2017

Resource-Efficient Scheduling Of Multiprocessor Mixed-Criticality Real-Time Systems

Jaewoo Lee
University of Pennsylvania, jaewoo@cis.upenn.edu

Resource-Efficient Scheduling Of Multiprocessor Mixed-Criticality Real-Time Systems

Abstract

Timing guarantee is critical to ensure the correctness of embedded software systems that interact with the physical environment. As modern embedded real-time systems evolves, they face three challenges: resource constraints, mixed-criticality, and multiprocessors. This dissertation focuses on resource-efficient scheduling techniques for mixed-criticality systems on multiprocessor platforms.

While Mixed-Criticality (MC) scheduling has been extensively studied on uniprocessor platforms, the problem on multiprocessor platforms has been largely open. Multiprocessor algorithms are broadly classified into two categories: global and partitioned. Global scheduling approaches use a global run-queue and migrate tasks among processors for improved schedulability. Partitioned scheduling approaches use per processor run-queues and can reduce preemption/migration overheads in real implementation. Existing global scheduling schemes for MC systems have suffered from low schedulability. Our goal in the first work is to improve the schedulability of MC scheduling algorithms. Inspired by the fluid scheduling model in a regular (non-MC) domain, we have developed the MC-Fluid scheduling algorithm that executes a task with criticality-dependent rates. We have evaluated MC-Fluid in terms of the processor speedup factor: MC-Fluid is a multiprocessor MC scheduling algorithm with a speed factor of 4/3, which is known to be optimal. In other words, MC-Fluid can schedule any feasible mixed-criticality task system if each processor is sped up by a factor of 4/3.

Although MC-Fluid is speedup-optimal, it is not directly implementable on multiprocessor platforms of real processors due to the fractional processor assumption where multiple task can be executed on one processor at the same time. In the second work, we have considered the characteristic of a real processor (executing only one task at a time) and have developed the MC-Discrete scheduling algorithm for regular (non-fluid) scheduling platforms. We have shown that MC-Discrete is also speedup-optimal.

While our previous two works consider global scheduling approaches, our last work considers partitioned scheduling approaches, which are widely used in practice because of low implementation overheads. In addition to partitioned scheduling, the work considers the limitation of conventional MC scheduling algorithms that drops all low-criticality tasks when violating a certain threshold of actual execution times. In practice, the system designer wants to execute the tasks as much as possible. To address the issue, we have developed the MC-ADAPT scheduling framework under uniprocessor platforms to drop as few low-criticality tasks as possible. Extending the framework with partitioned multiprocessor platforms, we further reduce the dropping of low-criticality tasks by allowing migration of low-criticality tasks at the moment of a criticality switch. We have evaluated the quality of task dropping solution in terms of speedup factor. In existing work, the speedup factor has been used to evaluate MC scheduling algorithms in terms of schedulability under the worst-case scheduling scenario. In this work, we apply the speedup factor to evaluate MC scheduling algorithms in terms of the quality of their task dropping solution under various MC scheduling scenarios. We have derived that MC-ADAPT has a speedup factor of 1.618 for task dropping solution.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Insup Lee

Second Advisor
Linh T. Phan

Keywords
Mixed-Criticality systems, Multiprocessor, Real-time systems, Schedulability analysis, Scheduling algorithm

RESOURCE-EFFICIENT SCHEDULING OF MULTIPROCESSOR MIXED-CRITICALITY REAL-TIME SYSTEMS

Jaewoo Lee

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2017

Supervisor of Dissertation

Insup Lee
Cecilia Fitler Moore Professor of
Computer and Information Science

Co-Supervisor of Dissertation

Linh T.X. Phan
Assistant Professor of
Computer and Information Science

Graduate Group Chairperson

Lyle Ungar, Professor of Computer and Information Science

Dissertation Committee

Oleg Sokolsky, Research Professor of Computer and Information Science

Rahul Mangharam, Associate Professor of Electrical and Systems Engineering

Joseph Devietti, Assistant Professor of Computer and Information Science

Insik Shin, Associate Professor (Korean Advanced Institute of Science and Technology)

RESOURCE-EFFICIENT SCHEDULING OF MULTIPROCESSOR MIXED-CRITICALITY REAL-TIME SYSTEMS

ABSTRACT

RESOURCE-EFFICIENT SCHEDULING OF MULTIPROCESSOR MIXED-CRITICALITY REAL-TIME SYSTEMS

Jaewoo Lee

Insup Lee

Linh T.X. Phan

Timing guarantee is critical to ensure the correctness of embedded software systems that interact with the physical environment. As modern embedded real-time systems evolves, they face three challenges: resource constraints, mixed-criticality, and multiprocessors. This dissertation focuses on resource-efficient scheduling techniques for mixed-criticality systems on multiprocessor platforms.

While Mixed-Criticality (MC) scheduling has been extensively studied on uniprocessor platforms, the problem on multiprocessor platforms has been largely open. Multiprocessor algorithms are broadly classified into two categories: global and partitioned. Global scheduling approaches use a global run-queue and migrate tasks among processors for improved schedulability. Partitioned scheduling approaches use per processor run-queues and can reduce preemption/migration overheads in real implementation. Existing global scheduling schemes for MC systems have suffered from low schedulability. Our goal in the first work is to improve the schedulability of MC scheduling algorithms. Inspired by the fluid scheduling model in a regular (non-MC) domain, we have developed the MC-Fluid scheduling algorithm that executes a task with criticality-dependent rates. We have evaluated MC-Fluid in terms of the processor speedup factor: MC-Fluid is a multiprocessor MC scheduling algorithm with a speed factor of 4/3, which is known to be optimal. In other words, MC-Fluid can schedule any feasible mixed-criticality task system if each processor is sped up by a factor of 4/3.

Although MC-Fluid is speedup-optimal, it is not directly implementable on multiprocessor platforms of real processors due to the fractional processor assumption where multiple task can be executed on one processor at the same time. In the second work, we have considered the characteristic of a real processor (executing only one task at a time) and have developed the MC-Discrete scheduling algorithm for regular (non-fluid) scheduling platforms. We have shown that MC-Discrete is also speedup-optimal.

While our previous two works consider global scheduling approaches, our last work considers partitioned scheduling approaches, which are widely used in practice because of low implementation overheads. In addition to partitioned scheduling, the work considers the limitation of conventional MC scheduling algorithms that drops all low-criticality tasks when violating a certain threshold of actual execution times. In practice, the system designer wants to execute the tasks as much as possible. To address the issue, we have developed the MC-ADAPT scheduling framework under uniprocessor platforms to drop as few low-criticality tasks as possible. Extending the framework with partitioned multiprocessor platforms, we further reduce the dropping of low-criticality tasks by allowing migration of low-criticality tasks at the moment of a criticality switch. We have evaluated the quality of task dropping solution in terms of speedup factor. In existing work, the speedup factor has been used to evaluate MC scheduling algorithms in terms of schedulability under the worst-case scheduling scenario. In this work, we apply the speedup factor to evaluate MC scheduling algorithms in terms of the quality of their task dropping solution under various MC scheduling scenarios. We have derived that MC-ADAPT has a speedup factor of 1.618 for task dropping solution.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1 : Introduction

Embedded software systems interact (i.e., sense and actuate) with the physical environment. Those systems require not only logical correctness but also temporal correctness. For example, when the autonomous vehicle decides to activate its brake based on the detection of road obstacles, the brake component should be activated within the predefined timing constraints, e.g., 100ms. Otherwise, the car may not avoid the collision, which could result in property damage, injury or even death. *Real-time systems* are software systems subject to timing constraints.

Traditional embedded real-time systems have been evolving into modern *Cyber-Physical Systems (CPS)*, which are intelligent systems interacting with the physical environment. CPS applications include automobiles (e.g., autonomous driving vehicles), avionics (e.g., unmanned aerial vehicles), and health-care applications (e.g., surgery robots). CPS are increasingly becoming complex systems with multiple subsystems of different criticalities.

There are three important challenges for modern real-time systems. The first challenge is resource constraint. Since most embedded systems are constrained by Size, Weight and Power (SWaP), we need resource-efficient scheduling techniques, which require less resources for the same functionality. For example, the automotive industrial standard AUTOSAR [3] aims to minimize the number of ECUs[1] and their wiring (through software components), due to the SWaP constraints of the automobile and manufacturing cost. A similar trend is also observed in the avionics industry (e.g., the Integrated Modular Avionics architecture [44]).

The second challenge is *Mixed-Criticality (MC)*. In real-time systems, components of different criticalities are often integrated under a shared platform. For example, the RTCA DO-178B software standard (see Table 1), which is applied by the US FAA (Federal Aviation Administration), specifies criticality levels of functions in aircraft, depending on the effects of their failures. The failure of criticality level A function may result in an aircraft crash,

[1] An Electronic Control Unit (ECU) is a physical container of an embedded system controlling a subsystem of an automobile

Level	Failure Condition	Interpretation
A	Catastrophic	Failure may cause a crash
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the plane due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers
C	Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)
D	Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload

Table 1: DO-178B, software considerations in airborne systems and equipment certification, published by RTCA, Incorporated

while the failure of criticality level E function results in no impact on aircraft safety. For safety assurance, we need MC scheduling techniques in which a low-criticality functionality cannot affect the correctness of a high-criticality functionality.

The third challenge is multiprocessor. Traditional uniprocessor platforms have the limitations on power consumption and heat dissipation. In 2004, Intel killed the plan to release the 4Ghz Pentium4 processor due to power and heat problems, and shifted all development teams to multicore processors. Overcoming the limitations, multiprocessor platforms are dominant across a wide range of domains (including CPS applications). Although multiprocessor scheduling is much studied on the conventional (non-MC) domain, it has not sufficiently matured on the MC domain.

1.1. The Goal of the Dissertation and Our Approaches

The goal of this dissertation is to develop resource-efficient MC scheduling techniques on multiprocessor platforms. An MC scheduling problem is the validation of multiple level timing constraints of the MC system. While high-criticality tasks must use the Worst-Case

Execution Time (WCET) estimate under pessimistic conditions, low-criticality tasks may use the WCET parameters under normal conditions. Then, the MC problem is to guarantee temporal correctness of both (i) all tasks under normal WCET assumptions and (ii) only high-criticality tasks under conservative WCET assumptions. While the MC scheduling problem has been extensively studied for uniprocessor platforms, it has largely remained open for the multiprocessor case.

Multiprocessor scheduling can be categorized depending on the resource allocation: *global* and *partitioned* scheduling approaches. While global scheduling approaches globally schedule tasks with a single run-queue, partitioned scheduling approaches statically assign tasks to separate per-processor run-queues and independently schedule tasks within a processor. Generally, neither global scheduling approaches nor partitioned scheduling approaches perform always better. Our work considers global approaches in the first two studies and partitioned approaches in the last study.

1.1.1. Fluid-based Mixed-Criticality Scheduling

Our goal is to develop a resource-efficient multiprocessor MC scheduling technique. A challenge is that existing multiprocessor MC scheduling approaches suffer from low schedulability due to known multiprocessor scheduling problems (e.g., Dhall's effect [22] and bin-packing problem [35]). There exists a solution to the problem in the non-MC domain: fluid scheduling model [9] (executing each task with each static rate), which is a global scheduling approach. However, its direct application is inefficient in the MC multiprocessor domain because it does not address the MC problem. EDF-VD [4] is a uniprocessor MC scheme with optimal *speedup factor*[2]. Inspired by the fluid scheduling model and EDF-VD, we develop a new multiprocessor MC scheduling algorithm, called *MC-Fluid*, which preserves the speedup-optimality of EDF-VD.

[2]MC scheduling algorithms are often evaluated by a metric of the processor speedup factor [36], which is a theoretical performance metric of a scheduling algorithm in comparison with an optimal algorithm.

1.1.2. Transforming Fluid-based Mixed-Criticality Scheduling into Discrete-time Platforms

Although MC-Fluid is a speedup-optimal scheduling algorithm for multiprocessor MC systems, it is not directly implementable on real hardware platforms due to the unrealistic assumption on fluid-based scheduling (i.e., executing a task with a fractional processor). In real hardware platforms, a processor executes only one task at a time. Our goal is to develop an MC scheduling technique considering realistic multiprocessor platforms. A challenge is how to transform the fluid schedule of MC tasks into a non-fluid schedule. In the non-MC domain, there have been scheduling techniques [9, 20, 38] that transform the fluid schedule into a non-fluid schedule without the loss of schedulability. Inspired by these results, we develop two non-fluid scheduling algorithms (transforming the MC-Fluid schedule into a non-fluid schedule), called *MC-DP-Fair* and *MC-Discrete*, both of whom have comparable schedulability with MC-Fluid.

1.1.3. Adaptive Mixed-Criticality Scheduling on Partitioned Multiprocessor Platforms

While global scheduling approaches (including MC-Fluid and MC-Discrete) shows high schedulability, they have large preemption/migration overheads and the implementation overheads with global run-queue[3] from a practical perspective. In the domain where implementation overheads are critical, partitioned scheduling approaches are adopted. In this work, we develop a new partitioned scheduling algorithm. By assigning tasks to processor statically, we eliminate migration between processors. By using a separate run-queue per processor, we reduce the overhead of the global shared run-queue. By directly applying uniprocessor MC scheduling techniques (e.g., EDF-VD [4]) for each processor, we can reduce the number of preemption in global MC scheduling algorithms such as MC-Fluid and MC-Discrete.

Conventional MC scheduling algorithms (including MC-Fluid and MC-Discrete) have the limitation for low-criticality tasks: they drops all low-criticality tasks at mode switch (any high-criticality task executes more than its normal WCET estimate). However, industrial

[3]In large systems, it could be excessive to manipulate a single run-queue for multiple processors.

demand is toward *adaptive MC scheduling*, which maximizes the survivability of each low-criticality task, rather than ignoring all of them after system-level mode-switch [19]. In this work, we develop an adaptive MC scheduling approach by dropping low-criticality tasks selectively under a task-level mode-switch in which each high-criticality task is independently mode-switched.

We present the *MC-ADAPT* scheduling framework for adaptive MC scheduling with partitioned scheduling approach. Extending the existing EDF-VD under task-level mode-switch mechanism, we have developed a uniprocessor adaptive MC scheduling algorithm that drops as few low-criticality tasks as possible. We evaluate the effectiveness of our algorithm for task dropping via the speedup factor. We extend our uniprocessor scheme with a semi-partitioned scheduling approach, which is a variant of partitioned scheduling approach, to further reduce task dropping via task migration when the system criticality changes.

1.2. Contributions

We make the following research contributions:

- We present MC-Fluid scheduling algorithm for multiprocessor MC systems (in Chapter 3). To our best knowledge, this is the first work to apply the fluid scheduling model (for global multiprocessor scheduling) to the MC domain. We develop the optimal algorithm for criticality-dependent execution rate assignment. We derive that the speedup factor of MC-Fluid is $4/3$, which is optimal for MC scheduling.

- Relaxing the fluid-based scheduling assumption, we present MC-DP-Fair and MC-Discrete scheduling algorithms for realistic multiprocessor platforms (in Chapter 4). Inspired by DP-Fair in the non-MC domain, we develop an MC-DP-Fair scheduling algorithm which preserves the speedup optimality of MC-Fluid. Overcoming the limitation of MC-DP-Fair (namely, the use of real-number deadlines in a low-criticality mode), we develop a MC-Discrete scheduling algorithm which uses only integer deadlines for a realistic task scheduler. Our simulation results show that our scheme

significantly outperforms other existing multiprocessor MC schemes.

- We present the MC-ADAPT scheduling framework for adaptive MC scheduling on partitioned multiprocessor platforms (in Chapter 5). We develop two new uniprocessor MC scheduling algorithms, called *EDF-AD* and *EDF-AD-E*, which drop low-criticality tasks selectively under task-level mode-switch. We apply the speedup factor for task dropping and derive that EDF-AD-E has a speedup factor of 1.618 for task dropping. We extend our algorithms into multiprocessor platforms with partitioned scheduling. Our simulation results show the effectiveness of our framework in terms of schedulability and resource utilization.

1.3. Organization

The rest of the dissertation is organized as follows: we describe the background, related work, and the summary of the existing work in Chapter 2. We present the fluid-based MC scheduling algorithm in Chapter 3. Relaxing the fluid scheduling assumption, we present the discrete, non-fluid variations of MC scheduling algorithms in Chapter 4. Addressing the limitations of global multiprocessor scheduling approaches and conventional MC scheduling approaches, we present adaptive MC scheduling framework with a partitioned multiprocessor scheduling approach in Chapter 5. Finally, we conclude the dissertation in Chapter 6.

CHAPTER 2 : Background and Related Work

In this chapter, we introduce the background of our research and related work.

2.1. Real-time Systems

Real-time systems require not only logical correctness but also temporal correctness. The temporal correctness in real-time systems generally means that the required computation is completed within a predefined deadline. The fundamental problem for real-time systems is how to assure temporal correctness. This problem can be divided into two sub-problems: scheduling algorithm (how to schedule tasks in a system) and its schedulability analysis (whether the system is schedulable by the scheduling algorithm).

Assumptions. The scheduling algorithm and its analysis are based on various underlying assumptions. A real-time system is generally constructed based on the task and platform models. The task model defines the timing requirement. An example of task model is periodic task model $\tau_i = (T_i, C_i)$, where T_i is an inter-arrival time between its instances (called *jobs*) and C_i is the Worst-Case Execution-Time (WCET) of a job. A task τ_i invokes a series of jobs, where a job has its WCET (C_i) and its deadline (which we assume equals to job release time plus T_i). On the other hand, platform model defines scheduling environments where tasks execute. An example of platform model is a preemptive multiprocessor platform where a higher-priority job can preempt a lower-priority job at any time instant and there are m processors that can execute $n \leq m$ jobs simultaneously.

