u_i^H$. Then, Eq. (6) does not hold when $w_i = 0$:

$$\theta_i^L \cdot 0 + \theta_i^H \cdot (T_i - 0) = \theta_i^H \cdot T_i,$$

which is smaller than C_i^H because $\theta_i^H \cdot T_i < u_i^H \cdot T_i = C_i^H$. However, since we assume that the task meets its deadline in HI-mode, Eq. (6) holds, which is a contradiction. Thus, we proved the claim ($\theta_i^H \geq u_i^H$).

Let $\theta_i^{L\prime}$ be the changed value of θ_i^L where $\theta_i^{L\prime} = \theta_i^H$. Then, it is required to show that Eq. (6) holds:

$$\forall w_i,\ \theta_i^H \cdot w_i + \theta_i^H \cdot (T_i - w_i) = \theta_i^H \cdot T_i,$$

which is greater than or equal to C_i^H because $\theta_i^H \geq u_i^H$. ■

Using the corollary below, we assume that $\theta_i^L \leq \theta_i^H$ for any task $\tau_i \in \tau_H$ in the rest of the paper.

Corollary 4. *For task-schedulability of a HI-task τ_i in HI-mode, we only need to consider the case where $\theta_i^L \leq \theta_i^H$.*

Proof: Task-schedulability of the task in HI-mode when $\theta_i^L > \theta_i^H$ is equivalent to the one when $\theta_i^L := \theta_i^H$ by Lemma 3. Thus, its task-schedulability in HI-mode is equivalent to the one when $\theta_i^L \leq \theta_i^H$. ∎

We derive task-schedulability in HI-mode based on the assumption that task-schedulability holds in LO-mode. The assumption is valid because the schedulability of a task in HI-mode is meaningless if the task is not schedulable in LO-mode.

Lemma 5 (Task-schedulability in HI-mode)**.** *Given a HI-task τ_i satisfying task-schedulability in LO-mode, the task can meet its deadline in HI-mode iff*

$$\frac{u_i^L}{\theta_i^L} + \frac{(u_i^H - u_i^L)}{\theta_i^H} \leq 1. \tag{8}$$

Proof: Consider a carry-over job of the task. We first derive the range of a valid w_i and derive task-schedulability in HI-mode by using the range.

Consider the range of w_i, which is $[0, T_i]$. We can further reduce the range by using task-schedulability in LO-mode. The execution amount of the job in LO-mode from its release time to any time instant cannot exceed its LO-WCET (C_i^L)[4]. Thus, its execution amount from its release time to mode switch also cannot exceed C_i^L:

$$E_i^L(w_i) \leq C_i^L \;\Rightarrow\; \theta_i^L \cdot w_i \leq C_i^L. \tag{9}$$

Combining $0 \leq w_i \leq T_i$ and Eq. (9), we have $0 \leq w_i \leq \min(C_i^L/\theta_i^L, T_i)$. Now, we need to compare C_i^L/θ_i^L and T_i. Since task-schedulability in LO-mode holds, we have $\theta_i^L \geq u_i^L$ by Lemma 2. Then,

$$\theta_i^L \geq C_i^L/T_i \;\Rightarrow\; T_i \geq C_i^L/\theta_i^L. \quad \text{(multiplying by } T_i/\theta_i^L\text{)}$$

Thus, the range of valid w_i is $0 \leq w_i \leq C_i^L/\theta_i^L$.

We know that task-schedulability in HI-mode is Eq. (6), which is rewritten to:

$$\begin{aligned}
&\forall w_i,\ \theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H \\
\Leftrightarrow\ &\forall w_i,\ \theta_i^H \cdot T_i \geq (\theta_i^H - \theta_i^L) \cdot w_i + C_i^H \\
\Leftrightarrow\ &\theta_i^H \cdot T_i \geq (\theta_i^H - \theta_i^L) \cdot C_i^L/\theta_i^L + C_i^H \qquad (\because \theta_i^H - \theta_i^L \geq 0)^5 \\
\Leftrightarrow\ &\theta_i^H \cdot T_i \geq C_i^L \cdot \theta_i^H/\theta_i^L + C_i^H - C_i^L \\
\Leftrightarrow\ &1 \geq u_i^L/\theta_i^L + (u_i^H - u_i^L)/\theta_i^H. \qquad \text{(dividing by } \theta_i^H \cdot T_i\text{)}
\end{aligned}$$

Thus, the task can meet its deadline in HI-mode iff Eq. (6) holds iff Eq. (8) holds. ∎

MC-schedulability. Since task schedulability is derived in Lemmas 2 and 5, and platform feasibility condition is straightforward from Lemma 1, we can prove Theorem 1. In Example 1, we show an application of Theorem 1.

Proof of Theorem 1: ($\Leftarrow$) We need to show that the task set satisfies both task-schedulability and platform feasibility. Since we assume Eq. (1), by Lemma 2, task-schedulability holds in LO-mode. Since we assume Eq. (2), by Lemma 5, task-schedulability holds in HI-mode. Since we assume Eqs. (3) and (4), by Lemma 1, platform feasibility holds in both LO-mode and HI-mode, respectively.

($\Rightarrow$) We will prove the contrapositive: if any of the conditions do not hold, then the task set is not MC-schedulable. If Eq. (1) does not hold, by Lemmas 2, task-schedulability does not hold in LO-mode. If Eq. (2) does not hold, by Lemmas 5, task-schedulability does not hold in HI-mode. If Eq. (3) or Eq. (4) does not hold, by Lemma 1, platform feasibility does not hold. ∎

[4]The job triggers mode switch if the job execute for more than C_i^L.

[5]We already assumed that $\theta_i^L \leq \theta_i^H$ by Corollary 4.

TABLE II
EXAMPLE TASK SET AND ITS EXECUTION RATE ASSIGNMENT.

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H	θ_i^L	θ_i^H
τ_1	10	2	8.5	HI	0.2	0.85	0.571	1
τ_2	20	5	10	HI	0.25	0.5	0.472	0.531
τ_3	30	4.5	9	HI	0.15	0.3	0.283	0.319
τ_4	40	4	6	HI	0.1	0.15	0.15	0.15
τ_5	50	10	10	LO	0.2	0.2	0.2	

Example 1. *Consider a two-processor system where its task set τ and its execution rate assignment are given as shown in Table II. For MC-schedulablility, by Theorem 1, we need to show that Eqs.* (1), (2), (3), *and* (4) *hold. We can easily check Eq.* (1). *We show that Eq.* (2)*:* $0.2/0.571 + 0.65/1 \leq 1$ *for* τ_1, $0.25/0.472 + 0.25/0.531 \leq 1$ *for* τ_2, $0.15/0.283 + 0.15/0.319 \leq 1$ *for* τ_3, *and* $0.1/0.15 + 0.05/0.15 \leq 1$ *for* τ_4. *We show that* $\sum_{\tau_i \in \tau} \theta_i^L = 1.676 \leq 2$ *for Eq.* (3) *and* $\sum_{\tau_i \in \tau_H} \theta_i^H = 2.0 \leq 2$ *for Eq.* (4).

V. The Execution Rate Assignment

In the previous section, we looked at schedulability analysis of MC-Fluid when a task set and its rate assignment are given. In this section, we seek the rate assignment and construct an optimization problem based on MC-Fluid schedulability conditions. We first define the *optimality* of a rate assignment algorithm, in a similar way to Davis et al. [25].

Definition 5. *A task set τ is called **MC-Fluid-feasible** if there exists an execution rate assignment that the task set is schedulable under MC-Fluid with. An execution rate assignment algorithm $\mathbb{A}$ is called **optimal** if $\mathbb{A}$ can find a schedulable assignment for all MC-Fluid-feasible task sets. For brevity, we refer to an execution rate assignment as an "assignment" and say that a task is feasible when the task is MC-Fluid-feasible.*

Due to complexity, we cannot check all execution rates one by one. To find an optimal assignment efficiently, we formulate the assignment problem as an optimization problem.

We first present a lemma for an optimal assignment of θ_i^L (Lemma 6) and later present the optimization problem for an optimal assignment of θ_i^H.

Lemma 6 (An optimal assignment of θ_i^L)**.** *If a task set τ is feasible, there is a schedulable assignment where (i) $\theta_i^L := u_i^L$ for a task $\tau_i \in \tau_L$ and (ii) $\theta_i^L := \frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L}$ for $\tau_i \in \tau_H$.*

Proof: Since the task set is feasible, there exists a schedulable assignment (denoted by $\mathbb{A}$) satisfying Eqs. (1), (2), (3), and (4) by Theorem 1.

(i) Consider a task $\tau_i \in \tau_L$ and its LO-execution rate (θ_i^L) in $\mathbb{A}$. We will show that τ is still schedulable after reassignment of θ_i^L. If we reassign θ_i^L, it only affects Eqs. (1) and (3).

Let θ_i^{L*} be the value of θ_i^L in $\mathbb{A}$. Since $\mathbb{A}$ is schedulable, Eq. (1) holds with θ_i^{L*}, which is $\theta_i^{L*} \geq u_i^L$. Suppose that we reassign $\theta_i^L := u_i^L$. Then, Eq. (1) still holds because $\theta_i^L \geq u_i^L$. Eq. (3) still holds because the decreased θ_i^L (from θ_i^{L*} to u_i^L) does not increase the sum of execution rates.

(ii) Consider a task $\tau_i \in \tau_H$ and its LO-execution rate (θ_i^L) in $\mathbb{A}$. We will show that τ is still schedulable after reassignment of θ_i^L. If we reassign θ_i^L, it only affects Eqs. (1), (2), and (3).