Scheduling Algorithms. We can categorize scheduling algorithms depending on the number of processors: uniprocessor and multiprocessor scheduling algorithms. While uniprocessor scheduling has been extensively studied, due to the limitation of uniprocessor systems, multiprocessor scheduling has received much attention recently. Multiprocessor scheduling algorithms can be classified into global and partitioned scheduling approaches. Global scheduling approaches allow task migration between processors with a single run-queue.

Partitioned scheduling approaches statically assign tasks into separate per-processor run-queues and do not allow task migration between processors. After partitioning, we can apply a uniprocessor scheduling technique within each processor.

Performance Metrics. A scheduling algorithm and its analysis can be evaluated in various performance metrics: acceptance ratio for random task sets, speedup factor, deadline miss ratio, etc. The acceptance ratio for random task sets is an empirical performance metric to represent the ratio of task sets schedulable by the scheduling algorithm for randomly-generated task sets. The deadline miss ratio for a given system is the ratio of the number of missed deadlines to the total number of deadlines in the observed time window. Recently, many studies evaluate their scheduling solution in terms of the speedup factor [36]. The factor $\alpha \in \mathbb{R}$ s.t. $\alpha \geq 1$ compares the worst-case behavior of different solution for solving the same problem. The smaller speedup factor of the solution indicates that the behavior of the algorithm is closer to that of the optimal solution.

2.2. Mixed-Criticality Scheduling

Mixed-Criticality (MC) systems are a part of real time systems. An MC system integrates functions with different criticalities (or importance) in a single shared platform. Good examples are Integrated Modular Avionics (IMA) [44] in avionics and AUTOSAR [3] in automotive systems. In MC systems, a different function may have a different criticality. A scheduling problem for MC systems is the validation of the temporal correctness of the system on multiple levels. Many MC systems are subject to certification requirements: the temporal correctness of their safety-critical functionalities must be certified with extremely pessimistic assumptions by certification authorities (CAs). On the other hand, system designers are concerned about the temporal correctness of all functionalities with normal assumptions. For example, consider an unmanned aerial vehicle (UAV) with flight-critical (high-critical) functions (e.g., engines) and mission-critical (low-critical) functions (e.g., surveillance cameras). System designers of the UAV examine the temporal correctness of all functions by empirical measurements assuming typical runtime conditions. To operate

the UAV, it is mandatory that civilian CAs, such as the US FAA, must certify flight-critical functions. The CA usually examines flight-critical functions of the UAV by rigorous static timing analysis assuming extreme runtime conditions (e.g., all cache miss).

For simplicity, we consider dual criticality levels: high-criticality (HI) and low-criticality (LO). Although some MC schemes are extended into arbitrary levels of criticality, most standard MC schemes only consider dual criticality levels for simplicity of representation.

Task Model. We define an MC task model: an MC task τ_i is characterized by $(T_i, C_i^L, C_i^H, \chi_i)$, where

- T_i is the minimum inter-job separation time,
- C_i^L is an LO-criticality WCET (LO-WCET),
- C_i^H is a HI-criticality WCET (HI-WCET), and
- χ_i is a task criticality level (HI or LO).

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. Since HI-WCETs are based on conservative assumptions, we assume that $0 < C_i^L \leq C_i^H \leq T_i$. Each MC task is either a LO-criticality task (LO-task) or a HI-criticality task (HI-task) depending on its task criticality level.

Task Sets. We consider an 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 $\tau_L \stackrel{\mathsf{def}}{=} \{\tau_i \in \tau | \chi_i = LO\}$ and $\tau_H \stackrel{\mathsf{def}}{=} \{\tau_i \in \tau | \chi_i = HI\}$.

Behavior Model. We assume some degree of uncertainty on the execution time of different jobs for a task. We consider task-level criticality mode (*task mode*). Each HI-task τ_i has its own task mode (denoted as M_i) that indicates its behavior. A task τ_i is said to be in LO-mode ($M_i = \mathsf{LO}$) if no job of the task has executed more than its LO-WCET (C_i^L), and in HI-mode ($M_i = \mathsf{HI}$) otherwise.

In Chapters 3 and 4, we assume system-level mode-switch, which is widely used in conventional MC scheduling approaches. 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 every HI-task τ_i has $M_i = \mathsf{LO}$. In HI-mode, we assume that every HI-task τ_i has $M_i = \mathsf{HI}$. We assume the following scenario:

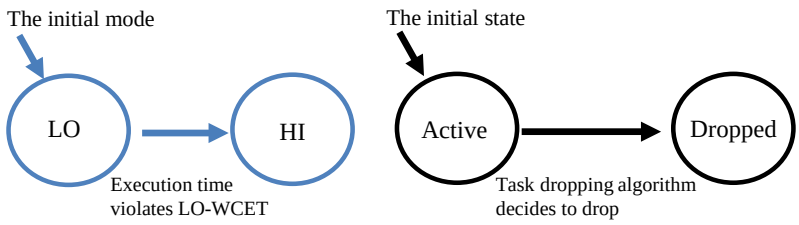
- The system starts in LO-mode. In LO-mode, jobs of all LO- and HI-tasks are released.
- If a job of any HI-task $\tau_i \in \tau_H$ executes for more than its LO-WCET (C_i^L), the system switches the system mode from LO to HI (called *mode-switch*). At mode-switch, the system immediately discards all 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 any further. Therefore, we assume that $C_i^L = C_i^H$ for each LO-task $\tau_i \in \tau_L$ without loss of generality.

In Chapter 5, we assume task-level mode switch (an individual task changes its task mode independently), which is adopted in recent adaptive MC scheduling approaches. Each HI-task starts in LO-mode, and switches to HI-mode when its execution time violates C_i^L (called *mode switch*) (see Fig. 1a). It is worth noting that system-level mode switch, which is adopted in most existing MC schemes, is a special case of task-level mode switch.

We introduce the execution state of an LO-task (see Fig. 1b): each LO-task is in either an *active* state or *dropped* state. Initially, all LO-tasks are in the active state. On mode switch (from the active state to the dropped state), some LO-tasks may be dropped in order to support HI-tasks with their additional resource requests.

System Goal. MC systems require different levels of assurance. We define the schedulability requirement of MC systems as follows:



(a) The task mode of a HI-task (b) The execution state of an LO-task

Figure 1: The behavioral model of tasks

Definition 2.1 (MC-schedulability)**.** *For a given task set, the system is MC-schedulable if*

- ***Condition A****: HI-tasks are always schedulable.*
- ***Condition B****: LO-tasks are schedulable if no HI-task has shown HI-behavior.*

In Chapters 3 and 4, we consider MC-schedulability as the system goal. In Chapter 5, we consider a different system goal. It is generally important to maximize the performance of LO-tasks [18]. Recent adaptive MC scheduling approaches also consider the survivability of each LO-task as well as MC-schedulability. Thus, the system goal of adaptive MC scheduling is to drop as few LO-tasks as possible under MC-schedulability.

Utilization. LO- and HI-task utilizations of a task τ_i are defined as

$$u_i^L \stackrel{\mathsf{def}}{=} C_i^L/T_i \text{ and}$$
$$u_i^H \stackrel{\mathsf{def}}{=} C_i^H/T_i,$$

respectively.

System-level utilizations of a task set τ are defined as

$$U_L^L \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L,$$
$$U_H^L \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L, \text{ and}$$
$$U_H^H \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H.$$

We define criticality-dependent system utilizations. The LO-criticality and HI-criticality utilizations are defined as $U_L^L + U_H^L$ and U_H^H, respectively.

2.3. Related Work

Since the seminal work of Vestal [49], a vast amount of work has been studied for MC scheduling (see [19] for a survey). In this section, we will look at our related work on uniprocessor MC scheduling, multiprocessor MC scheduling, and adaptive MC scheduling.

2.3.1. Mixed-Criticality Scheduling on Uniprocessor Platforms

Vestal [49] introduced an MC scheduling problem with multiple levels of execution time estimates. Vestal observed that HI-tasks tend to have more pessimistic Worst-Case Execution Time (WCET) estimates than LO-tasks. While HI-tasks must use WCET parameters from extremely conservative WCET tools such as static analysis, LO-tasks may use WCET parameters from common measurement-based WCET tools. Based on this observation, Vestal proposed a task model with multiple WCET parameters, which are different viewpoints of WCET for the same executable-code.

There are rich literatures for Vestal's MC task model. Vestal found that neither a rate-monotonic algorithm nor a criticality-monotonic algorithm is optimal in the fixed-priority (FP) scheduling with the MC task model [49]. He applied Optimal Priority Assignment (OPA) [2], which is optimal in the FP scheduling. Dorin et al. [23] presented the formal proof that the OPA algorithm can find an optimal assignment of task priorities for Vestal's

response time analysis. However, Baruah and Vestal [8] showed that the OPA algorithm is not yet optimal in general MC scheduling and proposed a hybrid-priority scheduling scheme combining the feature of FP algorithms and Earliest Deadline First (EDF) algorithms.

Recent MC schemes have advanced Vestal's task model with runtime monitoring, which is a platform feature to monitor execution-times of MC jobs on runtime. With runtime monitoring features, the system can continuously monitor the current execution-times of jobs and cancel jobs if they execute more than their expected WCETs. This mechanism can potentially improve the performance of MC systems.

Some early studies for runtime monitoring [7, 31, 39] addressed priority assignment problems for an individual job. These approaches share the principle that each job has a single priority regardless of the system mode. The first paper to consider runtime monitoring was the Own Criticality-Based Priority (OCBP) algorithm [7] for the restricted problem of scheduling a finite set of MC jobs. Because scheduling MC jobs even for dual-criticality systems is strongly NP-hard [5], OCBP presented a sufficient, rather than exact, schedulability analysis. It was later extended to sporadic task systems [31, 39]. On runtime, OCBP-load [39] recomputes priority assignment considering a busy interval (a time interval which has no idle instant). OCBP-priority [31] reduces high runtime overheads of the OCBP-load.

The above principle is extended into task-level priority assignment. The Adaptive Mixed-Criticality (AMC) approach [13] proposed a fixed-priority scheduling to find a single task-level priority regardless of the system mode. A body of studies [4, 10, 24, 25, 26] have considered another principle that a job may have different priorities in different modes. Since workload characteristics are different in LO and HI modes, these approaches find a best job priority for each mode, which leads to better schedulability. The first such approach was the Earliest Deadline First with Virtual Deadlines (EDF-VD) [10], which proposed the MC-EDF algorithm and its analysis. Multiplying the real deadline by a single system-wide parameter, EDF-VD computes virtual-deadlines. By using task-level parameters, Ekberg and Yi [25, 26] proposed another virtual deadline scheme. Easwaran [24] presented a tighter

schedulability analysis for task-level virtual assignment schemes.

Speedup factor [36] is widely used to evaluate MC scheduling algorithms [4, 12, 6, 37, 41]. In uniprocessor MC scheduling, Baruah et al. [4] proposed EDF-VD with a speedup factor of $4/3 (\approx 1.333)$, which is optimal.

2.3.2. Mixed-Criticality Scheduling on Multiprocessor Platforms

Unlike the uniprocessor case, the multiprocessor case has not been studied much. There are some global multiprocessor scheduling approaches [1, 40, 43]. Anderson et al. [1] first considered multiprocessor MC scheduling with a two-level hierarchical scheduler. Pathan [43] proposed a global fixed-priority multiprocessor scheduling algorithm for MC task systems. Li et al. [40] introduced a global scheduling algorithm with a speedup factor of $1 + \sqrt{5}$ (≈ 3.236).

Recently, researchers have studied partitioned scheduling approaches that can directly apply the result of uniprocessor MC scheduling [11, 28, 46]. Baruah et al. [11] presented a partitioned scheduling algorithm with a speedup factor of $8/3$ (≈ 2.666). Gu et al. [28] proposed a partitioned scheduling approach considering task-level virtual deadline assignment based on Ekberg and Yi [25]. Ren and Phan [46] proposed another partitioned scheduling approach considering multiple parameters (period and criticality-dependent utilizations), which reduces resource overbooking in MC scheduling. However, the global scheduling approaches suffers from Dhall's effect [22], and the partitioned scheduling approaches suffers from the bin-packing problem, similar to non-MC multiprocessor scheduling.

Overcoming the limitation of previous multiprocessor MC scheduling, MC-Fluid [37, 45, 6] were proposed based on the fluid scheduling model, which is the optimal scheduling model in non-MC multiprocessor scheduling and does not suffer from Dhall's effect or the bin-packing problem. Lee et al. [37] first proposed MC-Fluid with a speedup factor of $(1 + \sqrt{5})/2$. Later, Baruah et al. [6] improved its speedup factor to $4/3$ (optimal) through a simplified version of MC-Fluid, called MCF. Ramanathan and Easwaran [45] proposed another simplified version of MC-Fluid with better performance than MCF. Chapters 3 and 4 are based on our previous

work [37].

2.3.3. Adaptive Mixed-Criticality Scheduling

Conventional MC scheduling algorithms consider only the schedulability of HI-tasks when any task exceeds its LO-WCET while the industrial demand is toward adaptive MC scheduling, which also considers the survivability of each LO-task [18]. Most of the existing MC scheduling approaches employ an assumption of *system-level mode switch* that when a task violates its LO-WCET, all the other HI-tasks also show the same behavior simultaneously. There are scheduling techniques to execute LO-tasks longer by adjusting the transition between LO-mode and HI-mode [47, 29, 16]. Santy et al. [47] proposed a method that recomputes LO-WCETs of HI-tasks as long as they do not compromise the deadlines of HI-tasks. Gu et al. proposed a method to delay mode-switch with runtime computation [29]. Bate et al. [16] presented a scheduling protocol for returning to the LO-mode so as to resume the execution of LO-tasks. Other works [18, 27, 29, 34, 41, 48] provided degraded service to LO-tasks after system-level mode switch, which includes stretching their periods [18, 34, 48], lowering their priorities [18], skipping their jobs [27], or reducing their execution times [41, 29]. However, all the above studies share the assumption of system-level mode switch, which results that resources are still under-utilized in practice.

Relaxing the assumption of system-level mode switch, recent studies [33, 46, 30] considered task-level mode switch that enables LO-tasks to be penalized selectively in the event of individual mode switch. Huang et al. [33] proposed offline mapping from each HI-tasks to multiple LO-tasks: when the HI-task mode switches, the connected LO-tasks are dropped. Ren and Phan [46] proposed a similar technique with exclusive task grouping where each group has at most one HI-task. Gu et al. [30] also presented a task grouping technique that allows multiple HI-tasks. Within the predefined tolerance limit of a task group, it drops only LO-tasks within the task group; if it exceeds this tolerance limit, it drops all of the entire LO-tasks in the system.

Our work (Chapter 5) differs from these approaches in that it makes an online task dropping decision differently by adapting to the dynamically changing system state under task-level mode switch. It is worth noting that the system state comprises of not only the information considered by the above approaches but also new additional information, such as dropped LO-tasks. Since the speedup factor for the MC scheduling problem cannot evaluate the quality of task dropping, we apply the speedup factor for task dropping. Then, we evaluate the speedup factor of MC-ADAPT for task dropping, which is 1.618.

2.4. Recapitulation of Existing Work

We review the fluid scheduling platform to develop the MC-Fluid scheduling framework (Chapter 2.4.1). To transform the MC-Fluid schedule to a non-fluid schedule on realistic multiprocessor platforms, we review a DP-Fair scheduling technique that transforms any fluid schedule to a non-fluid schedule (Chapter 2.4.2). For adaptive MC scheduling on partitioned multiprocessor platforms, we review a uniprocessor EDF-VD scheduling algorithm (Chapter 2.4.3).

2.4.1. Fluid Scheduling Model

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 as the *fluid scheduling platform* [9, 32].

Definition 2.2 (Fluid scheduling platform [32])**.** *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 each task with its *execution rate*.

Definition 2.3 (Execution rate)**.** *A task τ_i is said to be executed with the* ***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 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. $t_1 \leq t_2$[1].

The 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 [9, 20, 38, 50] 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 the 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 $\sum_{\tau_i \in \tau} \theta_i \leq m$.

Lemma 2.1 presents platform feasibility for the fluid model, which is also applicable on MC systems.

Lemma 2.1 (Platform feasibility, from [9])**.** *Given a task set τ, all tasks can be executed with their execution rates iff $\sum_{\tau_i \in \tau} \theta_i \leq m$.*

2.4.2. DP-Fair Scheduling Model

Many fluid-based scheduling algorithms, including MC-Fluid, rely on the fractional (fluid) processor assumption. Due to this assumption, the algorithms cannot schedule tasks on real (non-fluid) hardware platforms. Overcoming the limitation of fluid-based algorithms, several approaches (e.g.,[9, 20, 38, 50]) have been introduced to construct a non-fluid schedule, while holding an equivalent schedulability to that of a fluid-based schedule.

Such approaches differ in the unit of a time interval over which they enforce the equivalence of fluid-based and non-fluid schedules. Quantum-based approaches (e.g., [9]) identify the

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

minimal scheduling unit (i.e., a time quantum) in hardware platforms: for every time quantum, they enforce the execution of every task to satisfy that the difference of the execution amount between the actual schedule and the fluid schedule is no greater than 1. Deadline partitioning approaches (e.g., [20, 38, 50]) enforce every task to meet the fluid scheduling requirement at only all distinct deadlines of the system, which suffices with respect to schedulability.

DP-Fair enforces the fluid requirement at every *time slice*. A time slice is defined as a time interval between two consecutive *Deadline Partitions (DPs)*, where a DP is defined as a distinct release time or deadline from all jobs in the system. For a time slice, DP-Fair ensures that every task executes for its execution requirement in fluid platforms by the end of the interval.

We present how to allocate resources to each task in a time slice. Let l be the length of the time slice. For a non-MC task τ_i, DP-Fair allocates $l \cdot \delta_i$ where δ_i is density of the task ($\delta_i = C_i/T_i$ where T_i is its period and C_i is its WCET).

Next, we need to generate a schedule of tasks based on execution allocation of tasks for a time slice. We need to consider non-parallel execution constraints that no task executes in more than one processor. DP-Fair applies McNaughton's algorithm [42]. The algorithm consecutively fills resources of a processor with tasks one by one. If the processor cannot accept a task, the algorithm splits the target task by the remaining capacity of the processor and the remaining execution amount of the task. Then, the algorithm allocates the first part of the task into the processor and the second part of the task to a new processor. This ensures that each task executes on at most one processor at a time.

The following lemmas recapitulate the schedulability properties of DP-Fair.

Lemma 2.2 (from [38])**.** *Given a non-MC task set τ and a time slice, if the task set is scheduled within the time slice under DP-Fair and $\sum_{i \in \tau} \delta_i \leq m$, then the required execution amount of each task within the time slice can be executed until the end of the time slice.*

Lemma 2.3 (from [38]). *A non-MC task set τ is schedulable under DP-Fair iff $\sum_{\tau_i \in \tau} \delta_i \leq m$.*

2.4.3. EDF-VD Scheduling Algorithm

We recapitulate EDF-VD [4] for the implicit-deadline task model, whose algorithm and analysis are simple while being speedup-optimal. Due to its simplicity, EDF-VD is extended into various directions (e.g., constrained-deadline task model [25, 24], multiprocessor platform [11, 6, 37], and imprecise computation model [41]).

EDF-VD considers a system-level mode switch that when a single HI-task switches to HI-mode, all the other HI-tasks switch to HI-mode simultaneously. Upon such an event, it changes the system mode from LO-mode to HI-mode and drops all the LO-tasks. Capturing the characteristics of MC tasks that HI-tasks are subject to with different WCET requirements in different modes, EDF-VD assigns different priorities to a HI-task in different modes (*virtual deadline* (VD) in the LO mode and real deadline in HI-mode).

We now explain how EDF-VD assigns VDs. For a HI-task τ_i, the VD of the task (V_i) is assigned by $V_i = xT_i$ where x is the VD coefficient[2] ($x \in \mathbb{R}$ s.t. $0 < x \leq 1$) with $x = U_H^L/(1 - U_L^L)$.

The schedulability analysis of EDF-VD consists of the following lemmas. We will reuse Lemma 2.4 for our new algorithm.

Lemma 2.4 (from [4]). *A task set τ is schedulable by EDF-VD when all HI-tasks are in LO-mode if*

$$U_L^L + \frac{U_H^L}{x} \leq 1. \tag{2.1}$$

Lemma 2.5 (from [4]). *A task set τ is schedulable by EDF-VD when any HI-tasks are in*

[2]The computation of VD coefficient is derived from Eq. (2.1) of Lemma 2.4.

HI-mode if

$$xU_L^L + U_H^H \leq 1. \tag{2.2}$$

By Lemmas 2.4 and 2.5, a given task τ is MC-schedulable by EDF-VD if Eqs. (2.1) and (2.2) hold.

CHAPTER 3 : Fluid-based Mixed-Criticality Scheduling

In this chapter, we focus on global multiprocessor scheduling approaches to minimize the number of the required processors for a given MC system.

3.1. The Overview of the MC-Fluid Scheduling Framework

While MC scheduling has been extensively studied for the uniprocessor case, the multiprocessor case has received little attention. In non-MC multiprocessor scheduling, many optimal scheduling algorithms [9, 20, 38] are based on the fluid scheduling model, where each task executes in proportion to a static rate (i.e., task utilization). While its proportional progress is still applicable on MC systems, a single static rate is inefficient because characteristics of MC systems change over time. For example, consider a task τ_i s.t. $u_i^L < u_i^H$. If the task executes with a rate of u_i^L, then it misses deadlines after mode-switch. If the task executes with a rate of u_i^H (the worst-case reservation approach), the task wastes its resources when there is no mode-switch. Using criticality-dependent execution rates, we can find an efficient fluid scheduling algorithm for MC systems.

We propose a new fluid scheduling algorithm for MC systems, called *MC-Fluid*, where each task executes with its pre-assigned rate depending on the system-level criticality-mode (the task initially executes with its low-criticality rate. As the criticality-mode changes (from LO-mode to HI-mode) at runtime, the task executes with its high-criticality rate). A central challenge that we address in this chapter is how to determine criticality-dependent execution rates of all the tasks, given that the time instance when the system criticality-level changes is unknown. Our goal is to optimally allocate criticality-dependent execution rates to each task within the problem domain.

There have been different rate assignment strategies for MC-Fluid. In our conference version [37], we presented the OERA (Optimal Execution Rate Assignment) algorithm to determine the optimal execution rates in polynomial time. Recently, Baruah et al. [6] presented

Table 2: Execution rate assignment algorithms in this chapter

Algorithm	**Approach**	**Optimality under dual-rate scheduling**	**Complexity**
OERA (Ch. 3.5.1)	Convex optimization	optimal	$O(n \log n)$
MC-Derivative (Ch. 3.5.2)	Derivative	optimal	$O(n \log n)$
MCF [6]	Approximation	suboptimal	$O(n)$

the *MCF* rate assignment algorithm with linear complexity, which is a simplified version of OERA, and showed that MCF has a speedup factor of 4/3 meaning that MCF can schedule any feasible MC task set if each processor is sped up by a factor of 4/3 (refer Performance Metrics in Chapter 2.1). Since any MC scheduling algorithm cannot have a lower speedup factor than 4/3 [4], MCF is optimal in terms of the speedup factor. Since the OERA algorithm is optimal in terms of schedulability (i.e., all task sets schedulable by MCF are also schedulable by OERA), the speedup result of MCF is also applicable to OERA: the OERA algorithm has an optimal speedup factor.

In this chapter, we present two rate assignment algorithms. First, the OERA algorithm computes the rate assignment by convex optimization. Compared to the conference version [37], we reduce the complexity of the algorithm to $O(n \log n)$. Second, the MC-Derivative algorithm computes the rate assignment by the first derivative principles. We show that both algorithms assigns execution rates optimally under the MC-Fluid framework (with LO- and HI-rates). Since the speedup factor of the suboptimal MCF is optimal, the speedup factor of both OERA and MC-Derivative is optimal. Table 2 summarizes the performance of our rate assignment algorithms with respect to complexity and optimality

Contributions. Our contributions in this chapter are summarized as follows:

- We present a fluid model-based multiprocessor MC scheduling algorithm, called MC-Fluid, with criticality-dependent execution rates per task (Chapter 3.2) and analyze its exact schedulability (Chapter 3.3).
- We propose three different rate assignment algorithms (Chapter 3.5). OERA and MC-

Derivative algorithms optimally assigns execution rates with $O(n \log n)$ complexity. OERA algorithm is based on the convex optimization framework (Chapter 3.5.1) while MC-Derivative algorithm is based on the first derivative principles (Chapter 3.5.2).

- We derive the speedup factor of MC-Fluid, which is 4/3, optimal in multiprocessor MC scheduling (Chapter 3.6).

3.2. The MC-Fluid Scheduling Algorithm

The fluid scheduling algorithm with a single static execution rate per task is inefficient in resource utilization on MC systems. We can 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 characteristics of MC tasks can change substantially at mode-switch. According to 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 (the execution requirement for a HI-task is changed from C_i^L to C_i^H).

For example, consider a HI-task τ_i with $T_i = 5$, $C_i^L = 2$, and $C_i^H = 3$. When a task executes with a rate of $u_i^H (= 0.6)$, it wastes 0.2 utilization if there is no mode-switch. 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. When the task executes with a rate of 0.5, it completes to execute its LO-WCET at relative time 4 and it can still execute one more time unit until relative time 5, which satisfies the HI-WCET requirement. Then, we can reduce 0.1 utilization compared to the previous case. Then, we wonder whether the task is schedulable with a rate smaller than 0.5. Assume that the task executes with a rate of $0.5 - \epsilon$ for any small real number $\epsilon > 0$. While the task completes to execute its LO-WCET at relative time $2/(0.5 - \epsilon)$, it cannot

execute anymore time until relative time 5:

$$\frac{C_i^H - C_i^L}{T_i - \text{The LO-WCET complete time}} = \frac{1}{5 - 2/(0.5 - \epsilon)} > \frac{1}{5 - 2/0.5} = 1.$$

Therefore, the task cannot meet the MC requirement with a rate smaller than 0.5.

We propose a fluid scheduling algorithm, called *MC-Fluid*, 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 3.1 (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 all LO-tasks $\forall \tau_i \in \tau_L$.*

The *execution amount* of a job indicates the processor resources consumed by the job within a time interval, based on Def. 3.1.

Definition 3.2. ***Execution amounts*** *of a job of a task $\tau_i \in \tau$ 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: $E_i^L(t) \stackrel{\text{def}}{=} \theta_i^L \cdot t$ and $E_i^H(t) \stackrel{\text{def}}{=} \theta_i^H \cdot t$.*

3.3. 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 LO-mode, we need task-schedulability (Eq. (3.1)) and platform feasibility (Eq. (3.3)). Similarly, in HI-mode, we need Eq. (3.2) and Eq. (3.4).

Theorem 3.1 (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

$$\forall \tau_i \in \tau,\ \theta_i^L \geq u_i^L, \tag{3.1}$$

$$\forall \tau_i \in \tau_H,\ \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} \leq 1, \tag{3.2}$$

$$\sum_{\tau_i \in \tau} \theta_i^L \leq m, \tag{3.3}$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m. \tag{3.4}$$

To prove Theorem 3.1, we need to derive task-schedulability and platform feasibility in each mode.

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

Lemma 3.1. *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 $\theta_i^L \geq u_i^L$,

$$\theta_i^L \cdot T_i \geq u_i^L \cdot T_i \ \Rightarrow\ E_i^L(T_i) \geq C_i^L. \qquad \text{(by Def. 3.2)}$$

($\Rightarrow$) 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. 2. We define a *carry-over job* as a job released in LO-mode and finished in HI-mode as shown in Fig. 2. 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

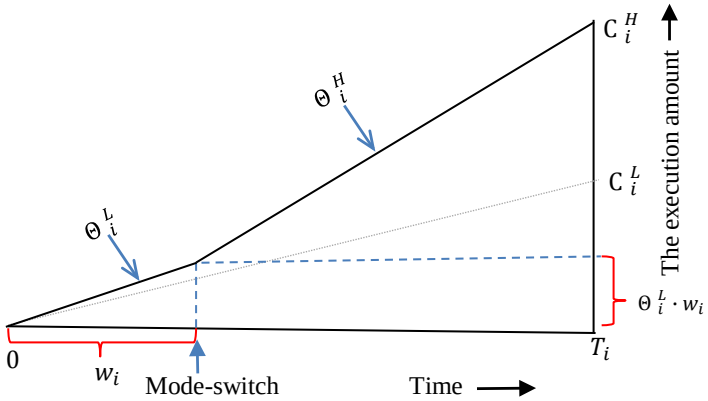


Figure 2: The model of a carry-over job of a task $\tau_i \in \tau_H$ where mode-switch happens at w_i and the job is executed with θ_i^L and θ_i^H in LO- and HI-mode, respectively (the execution amount of the job until its deadline (T_i) should be C_i^H to meet its deadline).

LO-mode and finished in LO-mode, 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). The second case is a special case of the carry-over job when $w_i = 0$, which means that the job is released at mode-switch.

To derive task-schedulability for a carry-over job of τ_i, we need to know the relative time of mode-switch (w_i). We first derive task-schedulability condition with a given w_i. Since mode-switch triggers in the middle of its execution, the job is executed with θ_i^L before mode-switch and with θ_i^H after 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. 3.2. 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):

$$\theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H. \tag{3.5}$$

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. (3.5) for any w_i in $[0, T_i]$:

$$\forall w_i,\ \theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H. \tag{3.6}$$

To sum up, task-schedulability in HI-mode is Eq. (3.6).

Now, we will derive task-schedulability in HI-mode (Eq. (3.6)) 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 3.2 considers the first case ($\theta_i^L > \theta_i^H$).

Lemma 3.2. *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. (3.6), we have

$$\forall w_i,\ \theta_i^H \cdot w_i + \theta_i^H \cdot (T_i - w_i) = \theta_i^H \cdot T_i \geq C_i^H. \tag{3.7}$$

Let $\theta_i^{L'}$ be the changed value of θ_i^L where $\theta_i^{L'} > \theta_i^H$. To show that the task can meet its deadline in HI-mode with $\theta_i^{L'}$, we need to show that Eq. (3.6) holds: $\forall w_i$,

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

which is greater than or equal to C_i^H by Eq. (3.7).

($\Rightarrow$) 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. (3.6) 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. (3.6) holds, which is a contradiction. Thus, we proved the claim ($\theta_i^H \geq u_i^H$).

Let $\theta_i^{L'}$ be the changed value of θ_i^L where $\theta_i^{L'} = \theta_i^H$. Then, it is required to show that Eq. (3.6) 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 chapter.

Corollary 3.1. *For the 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 3.2. Thus, its task-schedulability in HI-mode is equivalent to the one when $\theta_i^L \leq \theta_i^H$. □

The following lemma presents task-schedulability in HI-mode by using the assumption that

task-schedulability in LO-mode holds. It is a valid assumption because we eventually consider task-schedulability in both LO- and HI-mode for MC-schedulability.

Lemma 3.3 (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*

$$\frac{u_i^L}{\theta_i^L} + \frac{(u_i^H - u_i^L)}{\theta_i^H} \leq 1. \tag{3.8}$$

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)[1]. Thus, its execution amount from its release time to mode-switch also cannot exceed C_i^L:

$$E_i^L(w_i) \leq C_i^L \;\Rightarrow\; \theta_i^L \cdot w_i \leq C_i^L. \tag{3.9}$$

Combining $0 \leq w_i \leq T_i$ and Eq. (3.9), we have $0 \leq w_i \leq \min(C_i^L/\theta_i^L, T_i)$. Since task-schedulability in LO-mode holds, we have $\theta_i^L \geq u_i^L$ by Lemma 3.1. Then,

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

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

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

We know that task-schedulability in HI-mode is Eq. (3.6), which is rewritten to:

$$
\begin{aligned}
& \forall w_i,\ \theta_i^L \cdot w_i + \theta_i^H \cdot (T_i - w_i) \geq C_i^H \\
\Leftrightarrow\ & \forall w_i,\ \theta_i^H \cdot T_i \geq (\theta_i^H - \theta_i^L) \cdot w_i + C_i^H \\
\Leftrightarrow\ & \theta_i^H \cdot T_i \geq (\theta_i^H - \theta_i^L) \cdot C_i^L / \theta_i^L + C_i^H \qquad (\because \theta_i^H - \theta_i^L \geq 0)^2 \\
\Leftrightarrow\ & \theta_i^H \cdot T_i \geq C_i^L \cdot \theta_i^H / \theta_i^L + C_i^H - C_i^L \\
\Leftrightarrow\ & 1 \geq u_i^L / \theta_i^L + (u_i^H - u_i^L) / \theta_i^H. \qquad \text{(dividing by } \theta_i^H \cdot T_i)
\end{aligned}
$$

Thus, the task can meet its deadline in HI-mode iff Eq. (3.6) holds iff Eq. (3.8) holds. □

MC-schedulability. Since task schedulability is derived in Lemmas 3.1 and 3.3, and platform feasibility condition is straight-forward from Lemma 2.1, we can prove Theorem 3.1.

Proof of Theorem 3.1. (⇐) We need to show that the task set satisfies both task-schedulability and platform feasibility in both LO- and HI-mode. We show that task-schedulability holds in both LO- and HI-mode. In LO-mode, it holds by Lemma 3.1 with Eq. (3.1). In HI-mode, it holds by Lemma 3.3 with Eq. (3.2) and task-schedulability in LO-mode.

We show that platform feasibility holds in both LO- and HI-mode. In LO-mode, it holds by Lemma 2.1 with Eq. (3.3). In HI-mode, it holds by Lemma 2.1 with Eq. (3.4).

(⇒) We will prove the contrapositive: if any of the conditions does not hold, then the task set is not MC-schedulable. If Eq. (3.1) and Eq. (3.2) do not hold, task-schedulability in LO-mode and in HI-mode do not hold by Lemma 3.1 and Lemma 3.3, respectively. If Eq. (3.3) and Eq. (3.4) do not hold, platform feasibility in LO-mode and in HI-mode do not hold by Lemma 2.1, respectively. □

We apply Theorem 3.1 in the following example.

Example 3.1. *Consider a two-processor system where its task set τ and its execution rate*

[2] We already assumed that $\theta_i^L \leq \theta_i^H$ by Corollary 3.1.

Table 3: An 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	

assignment is given as shown in Table 3. To show that it is MC-schedulable, we need to show that Eqs. (3.1)*,* (3.2)*,* (3.3) *and* (3.4) *hold by Theorem 3.1. We can easily check that Eq.* (3.1) *holds. We show that Eq.* (3.2) *holds:* $\{0.2/0.571 + (0.85 - 0.2)/1\} \leq 1$ *for* τ_1*,* $\{0.25/0.472 + (0.5 - 0.25)/0.531\} \leq 1$ *for* τ_2*,* $\{0.15/0.283 + (0.3 - 0.15)/0.319\} \leq 1$ *for* τ_3*, and* $\{0.1/0.15 + (0.15 - 0.1)/0.15\} \leq 1$ *for* τ_4*. We can check that* $\sum_{\tau_i \in \tau} \theta_i^L = \{0.571 + 0.472 + 0.283 + 0.15 + 0.2\} \leq 2$ *for Eq.* (3.3) *and* $\sum_{\tau_i \in \tau_H} \theta_i^H = \{1 + 0.531 + 0.319 + 0.15\} \leq 2$ *for Eq.* (3.4)*.*

3.4. The Execution Rate Assignment

In the previous section, we looked at the schedulability analysis of MC-Fluid for a given task set and a given execution rate assignment. In this section, we find rate assignment for a given task set, based on the MC-Fluid schedulability analysis. We first define the *optimality* of a rate assignment algorithm, in a similar way to Davis et al.[21].