Let θ_i^{L*} and θ_i^{H*} be the value of θ_i^L and θ_i^H in $\mathbb{A}$, respectively. Since Eq. (2) holds for θ_i^{L*} and θ_i^{H*}, we have

$$\frac{u_i^H - u_i^L}{\theta_i^{H*}} \le 1 - \frac{u_i^L}{\theta_i^{L*}} \Rightarrow \frac{u_i^H - u_i^L}{\theta_i^{H*}} < 1, \tag{10}$$

since $u_i^L/\theta_i^{L*} > 0$. Eq. (2) with θ_i^{L*} and θ_i^{H*} is rewritten to:

$$\begin{aligned}
&\frac{u_i^L}{\theta_i^{L*}} \le \frac{\theta_i^{H*} - u_i^H + u_i^L}{\theta_i^{H*}} \\
\Leftrightarrow\ & u_i^L \cdot \theta_i^{H*} \le \theta_i^{L*}(\theta_i^{H*} - u_i^H + u_i^L) \quad \text{(multiplying by } \theta_i^{H*} \cdot \theta_i^{L*}\text{)} \\
\Leftrightarrow\ & \frac{u_i^L \cdot \theta_i^{H*}}{\theta_i^{H*} - u_i^H + u_i^L} \le \theta_i^{L*}. \qquad (\because \theta_i^{H*} > u_i^H - u_i^L \text{ by Eq. (10)})
\end{aligned}$$

Since $\mathbb{A}$ is schedulable, Eqs. (1), (2), and (3) hold with θ_i^{L*}. Suppose that we reassign $\theta_i^L := \frac{u_i^L \cdot \theta_i^{H*}}{\theta_i^{H*} - u_i^H + u_i^L}$. Then, Eq. (1) still holds because $\theta_i^L \ge u_i^L$. Eq. (2) still holds because the value of the reassigned θ_i^L is the minimum value satisfying Eq. (2). Eq. (3) still holds because the decreased θ_i^L does not increase the sum of execution rates. ■

Based on Lemma 6 and Theorem 1, we can present a feasibility condition, from which we can find an optimal assignment of θ_i^H.

Theorem 2 (Feasibility). *A task set τ is feasible iff there exists an assignment of θ_i^H for $\forall \tau_i \in \tau_H$ satisfying*

$$u_i^H \le \theta_i^H \le 1, \tag{11}$$

$$U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} \le m, \tag{12}$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \le m. \tag{13}$$

Proof: ($\Leftarrow$) To show that the task set is feasible, we need to show that there exists a schedulable assignment. Consider an assignment where $\theta_i^L := u_i^L$ for each task $\tau_i \in \tau_L$, $\theta_i^L := u_i^L \cdot \theta_i^H/(\theta_i^H - u_i^H + u_i^L)$ for each task $\tau_i \in \tau_H$, and θ_i^H for each task $\tau_i \in \tau_H$ satisfying Eqs. (11), (12), and (13).

We first show that $\theta_i^L \le 1$ for each $\tau_i \in \tau$ and $\theta_i^H \le 1$ for each $\tau_i \in \tau_H$ according to the definition of execution rates (Def. 2). For $\tau_i \in \tau_L$, we have $\theta_i^L \le 1$ by the selection of θ_i^L. For $\tau_i \in \tau_H$, we have $\theta_i^H \le 1$ from Eq. (11). Since we assumed $\theta_i^L \le \theta_i^H$ by Corollary 4, we have $\theta_i^L \le 1$ from Eq. (11).

To show that this assignment is schedulable by Theorem 1, we need to show that it satisfies Eqs. (1), (2), (3), and (4). We know that θ_i^L for $\forall \tau_i \in \tau$ satisfies Eq. (1) and θ_i^L for $\forall \tau_i \in \tau_H$ satisfies Eq. (2). We can rewrite Eq. (3) to:

$$\begin{aligned}
\sum_{\tau_i \in \tau} \theta_i^L \le m &\Leftrightarrow \sum_{\tau_i \in \tau_L} \theta_i^L + \sum_{\tau_i \in \tau_H} \theta_i^L \le m \\
&\Leftrightarrow \sum_{\tau_i \in \tau_L} u_i^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L} \le m \\
&\Leftrightarrow U_L^L + \sum_{\tau_i \in \tau_H} \left(u_i^L + \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} \right) \le m,
\end{aligned}$$

which is Eq. (12). Since Eq. (3) is equivalent to Eq. (12), the equation holds. Eq. (4) holds from Eq. (13).

($\Rightarrow$) Since the task set is feasible, there is a schedulable assignment where $\theta_i^L = u_i^L$ for $\forall \tau_i \in \tau_L$ and $\theta_i^L = u_i^L \cdot \theta_i^H/(\theta_i^H - u_i^H + u_i^L)$ by Lemma 6. We need to show that θ_i^H for $\forall \tau_i \in \tau_H$ in the assignment satisfies Eqs. (11), (12), and (13).

Consider θ_i^H of a task $\tau_i \in \tau_H$. We know $\theta_i^H \le 1$ from the definition of execution rates (Def. 2). Since we assumed that $\theta_i^L \le \theta_i^H$ by Corollary 4, we rewrite $\theta_i^L \le \theta_i^H$ to:

$$\frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L} \le \theta_i^H \Rightarrow u_i^L \le \theta_i^H - u_i^H + u_i^L,$$

which is $u_i^H \le \theta_i^H$. Then, Eq. (11) holds.

Since the assignment is schedulable, Eqs. (3) and (4) hold by Theorem 1. We already proved that Eq. (12) holds from Eq. (3) in Case ($\Leftarrow$). Eq. (13) holds from Eq. (4). ■

From feasibility condition, we know that the assignment of θ_i^H satisfying the conditions in Theorem 2 is optimal. For an optimal assignment of θ_i^H, we present the optimization problem based on Theorem 2.

Definition 6 (The assignment problem). *Given a task set τ, we define a non-negative real number X_i for each task $\tau_i \in \tau_H$ such that $\theta_i^H := u_i^H + X_i^*$ and X_i^* is an optimal point of X_i on the following optimization problem:*

$$\begin{aligned}
\textit{minimize} \quad & \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L} \\
\textit{subject to} \quad & \sum_{\tau_i \in \tau_H} X_i \le m - U_H^H, && \text{(CON1)} \\
& \forall \tau_i \in \tau_H,\ X_i \ge 0, && \text{(CON2)} \\
& \forall \tau_i \in \tau_H,\ X_i \le 1 - u_i^H. && \text{(CON3)}
\end{aligned}$$

We present that the assignment of θ_i^H by solving the optimization problem is optimal according to Def. 5.

Lemma 7. *If a rate assignment constructs a solution for the assignment problem in Def. 6, the rate assignment is optimal.*

Proof: Consider a feasible task set. Then, by Theorem 2, some assignment satisfies Eqs. (11), (12), and (13). We claim that the assignment satisfies CON1, CON2, and CON3. If the assignment does not satisfy CON1, then Eq. (13) does not hold. If the assignment does not satisfy CON2 or CON3, then Eq. (11) does not hold.

We claim that if Eq. (12) does not hold with the assignment by Def. 6, then Eq. (12) does not hold with any assignment satisfying CON1, CON2, and CON3. Suppose that we have a solution to the optimization problem in Def. 6. For $\tau_i \in \tau_H$, let X_i^* be the optimal value of X_i for the problem. Let OBJ^* be the value of the objective function with X_i^*. Suppose that the solution does not satisfy Eq. (12): $\mathsf{OBJ}^* > m - U_L^L - U_H^L$. For any set of X_i satisfying CON1, CON2, and CON3, we have $\sum_{\tau_i \in \tau_H} u_i^L(u_i^H - u_i^L)/(X_i + u_i^L) \ge \mathsf{OBJ}^*$ by the definition of the optimization problem and thus Eq. (12) does not hold: $\sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L} \ge \mathsf{OBJ}^* > m - U_L^L - U_H^L$. ■

VI. RATE ASSIGNMENT ALGORITHM

In previous section, we constructed the optimization problem (Def. 6) for rate assignment. In this section, we assign execution rates by solving the optimization problem. In Sec. VI-A, we present MC-Derivative, a new optimal rate assignment algorithm using the first derivative principles. In Sec. VI-B, we briefly outline the applicability of the MC-Derivative strategy for a multi-criticality system.

A. *MC-Derivative Algorithm*

The motivation of the MC-Derivative rate assignment algorithm is to address the limitation on the scalability of our earlier rate assignment algorithm [12]. In [12], we presented the OERA rate assignment algorithm by solving the problem in Def. 6 using a convex optimization framework. Extending such optimization problem for a multi-criticality system is challenging because it may result in serious degradation in the schedulability performance (refer Sec. VI-B). Thus, instead of applying the existing optimization framework, we develop a simple approach to determine the optimal execution rates. MC-Derivative assigns execution rates by using the first derivative principles, which has a lower complexity compared to OERA and is easily extensible to a multi-criticality system.

Intuition. We recall the objective function of the assignment problem mentioned in Def. 6 as $f(X)$:

$$f(X) = \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L}$$

where X (or $\{X_i\}$) denotes the vector of X_i values. The objective function can be rewritten as separable function $f(X) = \sum f_i(X_i)$, where $f_i(X_i) = \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L}$. Our aim is to minimize the objective function $f(X)$. Since the objective function is differentiable and convex, the first partial derivative technique is applicable to find the optimal solution.