Definition 3.3. *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.*

We will construct an efficient and optimal assignment algorithm. While a naive assignment algorithm checking all combinations of execution rates is optimal, its complexity is intractable because possible real-number execution rates are infinite. We formulate the execution rate assignment problem as an optimization problem.

To find an optimal rate assignment algorithm, we consider an optimal assignment of θ_i^L and θ_i^H. The following lemma presents an optimal assignment of θ_i^L.

Lemma 3.4 (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. (3.1), (3.2), (3.3), and (3.4) by Theorem 3.1.

(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. (3.1) and (3.3).

Let θ_i^{L*} be the value of θ_i^L in $\mathbb{A}$. Since $\mathbb{A}$ is schedulable, Eq. (3.1) 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. (3.1) still holds because $\theta_i^L \geq u_i^L$. Eq. (3.3) still holds because the decreased θ_i^L (from θ_i^{L*} to u_i^L) does not increase the sum of the 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. (3.1), (3.2), and (3.3).

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. (3.2) holds for θ_i^{L*} and θ_i^{H*}, we have

$$\frac{u_i^H - u_i^L}{\theta_i^{H*}} \leq 1 - \frac{u_i^L}{\theta_i^{L*}} \;\Rightarrow\; \frac{u_i^H - u_i^L}{\theta_i^{H*}} < 1, \tag{3.10}$$

since $u_i^L / \theta_i^{L*} > 0$. Eq. (3.2) with θ_i^{L*} and θ_i^{H*} is rewritten to:

$$\begin{aligned}
&\frac{u_i^L}{\theta_i^{L*}} \leq \frac{\theta_i^{H*} - u_i^H + u_i^L}{\theta_i^{H*}} \\
\Leftrightarrow\; & u_i^L \cdot \theta_i^{H*} \leq \theta_i^{L*}(\theta_i^{H*} - u_i^H + u_i^L) \quad \text{(multiplying by } \theta_i^{H*} \cdot \theta_i^{L*}\text{)} \\
\Leftrightarrow\; & \frac{u_i^L \cdot \theta_i^{H*}}{\theta_i^{H*} - u_i^H + u_i^L} \leq \theta_i^{L*}. \qquad (\because \theta_i^{H*} > u_i^H - u_i^L \text{ by Eq. (3.10)})
\end{aligned}$$

Since $\mathbb{A}$ is schedulable, Eqs. (3.1), (3.2), and (3.3) 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. (3.1) still holds because $\theta_i^L \geq u_i^L$. Eq. (3.2) still holds because the value of the reassigned θ_i^L is the minimum value satisfying Eq. (3.2). Eq. (3.3) still holds because the decreased θ_i^L does not increase the sum of the execution rates. □

We define the optimal assignment of θ_i^L mentioned in Lemma 3.4 as $\mathsf{computeLoRate_i}(\theta_i^H)$: for each task τ_i,

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

To find an optimal assignment of θ_i^H, we first consider feasibility. Based on Lemma 3.4 and Theorem 3.1, the following theorem presents a feasibility condition.

Theorem 3.2 (Feasibility)**.** *A task set τ is feasible iff there exists an assignment of θ_i^H for $\forall \tau_i \in \tau_H$ satisfying*

$$u_i^H \leq \theta_i^H \leq 1, \tag{3.11}$$

$$U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L (u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} \leq m, \tag{3.12}$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m. \tag{3.13}$$

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. (3.11), (3.12), and (3.13).

We first show that $\theta_i^L \leq 1$ for each $\tau_i \in \tau$ and $\theta_i^H \leq 1$ for each $\tau_i \in \tau_H$ according to the definition of execution rates (Def. 2.3). For $\tau_i \in \tau_L$, we have $\theta_i^L \leq 1$ by the selection of θ_i^L. For $\tau_i \in \tau_H$, we have $\theta_i^H \leq 1$ from Eq. (3.11). Since we assumed $\theta_i^L \leq \theta_i^H$ by Corollary 3.1,

we have $\theta_i^L \leq 1$ from Eq. (3.11).

To show that this assignment is schedulable by Theorem 3.1, we need to show that it satisfies Eqs. (3.1), (3.2), (3.3), and (3.4). We know that θ_i^L for $\forall \tau_i \in \tau$ satisfies Eq. (3.1) and θ_i^L for $\forall \tau_i \in \tau_H$ satisfies Eq. (3.2). We can rewrite Eq. (3.3) to:

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

which is Eq. (3.12). Since Eq. (3.3) is equivalent to Eq. (3.12), the equation holds. Eq. (3.4) holds from Eq. (3.13).

($\Rightarrow$) 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 3.4. We need to show that θ_i^H for $\forall \tau_i \in \tau_H$ in the assignment satisfies Eqs. (3.11), (3.12), and (3.13).

Consider θ_i^H of a task $\tau_i \in \tau_H$. We know $\theta_i^H \leq 1$ from the definition of execution rates (Def. 2.3). Since we assumed that $\theta_i^L \leq \theta_i^H$ by Corollary 3.1, we rewrite $\theta_i^L \leq \theta_i^H$ to:

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

which is $u_i^H \leq \theta_i^H$. Then, Eq. (3.11) holds.

Since the assignment is schedulable, Eqs. (3.3) and (3.4) hold by Theorem 3.1. We already proved that Eq. (3.12) holds from Eq. (3.3) in Case ($\Leftarrow$). Eq. (3.13) holds from Eq. (3.4). $\square$

From feasibility condition, we know that the assignment of θ_i^H satisfying the conditions in Theorem 3.2 is optimal. However, it is not easy to find such an assignment. We will find an optimal assignment algorithm by solving an equivalent optimization problem based on

Theorem 3.2.

Definition 3.4 (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:*

$$\begin{aligned}
&\textit{minimize} \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L} \\
&\textit{subject to} \sum_{\tau_i \in \tau_H} X_i - m + U_H^H \leq 0, && (\mathsf{CON}1) \\
&\forall \tau_i \in \tau_H, \ -X_i \leq 0, && (\mathsf{CON}2) \\
&\forall \tau_i \in \tau_H, \ X_i - 1 + u_i^H \leq 0. && (\mathsf{CON}3)
\end{aligned}$$

The following lemma shows that the assignment of θ_i^H by solving the optimization problem (Def. 3.4) is optimal.

Lemma 3.5. *If a rate assignment constructs a solution for the problem in Def. 3.4, the rate assignment is optimal (according to Def. 3.3).*

Proof. We claim that if the assignment by Def. 3.4 cannot satisfy Eqs. (3.11), (3.12), and (3.13), no assignment can satisfy those equations. Then, τ is not feasible by Theorem 3.2.

Suppose that we have a solution to the optimization problem in Def. 3.4. For each task $\tau_i \in \tau_H$, 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^* for $\forall \tau_i \in \tau_H$. By Def. 3.4, we have an assignment where $\theta_i^H := u_i^H + X_i^*$ to for $\forall \tau_i \in \tau_H$. Then, Eq. (3.11) holds from $\mathsf{CON}2$ and $\mathsf{CON}3$ and Eq. (3.13) holds from $\mathsf{CON}1$.

Suppose that Eq. (3.12) does not hold with the assignment, meaning $\mathsf{OBJ}^* > m - U_L^L - U_H^L$. For any set of X_i satisfying $\mathsf{CON}1$, $\mathsf{CON}2$, and $\mathsf{CON}3$, we have $\sum_{\tau_i \in \tau_H} u_i^L(u_i^H - u_i^L)/(X_i + u_i^L) \geq \mathsf{OBJ}^*$ by the definition of the optimization problem and thus Eq. (3.12) does not

hold,

$$\sum_{\tau_i \in \tau_H} \frac{u_i^L (u_i^H - u_i^L)}{X_i + u_i^L} \geq \mathsf{OBJ}^* > m - U_L^L - U_H^L.$$

Thus, no assignment can satisfy Eqs. (3.11), (3.12), and (3.13). □

3.5. Rate Assignment Algorithms

In Chapter 3.5.1, we formulated the optimization problem to construct an optimal assignment algorithm. In this section, we present two different execution rate assignment algorithms (namely OERA and MC-Derivative) to solve the optimization problem. First, OERA solves the optimization problem using convex optimization. Second, MC-Derivative computes the execution rates using the first derivative principles. The structure of those execution rate assignment algorithms is as follows:

- Compute θ_i^H for all $\tau_i \in \tau_H$ using the respective HI-execution rate assignment algorithms,
- **If** $\sum_{\tau_i \in \tau_H} \theta_i^H \leq m$ compute θ_i^L (by Lemma 3.4) **else** declare failure;

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

- **If** $\sum_{\tau_i \in \tau} \theta_i^L \leq m$ declare success **else** declare failure.

For brevity of the following sections, we abbreviate $\forall \tau_i \in \tau_H$ as $\forall \tau_i$ because only HI-tasks are considered in Def. 3.4.

3.5.1. The OERA Algorithm

Any optimization problem can be rewritten in its dual form using *Lagrange multiplier* (Chapter 5, Boyd et al. [17]), a technique to find a solution for a constrained optimization problem. Using Lagrangian multipliers, we can transform any optimization problem with constraints into its dual problem without constraints. In particular, when its objective function and constraints are differentiable and convex, we can apply *Karush-Kuhn-Tucker* (KKT) conditions (Chapter 5.5.3, [17]), which is a set of optimality conditions for an optimization problem with constraints.

Lemma 3.6 (KKT conditions[17])**.** *Let $\overline{x}$ be a vector of x_i for $i = 1, \cdots, n$ where $n = |\overline{x}|$ and x_i^* be the value of x_i. Consider an optimization problem:*

$$\textit{minimize } f(\overline{x}) \textit{ subject to } g_j(\overline{x}) \leq 0 \textit{ for } j = 1, \cdots, N$$

where N is the number of $g_j(\overline{x})$ and all of $f(\overline{x})$ and $g_j(\overline{x})$ for $\forall j$ are differentiable and convex. Then, $\overline{x^*}$ minimizes $f(\overline{x})$ iff there exists $\overline{\lambda^*}$ s.t.

$$\forall j,\ g_j(\overline{x^*}) \leq 0, \tag{3.14}$$

$$\forall j,\ \lambda_j^* \cdot g_j(\overline{x^*}) = 0, \tag{3.15}$$

$$\forall i,\ \frac{\partial f(\overline{x^*})}{\partial x_i} + \sum_j (\lambda_j^* \frac{\partial g_j(\overline{x^*})}{\partial x_i}) = 0, \tag{3.16}$$

$$\forall j,\ \lambda_j^* \geq 0, \tag{3.17}$$

where λ_j is a Lagrange multiplier, λ_j^* is the value of λ_j, and $\overline{\lambda}$ is a vector of λ_j.

Lemma 3.7 applies KKT conditions (Lemma 3.6) to the optimization problem in Def. 3.4 because the objective function and all the constraints are differentiable and convex. Then, we only need to find the value of Lagrange multipliers satisfying KKT conditions for the optimal solution. We use ψ, λ_i, and ν_i as Lagrange multipliers for CON1, CON2, and CON3, respectively. We denote a vector of X_i, λ_i, and ν_i by $\overline{X}$, $\overline{\lambda}$, and $\overline{\nu}$, and denote the value of

X_i, λ_i, and ν_i by X_i^*, λ_i^*, and ν_i^*, respectively. We define, for each task τ_i,

$$\mathsf{Cost}_i(x) \stackrel{\mathsf{def}}{=} \frac{u_i^L(u_i^H - u_i^L)}{(x + u_i^L)^2} \quad \text{where } x \in \mathbb{R}.$$

Lemma 3.7. *Consider the optimization problem in Def. 3.4. $\overline{X^*}$ minimizes $f(\overline{X})$ iff there exist ψ^*, $\overline{\lambda^*}$, and $\overline{\nu^*}$ s.t.*

$$\sum_{\tau_i} X_i^* - m + U_H^H \leq 0, \tag{3.18}$$

$$\forall \tau_i, \ -X_i^* \leq 0, \qquad X_i^* - 1 + u_i^H \leq 0, \tag{3.19}$$

$$\psi^*(\sum_{\tau_i} X_i^* - m + U_H^H) = 0, \tag{3.20}$$

$$\forall \tau_i, \ \lambda_i^*(-X_i^*) = 0, \qquad \nu_i^*(X_i^* - 1 + u_i^H) = 0, \tag{3.21}$$

$$\forall \tau_i, \ -\mathsf{Cost}_i(X_i^*) + \psi^* - \lambda_i^* + \nu_i^* = 0, \tag{3.22}$$

$$\psi^* \geq 0 \wedge \forall \tau_i(\lambda_i^* \geq 0 \wedge \nu_i^* \geq 0), \tag{3.23}$$

where $f(\overline{X}) = \sum_{\tau_i} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L}$.

Proof. We will derive KKT conditions for the optimization problem in Def. 3.4: $f(\overline{X}) = \sum_{\tau_i} \frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L}$ and $g_1(\overline{X}) = \sum_{\tau_i} X_i - m + U_H^H$ for $\mathsf{CON1}$; for $\forall \tau_i$, $g_{2,i}(\overline{X}) = -X_i$ for $\mathsf{CON2}$ and $g_{3,i}(\overline{X}) = X_i - 1 + u_i^H$ for $\mathsf{CON3}$.

We know that $f(\overline{X})$ and all constraints are differentiable. To show that they are convex, we need to show that their second derivatives are no smaller than zero:

$$\forall \tau_i, \ \forall X_i \ , \frac{\partial^2 f(\overline{X})}{\partial (X_i)^2} = \frac{2 \cdot u_i^L(u_i^H - u_i^L)}{(X_i + u_i^L)^3} \geq 0 \text{ and}$$
$$\forall \tau_i, \ \forall \tau_j, \ \forall X_i, \ \frac{\partial^2 g_1(\overline{X})}{\partial (X_i)^2} = \frac{\partial^2 g_{2,j}(\overline{X})}{\partial (X_i)^2} = \frac{\partial^2 g_{3,j}(\overline{X})}{\partial (X_i)^2} = 0.$$

We derive KKT conditions for the problem by Lemma 3.6:

- We denote Lagrange multipliers for $g_1(\overline{X})$, $g_{2,i}(\overline{X})$ and $g_{3,i}(\overline{X})$ by ψ, λ_i and ν_i, respectively.
- Eq. (3.14) for $g_1(\overline{X})$ is Eq. (3.18). Eq. (3.14) for $g_{2,i}(\overline{X})$ and $g_{3,i}(\overline{X})$ is Eq. (3.19).
- Eq. (3.15) for $g_1(\overline{X})$ is Eq. (3.20). Eq. (3.15) for $g_{2,i}(\overline{X})$ and $g_{3,i}(\overline{X})$ is Eq. (3.21).
- Eq. (3.16) is $\forall \tau_i,\ \partial f(\overline{X})/\partial X_i + \psi - \lambda_i + \nu_i = 0$, which is Eq. (3.22).
- Eq. (3.17) for $g_1(\overline{X})$, $g_{2,i}(\overline{X})$ and $g_{3,i}(\overline{X})$ is Eq. (3.23).

By Lemma 3.6, $\overline{X^*}$ minimizes $f(\overline{X})$ iff there exist ψ^*, $\overline{\lambda^*}$, and $\overline{\nu^*}$ satisfying Eqs. (3.18), (3.19), (3.20), (3.21), (3.22), and (3.23). □

Example 3.2. *Consider the system in Example 3.1. We need to solve the optimization problem in Def. 3.4. Suppose that we have $\overline{X^*}$ satisfying KKT conditions in Lemma 3.7 for the system: $X_1^* = 0.2$, $X_2^* = 0.2$, and $X_3^* = 0$. Eq.* (3.18) *holds:* $0.2 + 0.2 - 2 + 1.6 = 0$. *We can easily check that Eqs.* (3.19) *and* (3.20) *hold. To satisfy Eq.* (3.21)*, we have $\lambda_1^* = \lambda_2^* = \nu_2^* = \nu_3^* = 0$. By Eq.* (3.22)*, we have $-\frac{0.15}{(0.2+0.3)^2} + \psi^* + \nu_1^* = 0$, $-\frac{0.12}{(0.2+0.4)^2} + \psi^* = 0$, and $\psi^* - \lambda_3^* = 0$. Then, we have $\psi^* = 1/3$, $\nu_1^* = 4/15$, and $\lambda_3^* = 1/3$. Eq.* (3.23) *holds with ψ^*, $\overline{\lambda^*}$, and $\overline{\nu^*}$. Thus, $\overline{X^*}$ is the solution to the problem in Def. 3.4 by Lemma 3.7.*

By Def. 3.4, we have $\theta_1^H := 1$, $\theta_2^H := 0.9$, and $\theta_3^H := 0.1$. We can assign θ_i^L for $\forall \tau_i \in \tau$ by Lemma 3.4. Since this calculated assignment is the same as the assignment in Example 3.1, we conclude that the assignment in Example 3.1 is optimal.

The OERA algorithm. General KKT conditions are not easily solvable because we cannot find the feasible values of the Lagrange multipliers for the conditions. We propose the *Optimal Execution Rate Assignment (OERA) algorithm* to solve our KKT conditions. The intuition is that we first independently analyze λ_i^*, ν_i^*, and X_i^* for each task τ_i for a given ψ^* and later find ψ^* satisfying our KKT conditions. In our KKT conditions (Lemma 3.7), we know that λ_i^* and ν_i^* for each task τ_i depend only on X_i^*. Since only ψ^* depends on $\overline{X^*}$, we can find ψ^* with each independently analyzed X_i^*. Based on this observation, we

develop a simple greedy algorithm.

Before presenting the OERA algorithm, the following lemma shows that we can independently analyze λ_i^*, ν_i^*, and X_i^* for each task τ_i.

Lemma 3.8. *Given a task τ_i and ψ (≥ 0), if we assign $X_i :=$*

$$\mathsf{Cal\_X_i}(\psi) \stackrel{\mathsf{def}}{=} \begin{cases} 0 & \text{if } \psi \geq \mathsf{Cost}_i(0), \\ 1 - u_i^H & \text{if } \psi < \mathsf{Cost}_i(1-u_i^H), \\ \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{\psi}} - u_i^L & \text{otherwise}, \end{cases}$$

then Eqs. (3.19)*,* (3.21)*,* (3.22)*, and* (3.23) *hold with some λ_i (≥ 0) and ν_i (≥ 0).*

Proof. To satisfy Eq. (3.19), we assume that $0 \leq X_i \leq 1 - u_i^H$. Then, to prove this lemma, we only need to show that Eqs. (3.21) and (3.22) hold in each case of $\mathsf{Cal\_X_i}(\psi)$.

Case ($\psi \geq \mathsf{Cost}_i(0)$)**.** For some $\nu_i \geq 0$, Eq. (3.22) is simplified to:

$$\psi \leq \mathsf{Cost}_i(X_i) + \lambda_i, \tag{3.24}$$

by removing ν_i. Note that $\mathsf{Cost}(X_i) \leq \mathsf{Cost}(0)$ where $0 \leq X_i \leq 1 - u_i^H$. To satisfy Eq. (3.24), λ_i should satisfy that $\lambda_i \geq \psi - \mathsf{Cost}_i(X_i) \geq \psi - \mathsf{Cost}_i(0)$, which is greater than or equal to 0 from the assumption. To satisfy Eq. (3.21), we assign $\nu_i := 0$ and $X_i := 0$. Thus, Eqs. (3.21), (3.22), and (3.23) hold.

Case ($\psi < \mathsf{Cost}_i(1-u_i^H)$)**.** For some $\lambda_i \geq 0$, Eq. (3.22) is simplified to:

$$\psi + \nu_i \geq \mathsf{Cost}_i(X_i), \tag{3.25}$$

by removing λ_i. Note that $\mathsf{Cost}(1 - u_i^H) \leq \mathsf{Cost}(X_i)$ where $0 \leq X_i \leq 1 - u_i^H$. To satisfy Eq. (3.25), ν_i should satisfy that $\nu_i \geq \mathsf{Cost}_i(X_i) - \psi \geq \mathsf{Cost}_i(1 - u_i^H) - \psi$, which is greater than 0 from the assumption. To satisfy Eq. (3.21), we assign $\lambda_i := 0$ and $X_i := 1 - u_i^H$.

Thus, Eqs. (3.21), (3.22), and (3.23) hold.

Case ($\mathsf{Cost}_i(1 - u_i^H) \leq \psi < \mathsf{Cost}_i(0)$)**.** To satisfy Eq. (3.21), we assign $\lambda_i := 0$ and $\nu_i := 0$. Then, to satisfy Eq. (3.22), we need to satisfy $\psi = \mathsf{Cost}_i(X_i)$, from which, X_i can be computed:

$$\psi = \frac{u_i^L(u_i^H - u_i^L)}{(X_i + u_i^L)^2} \;\Leftrightarrow\; X_i = \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{\psi}} - u_i^L.$$

Thus, Eqs. (3.21), (3.22), and (3.23) hold.

Combining three cases, we showed that Eqs. (3.19), (3.21), and (3.22) hold with some positive λ_i and ν_i. □

Algorithm 3.1 describes the OERA algorithm. If we know ψ satisfying Eqs. (3.18) and (3.20) of our KKT conditions, we satisfy the other equations by assigning $X_i := \mathsf{Cal\_X_i}(\psi^*)$ for $\forall\tau_i$. To find such a ψ, we will first consider $\psi > 0$ and later $\psi = 0$. Suppose that $\psi > 0$. Then, we only need to satisfy the condition that $\sum_{\tau_i} X_i - m + U_H^H = 0$, which is

$$\mathsf{Sum\_Cal\_X}(\psi^*) = m - U_H^H \tag{3.26}$$

where $\mathsf{Sum\_Cal\_X}(\psi^*) = \sum_{\tau_i} \mathsf{Cal\_X_i}(\psi^*)$.

The LHS of the required condition (Eq. (3.26)) is a piecewise linear function of $1/\sqrt{\psi}$. According to Lemma 3.8, if ψ is greater than or equal to $\mathsf{Cost}_i(0)$ or smaller than $\mathsf{Cost}_i(1 - u_i^H)$, $\mathsf{Cal\_X_i}(\psi)$ is a constant; otherwise, $\mathsf{Cal\_X_i}(\psi)$ is a linear function of $1/\sqrt{\psi}$. To find the range of each linear function in $\mathsf{Sum\_Cal\_X}(\psi^*)$, we define an ordered set of critical points:

$$G \stackrel{\mathsf{def}}{=} \{x \in \mathbb{R} | x = \mathsf{Cost}_i(0) \text{ or } x = \mathsf{Cost}_i(1 - u_i^H) \text{ for } \forall\tau_i\}$$

where each element is sorted in increasing order. We denote the j-th element of G as G_j, which is a critical point for some task τ_i that $\mathsf{Cal\_X_i}(\psi)$ changes a constant to a linear

function of $1/\sqrt{\psi}$ or vice-versa (a linear function to constant) at $\psi = G_j$.

We will utilize the property of critical points that the function is a linear function within two consecutive critical points G_j and $G_j + 1$. We apply a binary search to find G_j^* s.t. $\mathsf{Sum\_Cal\_X}(G_{j+1}) \leq m - U_H^H$ and $m - U_H^H < \mathsf{Sum\_Cal\_X}(G_j)$. If it is found, there exist ψ^* satisfying Eq. (3.26) between two consecutive critical points: $G_j^* < \psi^* \leq G_{j+1}^*$ (Line 1). Since the LHS of the condition is just a linear function within the range, we can find ψ^* by solving the condition (Line 2) and OERA returns $\overline{X^*}$ where $X_i^* = \mathsf{Cal\_X_i}(\psi^*)$ (Line 3).

It is possible that there does not exist G_j^* satisfying Eq. (3.26). Suppose that $\psi = 0$ (Line 4). OERA returns $\overline{X^*}$ where $X_i^* = \mathsf{Cal\_X_i}(0)$ for $\forall \tau_i$ (Line 5).

Algorithm 3.1 The OERA algorithm

Input: τ_H, G, and m
Output: $\overline{X^*}$ satisfying KKT conditions in Lemma 3.7

1: **if** binary search G_j s.t. $\mathsf{Sum\_Cal\_X}(G_{j+1}) \leq m - U_H^H$ and $m - U_H^H < \mathsf{Sum\_Cal\_X}(G_j)$ **then**
2: find ψ^* by solving Eq. (3.26) **return** $\overline{X^*}$ where $X_i = \mathsf{Cal\_X_i}(\psi^*)$
3: **elsereturn** $\overline{X^*}$ where $X_i = \mathsf{Cal\_X_i}(0)$
4: **end if**

Example 3.3. *Consider KKT conditions in Example 3.2. We have* $G = \{0.245, 0.6, 0.75, 1.667\}$. *We apply Algorithm 3.1. In Fig. 3, X-axis represents* $1/\sqrt{\psi}$. *Blue (solid) line is* $\mathsf{Sum\_Cal\_X}(\psi)$, *which is a piecewise linear function where its slope is changed at each* $1/\sqrt{G_j}$ *where* $G_j \in G$. *Red (dashed) line is a constant function of* $m - U_H^H$. *When* $G_1(= 0.245) \leq \psi^* < G_2(= 0.6)$, *the condition in Line 2 holds:* $\mathsf{Sum\_Cal\_X}(G_2) \leq m - U_H^H < \mathsf{Sum\_Cal\_X}(G_1)$, *which is* $0.5 \leq 0.4 < 0.247$. *In other words, a crossing point between the blue line and the red line is located in* $(1/\sqrt{G_2}, 1/\sqrt{G_1}]$. *Then, we can find* ψ^* *by solving Eq.* (3.26), *where the blue line meets the red line at* $1/\sqrt{\psi^*}$. *Since* $0.2 + \sqrt{0.12/\psi^*} - 0.4 = 0.4$, *we have* $1/\sqrt{\psi^*} = \sqrt{3}$. *Finally,* $\psi^* = 1/3$, $X_1^* = 0.2$, $X_2^* = \sqrt{\frac{0.12}{1/3}} - 0.4 = 0.2$, *and* $X_3^* = 0$. *We conclude that OERA can find* $\overline{X^*}$ *satisfying the KKT conditions in Lemma 3.7 for Example 3.2.*

The following theorem proves the correctness of OERA.

Theorem 3.3. *Algorithm 3.1 (OERA) can find* $\overline{X^*}$ *satisfying KKT conditions in Lemma 3.7.*

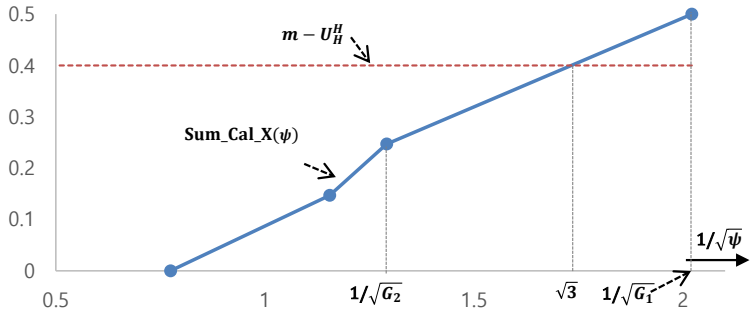


Figure 3: The plot of $\mathsf{Sum\_Cal\_X}(\psi)$ in respect to $1/\sqrt{\psi}$ on Example 3.3, which is a piecewise linear function (note that there is ψ^* s.t. $\mathsf{Sum\_Cal\_X_i}(\psi^*) = 0.4$ where $G_1 \leq \psi^* < G_2$).

Proof. If we know ψ^* satisfying Eqs. (3.18) and (3.20), we can assign $X_i := \mathsf{Cal\_X_i}(\psi^*)$ for each τ_i, which satisfies Eqs. (3.19), (3.21), (3.22), and (3.23) by Lemma 3.8. To find such a ψ^*, we will divide cases depending on whether $\mathsf{Sum\_Cal\_X}(0) \leq m - U_H^H$.

Case ($\mathsf{Sum\_Cal\_X}(0) \leq m - U_H^H$)**.** Then, we assign $\psi = 0$, which satisfies Eq. (3.20). From the assumption, Eqs. (3.18) holds.

Case ($\mathsf{Sum\_Cal\_X}(0) > m - U_H^H$)**.** To satisfy Eqs. (3.18) and (3.20), we only need to find ψ^* satisfying Eq. (3.26). Although the LHS of Eq. (3.26) is a piecewise linear function of $1/\sqrt{\psi}$, it is a linear function within two consecutive critical points G_j and $G_j + 1$. We apply a binary search to find G_j^* s.t. $\mathsf{Sum\_Cal\_X}(G_{j+1}) \leq m - U_H^H$ and $m - U_H^H < \mathsf{Sum\_Cal\_X}(G_j)$. If G_j^* is found, we can find a unique ψ^* because $\mathsf{Sum\_Cal\_X}(\psi)$ is a linear function within $[G_j^*, G_{j+1}^*)$.

We need to check $\mathsf{Sum\_Cal\_X}(\max_j G_j) \leq m - U_H^H$ and $\mathsf{Sum\_Cal\_X}(\min_j G_j) > m - U_H^H$. If it does not hold, G_j^* may not exist. We know that $\mathsf{Sum\_Cal\_X}(\max_j G_j) = \sum_{\tau_i} 0 = 0$, which is smaller than or equal to $m - U_H^H$. However, we do not know whether $\mathsf{Sum\_Cal\_X}(\min_j G_j)$ is larger than $m - U_H^H$ or not.

We claim that for a task τ_i, $\mathsf{Cal\_X_i}(0) = \mathsf{Cal\_X_i}(\min_j G_j)$. If $\min_j G_j = 0$, we have $\mathsf{Cal\_X_i}(0) = \mathsf{Cal\_X_i}(\min_j G_j)$. Otherwise, we have $\min_j G_j > 0$. When $\min_j G_j \neq \mathsf{Cost}_i(1 - u_i^H)$, by the second case of Lemma 3.8, we know that $\mathsf{Cal\_X_i}(0) = \mathsf{Cal\_X_i}(\min_j G_j)$. When $\min_j G_j =$

$\mathsf{Cost}_i(1-u_i^H)$, we need to apply the third case of Lemma 3.8. Since $\mathsf{Cal\_X_i}(\psi)$ calculates X_i s.t. $\psi = \mathsf{Cost}(X_i)$ in the third case of Lemma 3.8, we have $\mathsf{Cal\_X_i}(\mathsf{Cost}_i(1-u_i^H)) = 1-u_i^H$, from which we have $\mathsf{Cal\_X_i}(\min_j G_j) = 1-u_i^H$ by using $\mathsf{Cost}_i(1-u_i^H) = \min_j G_j$. Since $\mathsf{Cal\_X_i}(0) = 1-u_i^H$, we have $\mathsf{Cal\_X_i}(0) = \mathsf{Cal\_X_i}(\min_j G_j)$. Thus, we proved the claim.

Since $\mathsf{Sum\_Cal\_X}(\min_j G_j) = \mathsf{Sum\_Cal\_X}(0)$ and $\mathsf{Sum\_Cal\_X}(0) > m - U_H^H$, we confirm that G_j^* exists. □

Algorithm Complexity. We consider the complexity of Algorithm 3.1 (OERA). Let n be $|\tau|$. Since the computation of $\mathsf{Sum\_Cal\_X_i}(\psi)$ for a given ψ takes $O(n)$ times, the binary search takes $O(n \log n)$. Additionally, finding a solution for Eq. (3.26) takes $O(n)$. Since $O(n \log n) + O(n) = O(n \log n)$, OERA has linearithmic time-complexity ($O(n \log n)$) .

3.5.2. The MC-Derivative Algorithm

In previous section, we presented a rate assignment algorithm by solving the problem in Def. 3.4 using a convex optimization framework. Extending and solving such optimization problem for a larger system such as a multi-criticality system without compromising on the schedulability performance and runtime complexity can be quite challenging. To overcome the shortcoming of the complex convex optimization framework, we require a simple straightforward approach to determine the optimal execution rates. Thus, we propose MC-Derivative which assigns execution rates by using the first derivative principles, instead of using convex optimization technique.

The MC-Derivative rationale. We recall the objective function of the assignment problem mentioned in Def. 3.4 as $f(X)$:

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

and X 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

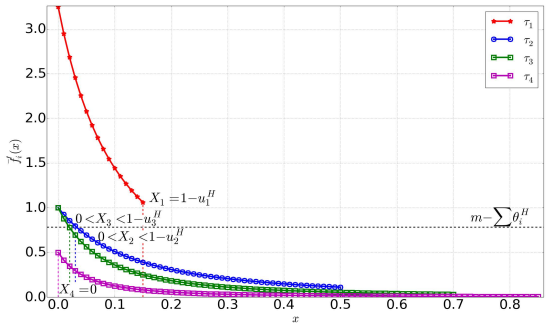


Figure 4: The plot of $\bar{f}_i'(X_i = x)$ varying $x \in \mathbb{R}$ for the task set in Table 3

function $f(X)$, and we denote the resulting X_i as X_i^* (the optimal value of X_i). Since the objective function is differentiable and convex, we can apply the first partial derivative techniques to find the optimal solution.

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

$$\bar{f}_i'(X_i) = \frac{u_i^L (u_i^H - u_i^L)}{(X_i + u_i^L)^2} \tag{3.27}$$

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

We present the rationale behind MC-Derivative. It minimizes the objective function by assigning the execution rates to the tasks in the order of their *maximum first partial derivative*. Initially, it assigns $X_i = 0$ for each HI-task, the minimum value according to CON2 in the assignment problem (Def. 3.4). This assignment makes the objective function the largest. Then, MC-Derivative minimizes the total sum by increasing the X_i of the tasks with a larger $\bar{f}_i'(X_i)$. We present the application of MC-Derivative strategy with an example.

Example 3.4. *Consider a sample task set in Table 3. Fig. 4 plots $\bar{f}_i'(x)$ of the corresponding task set for $x \in \mathbb{R}$. It can be observed that task τ_1 has a larger $\bar{f}_i'(x)$ compared to that of task*

τ_2, τ_3 and τ_4. Hence, MC-Derivative initially picks task τ_1 for X_i assignment then followed by task τ_2, τ_3 and τ_4. MC-Derivative first assigns the minimum required execution rate, $X_i = 0$ for each task. Then, MC-Derivative increases X_1 until its $\bar{f}'_i(X_i)$ of τ_1 becomes equal to that of task τ_2/τ_3 or until X_1 becomes $1 - u_1^H$. Then, the remaining system utilization $(m - \sum_{1 \leq i \leq 4}(u_i^H + X_i))$, called Slack, is allocated to the remaining tasks proportionately such that the updated $\bar{f}'_i(X_i)$ values of tasks are equal. When X_1 becomes the maximum value $(1 - u_1^H)$, we allocate Slack to τ_2 and τ_3. Since $\bar{f}'_4(X_4)$ is lower than the updated $\bar{f}'_i(X_i)$ of task τ_2 and τ_3, X_4 remains as 0.