The first partial derivative of the objective function is defined as $f_i'(X_i) = \frac{\partial f(X)}{\partial X_i}$, which indicates the change of the objective function value with respect to the change in X_i. For ease of understanding, since $f_i'(X_i)$ is a negative function within the range of X_i ($0 \leq X_i \leq 1 - u_i^H$ by CON2 and CON3 in Def. 6), we consider the absolute value of $f_i'(X_i)$ denoted as $\bar{f}_i'(X_i)$:

$$\bar{f}_i'(X_i) = \frac{u_i^L(u_i^H - u_i^L)}{(X_i + u_i^L)^2} \tag{14}$$

Thus, larger $\bar{f}_i'(X_i)$ indicates larger decrease of $f(X)$.

To minimize the objective function ($f(X)$), we utilize the first partial derivative. We initialize $X_i = 0$ for all tasks, which makes the value of $f(X)$ the largest. Then, we increase X_i of the task with the largest partial derivative ($\bar{f}_i'(X_i)$), which brings the largest decrease of $f(X)$. We repeat the above procedure as long as CON1 holds, which minimizes $f(X)$. Let Γ be the value of the partial derivative when CON1 just holds. We present the application of MC-Derivative strategy with an example.

Example 2. *Consider a sample task set in Table II. Fig. 2 plots $\bar{f}_i'(x)$ of the corresponding task set for $x \in \mathbb{R}^+$. MC-Derivative first assigns the minimum required execution rate, $X_i = 0$ for each task. Then, MC-Derivative increases X_1 until its $\bar{f}_i'(X_i)$ is the largest among tasks, X_1 becomes its maximum value ($1 - u_1^H$), or Slack (the remaining system utilization, $m - \sum_{i=1}^{4}(u_i^H + X_i)$) is zero. In the example, X_1 becomes the maximum value. Next, τ_2 and τ_3 have the next largest $\bar{f}_i'(X_i)$. X_2 or X_3 is increased until its $\bar{f}_i'(X_i)$ is the largest among tasks except τ_1 (X_1 cannot be increased any further). Before τ_4 has the largest partial derivative ($\bar{f}_4'(0)$), Slack becomes zero, where MC-Derivative ends.*

Note that $\bar{f}_4'(0) < \Gamma < \bar{f}_2'(0) = \bar{f}_3'(0)$ (the black dotted line in Fig. 2). Thus, MC-Derivative assigns $X_1 = 1 - u_1^H$ and $X_4 = 0$ because $\Gamma < \bar{f}_1'(1 - u_1^H)$ and $\Gamma > \bar{f}_4'(0)$. For τ_2 and τ_3, MC-Derivative assign X_i between 0 and $1 - u_i^H$ because $\bar{f}_i'(1 - u_i^H) < \Gamma < \bar{f}_i'(0)$. Then, we can compute the rate assignment by techniques in Sec. V.

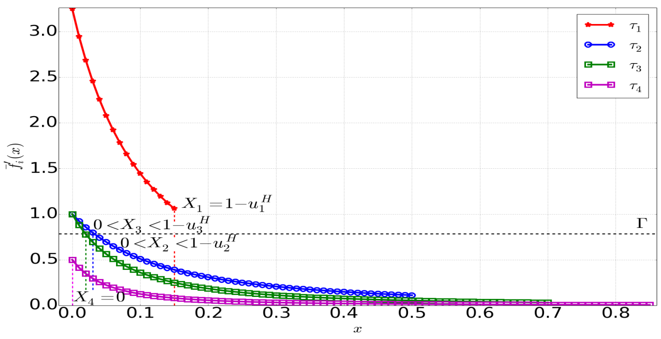


Fig. 2. The plot of $\bar{f}_i'(X_i = x)$ varying $x \in \mathbb{R}^+$ for the task set in Table II

Algorithm Description. For a given task set with $U_H^H \leq m$, Algorithm 1 computes the optimal LO- and HI- execution rates of tasks. To assign rates optimally, we need to know Γ. In Line 1, Algorithm 2 divides all possible range of the partial derivative into a number of ranges and checks whether Γ belongs to one of the ranges. Let Γ' be the largest value of the partial derivative in all the ranges of Γ s.t. $\Gamma' \geq \Gamma$. Then, we assign $\{X_i\}$ based on Γ' (increasing each X_i as long as $\bar{f}_i'(X_i) \geq \Gamma'$). For CON3 (the constraint of total increase of X_i for all HI-tasks), we define Slack $:= m - U_H^H - \sum X_i$, which indicates the remaining increase of X_i for any task. In Lines 2-4, if Slack is positive (meaning $\Gamma' \neq \Gamma$), we find Γ based on Slack and the assigned $\{X_i\}$ with Γ'. Lines 5-6 compute the LO- and HI- execution rates from $\{X_i\}$ by Def. 6 and the function computeLoRate$_i(\theta_i^H)$, which computes the optimal assignment of θ_i^L based on Lemma 6 in Sec. V: for a task τ_i,

$$\text{computeLoRate}_i(\theta_i^H) = \begin{cases} \frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L}, & \text{if } \tau_i \in \tau_H, \\ u_i^L, & \text{otherwise.} \end{cases}$$

Using the output of Algorithm 1, we can check the MC-schedulability of the input task set: since the value of the objective function in Def. 6 constructs the sum of LO-execution rates of HI-tasks ($\sum_{\tau_i \in \tau_H} \theta_i^L = f(X) + U_H^L$) and LO-mode platform feasibility is $\sum_{\tau_i \in \tau} \theta_i^L \leq m$, the task set is feasible if $\sum_{\tau_i \in \tau_H} \theta_i^L + U_L^L \leq m$.

Algorithm 2 determines the range of Γ. To do this, we divide all possible range of the partial derivative by the $\bar{f}_i'(0)$ and $\bar{f}_i'(1 - u_i^H)$ values of each HI-task. It is because the assignment of X_i for each task is different depending on the values: X_i is increased to the maximum ($1 - u_i^H$) if $\bar{f}_i'(1 - u_i^H) > \Gamma$, never increased if $\bar{f}_i'(0) < \Gamma$, and increased as long as $\bar{f}_i'(X_i) \geq \Gamma$ otherwise. To find the range of Γ, Line 3 computes $\bar{f}_i'(0)$ and $\bar{f}_i'(1 - u_i^H)$ of each task and adds their distinct values into F (the size of F is at most $2 \times |\tau_H|$ elements). Line 4 sorts F in the increasing order of $\bar{f}_i'(X_i)$.

Algorithm 1 The MC-Derivative algorithm

```
Input: $\tau$ s.t. $U_H^H \le m$
Output: $\{\theta_i^H\}$ for $\tau_H$ and $\{\theta_i^L\}$ for $\tau$
1: Assign $\{X_i\}$ and compute Slack using Algorithm 2
2: if (Slack $> 0$) then
3:     Modify $\{X_i\}$ using Algorithm 3 with Slack
4: end if
5: for each $\tau_i \in \tau_H$, assign $\theta_i^H := u_i^H + X_i$
6: for each $\tau_i \in \tau$, assign $\theta_i^L :=$ computeLoRate$_i(\theta_i^H)$
```

Lines $5-13$ of Algorithm 2 determine whether Γ belongs in the range (F_j, F_{j+1}). For a HI-task, X_i is assigned with $X_i^{\triangle}$ such that $\bar{f}_i'(X_i^{\triangle}) = F_j$. Then, by Eq. (14), we have

$$X_i^{\triangle} = \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{F_j}} - u_i^L \tag{15}$$

For the $\{X_i\}$ assignment (Line 7) to be feasible, the task is assigned a maximum X_i, which is $1 - u_i^H$ and a minimum X_i, which is 0 according to CON3 and CON2 in Def. 6 respectively. Line 8 computes Slack. If the range of Γ is $[0, F_1]$, Slack based on F_1 is positive. If the range of Γ is $(F_{j-1}, F_j]$ s.t. $j \ge 2$, Slack based on F_{j-1} is a negative value and Slack based on F_j is a positive value or zero. Since we sort F in an increasing order, we can find the range of Γ when Slack based on F_j is non-negative for the first time (Lines 9-12). Since F_j is the largest value in the range of Γ, we have $\Gamma' = F_j$. If Slack is positive, it is possible to increase the X_i of τ_{REM}, which are tasks that have the partial derivative such that $\bar{f}_i'(1 - u_i^H) < F_j \le \bar{f}_i'(0)$ (i.e., $0 \le X_i^{\triangle} < 1 - u_i^H$). In Line 10, we identify which task belongs to τ_{REM}. The FOR loop exits either when the condition in Line 9 holds (Slack is non-negative) or X_i of all tasks is zero (in this case, Slack is also non-negative since $U_H^H \le m$).

Algorithm 2 Determination of the range of Γ

```
Input: $\tau_H$
Output: $\tau_{\mathsf{REM}}$, Slack, and $\{X_i\}$ assigned with $\Gamma'$
1: Initialize $\tau_{\mathsf{REM}} = \emptyset$
2: for each $\tau_i$, assign $X_i = 0$
3: Let $F$ be the set of the value of $\bar{f}_i'(X_i)$ for each $\tau_i$ at
   $X_i = 0$ and $X_i = 1 - u_i^H$
4: Sort $F$ in increasing order
5: for j := 1 to $|F|$ (the size of $F$) do
6:     for each $\tau_i$, compute $X_i^{\triangle}$ s.t. $\bar{f}_i'(X_i^{\triangle}) = F_j$ (by
   Eq. (15))
7:     for each $\tau_i$, assign $X_i := \min(1 - u_i^H, \max(0, X_i^{\triangle}))$
8:     Let Slack = $m - U_H^H - \sum_{\tau_i \in \tau_H} X_i$
9:     if $\big($Slack $\ge 0\big)$ then
10:        for each $\tau_i$, add the task into $\tau_{\mathsf{REM}}$ if $0 \le X_i^{\triangle} <$
   $1 - u_i^H$
11:        Break
12:    end if
13: end for
```

Algorithm 3 modifies $\{X_i\}$ with Slack. Line 1 finds Γ based on Slack and the assignment of $\{X_i\}$ from Algorithm 2. Lines 2-4 assign $\{X_i\}$ of τ_{REM} such that $\bar{f}_i'(X_i) = \Gamma$ (formally proven in Lemma 9).