Note that, the slack (the black dotted line in Fig. 4) lies below $\bar{f}'_1(1 - u_1^H)$, $\bar{f}'_2(0)$ and $\bar{f}'_3(0)$, and above $\bar{f}'_2(1 - u_2^H)$, $\bar{f}'_3(1 - u_3^H)$ and $\bar{f}'_4(0)$. Therefore, MC-Derivative assigns $X_1 = 1 - u_1^H$ (i.e., $\theta_1^H = 1$), $0 < X_2 < 1 - u_2^H$ (i.e., $u_2^H < \theta_2^H < 1$), $0 < X_3 < 1 - u_3^H$ (i.e., $u_3^H < \theta_3^H < 1$) and $X_4 = 0$ (i.e., $\theta_4^H = u_4^H$).

In MC-Derivative, we utilize the classification of HI-tasks (Def. 3.5) based on their final X_i assignment.

Definition 3.5 (HI-tasks Classification)**.** *Without any loss of generality, tasks in τ_H are categorized into three sets (τ_{MAX}, τ_{MIN} and τ_{REM})[3] based on their final X_i values:*

$$\tau_{\mathsf{MAX}} \stackrel{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ X_i = 1 - u_i^H\},$$
$$\tau_{\mathsf{REM}} \stackrel{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ 0 < X_i < 1 - u_i^H\},$$
$$\tau_{\mathsf{MIN}} \stackrel{\mathsf{def}}{=} \{\tau_i \in \tau_H |\ X_i = 0\}.$$

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

For the example task set in Table 3, $\tau_{\mathsf{MAX}} = \{\tau_1\}$, $\tau_{\mathsf{REM}} = \{\tau_2, \tau_3\}$ and $\tau_{\mathsf{MIN}} = \{\tau_4\}$.

The MC-Derivative algorithm. Algorithm 3.2 gives the overall flow for the rate assignment strategy. The algorithm only inputs task set with $U_H^H \leq m$ for computing the rates,

[3] τ_{MAX} denotes the set of HI-tasks with the maximum rate assignment (i.e., $\theta_i^H = 1$), τ_{MIN} denotes the set of HI-tasks with the minimum rate assignment (i.e., $\theta_i^H = u_i^H$) and τ_{REM} denotes the set of remaining HI-tasks other than τ_{MAX} and τ_{MIN}.

and declares failure otherwise. Line 1 computes the X_i for all HI-tasks and τ_{REM} using Algorithm 3.3. Line 2 assigns the θ_i^H rate to the HI-tasks. Lines $3-5$ assign the slack (output of Algorithm 3.3) to the HI-tasks in τ_{REM}. Line 6 computes the LO-execution rate of the tasks using $\mathsf{computeLoRate_i}(\theta_i^H)$ defined in Sec. 3.4. Finally, we can check the LO-mode platform feasibility by $\sum_{\tau_i \in \tau} \theta_i^L \leq m$.

Algorithm 3.2 MC-Derivative Algorithm

Input: τ where $U_H^H \leq m$
Output: θ_i^H for each $\tau_i \in \tau_H$ and θ_i^L for each $\tau_i \in \tau$
1: Compute X_i and τ_{REM} using Algorithm 3.3
2: For each $\tau_i \in \tau_H$, assign $\theta_i^H := u_i^H + X_i$
3: **if** (Slack > 0) **then**
4: For each $\tau_i \in \tau_{\mathsf{REM}}$, assign θ_i^H using Algorithm 3.4
5: **end if**
6: For each $\tau_i \in \tau$, $\theta_i^L := \mathsf{computeLoRate_i}(\theta_i^H)$

Algorithm 3.3 classifies the HI-tasks into τ_{MAX}, τ_{MIN} and τ_{REM}. Line 2 assigns the minimum valid X_i for all the HI-tasks. Line 3 determines the minimum and maximum $\bar{f}_i'(x)$ $\big(2 \times |\tau_H|$ elements of $F\big)$ for all the HI-tasks. Line 4 sorts all the HI-tasks in increasing order of $\bar{f}_i'(x)$. Lines $5-13$ determine the range within which the slack (computed in Line 8) lies such that the tasks with a larger $\bar{f}_i'(x)$ is assigned a maximum X_i, which is $1-u_i^H$ according to CON3 in Def. 3.4. We consider each member (F_j)[4] of this sorted list (Line 5). For a HI-task, X_i is assigned with x such that $\bar{f}_i'(x) = F_j$ and $x \geq 0$. When $\Gamma := F_j$, by Eq. (3.27), we have

$$x = \max(0, \sqrt{\frac{u_i^L(u_i^H - u_i^L)}{\Gamma}} - u_i^L). \tag{3.28}$$

Lines 10 and 14 determine the set of tasks (τ_{REM}) that get the remaining slack in the system after the X_i assignment. That is, tasks with $\bar{f}_i'(0) > F_j > \bar{f}_i'(1-u_i^H)$ are added to τ_{REM}. Line 10 adds all τ_k s.t. $\bar{f}_k'(x) = F_j$ to τ_{REM} for slack assignment if $X_k = 0$ because the slack lies below $\bar{f}_k'(0)$.

Algorithm 3.3 (Line 4) sorts F in increasing order to assign a maximum value of X_i to

[4]If two tasks have the same parameters, there exist F_j and F_{j+1} such that $F_j = F_{j+1}$ with the different task index.

all the HI-tasks in the first iteration. If the assigned rates are not feasible (i.e., slack is negative), then the above process is repeated for the next HI-task τ_k in the sorted list. By doing so, MC-Derivative assigns a maximum value of X_i to the tasks such that CON1 of Def. 3.4 holds (the sum of the HI-execution rates is no greater than the system capacity). That is, tasks with $\bar{f}_i'(1-u_i^H) \geq F_j$ ($\tau_i \in \tau_{\mathsf{MAX}}$) is assigned a maximum X_i of $1-u_i^H$. The FOR loop exits either when the condition in Line 9 holds (assigned rates are feasible) or without any modification to the initial assignment of X_i (Line 2), which is always feasible.

Algorithm 3.3 HI-Tasks Classification

Input: τ_H and τ_{REM}
Output: X_i for each $\tau_i \in \tau_H$, τ_{REM}, and Slack
1: Initialize $\tau_{\mathsf{REM}} = \emptyset$
2: For each $\tau_i \in \tau_H$, assign $X_i = 0$
3: Let F be the set of $\bar{f}_i'(x)$ values at $x = 0$ and $x = 1 - u_i^H$
4: Sort F in increasing order
5: **for** j := 1 to $|F|$ (the size of F) **do**
6: Let k be the task index of F_j (the j-th member in F)
7: For each τ_i s.t. $\tau_i \neq \tau_k$, $X_i := \min(1-u_i^H, x)$ s.t. $\bar{f}_i'(x) = F_j$ (by Eq. (3.28))
8: Let $\mathsf{Slack} = m - \sum_{\tau_i \in \tau_H}(u_i^H + X_i)$
9: **if** $(\mathsf{Slack} \geq 0)$ **then**
10: If $X_k = 0$, then $\forall \tau_k$ s.t. $\bar{f}_k'(x) = F_j$ add τ_k in τ_{REM}
11: Break
12: **end if**
13: **end for**
14: For each $\tau_i \in \tau_H$ s.t. $0 < X_i < 1 - u_i^H$, add τ_i in τ_{REM}

Algorithm 3.4 allocates Slack (the remaining utilization after Algorithm 2) to tasks in τ_{REM}. The algorithm receives Slack (computed in Line 3 of Algorithm 3.2) as a input and allocates it to all the HI-tasks in τ_{REM} proportionately, such that their updated $\bar{f}_i'(x)$ values are equal (formally proven in Lemma 3.10).

Algorithm 3.4 Slack Allocation for τ_{REM}

Input: τ_{REM}, Slack, and X_i
Output: θ_i^H for each $\tau_i \in \tau_{\mathsf{REM}}$
1: **for** τ_i in τ_{REM} **do**
2: Assign $\theta_i^H := u_i^H + x$ s.t. $\bar{f}_i'(x) = \Gamma$ where $\Gamma = \left(\frac{\sum_{\tau_j \in \tau_{\mathsf{REM}}} \sqrt{u_j^L(u_j^H - u_j^L)}}{\mathsf{Slack} + \sum_{\tau_j \in \tau_{\mathsf{REM}}} (X_j + u_j^L)} \right)^2$
3: **end for**

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. 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 3.9. Finally, we prove the optimality in Theorem 3.4.

The tasks in τ_{MAX} cannot be assigned any additional X_i since $X_i \leq 1 - u_i^H$ from CON3 (i.e., $\theta_i^H \leq 1$). The X_i of the tasks in τ_{MIN} cannot be reduced any further because $X_i \geq 0$ from CON2 (i.e., $\theta_i^H \geq u_i^H$). Therefore, there are only two cases to consider for rate assignments different from MC-Derivative.

Case 1: The X_i of the task $\tau_p \in \tau_{\mathsf{MAX}}$ is partially reassigned[5] 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 partially reassigned to task $\tau_q \in \tau_{\mathsf{MIN}}$.

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

Before proving these cases, we introduce an auxiliary lemma (Lemma 3.9) showing the property of $\bar{f}_i'(X_i^*)$ of any two tasks for a valid reassignment of execution rates. Using the property, Theorem 3.4 proves the optimality of MC-Derivative.

Lemma 3.9. *For any two tasks τ_p and τ_q, assume that their X_i's are assigned by Algorithm 3.2. If the X_i of τ_p is partially reassigned to the one of τ_q, we have $\bar{f}_p'(X_p^*) \geq \bar{f}_q'(X_q^*)$.*

Proof. We need to show $\bar{f}_p'(X_p^*) \geq \bar{f}_q'(X_q^*)$ for the two cases. Consider Case 1. For any two tasks τ_i and τ_j such that $\bar{f}_i'(X_i) > \bar{f}_j'(X_j)$, Lines $5 - 12$ of Algorithm 3.3 ensure task τ_i is assigned an X_i such that $\bar{f}_i'(X_i)$ is decreased to match $\bar{f}_j'(X_j)$. The task $\tau_q \in \tau_{\mathsf{MIN}}$ does not receive any execution rate ($X_q^* = 0$) apart from its HI-utilization u_q^H because its $\bar{f}_q'(X_q^*)$ is lower than $\bar{f}_p'(X_p^*)$ of any task $\tau_p \in \{\tau_{\mathsf{MAX}}, \tau_{\mathsf{REM}}\}$ until the slack gets assigned. In the case of $\tau_p \in \tau_{\mathsf{MAX}}$ and $\tau_q \in \tau_{\mathsf{REM}}$, the task τ_p is assigned a maximum execution rate ($X_p^* = 1 - u_p^H$) and cannot be assigned any further. This implies $\bar{f}_p'(X_p)$ cannot be decreased any further

[5]Denote X_i of τ_p is decreased by ϵ and that of τ_q is increased by ϵ.

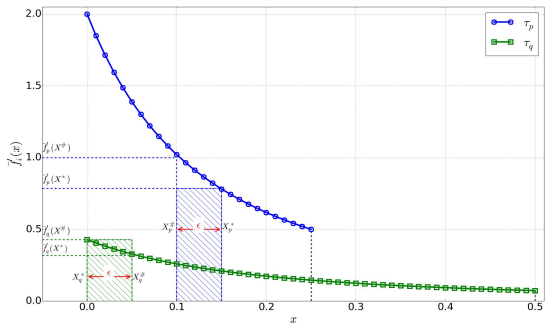


Figure 5: The partial reassignment of the execution rate from τ_p to τ_q

which means $\bar{f}'_p(X^*_p) > \bar{f}'_q(X^*_q)$.

We 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 $\bar{f}'_p(X^*_p) = \bar{f}'_q(X^*_q)$ by Line 2 of Algorithm 3.4.

□

Theorem 3.4. *MC-Derivative is an optimal rate assignment algorithm.*

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

Consider the reassignment of rate from task τ_p to task τ_q as shown in Fig. 5. 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 reassigned, 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 $\epsilon \in [0, \min(X^*_p, 1 - u^H_q - X^*_q)]$ since for any task τ_i, $X_i \in [0, 1 - u^H_i]$.

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

that

$$
\begin{aligned}
& f_p(X_p^{\#}) + f_q(X_q^{\#}) \geq f_p(X_p^*) + f_q(X_q^*) \\
\Leftrightarrow\ & f_p(X_p^{\#}) - f_p(X_p^*) \geq f_q(X_q^*) - f_q(X_q^{\#}) \\
\Leftrightarrow\ & \int_{X_p^{\#}}^{X_p^*} \bar{f}_p'(x)dx \geq \int_{X_q^*}^{X_q^{\#}} \bar{f}_q'(x)dx. \qquad (3.29)
\end{aligned}
$$

For task τ_i and X_1 and X_2 s.t. $X_1 < X_2$, note that $\int_{X_1}^{X_2} \bar{f}_i'(x)dx \geq \min_{X_1 \leq x \leq X_2} \bar{f}_i'(x) \cdot (X_2 - X_1)$ and $\int_{X_1}^{X_2} \bar{f}_i'(x)dx \leq \max_{X_1 \leq x \leq X_2} \bar{f}_i'(x) \cdot (X_2 - X_1)$ (see the shaded rectangle in Fig. 5). To show that Eq. (3.29), we only need to show that

$$
\begin{aligned}
& \min_{X_p^{\#} \leq x \leq X_p^*} \bar{f}_p'(x)(X_p^{\#} - X_p^*) \geq \max_{X_q^* \leq x \leq X_q^{\#}} \bar{f}_q'(x)(X_q^* - X_q^{\#}) \\
\Leftrightarrow\ & \min_{X_p^{\#} \leq x \leq X_p^*} \bar{f}_p'(x)\epsilon \geq \max_{X_q^* \leq x \leq X_q^{\#}} \bar{f}_q'(x)\epsilon \\
\Leftrightarrow\ & \min_{X_p^{\#} \leq x \leq X_p^*} \bar{f}_p'(x) \geq \max_{X_q^* \leq x \leq X_q^{\#}} \bar{f}_q'(x). \qquad (3.30)
\end{aligned}
$$

Note that 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 3.9, Eq. (3.30) holds. □

Correctness. Before presenting the correctness of MC-Derivative, we present the correctness of Algorithm 3.4. The following lemma shows the correctness for proportional allocation of slack to the tasks in τ_{REM} by Algorithm 3.4.

Lemma 3.10. *Given a slack MC-Derivative assigns X_i to the tasks in τ_{REM} such that their $\bar{f}_i'(X_i)$ are equal.*

Proof. Let X_i^* denote the assignment of X_i to the tasks in τ_{REM} such that their $\bar{f}_i'(X_i)$ are equal. Let Γ $\big($defined as $\bar{f}_i'(X_i^*)\big)$ be the updated $\bar{f}_i'(X_i)$ value of the tasks in τ_{REM}. Then for each $\tau_i \in \tau_{\mathsf{REM}}$, we can compute Γ from the relation between the sum of X_i (intermediate assignment in Line 7 of Algorithm 3.3) and the sum of X_i^* (final assignment in Line 2 of

Algorithm 3.4). Since the Slack is assigned only to the tasks in τ_{REM}, $\mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} X_i = \sum_{\tau_i \in \tau_{\mathsf{REM}}} X_i^*$. Then,

$$\mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i^* + u_i^L)$$

$$\Leftrightarrow \mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} \sqrt{\frac{u_i^L (u_i^H - u_i^L)}{\bar{f}_i'(X_i^*)}}$$

$$\big(\text{by Eq. (3.28)}\big)$$

$$\Leftrightarrow \mathsf{Slack} + \sum_{\tau_i \in \tau_{\mathsf{REM}}} (X_i + u_i^L) = \sum_{\tau_i \in \tau_{\mathsf{REM}}} \sqrt{\frac{u_i^L (u_i^H - u_i^L)}{\Gamma}}$$

$$\big(\text{by } \bar{f}_i'(X_i^*) = \Gamma\big)$$

Thus, Algorithm 3.4 assigns X_i to the tasks in τ_{REM} such that $\bar{f}_i'(x) = \Gamma$. □

Now, we present the correctness of MC-Derivative. Line 7 in Algorithm 3.3 ensures CON2 and CON3 in Def. 3.4 (i.e., $u_i^H \leq \theta_i^H \leq 1$). Line 9 in Algorithm 3.3 guarantees CON1 (i.e., the sum of the HI-execution rate does not exceed the system capacity).

Complexity. MC-Derivative algorithm has a run-time complexity of $O(n \log n)$ in the number of tasks in τ. X_i computation can be done in $O(n \log n)$ time complexity. Assigning an initial value to X_i can be done in $O(n)$ time. Sorting F in Line 4 of Algorithm 3.3 can be done in $O(n \log n)$ time complexity. Selecting a task with the least $\bar{f}_i'(x)$ that satisfies the condition in Line 9 of Algorithm 3.3 can be done using a binary search. Thus, the outer FOR loop of Line 5 of Algorithm 3.3 runs at most $O(\log n)$ times. Computing X_i in Line 7 of Algorithm 3.3 consumes $O(n)$ time. Proportionately allocating the remaining slack to the tasks can also be done in $O(n)$ time. Determining the HI-execution rates and the LO-execution rates can also be done in $O(n)$ time.

3.6. The Speedup Factor

In this section, we quantify the effectiveness of MC-Fluid via the metric of processor speedup factor [36]. In general, the speedup factor of an algorithm $\mathbb{A}$ is defined as a real number α (≥ 1) such that any task set schedulable by an optimal clairvoyant algorithm[6]. on m speed-*1* processors is also schedulable by $\mathbb{A}$ on m speed-α processors. In other words, a task set that is clairvoyantly schedulable on m speed-$(1/\alpha)$ processors is also schedulable by $\mathbb{A}$ on m speed-*1* processors.

The following lemma shows $\alpha \geq 4/3$ for any algorithm.

Lemma 3.11. *Any non-clairvoyant MC scheduling algorithm on multicores cannot have a speedup factor better than* 4/3*.*

Proof. It directly follows from Theorem 5 of Baruah et al. [4]. □

The following theorem shows that α for MC-Fluid is 4/3, which means that MC-Fluid has an optimal speedup factor.

Theorem 3.5. *Given a task set τ that is clairvoyantly MC-schedulable on m speed-(3/4) processors, the task set is MC-schedulable on m speed-1 processors by MC-Fluid.*

Proof. We consider the special assignment method: $\theta_i^H := u_i^H/x$ where x is a real number s.t. $0 \leq x \leq 1$. If it is a valid schedulable assignment, the OERA algorithm can find the assignment because OERA is the optimal assignment algorithm. By Lemma 3.4, we pick execution rates as $\theta_i^L = u_i^L$ for $\tau_i \in \tau_L$ and $\theta_i^L = u_i^L \cdot \theta_i^H/(\theta_i^H - u_i^H + u_i^L)$ for $\tau_i \in \tau_L$. To show that τ is MC-schedulable by MC-Fluid, we need to satisfy Eqs. (3.1), (3.2), (3.3), and (3.4) of Theorem 3.1. By the selection of the execution rates, Eqs. (3.1) and (3.2) hold.

Since τ is clairvoyantly schedulable on m speed-*(3/4)* processors, we know that $U_L^L + U_H^L \leq 3m/4$ and $U_H^H \leq 3m/4$. Since the task is assumed to be executable on a speed-*(3/4)*

[6]In MC systems, a clairvoyant (fluid or non-fluid) scheduling algorithm is the one that knows the time instant of mode-switch before runtime scheduling. However, it is only possible for imaginary algorithms because we cannot know the mode-switch instant in advance.

processor, we know that each task τ_i has $u_i^L \le u_i^H \le 3/4$.

To be a valid execution rate by Def. 2.3, HI-execution rate θ_i^H should satisfy $\theta_i^H \le 1$:

$$\frac{u_i^H}{x} \le 1 \;\Leftarrow\; \frac{3}{4} \le x. \qquad (\text{by } u_i^H \le \frac{3}{4})$$

Thus, we need $x \ge 3/4$ to be a valid execution rate. Since we assume that $\theta_i^L \le \theta_i^H$ by Lemma 3.2, LO-execution rates are also valid.

We show that Eq. (3.4) holds ($\sum_{\tau_i \in \tau_H} \theta_i^H \le m$):

$$\sum_{\tau_i \in \tau_H} \frac{u_i^H}{x} = \frac{U_H^H}{x} \le \frac{3m}{4} \cdot \frac{4}{3} \le m,$$

because $U_H^H \le \frac{3m}{4}$ and $1/x \le 4/3$.

Before checking Eq. (3.3), we claim that $\theta_i^L \le u_i^L + \frac{1-x}{x} u_i^H$ for $\tau_i \in \tau_H$:

$$\begin{aligned}
& \frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L} \le u_i^L + \frac{1-x}{x} u_i^H \\
\Leftrightarrow\; & \frac{u_i^L \cdot u_i^H/x}{u_i^H/x - u_i^H + u_i^L} \le u_i^L + \frac{1-x}{x} u_i^H \quad (\text{since } \theta_i^H = \tfrac{u_i^H}{x}) \\
\Leftrightarrow\; & \frac{u_i^L \cdot u_i^H}{x \cdot u_i^L + (1-x) u_i^H} \le u_i^L + \frac{1-x}{x} u_i^H \\
\Leftrightarrow\; & u_i^L \cdot u_i^H \le x(u_i^L)^2 + 2(1-x) u_i^L \cdot u_i^H + \frac{(1-x)^2}{x} (u_i^H)^2 \\
\Leftrightarrow\; & 0 \le x^2 (\frac{u_i^L}{u_i^H})^2 + x(1-2x)\frac{u_i^L}{u_i^H} + (1-x)^2 \\
\Leftrightarrow\; & 0 \le x^2(\beta-1)^2 + x(\beta-2) + 1 \quad (\text{by replacing } \beta = \tfrac{u_i^L}{u_i^H}) \\
\Leftrightarrow\; & 0 \le (x(\beta-1) - 1/2)^2 - x + 3/4 \\
\Leftarrow\; & 0 \le 3/4 - x, \qquad (\because (x(\beta-1) - 1/2)^2 \ge 0)
\end{aligned}$$

which is true if $x \le 3/4$. We assume that $x \le 3/4$ for the claim.

We show that Eq. (3.3) holds ($\sum_{\tau_i \in \tau_H} \theta_i^L \leq m$):

$$
\begin{aligned}
& U_L^L + \sum_{\tau_i \in \tau_H} \theta_i^L \\
= \; & U_L^L + \sum_{\tau_i \in \tau_H} \left(u_i^L + \frac{1-x}{x} u_i^H \right) \\
= \; & U_L^L + U_H^L + \frac{1-x}{x} U_H^H \\
\leq \; & \frac{3m}{4} + \frac{1-x}{x} \cdot \frac{3m}{4} \qquad (\text{by } U_H^H \leq \tfrac{3m}{4},\, U_L^L + U_H^L \leq \tfrac{3m}{4}) \\
\leq \; & \frac{3m}{4} + \frac{1}{3} \cdot \frac{3m}{4} \qquad (\text{by } \tfrac{1-x}{x} \leq \tfrac{1}{3} \text{ from } x \leq \tfrac{3}{4}) \\
\leq \; & m
\end{aligned}
$$

In sum, to have a solution, we need both $x \leq 3/4$ and $x \geq 3/4$, which is possible at $x = 3/4$. □

Baruah et al. [4] showed that any non-clairvoyant uniprocessor algorithm for a dual-criticality task set cannot have a speedup factor better than 4/3. This result is also applicable for multiprocessors. Since MC-Fluid has a speedup factor of 4/3, it is an optimal multicore MC scheduling algorithm in terms of the speedup factor.

3.7. Summary

We presented a multiprocessor MC 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 an optimal execution rate assignment algorithm to determine the execution rates in $O(n \log n)$ time. Furthermore, we also presented that MC-Fluid has an optimal speedup factor.

CHAPTER 4 : Transforming Fluid-based Mixed-Criticality Scheduling into Discrete-time Platforms

In the previous chapter, we developed a resource-efficient MC-Fluid scheduling algorithm for multiprocessor MC systems. However, it cannot be not directly implemented on real hardware platforms because MC-Fluid assumes that a task executes on a fractional processor. In real hardware platforms, a processor only executes one task at a time. In this chapter, we develop MC scheduling techniques that can be implemented on multiprocessor platforms.

4.1. The Overview of the MC-Discrete Scheduling Framework

One challenge that we face is how to map the MC-Fluid schedule to a non-fluid schedule. Inspired by schedule transformation techniques that the optimal fluid schedule into a non-fluid schedule, called DP-Fair (Deadline Partitioning Fair), without the loss of schedulability [9, 20, 38, 50], we develop the MC-DP-Fair scheduling algorithm with real-number deadlines and the MC-Discrete scheduling algorithm with integer deadlines. MC-DP-Fair preserves the speedup optimality in MC-Fluid while MC-Discrete mode-switches only at integer time points, which is preferred in practice.

Contributions. Our contributions in this chapter are summarized as follows:

- We develop the MC-DP-Fair algorithm, which constructs a non-fluid schedule from the MC-Fluid schedule and preserves the speedup-optimality of MC-Fluid (Chapter 4.2).
- We develop the MC-Discrete algorithm, which transforms the MC-Fluid schedule and allows mode-switch in integer time points (Chapter 4.3).
- Simulation results show that MC-DP-Fair and MC-Discrete outperform other existing approaches (Chapter 4.4).

4.2. The MC-DP-Fair Scheduling Algorithm and Schedulability Analysis

Building upon the DP-Fair algorithm, we propose the *MC-DP-Fair* scheduling algorithm, which constructs a non-fluid schedule based on MC-Fluid. To extend MC-Fluid with the DP-Fair algorithm, we have the following questions:

- How to compute LO-density (density in LO-mode) and HI-density from LO- and HI-execution rates?
- How to determine whether a given task set is MC-schedulable by MC-DP-Fair?

To answer the first question, we utilize a concept of virtual deadlines (VDs), which is introduced in EDF-VD [4]. To be compatible with DP-Fair, we will transform execution rates of tasks into (virtual) deadlines, which may be different from their real deadlines.

With VDs, we present the MC-Discrete algorithm as follows:

Definition 4.1. *MC-DP-Fair is defined with per-task virtual deadlines and a special DP* Γ*, which is the earliest DP after mode-switch. For each task* τ_i*, we define a VD:* $V_i \in \mathbb{R}$ *s.t.* $0 < V_i \leq T_i$*. In MC-DP-Fair, each task executes with its VD before* Γ *and with its real deadline after* Γ*, according to DP-Fair.*

To utilize the analysis of MC-Fluid, we need to check whether the execution amount of tasks in MC-Fluid is the one in MC-DP-Fair. In non-MC domain, the execution amounts of tasks in fluid platforms is the same as the one in DP-Fair at every DP, according to Lemma 2.2. In MC domain, MC-DP-Fair can utilize the results of MC-Fluid because MC-DP-Fair adopts the delayed mode-switch at Γ (scheduling policy is changed not at mode-switch but at Γ).

We will show the computation of LO-density and HI-density with VD and discuss VD assignment, utilizing the execution rates of MC-Fluid. To answer the second question, we will present the schedulability analysis later.

Density Computation. We can compute LO-density: for each $\tau_i \in \tau$, $\delta_i^L \stackrel{\text{def}}{=} C_i^L / V_i$. We

need to compute HI-density differently depending on the time instant of mode-switch is a DP or not. We claim that we do not need to consider the latter case: mode-switch happens in the middle of a time slice. Note that Γ indicates the end of this time slice. Due to the delayed scheduling policy changed, MC-DP-Fair executes HI-tasks for the amount of their required remaining execution (calculated based on C_i^L) until Γ. Then, the case is equivalent to the former case (mode-switch happens at Γ).

Now, we only need to compute HI-density when mode-switch happens at a DP. To do this, we need to know the remaining time to the deadline of the carry-over job, which is active at mode-switch, and the remaining execution amount of the job. The former is calculated by $T_i - w_i$ where w_i is the time interval length from its release time to the boundary. The latter is calculated by $C_i^H - E_i^L(w_i)$ where $E_i^L(t)$ is the execution amount of the job for any timer interval t ($E_i^L(t) = \delta_i^L \cdot t$). Using these values, we can compute δ_i^H:

$$\delta_i^H \stackrel{\text{def}}{=} \frac{C_i^H - E_i^L(w_i)}{T_i - w_i} = \frac{C_i^H - \delta_i^L \cdot w_i}{T_i - w_i}. \tag{4.1}$$

Virtual Deadline Assignment. Although LO-density is fixed for a given VD, HI-density varies on runtime. Intuitively, to utilize MC-Fluid analysis, LO-density for each task is no greater than θ_i^L and HI-density for each HI-task is no greater than θ_i^H. Since we know the optimal rate assignment of MC-Fluid, we propose a VD assignment based on the rate assignment:

Definition 4.2 (MC-DP-Fair VD assignment)**.** *We assign $V_i := T_i$ for an LO-task $\tau_i \in \tau_L$ and $V_i := V_i^*$ where $V_i^* = C_i^L/\theta_i^{L*}$ for a HI-task $\tau_i \in \tau_H$ where θ_i^{L*} is the value of θ_i^L in the optimal assignment for MC-Fluid.*

Lemma 4.1 validates the correctness of the VD assignment.

Lemma 4.1. *Given an MC task set τ, we assume the virtual deadline assignment by Def. 4.2. Thus, we have (i) $\delta_i^L \le \theta_i^{L\triangle}$ for each task $\tau_i \in \tau$ and (ii) $\delta_i^H \le \theta_i^{H\triangle}$ for each task $\tau_i \in \tau_H$, where $\theta_i^{L\triangle} = C_i^L/V_i^*$ and $\theta_i^{H\triangle} = (u_i^H - u_i^L)/(1 - u_i^L/\theta_i^{L\triangle})$.*

Proof. (i) We show that $\delta_i^L \leq \theta_i^{L\triangle}$ for $\tau_i \in \tau$: we have $\delta_i^L = C_i^L/T_i = u_i^L = \theta_i^{L\triangle}$ for $\tau_i \in \tau_L$ by Lemma 3.4 and $\delta_i^L = C_i^L/V_i = \theta_i^{L\triangle}$ for $\tau_i \in \tau_H$.

(ii) We will show that $\delta_i^H \leq \theta_i^{H\triangle}$ for $\tau_i \in \tau_H$. Since δ_i^H varies on w_i in Eq. (4.1), we first derive the maximum δ_i^H and later show that the value is no greater than $\theta_i^{H\triangle}$.

To find the maximum δ_i^H, consider derivative of Eq. (4.1):

$$\frac{d\ \delta_i^H}{d\ w_i} = \frac{-\delta_i^L(T_i - w_i) + (C_i^H - \delta_i^L \cdot w_i)}{(T_i - w_i)^2} = \frac{C_i^H - \delta_i^L \cdot T_i}{(T_i - w_i)^2},$$

which is greater than or equal to 0 if $u_i^H \geq \delta_i^L$.

To show that the derivative is non-negative, we show that $u_i^H \geq \delta_i^L$. Since $\theta_i^{L\triangle} \geq \delta_i^L$ from Case (i), we only need to show $u_i^H \geq \theta_i^{L\triangle}$: since $\theta_i^{L\triangle} = \frac{u_i^L \cdot \theta_i^{H\triangle}}{\theta_i^{H\triangle} - u_i^H + u_i^L}$ by Lemma 3.4,

$$\begin{aligned} u_i^H \geq \frac{u_i^L \cdot \theta_i^{H\triangle}}{\theta_i^{H\triangle} - u_i^H + u_i^L} &\Leftrightarrow u_i^H(\theta_i^{H\triangle} - u_i^H + u_i^L) \geq u_i^L \cdot \theta_i^{H\triangle} \\ &\Leftrightarrow \theta_i^{H\triangle}(u_i^H - u_i^L) \geq u_i^H(u_i^H - u_i^L) \\ &\Leftrightarrow \theta_i^{H\triangle} \geq u_i^H, \end{aligned}$$

which is true because θ_i^{H*} satisfies Eq. (3.11), which is $\theta_i^{H*} \geq C_i^L/T_i$, by Theorem 3.2 and $\theta_i^{H\triangle} \geq C_i^L/\lfloor T_i \rfloor = C_i^L/T_i$.

From the derivative of δ_i^H, we can find the inequality that δ_i^H is maximized at the maximum w_i because $\frac{d\ \delta_i^H}{d\ w_i} \geq 0$ and thus δ_i^H is an increasing function of w_i: since $w_i \leq V_i$,

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

which is $\theta_i^{H\triangle}$ by Lemma 3.4. Thus, we conclude $\delta_i^H \leq \theta_i^{H\triangle}$. □

Schedule Generation. Since we assigned VDs, we can calculate how much resources are allocated to each task for a time slice. Due to the delayed mode-switch of MC-DP-Fair, the resource allocation is not changed within the time slice. Then, we can directly use the schedule generation of DP-Fair, which is McNaughton's algorithm [42] satisfying a sequential execution constraint (or called a non-parallel execution constraint) on multiprocessor platforms.

Schedulability Analysis. Since MC tasks are subject to different execution time requirements (and thereby different densities), we extend Lemma 2.3 for MC systems as follows.

Lemma 4.2. *Given δ_i^L and δ_i^H for each task $\tau_i \in \tau$, an MC task set τ is MC-schedulable iff*

$$\sum_{\tau_i \in \tau} \delta_i^L \leq m, \tag{4.2}$$

$$\sum_{\tau_i \in \tau_H} \delta_i^H \leq m. \tag{4.3}$$

Proof. Eq. (4.2) is the LO-schedulability condition by Lemma 2.3 with LO-mode. Eq. (4.3) is the HI-schedulability condition by Lemma 2.3 with HI-mode. Since MC-schedulability implies both LO- and HI-schedulability, the task set is MC-schedulable iff Eqs. (4.2) and (4.3) hold. □

Based on Lemmas 4.1 and 4.2, Theorem 4.1 presents that MC-DP-Fair has the same schedulability as MC-Fluid.

Theorem 4.1. *An MC task set τ is schedulable by MC-DP-Fair with the virtual deadline assignment by Def. 4.2 iff τ is MC-Fluid-feasible.*

Proof. ($\Rightarrow$) To show that τ is feasible, we need to show that there exists an assignment satisfying Eq. (3.11), (3.12), and (3.13) by Theorem 3.2. Since θ_i^{H*} for $\forall \tau_i \in \tau_H$ can only violate Eq. (3.12) according to Def. 3.4, we need to show that Eq. (3.12) holds.

Since the task set is schedulable by MC-DP-Fair with Def. 4.2, we know that Eq. (4.2) holds

by Lemma 4.2. Eq. (4.2) is rewritten to:

$$\begin{aligned}
\sum_{\tau_i \in \tau} \delta_i^L \leq m \Leftrightarrow & \sum_{\tau_i \in \tau_L} C_i^L/T_i + \sum_{\tau_i \in \tau_H} C_i^L/V_i \leq m \\
\Leftrightarrow & \; U_L^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L \cdot \theta_i^{H*}}{\theta_i^{H*} - u_i^H + u_i^L} \leq m \\
\Leftrightarrow & \; U_L^L + \sum_{\tau_i \in \tau_H} \left(u_i^L + \frac{u_i^L (u_i^H - u_i^L)}{\theta_i^{H*} - u_i^H + u_i^L} \right) \leq m \\
\Leftrightarrow & \; U_L^L + U_H^L + \sum_{\tau_i \in \tau_H} \frac{u_i^L (u_i^H - u_i^L)}{\theta_i^{H*} - u_i^H + u_i^L} \leq m,
\end{aligned}$$

which is Eq. (3.12).