Algorithm 3 Modification of X_i with Slack

```
Input: $\tau_H$, $\tau_{\mathsf{REM}}$, Slack, and $\{X_i\}$
Output: $\{X_i^+\}$ (the vector of the modified $X_i$)
1: Compute $\Gamma$ by Eq. (18) with $\{X_i\}$ and Slack.
2: for each $\tau_i \in \tau_{\mathsf{REM}}$ do
3:     Assign $X_i^+$ s.t. $\bar{f}_i'(X_i^+) = \Gamma$ (by Eq. (15))
4: end for
5: for each $\tau_i \in \tau_H \setminus \tau_{\mathsf{REM}}$, assign $X_i^+ := X_i$
```

Optimality. We will show that the total sum of the objective function by MC-Derivative is always smaller than or equal to the one by any rate assignments. Let us first determine the possible cases for rate assignments different from MC-Derivative. To do so, we classify the HI-tasks in a task set based on their $\bar{f}_i'(X_i)$ values in Def. 7. Then, we derive the property of $\bar{f}_i'(X_i)$ of any two tasks in these classes for a valid rate assignment different from MC-Derivative in Lemma 8. Finally, we prove the optimality in Theorem 3.

Definition 7 (HI-tasks Classification)**.** *Without any loss of generality, tasks in τ_H are categorized into three sets (τ_{MAX}, τ_{MIN} and τ_{REM})*[6] *based on their $\bar{f}_i'(X_i)$ values:*

$$\begin{aligned}
\tau_{\mathsf{MAX}} &\overset{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ \bar{f}_i'(X_i = 1 - u_i^H) \ge \Gamma\},\\
\tau_{\mathsf{REM}} &\overset{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ \bar{f}_i'(0 < X_i < 1 - u_i^H) = \Gamma\},\\
\tau_{\mathsf{MIN}} &\overset{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ \bar{f}_i'(X_i = 0) \le \Gamma\}.
\end{aligned}$$

Note that τ_{MAX}, τ_{REM}, and τ_{MIN} are mutually exclusive, and $\tau_H = \tau_{\mathsf{MAX}} \cup \tau_{\mathsf{REM}} \cup \tau_{\mathsf{MIN}}$.

For the example task set in Table II, $\tau_{\mathsf{MAX}} = \{\tau_1\}$, $\tau_{\mathsf{REM}} = \{\tau_2, \tau_3\}$ and $\tau_{\mathsf{MIN}} = \{\tau_4\}$.

Before proving the optimality, we introduce an auxiliary lemma. Consider an assignment algorithm $\mathbb{A}$ that assigns $\{X_i\}$ differently from Algorithm 1. We can regard that $\{X_i\}$ in $\mathbb{A}$ is constructed by the transfer[7] of $\{X_i\}$ in Algorithm 1. Lemma 8 shows the property of $\bar{f}_i'(X_i)$ of any two tasks for a valid transfer of X_i.

Lemma 8. *For any two tasks τ_p and τ_q, assume that X_p and X_q are assigned by Algorithm 1. If a valid transfer from X_p to X_q is possible, then it must be $\bar{f}_p'(X_p) \ge \bar{f}_q'(X_q)$.*

Proof: The X_i of task $\tau_i \in \tau_{\mathsf{MAX}}$ cannot be increased any further since $X_i \le 1 - u_i^H$ from CON3 (i.e., $\theta_i^H \le 1$). The X_i of task $\tau_i \in \tau_{\mathsf{MIN}}$ cannot be decreased any further because $X_i \ge 0$ from CON2 (i.e., $\theta_i^H \ge u_i^H$). Therefore, there are only two cases to consider for a valid rate transfer.

Case 1: The X_i of the task $\tau_p \in \tau_{\mathsf{MAX}}$ is transferable to the one of task $\tau_q \in \{\tau_{\mathsf{REM}}, \tau_{\mathsf{MIN}}\}$ or the X_i of the task $\tau_p \in \tau_{\mathsf{REM}}$ is transferable to the one of task $\tau_q \in \tau_{\mathsf{MIN}}$.

Case 2: The X_i of the task $\tau_p \in \tau_{\mathsf{REM}}$ is transferable to the one of another task $\tau_q \in \tau_{\mathsf{REM}}$.

We need to show $\bar{f}_p'(X_p) \ge \bar{f}_q'(X_q)$ for the two cases. We show that $\bar{f}_p'(X_p) \ge \bar{f}_q'(X_q)$ for Case 1 using Def. 7. Consider $\tau_p \in \tau_{\mathsf{MAX}}$ and $\tau_q \in \{\tau_{\mathsf{REM}}, \tau_{\mathsf{MIN}}\}$. If $\tau_q \in \tau_{\mathsf{REM}}$, we have $\bar{f}_p'(X_p) \ge \Gamma = \bar{f}_q'(X_q)$. If $\tau_q \in \tau_{\mathsf{MIN}}$, we have $\bar{f}_p'(X_p) \ge \Gamma \ge$

[6]By Algorithm 1, X_i of $\tau_i \in \tau_{\mathsf{MAX}}$ has the maximum value (i.e., $X_i = 1 - u_i^H$), X_i of $\tau_i \in \tau_{\mathsf{MIN}}$ has the minimum value (i.e., $X_i = 0$).

[7]For any small real number ϵ and two tasks, τ_p and τ_q s.t. $\tau_p \neq \tau_q$, the transfer from X_p to X_q is defined as X_p is decreased by ϵ and X_q is increased by ϵ.

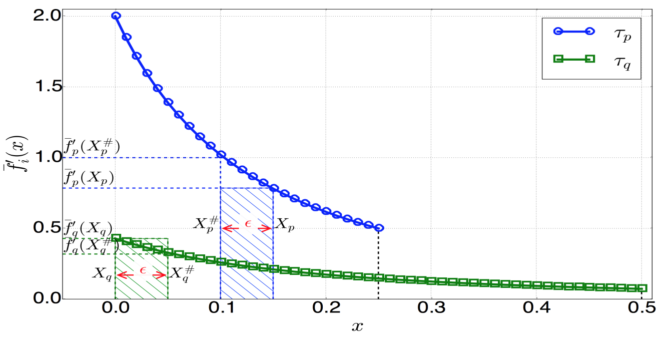


Fig. 3. Transfer of the execution rate from τ_p to τ_q

$\bar{f}'_q(X_q)$. In the case of $\tau_p \in \tau_{\mathsf{REM}}$ and $\tau_q \in \tau_{\mathsf{MIN}}$, we have $\bar{f}'_p(X_p) = \Gamma \geq \bar{f}'_q(X_q)$.

Next, we will show that $\bar{f}'_p(X_p) \geq \bar{f}'_q(X_q)$ for Case 2. Since $\tau_p, \tau_q \in \tau_{\mathsf{REM}}$, we have $\bar{f}'_p(X_p) = \bar{f}'_q(X_q) = \Gamma$ by Line 3 of Algorithm 3. ■

Theorem 3. *MC-Derivative is an optimal rate assignment algorithm.*

Proof: We show that if the execution rates are transferred, then the resulting sum of the objective will be greater than the one determined: $f(X^{\#}) \geq f(X)$ where $\{X_i^{\#}\}$ denotes the updated $\{X_i\}$ after the transfer of execution rates such that $X_i^{\#} \neq X_i$.

Consider the transfer of rate from task τ_p to task τ_q as shown in Fig. 3. The execution rates assigned by MC-Derivative to tasks τ_p and τ_q is given by X_p and X_q respectively. Let ϵ denote the amount of execution rate transferred, and $X_p^{\#}$ and $X_q^{\#}$ denote the corresponding updated execution rates. Then, $X_p^{\#} := X_p - \epsilon$ and $X_q^{\#} := X_q + \epsilon$ where $\epsilon \in [0, \min(X_p, 1 - u_q^H - X_q)]$ since for any task τ_i, $X_i \in [0, 1 - u_i^H]$.

To show that the sum of the objective is greater after transfer, we only need to show that

$$\begin{aligned} & f_p(X_p^{\#}) + f_q(X_q^{\#}) \geq f_p(X_p) + f_q(X_q) \\ \Leftrightarrow\ & f_p(X_p^{\#}) - f_p(X_p) \geq f_q(X_q) - f_q(X_q^{\#}) \\ \Leftrightarrow\ & \int_{X_p}^{X_p^{\#}} \bar{f}'_p(x)dx \geq \int_{X_q^{\#}}^{X_q} \bar{f}'_q(x)dx. \end{aligned} \tag{16}$$

For a task τ_i and monotonically non-increasing function $\bar{f}'_i(x)$ with X_1 and X_2 s.t. $X_1 < X_2$, note that $\int_{X_2}^{X_1} \bar{f}'_i(x)dx \geq \min_{X_1 \leq x \leq X_2} \bar{f}'_i(x) \cdot (X_2 - X_1)$ and $\int_{X_2}^{X_1} \bar{f}'_i(x)dx \leq \max_{X_1 \leq x \leq X_2} \bar{f}'_i(x) \cdot (X_2 - X_1)$.

To show Eq. (16), we only need to show that

$$\begin{aligned} & \min_{X_p^{\#} \leq x \leq X_p} \bar{f}'_p(x) \cdot (X_p^{\#} - X_p) \geq \max_{X_q \leq x \leq X_q^{\#}} \bar{f}'_q(x) \cdot (X_q - X_q^{\#}) \\ \Leftrightarrow\ & \min_{X_p^{\#} \leq x \leq X_p} \bar{f}'_p(x) \cdot \epsilon \geq \max_{X_q \leq x \leq X_q^{\#}} \bar{f}'_q(x) \cdot \epsilon \\ & \qquad \text{(see the shaded rectangle in Fig. 3)} \\ \Leftrightarrow\ & \min_{X_p^{\#} \leq x \leq X_p} \bar{f}'_p(x) \geq \max_{X_q \leq x \leq X_q^{\#}} \bar{f}'_q(x). \end{aligned} \tag{17}$$

Note that as both $\bar{f}'_p(X_p)$ and $\bar{f}'_q(X_q)$ are monotonically non-increasing functions, we have $\min_{X_p^{\#} \leq x \leq X_p} \bar{f}'_p(x) = \bar{f}'_p(X_p)$ and $\max_{X_q \leq x \leq X_q^{\#}} \bar{f}'_q(x) = \bar{f}'_q(X_q)$. Since we know $\bar{f}'_p(X_p) \geq \bar{f}'_q(X_q)$ by Lemma 8, Eq. (17) holds. ■

Correctness. The MC-Derivative algorithm is correct if it satisfies CON1, CON2, and CON3 in Def. 6. For CON2 and CON3 in Def. 6 (i.e., $0 \leq X_i \leq 1 - u_i^H$), we identify τ_{MAX} and τ_{MIN} in Line 7 of Algorithm 2. We only increase the X_i of τ_{REM} with $X_i \leq 1 - u^H$ in Algorithm 3.

For CON1 (i.e., $\sum X_i \leq m - U_H^H$), we have non-zero Slack after Algorithm 2. The following lemma computes Γ from the assignment of $\{X_i\}$ by Algorithm 2 and its positive Slack. If Γ is known, Algorithm 3 can compute $\{X_i\}$ where $\bar{f}'_i(X_i) \geq \Gamma$ for each τ_i and Slack is zero.

Lemma 9. *Given Γ', let $X_i^{\#}$ denote the assignment of X_i by Algorithm 2. When Slack for Γ' is positive, we can compute Γ such that Slack for Γ is zero:*

$$\Gamma = \left(\frac{\sum_{\tau_j \in \tau_{\mathsf{REM}}} \sqrt{u_j^L(u_j^H - u_j^L)}}{\mathsf{Slack} + \sum_{\tau_j \in \tau_{\mathsf{REM}}} (X_j^{\#} + u_j^L)} \right)^2 \tag{18}$$

Proof: Let X_i^{+} denote the assignment of X_i with Γ. Note that Slack for Γ' (F_j in F) is positive and Slack for F_{j-1} is negative by Algorithm 2. Since τ_{MAX} and τ_{MIN} are not changed within $(F_{j-1}, F_j]$, only tasks in τ_{REM} has a different X_i^{+} from $X_i^{\#}$. Then, we have $\sum_{\tau_i \in \tau_H \setminus \tau_{\mathsf{REM}}} X_i^{\#} = \sum_{\tau_i \in \tau_H \setminus \tau_{\mathsf{REM}}} X_i^{+}$. Since $\mathsf{Slack} + \sum_{\tau_i \in \tau_H} X_i^{\#} = \sum_{\tau_i \in \tau_H} X_i^{+}$ by the definition of Slack and Γ, we have

$$\begin{aligned} & \mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i^{\#} + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i^{+} + u_i^L) \\ \Leftrightarrow\ & \mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i^{\#} + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{\bar{f}'_i(X_i^{+})}} \\ & \qquad \text{(by Eq. (15))} \\ \Leftrightarrow\ & \mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i^{\#} + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{\Gamma}}, \\ & \qquad \text{(by } \bar{f}'_i(X_i^{+}) = \Gamma\text{)} \end{aligned}$$

which is Eq. (18). ■

Complexity. MC-Derivative algorithm has a complexity of $O(n \log n)$. Algorithm 2 has $O(n \log n)$ time complexity. Sorting F in Line 4 can be done in $O(n \log n)$ time complexity. The range of Γ (Lines 5-13) can be determined in $O(n \log n)$ time complexity. Although the outer FOR loop (Line 5) of Algorithm 2 takes $O(n) \cdot \mathbb{T}$ where $\mathbb{T}$ is the time taken in Lines 6-12, we can reduce it into $O(\log n) \cdot \mathbb{T}$: selecting the least $F_j \in F$ that satisfies the condition in Line 9 of Algorithm 2 can be done using binary search. For $\mathbb{T}$, computing $X_i^{\triangle}$ in Line 6 and assigning X_i in Line 7 of Algorithm 2 consumes $O(n)$ time. Algorithm 3 has $O(n)$ time complexity because modifying $\{X_i\}$ of the tasks with Slack such that $\bar{f}'_i(X_i) = \Gamma$ can be done in $O(n)$ time. In some domains such as online admission control, MC-Derivative is preferable due to its low complexity ($O(n \log n)$) while OERA has $O(n^2)$ complexity.

B. Discussion on extension to multi-criticality systems

In this section, we discuss the scalability of MC-Derivative (in terms of schedulability performance) to 3-criticality systems, compared to OERA. Consider an example task set with

TABLE III
EXAMPLE 3-CRITICALITY TASK SET SCHEDULED ON 2 PROCESSOR

	T_i	C_i^L	C_i^M	C_i^H	χ_i	u_i^L	u_i^M	u_i^H
τ_1	10	1	5	6	HI	0.1	0.5	0.6
τ_2	10	3	4	6	HI	0.3	0.4	0.6
τ_3	10	1	3	5	HI	0.1	0.3	0.5
τ_4	10	2	4	-	ME	0.2	0.4	-
τ_5	10	1.5	-	-	LO	0.15	-	-

TABLE IV
MULTI-CRITICALITY RATE ASSIGNMENT

	OERA			MC-Derivative		
	θ_i^L	θ_i^M	θ_i^H	θ_i^L	θ_i^M	θ_i^H
τ_1	0.5274	0.6214	0.6000	0.3911	0.6925	0.6000
τ_2	0.5493	0.5493	0.7359	0.6277	0.4000	0.7351
τ_3	0.4293	0.4293	0.6641	0.3278	0.5075	0.6649
τ_4	0.4000	0.4000		0.4000	0.4000	
τ_5	0.1500			0.1500		
$\sum$	2.0560	2.000	2.000	1.8966	2.000	2.000

3 criticality levels (low – LO, medium – ME and high – HI) as shown in Table III. The system operates as follows: the system initially starts in the LO-mode; if a ME-task or a HI-task executes more than C_i^L, the system switches to ME-mode; if a HI-task executes more than C_i^M, the system switches to HI-mode.

We can apply the OERA algorithm to a 3-criticality system if we construct the rate assignment problem by the *Karush-Kuhn-Tucker (KKT)* conditions [12]. Thus, we decompose it into two sub-problems[8]:

1) Minimize $\sum_{\tau_H} \theta_i^M$ subject to $\sum_{\tau_H} \theta_i^H \leq m$.
2) Minimize $\sum_{\tau_M \cup \tau_H} \theta_i^L$ subject to $\sum_{\tau_M \cup \tau_H} \theta_i^M \leq m$.

We apply convex optimization for each subproblem. For the first subproblem, we decide $\{\theta_i^M\}$ by optimization techniques. Based on the assignment of $\{\theta_i^M\}$, we decide $\{\theta_i^L\}$ for the second subproblem. As shown in Table IV, the resulting rate assignment for the example task set is unschedulable since $\sum \theta_i^L > 2$.

Although the extension of OERA cannot find a schedulable rate assignment for the example task set, the extension of MC-Derivative can find a schedulable rate assignment. Let $f(X)$ be $\sum_{\tau_M \cup \tau_H} \theta_i^L$. Our strategy is to increase θ_i^M or θ_i^H by partial derivative of $f(X)$. We can partially differentiate $f(X)$ with respect to θ_i^M and θ_i^H:

$$f_i''(X_i) = \frac{\partial^2 f(X)}{\partial \theta_i^M \partial \theta_i^H}.$$

We initially assign the minimum valid rate of $\theta_i^M = u_i^M$ and $\theta_i^H = u_i^H$ for all ME-tasks and HI-tasks. We increase the θ_i^M rate to the tasks in the order of their maximum $f_i''(X_i)$ derivative value as long as the ME-mode slack $(m - \sum_{\tau_M \cup \tau_H} \theta_i^M)$ is non-negative. Similarly, we increase the θ_i^H rate to the HI-tasks in the order of their maximum $f_i''(X_i)$ derivative value as long as the HI-mode slack $(m - \sum_{\tau_H} \theta_i^H)$ is non-negative. As shown in Table IV, the resulting rate assignment for the same example task set is schedulable ($\sum \theta_i^L \leq 2$, $\sum_{\tau_M \cup \tau_H} \theta_i^M \leq 2$, and $\sum_{\tau_H} \theta_i^H \leq 2$). Note that both OERA and MC-Derivative are sub-optimal solutions to the 3-criticality rate assignment problem.

[8]For task schedulability, each subproblem assumes that $u_i^L \leq \theta_i^L$, $u_i^M \leq \theta_i^M$, and $u_i^H \leq \theta_i^H$. Also, any rate is no larger than 1 by the definition of execution rate (Def. 2)

As shown in the above example on the 3-criticality system, the extended MC-Derivative is more flexible than the extended OERA. While OERA relies on the formal convex optimization framework, MC-Derivative can be freely extended for various directions (e.g., multi-criticality and online admission control for MC systems).

VII. CONSTRUCTING THE DISCRETE-TIME SCHEDULE FROM THE MC-FLUID SCHEDULE

In the previous section, we presented the optimal rate assignment of MC-Fluid and its exact schedulability analysis. In this section, we discuss the limitation of fluid scheduling platform and present how to overcome the limitation for MC-Fluid.

Many fluid-based scheduling algorithms, including MC-Fluid, rely on fractional (fluid) processor assumption, which is not directly applicable on real (non-fluid) hardware platforms. Overcoming the limitation of fluid-based algorithms, several approaches in the non-MC domain (e.g., [7], [9]–[11]) have been introduced to construct a non-fluid schedule with schedulability equivalent to a fluid-based schedule.

Such approaches are broadly classified into two categories: quantum-based approaches and deadline partitioning approaches. Quantum-based approaches (e.g., [7]) identify the minimal scheduling unit (i.e., a time quantum) in hardware platforms. Every time quantum, they enforce the execution of each task to satisfy that allocation error (the difference between the execution amount of the actual schedule and fluid schedule) is no greater than 1. On the other hand, deadline partitioning approaches (e.g., [9]–[11], [26]) divide the entire timeline by distinct release times and deadlines of the task system. Every time partition, they ensure that allocation error of each task is no greater than 1, which suffices optimality.

A. Recapitulation of Boundary Fair (BF) Algorithm

To construct a discrete-time schedule from the schedule of MC-Fluid, we choose BF scheduling algorithm ([11], [26]) because deadline partitioning approaches have lower numbers of preemption than quantum-based approaches and among them, only BF considers discrete execution units for realistic platforms.

Before presenting BF, we present the task model for discrete-time schedule and define the lag of a task. Each task τ_i is characterized by (T_i, C_i), where $T_i \in \mathbb{N}$ is the period and $C_i^L \in \mathbb{N}$ is the WCET.

Definition 8 (from [26])**.** *The lag of task τ_i at time t is the difference between the execution amount of a job of τ_i in the fluid schedule and the one in the actual schedule after transformation:*

$$\mathsf{lag_i}(t) \overset{\mathsf{def}}{=} \delta_i \cdot (t - a_i(t)) - \mathsf{AL_i}(a_i(t), t) \qquad (19)$$

where δ_i is the density (a ratio of execution time over deadline) of task τ_i, $a_i(t)$ is the release time of the active job of τ_i at time t, and $\mathsf{AL_i}(t_1, t_2)$ is the actual resource allocation of τ_i in time interval $[t_1,\ t_2)$.

Next, we explain the granularity of schedule transformation in BF. *Boundary time* is defined as a distinct release time or deadline of any job in the system. BF transforms the fluid schedule into an actual discrete-time schedule every boundary time. We explain how BF allocates resources for a time interval between two consecutive boundary times[9]. For each task τ_i, BF allocates mandatory execution amount (a rounded value of execution amount from the fluid schedule) and optional execution amount (up to one) to preserve the optimality of the fluid schedule. Mandatory execution amount of τ_i (denoted as M_i) for a time interval from b_k to $b_k + 1$ is computed as follows:

$$M_i(b_k, b_{k+1}) = \max(0, \lfloor \mathsf{lag_i}(b_k) + (b_{k+1} - b_k) \cdot \delta_i \rfloor).$$

If there are idle resources after allocating the mandatory execution amount for every task, BF allocates optional execution amount to some unfinished jobs (refer [26]). The following lemma shows a scheduling property of BF.

Lemma 10 (from [26]). *In BF, at any boundary time b_k, it holds for every task τ_i that $\mathsf{lag_i}(b_k) < 1$.*

If the execution time of a task is integer, by Lemma 10, the lag at the deadline of the task is always less than zero, which guarantees that the task meets its deadline.

The following lemma shows the schedulability condition of BF for a given task set.

Lemma 11 (from [26]). *A non-MC task set τ is schedulable by BF iff $\sum_{\tau_i \in \tau} \delta_i \leq m$.*

B. *MC-Discrete Scheduling Algorithm*

To extend BF into MC systems, we redefine our system model in MC systems. According to the assumption under BF, the task model is modified: a MC task τ_i has integer LO- and HI-WCETs ($C_i^L \in \mathbb{N}$ and $C_i^H \in \mathbb{N}$).

There are three challenges to extend BF to construct a discrete-time schedule equivalent to the MC-Fluid schedule. The first challenge is how to transform the execution rate of MC-Fluid into BF. Since the fluid scheduling model for non-MC systems executes a task at a constant rate, BF allocates resources to a task statically based on its density. However, since MC-Fluid switches the execution rate of a task at mode switch, in our discrete-time schedule, we will allocate resources to a task based on two densities, LO-density (LO-criticality density) and HI-density (HI-criticality density). A task initially receives resources based on its LO-density and based on its HI-density after the decision of density change (discussed later).

The second challenge is how to apply boundary times of the BF algorithm for MC systems. If we apply boundary times of BF, then the mode switch time instant may be delayed because the actual resource allocation between two boundary times could be different from the MC-Fluid schedule. Then, even if an HI-task executes with HI-density (transformed from HI-execution rate) after the mode switch, the task cannot meet its deadline due to the smaller remaining time to deadline, compared to that of the MC-Fluid schedule. To guarantee that mode switch does not happen beyond the worst-case mode switch time in MC-Fluid, we introduce virtual deadlines (VDs) indicating the time instant of the worst-case mode switch in MC-Fluid. By dividing the timeline by the VDs of tasks, the scheduler enforces each task to execute its LO-WCET until its VD. Then, the worst-case mode switch instant is no later than the one in the MC-Fluid schedule.

The third challenge is how to determine the time when the scheduler changes the density of a task, from LO-density to HI-density. Although the delay of the worst-case mode switch time is prevented, direct application of MC-Fluid (changing the density of a task at mode switch) may lead to deadline miss of a task because the executed time of the task until mode switch could be smaller than the one in the fluid schedule. Our previous work [12] based on DP-Fair [10] (allowing real-number execution time) guarantees the equivalence of the executed time by changing the density at the next boundary time after the mode switch. Then, since $\mathsf{lag_i}(b_k) = 0$ at every boundary time, HI-tasks receive the same amount of execution time as the fluid schedule. However, BF (allowing only discrete execution time) does not provide such a guarantee, i.e., $\mathsf{lag_i}(b_k)$ may be greater than 0 (Lemma 10)[10]. To solve the issue, we present a new policy for the density change: instead of globally switching the densities of all tasks at a certain boundary time, we independently switch the density of each task at its VD. Then, each task receives the same amount of execution time as the fluid schedule until its VD because $\mathsf{lag_i}(b_k) = 0$ at the VD of task τ_i.

Addressing the above challenges, we present a new scheduling algorithm, called MC-Discrete.

Definition 9. *MC-Discrete scheduling algorithm is defined with virtual deadlines. For each task τ_i, we define a VD: $V_i \in \mathbb{N}$ s.t. $0 < V_i \leq T_i$. At mode switch, MC-Discrete drops all LO-tasks. After the mode switch, MC-Discrete changes the density of a task at its VD: each task executes with its LO-density until the VD of the active job of τ_i at mode switch and with its HI-density after the VD of the job, according to BF.*

Density Computation.

For a LO-task, we assign its LO-density to task utilization as in the original BF: $\delta_i^L \overset{\mathsf{def}}{=} C_i^L / T_i$.

For a HI-task, we need to assign its LO-density and its HI-density. In MC-Discrete, to execute a HI-task for its LO-WCET until its VD, we assign LO-density as $\delta_i^L \overset{\mathsf{def}}{=} C_i^L / V_i$.

To compute HI-density (δ_i^H), assume that we know the interval length from the release time of the job to the time instant of actual density change (w_i) and its executed time ($E_i^L(w_i)$ where $E_i^L(t)$ be the execution amount of the job for time interval t with LO-density). Then, δ_i^H is the remaining execution amount over the remaining time to the deadline:

$$\delta_i^H \overset{\mathsf{def}}{=} \frac{C_i^H - E_i^L(w_i)}{T_i - w_i}. \tag{20}$$

Virtual Deadline Assignment. We present the VD assignment in Def. 10 based on the optimal rate assignment of MC-Fluid.

Definition 10 (VD assignment). *We assign $V_i := T_i$ for $\tau_i \in \tau_L$ and $V_i := \lfloor C_i^L / \theta_i^{L*} \rfloor$ for $\tau_i \in \tau_H$ where θ_i^{L*} is the optimal value of θ_i^L in MC-Fluid.*

If task τ_i in MC-Fluid is executed with θ_i^{L*}, the task can execute C_i^L at time C_i^L / θ_i^{L*}. By assigning the VD in MC-Discrete, we enforce the execution until the largest integer time before C_i^L / θ_i^{L*} in the discrete-time schedule. In the next

[9]We will discuss how to generate a schedule avoiding intra-job parallelism from the resource allocation at the end of this section.

[10]It is also possible that $\mathsf{lag_i}(b_k) < 0$ by Lemma 10. However, receiving more resource than the MC-Fluid does not produce any problem.

lemma, we find an equivalent LO- and HI-execution rates to LO- and HI-densities of MC-Discrete in order to apply the schedulability analysis of MC-Fluid.

Lemma 12. *Given a MC task set τ, if VDs are assigned by Def. 10, then (i) $\delta_i^L = u_i^L$ for $\tau_i \in \tau_L$ and (ii) $\delta_i^L = \theta_i^{L\triangle}$ and $\delta_i^H \le \theta_i^{H\triangle}$ for $\tau_i \in \tau_H$ where $\theta_i^{L\triangle} = C_i^L/\lfloor C_i^L/\theta_i^{L*}\rfloor$ and $\theta_i^{H\triangle} = (u_i^H - u_i^L)/(1 - u_i^L/\theta_i^{L\triangle})$.*

Proof: (i) We show that $\delta_i^L = u_i^L$ for $\tau_i \in \tau_L$: $\delta_i^L = C_i^L/V_i = C_i^L/T_i = u_i^L$.

(ii) We show that $\delta_i^L = \theta_i^{L\triangle}$ for $\tau_i \in \tau_H$: $\delta_i^L = C_i^L/V_i = \theta_i^{L\triangle}$. Next, we will show that $\delta_i^H \le \theta_i^{H\triangle}$ for $\tau_i \in \tau_H$. From the MC-Discrete algorithm, w_i is either 0 or V_i. When $w_i = 0$, we have $\delta_i^H = C_i^H/T_i \le \theta_i^{H\triangle}$. When $w_i = V_i$,

$$\delta_i^H = \frac{C_i^H - \delta_i^L \cdot V_i}{T_i - V_i} = \frac{C_i^H - C_i^L/V_i \cdot V_i}{T_i - C_i^L/\theta_i^{L\triangle}} = \frac{u_i^H - u_i^L}{1 - u_i^L/\theta_i^{L\triangle}},$$

which is $\theta_i^{H\triangle}$ by Lemma 6(ii). Thus, $\delta_i^H \le \theta_i^{H\triangle}$. ■

Schedulability Analysis. Since we allow only multiples of the scheduling quantum (integer) in VDs, MC-Discrete has different schedulability condition from MC-Fluid.

Theorem 4. *A MC task set τ is schedulable by MC-Discrete with the VD assignment by Def. 10 if*

$$U_L^L + \sum_{\tau_i \in \tau_H} \theta_i^{L\triangle} \le m.$$

Proof: To show that the task set is schedulable, we need to show schedulability in both LO-mode and HI-mode. Consider LO-mode. We consider $\sum_{\tau_i \in \tau} \delta_i^L$, which is $U_L^L + \sum_{\tau_i \in \tau_H} \theta_i^{L\triangle}$ by Lemma 12, which is less than or equal to m by the assumption. From $\sum_{\tau_i \in \tau} \delta_i^L \le m$, the task set is schedulable in LO-mode by Lemma 11.

Consider HI-mode. We claim that $\theta_i^{H\triangle} \le \theta_i^{H*}$:

$$\frac{u_i^H - u_i^L}{1 - u_i^L/\theta_i^{L\triangle}} \le \theta_i^{H*} \qquad \text{(by the definition of } \theta_i^{H\triangle})$$

$$\Leftrightarrow \frac{u_i^H - u_i^L}{1 - u_i^L/\theta_i^{L\triangle}} \le \frac{u_i^H - u_i^L}{1 - u_i^L/\theta_i^{L*}}, \qquad \text{(by Lemma 6(ii))}$$

which is true because $\theta_i^{L\triangle} = C_i^L/\lfloor C_i^L/\theta_i^{L*}\rfloor \ge \theta_i^{L*}$. Since $\delta_i^H \le \theta_i^{H\triangle}$ by Lemma 12, we have

$$\sum_{\tau_i \in \tau_H} \delta_i^H \le \sum_{\tau_i \in \tau_H} \theta_i^{H\triangle} \le \sum_{\tau_i \in \tau_H} \theta_i^{H*},$$

which is less than or equal to m because the optimal assignment satisfies CON1 in Def. 6. From $\sum_{\tau_i \in \tau_H} \delta_i^H \le m$, the task set is schedulable in HI-mode by Lemma 11. ■

Schedule Generation. We generate a discrete-time schedule of tasks based on their execution allocation within a time slice (the time interval from b_k to b_{k+1}), with the consideration of the sequential task execution requirement that no task executes in parallel. We can apply the schedule generation strategy of BF [11], [26]. In BF, McNaughton's algorithm [27] is applied: the algorithm consecutively fills resources of a processor with tasks one by one within a time slice; if the processor is full, the algorithm fills resources of a new processor from the last allocated task which has remaining execution amount.

VIII. Experiments and Results

In this section, we evaluate the schedulability performance of MC-Fluid framework (MC-Derivative in Sec. VI-A and MC-Discrete in Sec. VII). We compare the framework with other multiprocessor MC scheduling algorithms. These algorithms include GLO (global scheduling approach based on EDF-VD [18]), PAR (partitioned scheduling approach based on EDF-VD [20]) and FP (global fixed-priority scheduling approach [19]). We first describe our random task set generation procedure in the experimental setup section and then discuss the performance results of the algorithms.

A. Experiment Setup

Our experiments are carried out using the *MC-FairGen* task set generator [28], which methodically considers all the task set parameters that affect the performance of MC scheduling algorithms and generate fair task sets. The task parameters and the task set generation procedure are described as follows:

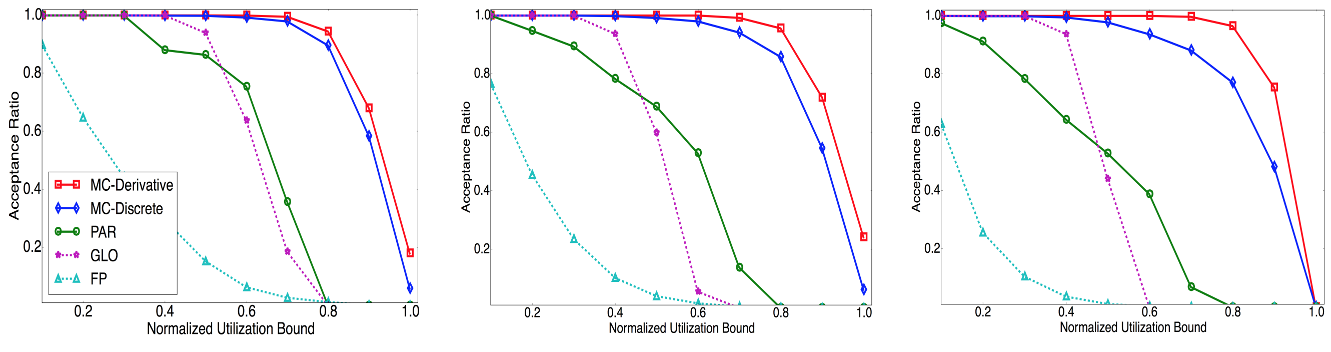
- $m \in \{2, 4, 8\}$ denotes the total number of processors.
- $u_{\min}$ $(= 0.0001)$ and $u_{\max}$ $(= 0.99)$ denote the minimum and maximum task utilization respectively.
- The normalized system-level utilization of HI-tasks in HI mode is given by $U_H^H/m \in [0.1, 0.2, \ldots, 1.0]$.
- The normalized system-level utilization of HI-tasks in LO mode is given by $U_H^L/m \in [0.05, 0.15, \ldots, U_H^H/m]$.
- The normalized system-level utilization of LO-tasks in LO mode is given by $U_L^L/m \in [0.05, 0.15, \ldots, 1 - (U_H^L/m)]$.
- $P_H \in [0.1, 0.2, \ldots, 0.9]$ denotes the percentage of HI-tasks in the system. We use the exact percentage of HI-tasks and not the probability to be an HI-task.
- The total number of tasks in the system is lower bounded and upper bounded by $m + 1$ and $10m$, respectively.
- T_i, the period of task τ_i is drawn uniformly from $[5, 100]$.
- The utilization values u_i^L and u_i^H are generated using standard techniques [28], [29].
- Execution requirements C_i^L and C_i^H are derived as $u_i^L \cdot T_i$ and $u_i^H \cdot T_i$, respectively.

$10,395$ different possible combinations of task parameters were considered and 50 task sets were generated for each combination, resulting in $519,750$ task sets in total.

B. Results

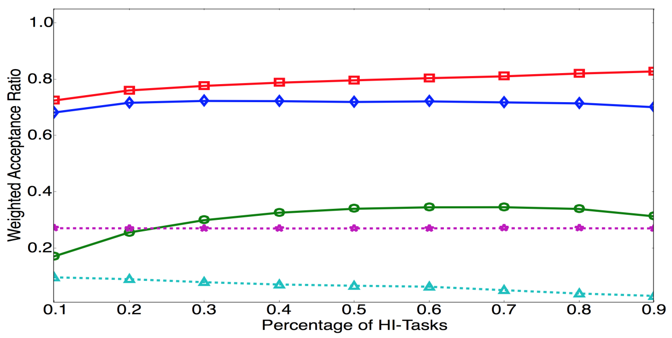
In Fig. 4, we present the overall schedulability of the algorithms. We plot the acceptance ratios of the algorithms i.e., fraction of schedulable task sets, versus normalized utilization bound U_B $(U_B = \max(U_H^H, U_H^L + U_L^L)/m)$ varying over $m \in \{2, 4, 8\}$. It can be observed that all the dual-rate scheduling algorithms perform significantly better than the existing multiprocessor MC scheduling algorithms. As m increases, the performance of all the algorithms decreases in general. But, the performance gap between the MC-Derivative and the other MC algorithms increases as m increases. This shows how well the MC-Derivative scales with m. As seen, though MC-Discrete incurs a very small loss in schedulability for allowing discrete mode switch instant it still performs much better than the existing algorithms.

To evaluate the performance of the algorithms with respect to specific individual task set parameters such as percentage of HI-tasks and maximum individual task utilization, we plot the

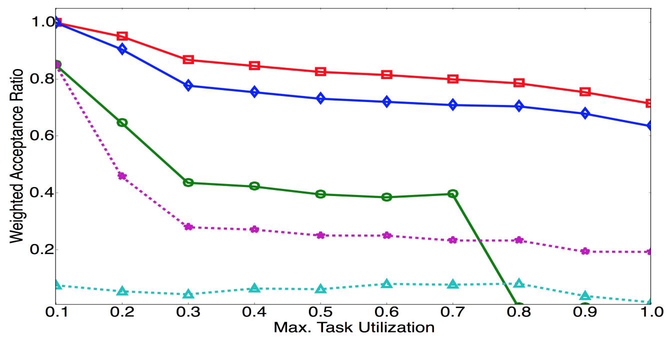


(a) $m = 2$ (b) $m = 4$ (c) $m = 8$

Fig. 4. Overall Schedulability of Multi-Core MC Algorithms



(a) Percentage of HI-Tasks



(b) Maximum Individual Task Utilization

Fig. 5. Varying Individual Task Set Parameters

Weighted Acceptance Ratio [30] of the algorithms in Fig. 5. The legend of Fig. 5 is the same as in Fig. 4. The weighted acceptance ratio is computed as:

$$\mathsf{W}(\mathbb{S}) \overset{\text{def}}{=} \frac{\sum_{U_B \in \mathbb{S}} \mathsf{A}(U_B) \cdot U_B}{\sum_{U_B \in \mathbb{S}} U_B}$$

where $\mathbb{S}$ is the set of U_B values and $\mathsf{A}(U_B)$ is the acceptance ratio for a specific U_B value. Each data point in these plots correspond to at least $15,000$ task sets.

In Fig. 5(a), we compare the weighted acceptance ratio of the algorithms for varying percentage of HI-tasks (P_H). In general, fluid scheduling algorithms (MC-Derivative and MC-Discrete) perform better than all the other algorithms regardless of P_H. The performance of MC-Discrete decreases as P_H increases. For a large number of HI-tasks in the system (high P_H), there are high resource overheads to compute the integer VDs.

In Fig. 5(b), we compare the weighted acceptance ratio of the algorithms for varying maximum individual task utilization ($u_{\max}$). All the algorithms perform well when the $u_{\max}$ values are small, and relatively poorly when they are large. The performance of the fluid algorithms decreases gradually as $u_{\max}$ increases. In the case of PAR, the performance of the algorithm drops significantly as $u_{\max}$ becomes greater than 0.7. This is true because the algorithm fails to schedule any task sets with $u_{\max}$ greater 0.75. In the case of GLO and FP, the performance degradation is gradual as $u_{\max}$ increases.

IX. Conclusion

We presented a multiprocessor mixed-criticality scheduling algorithm, called MC-Fluid, based on the fluid scheduling platform. Given LO- and HI-execution rates per task, we derived an exact schedulability analysis of MC-Fluid on the dual-criticality systems. We presented the MC-Derivative assignment algorithm to determine the optimal execution rates in $O(n \log n)$ time. We presented the MC-Discrete scheduling algorithm, which is a variant of MC-Fluid for mapping fluid schedules into practical discrete-time schedules. In the simulation, we showed that MC-Derivative and MC-Discrete outperform all the existing multiprocessor MC scheduling algorithms. As future work, we plan to extend the MC-Fluid framework to multi-criticality systems with MC-Derivative strategy.

Acknowledgement

This work was supported in part by NSF CNS-1329984, ONR N000141612195 and Global Research Laboratory Program (2013K1A1A2A02078326) through NRF, and the DGIST Research and Development Program (CPS Global Center) funded by the Ministry of Science, ICT & Future Planning. This work was also supported in part by BSRP (NRF-2015R1D1A1A01058713), IITP (2014-0-00065, Resilient Cyber-Physical Systems Research), NRF(2015M3A9A7067220), and DAPA/ADD (High-Speed Vehicle Research Center of KAIST) funded by the Korea Government (MEST/MSIT/MOTIE). This work was also funded in part by the Ministry of Education, Singapore, grant number ARC9/14.

References

[1] *ARINC653 - An Avionics Standard for Safe, Partitioned Systems.* Wind River Systems / IEEE Seminar, 2008.

[2] "ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary," Tech. Rep., Jul. 2009.

[3] P. Prisaznuk, "Integrated modular avionics," in *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, May 1992, pp. 39–45 vol.1.

[4] A. Burns and R. Davis, "Mixed criticality systems – a review," http://www-users.cs.york.ac.uk/burns/review.pdf, 2016, the seventh edition.

[5] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance," *Journal of ACM*, vol. 47, no. 4, pp. 617–643, Jul. 2000.

[6] W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, no. 1, pp. 177–185, 1974. [Online]. Available: http://dx.doi.org/10.1002/nav.3800210113

[7] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '93. New York, NY, USA: ACM, 1993, pp. 345–354.

[8] P. Holman and J. H. Anderson, "Adapting pfair scheduling for symmetric multiprocessors," *J. Embedded Comput.*, vol. 1, no. 4, pp. 543–564, Dec. 2005.

[9] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, Dec 2006, pp. 101–110.

[10] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-FAIR: A simple model for understanding optimal multiprocessor scheduling," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, July 2010, pp. 3–13.

[11] D. Zhu, D. Mosse, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, Dec 2003, pp. 142–151.

[12] 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 *Real-Time Systems Symposium (RTSS), 2014 IEEE*, Dec 2014, pp. 41–52.

[13] S. Baruah, A. Eswaran, and Z. Guo, "MC-Fluid: Simplified and optimally quantified," in *Real-Time Systems Symposium, 2015 IEEE*, Dec 2015, pp. 327–337.

[14] S. Ramanathan and A. Easwaran, "MC-Fluid: rate assignment strategies," in *Workshop of Mixed-Criticality Systems (WMC)*, December 2015.

[15] S. Baruah, V. Bonifaci, G. D'Angelo, 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 *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 145–154.

[16] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, Dec 2007, pp. 239–243.

[17] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *The Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009.

[18] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 166–175.

[19] R. Pathan, "Schedulability analysis of mixed-criticality systems on multiprocessors," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 309–320.

[20] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, vol. 50, no. 1, pp. 142–177, 2014.

[21] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.

[22] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, July 2015, pp. 25–34.

[23] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 135–144.

[24] I. Bate, A. Burns, and R. I. Davis, "A Bailout Protocol for Mixed Criticality Systems," in *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 259–268.

[25] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Systems*, vol. 47, no. 1, pp. 1–40, 2011.

[26] G. Nelissen, H. Su, Y. Guo, D. Zhu, V. Nélis, and J. Goossens, "An optimal boundary fair scheduling," *Real-Time Systems*, vol. 50, no. 4, pp. 456–508, 2014.

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

[28] S. Ramanathan and A. Easwaran, "Evaluation of Mixed-Criticality Scheduling Algorithms using a Fair Taskset Generator," in *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2016.

[29] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens, "Multi-Criteria Evaluation of Partitioned EDF-VD for Mixed-Criticality Systems Upon Identical Processors," in *Workshop on Mixed-Criticality Systems (WMC)*, December 2013.

[30] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," in *Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, July 2010, pp. 33–44.

Jaewoo Lee received an MS from Seoul National University, South Korea, in Computer Science and Engineering in 2008, and a Ph.D. from the University of Pennsylvania, USA, in Computer and Information Science in 2017. His research interests include cyber-physical systems, real-time embedded systems, and model-driven engineering.

Saravanan Ramanathan received the MSc degree from Nanyang Technological University, Singapore, in Embedded Systems in 2014, and the BE degree from College of Engineering Guindy, India, in Electronics and Communication in 2011. He is currently a Ph.D. candidate in School of Computer Science and Engineering at Nanyang Technological University, Singapore. His research interests include embedded real-time systems and cyber-physical systems.

Kieu-My Phan received B.S. degree from KAIST (Korea Advanced Institute of Science and Technology), South Korea, in Computer Science in 2013. Her research interests include system design and analysis with timing guarantees and resource management in real-time embedded systems and cyber-physical systems.

Arvind Easwaran is currently an assistant professor in School of Computer Science and Engineering at Nanyang Technological University, Singapore, where he joined in 2013. He received a Ph.D. from the University of Pennsylvania, USA, in Computer and Information Science in 2008. His research interests lie in cyber-physical systems, embedded real-time systems, and formal methods.

Insik Shin is currently an associate professor in School of Computing at KAIST, South Korea, where he joined in 2008. He received a Ph.D. from University of Pennsylvania, USA, in Computer and Information Science in 2006. His research interests lie in real-time embedded systems, mobile computing, and cyber-physical systems.

Insup Lee is the Cecilia Fitler Moore Professor of Computer and Information Science and the Director of PRECISE Center at the University of Pennsylvania, USA, where he has been since 1983. He received the Ph.D degree from the University of Wisconsin-Madison, USA, in Computer Science in 1983. His research interests lie in cyber-physical systems (CPS), real-time systems, embedded systems, formal methods and tools, medical device systems, and software engineering.