($\Leftarrow$) To show that the task set is schedulable, we need to show that Eqs. (4.2) and (4.3) hold by Lemma 4.2.

Since the task set is feasible, Eq. (3.12) with θ_i^{H*} holds by Theorem 3.2. We already showed that Eq. (3.12) is Eq. (4.2) in Case $\Rightarrow$.

Since the feasible task set is scheduled by MC-DP-Fair with Def. 4.2, we have $\delta_i^H \leq \theta_i^{H*}$ for each task $\tau_i \in \tau_H$ by Lemma 4.1. We show that Eq. (4.3) holds:

$$\sum_{\tau_i \in \tau_H} \delta_i^H \leq \sum_{\tau_i \in \tau_H} \theta_i^{H*} \leq m,$$

which is true because the optimal assignment satisfies CON1 in Def. 3.4 with $\theta_i^{H*} = X_i + u_i^H$. □

4.3. The MC-Discrete Scheduling Algorithm and Schedulability Analysis

We consider integer virtual deadlines, which are preferable for real hardware platforms, compared to real number virtual deadlines. We present the revised algorithm, called MC-Discrete, as follows:

Definition 4.3. *MC-Discrete is defined with per-task virtual deadlines and a special DP* Γ*, which is the earliest DP after mode-switch. For each task* τ_i*, we define a VD:* $V_i \in \mathbb{Z}$ *s.t.* $0 < V_i \leq T_i$*. Other rules are the same as MC-DP-Fair.*

As an initial step toward a discrete-time schedule, we discretize the time point of mode-switch of MC-Discrete (Γ is an integer time point because it is a real deadline or a virtual deadline of a task).

Virtual Deadline Assignment. We also revise the VD assignment which only allows integer VDs.

Definition 4.4 (MC-Discrete VD assignment)**.** *We assign* $V_i := T_i$ *for an LO-task* $\tau_i \in \tau_L$ *and* $V_i := V_i^*$ *where* $V_i^* = \lfloor C_i^L / \theta_i^{L*} \rfloor$ *for a HI-task* $\tau_i \in \tau_H$ *where* θ_i^{L*} *is the value of* θ_i^L *in the optimal assignment for MCF.*

Schedulability Analysis. MC-DP-Fair transforms the fluid schedule of MCF into a non-fluid schedule without any schedulability loss. Since we allow only multiples of the scheduling quantum (integer) in VDs, MC-DP-Fair incurs slight overheads compared to MC-DP-Fair. Since the schedulability analysis of MC-DP-Fair is not applicable, we derive sufficient schedulability conditions for MC-Discrete.

Theorem 4.2. *An MC task set* τ *is schedulable by MC-Discrete with the VD assignment by Def. 4.4 if*

$$\sum_{\tau_i \in \tau} \theta_i^{L\triangle} \leq m. \tag{4.4}$$

Proof. To show that the task set is schedulable, we need to show schedulability in LO-mode and schedulability in HI-mode.

Consider LO-mode. Since Eq. (4.4) ia assumed and $\delta_i^L \leq \theta_i^{L\triangle}$ by Lemma 4.1, we have $\sum_{\tau_i \in \tau} \delta_i^L \leq \sum_{\tau_i \in \tau} \theta_i^{L\triangle} \leq m$. Since $\sum_{\tau_i \in \tau} \delta_i^L \leq m$, the task set is schedulable in LO-mode by Lemma 2.3.

Consider HI-mode. Since $\theta_i^{L\triangle} \geq \theta_i^{L*}$ by the definition of $\theta_i^{L\triangle}$, we have $\theta_i^{H\triangle} \leq \theta_i^{H*}$. Since $\delta_i^H \leq \theta_i^{H\triangle}$ by Lemma 4.1 and we have

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

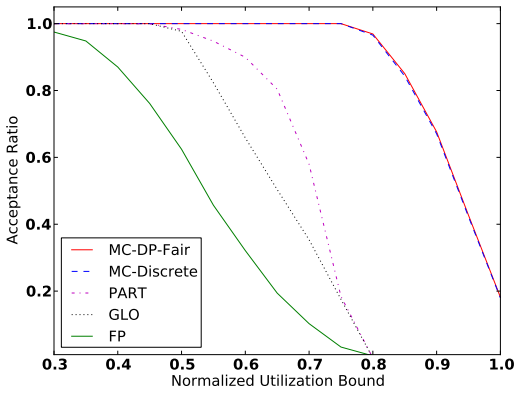
which is less than or equal to m because the optimal assignment satisfies CON1 in Def. 3.4. Since $\sum_{\tau_i \in \tau_H} \delta_i^H \leq m$, the task set is schedulable in HI-mode by Lemma 2.3. □

4.4. Evaluation

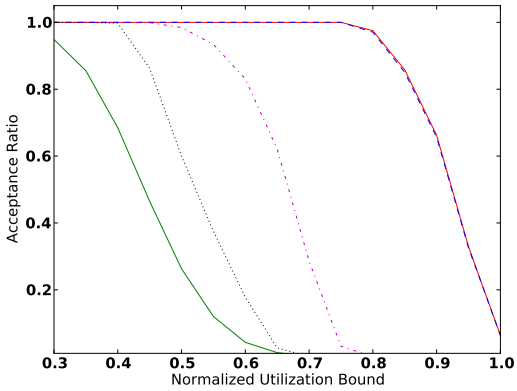
In this section we evaluate the schedulability of MC-DP-Fair (in Chapter 4.2) and MC-Discrete (in Chapter 4.3) using the randomly generated task sets and compare them with other multiprocessor MC scheduling algorithms. These algorithms include GLO [40] (a global scheduling approach based on EDF-VD), PART [11] (a partitioned scheduling approach based on EDF-VD) and FP [43] (a global fixed-priority scheduling approach). We first describe the task set generation procedure in the experimental setup section and then discuss the performance results of the algorithms.

Task Set Generation. We generate random task sets according to the workload-generation algorithm [40]. Let U^b be the upper bound of system utilization in both LO- and HI-mode. Input parameters are U^b, m (the number of processors), Z^b (the upper bound of task utilization), and P^c (the probability of task criticality). Initially, $m = 2$, $Z^b = 0.7$, and $P^c = 0.5$. We will also evaluate varying different input parameters. A random task is generated as follows (all task parameters are randomly drawn in uniform distribution):

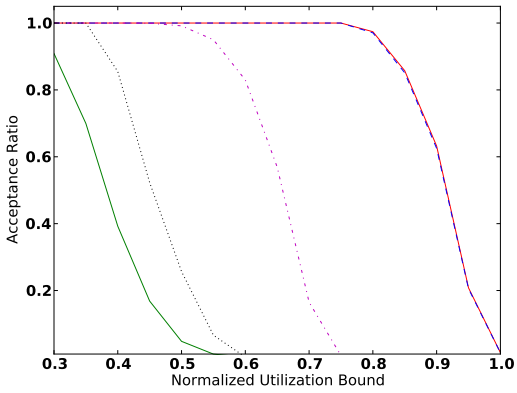
- u_i^L is a real number drawn from the range [0.02, Z^b].
- T_i is an integer drawn from the range [20, 300].
- R_i (the ratio of u_i^H/u_i^L) is a real number drawn from the range [1, 4].
- P_i (the probability that the task is a HI-task) is a real number from the range [0,1]. If



(a) m=2



(b) m=4



(c) m=8

Figure 6: The acceptance ratio with varying the normalized utilization bound (U^b/m) and the number of processors (m).

U^b/m / m	≤ 0.75	0.80	0.85	0.90	0.95	1.00
2	0	0.0035	0.0074	0.0057	0.0045	0.0019
4	0	0.0043	0.0071	0.0056	0.0039	0.0003
8	0	0.0033	0.0061	0.0069	0.0026	0.0004

Table 4: Acceptance ratio difference between MC-DP-Fair and MC-Discrete

$P_i < P^c$, set $\chi_i := LO$ and $C_i^L := \lfloor u_i^L \cdot T_i \rfloor$. Otherwise, set $\chi_i := HI$, $C_i^L := \lfloor u_i^L \cdot T_i \rfloor$, and $C_i^H := \lfloor u_i^L \cdot R_i \cdot T_i \rfloor$.

Repeat to generate a task in the task set until $\max(U_H^L + U_L^L,\ U_H^H)$ is larger than U^b. Then, discard the task added last.

Simulation Results. Fig. 6 shows the acceptance ratio (ratio of schedulable task sets) over varying $m \in \{2, 4, 8\}$ and normalized utilization bound U^b/m from 0.3 to 1.0 in increments of 0.05. Each data point is based on 10,000 task sets. The result shows that MC-Discrete outperforms previously known approaches. As shown in Table 4, MC-Discrete has a slightly lower acceptance ratio than MC-DP-Fair (the difference is no greater than 0.0074).

Fig. 7 and 8 show the effect of varying different parameters (P^c or Z^b). We use the weighted acceptance ratio [15] to reduce the number of dimensions in the plots. Let U_m^b be U^b/m and $A(U_m^b)$ be the acceptance ratio for U_m^b. The weighted acceptance ratio $W(S)$ is calculated to:

$$W(S) \stackrel{\text{def}}{=} \frac{\sum_{U_m^b \in S} \left(U_m^b \cdot A(U_m^b)\right)}{\sum_{U_m^b \in S} U_m^b},$$

where S is the set of U^b/m. The set S is the same as the one for Fig. 6 ($S = \{0.3, 0.35, \cdots, 1.0\}$). In Figs. 7 and 8, each data point is based on 15,000 task sets where 1,000 task sets are experimented for each U^b/m in S.

Fig. 7 shows the weighted acceptance ratio varying the upper bound of task utilization (Z^b). MC-Discrete and GLO are insensitive to Z^b while the performance of PART decreases as Z^b increases due to difficulty of scheduling large utilization tasks. On the other hand,

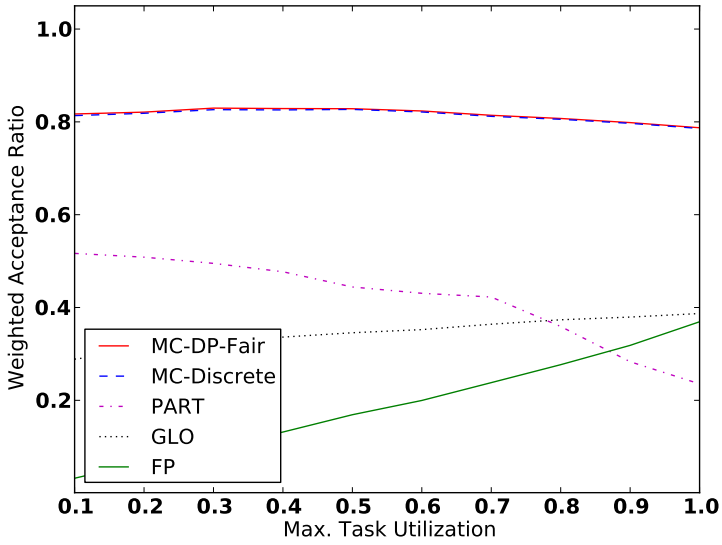


Figure 7: The weighted acceptance ratio with varying the upper bound of task utilization (Z^b).

the performance of FP increases as Z^b increases because interference-based analysis favors a smaller number of tasks[1].

Fig. 8 shows the weighted acceptance ratio varying the probability of task criticality (P^c). MC-DP-Fair and MC-Discrete can schedule all task sets when only LO-tasks or only HI-tasks are generated (i.e., $P^c = 0$ or $P^c = 1$) since MC-DP-Fair and MC-Discrete generalizing DP-Fair are optimal for the non-MC task model.

4.5. Summary

We presented the MC-DP-Fair and MC-Discrete scheduling algorithms, which is a variant of MC-Fluid for mapping fluid schedules into practical non-fluid schedules. While MC-DP-Fair allows mode switch at any time point, MC-Discrete allows mode switch only at integer time points, which is more practical for a realistic scheduler. We showed that our non-fluid MC-DP-Fair has the same schedulability as MC-Fluid on fluid scheduling model and MC-Discrete) are comparable to MC-Fluid. In simulation, we showed that MC-DP-Fair and MC-Discrete

[1]While FP uses interference-based analysis, all others use utilization-based analysis.

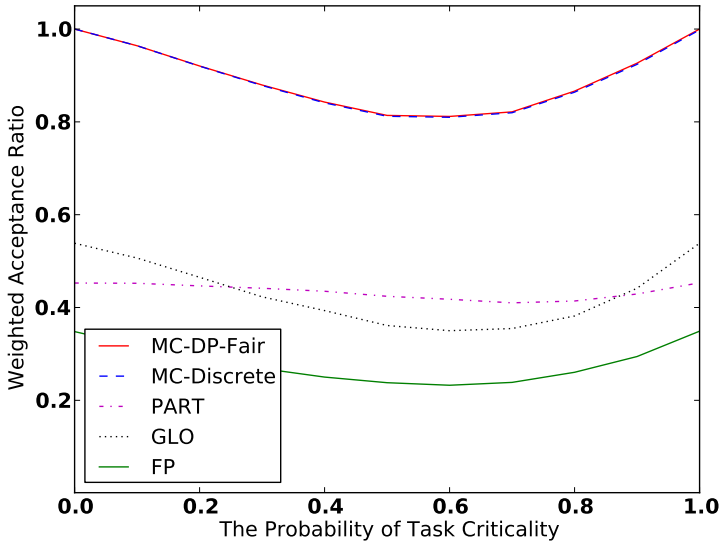


Figure 8: The weighted acceptance ratio with varying the probability of task criticality (P^c).

outperform all the existing multiprocessor MC scheduling algorithms.

CHAPTER 5 : Adaptive Mixed-Criticality Scheduling on Partitioned Multiprocessor Platforms

In the previous two chapters, we studied global scheduling approaches for multiprocessor MC systems. Although global MC scheduling approaches such as MC-Fluid and MC-Discrete are resource-efficient, their efficiency is limited on the platforms (e.g., large-scale systems) because implementation overheads can become high[1]. In this chapter, we focus on partitioned scheduling approaches, which assign tasks to processors statically and use a separate run-queue per processor. For adaptive MC scheduling on multiprocessors, we first present an adaptive MC scheduling algorithm for uniprocessor platforms and later extend it for partitioned multiprocessor platforms.

5.1. The Overview of the MC-ADAPT Scheduling Framework

Conventional MC scheduling approaches under system-level mode switch has the limitation that drop all LO-tasks at mode-switch. Recently, industrial demand is toward adaptive MC scheduling, which also considers the performance of LO-tasks [18]. In this chapter, we focus on adaptive MC scheduling.

Relaxing system-level mode switch assumptions, we consider *task-level mode switch*, where different tasks execute more than their LO-WCET at different times, independently from each other. Under task-level mode switch, some HI-tasks execute in the HI-mode while others remain in the LO-mode. This makes it possible to penalize some of the LO-tasks selectively in the event of mode switch, rather than all of them unnecessarily. Our goal is then to minimize the total number of LO-tasks to be dropped subject to the MC schedulability constraints. To achieve this, we seek to develop a new MC scheduling framework that dynamically determines which tasks to drop at runtime.

The task dropping decision under task-level mode switch is challenging since the system

[1]The overheads include preemption/migration overheads (in the case of MC-Discrete, preemption and migration happen at every time quantum in the worst case) and the overheads of global run-queue. The overhead could be severe for large-scale complex time-sensitive systems.

state is dynamically changing in the sense that a set of tasks in HI-mode (or LO-mode) as well as a set of tasks dropped (or active) change dynamically over time. In addition, when a HI-task requires additional resources due to its mode switch, the actual resources to be secured by dropping LO-tasks may vary depending on the current system state. There exist some recent studies [33, 46, 30] considering task dropping under task-level mode switch; however, all of them do not take the dynamic behavior of the system into full consideration since the tasks to be dropped are determined (and analyzed) at design time and then remain unchanged during runtime. Such a static decision has a considerable degree of pessimism, leading to unnecessary dropping of LO-tasks. This is because existing solutions cannot fully capture dynamic system states and incorporate it efficiently into the decision making of task dropping at runtime.

Our goal is adaptive MC scheduling under partitioned multiprocessor platforms. We first partition tasks into processors and schedule tasks within each processor by using a uniprocessor adaptive MC scheduling algorithm. Although existing work focused on partition algorithms for better schedulability, we would like to develop the multiprocessor scheduling technique for better performance of LO-tasks.

We have the following research questions to address these challenges.

- Q1. How can we analyze the impact of dynamic system state changes on the MC schedulability at runtime?
- Q2. How can we make adaptive decision on task dropping without sacrifice in the MC schedulability?
- Q3. How can we evaluate the quality of task dropping solution for MC scheduling algorithms?
- Q4. How can we schedule tasks for partitioned multiprocessor MC systems in order to reduce the dropping of LO-tasks?

We present a new MC scheduling framework, called *MC-ADAPT*, that makes online adaptive task dropping decisions according to dynamic system states under task-level mode switch. In particular, to address Q1, we develop a run-time schedulability analysis capable of capturing dynamic system states, which serves as a basis for online task dropping decisions. Our run-time analysis is efficient in the sense that it is sufficient to consider only the current system state without tracking the previous history of all system state changes when deciding which tasks to drop.

To address Q2, we design a new scheduling algorithm, called EDF-AD, by extending EDF-VD to support adaptive task dropping under task-level mode switch. EDF-AD utilizes the proposed run-time schedulability analysis to find a minimal set of LO-tasks to be dropped, so as to secure the additional resources requested by a mode-switching task at the current system state. We found that a straightforward extension of EDF-VD yields a counter-intuitive result – the extension with task-level mode switch would not dominate EDF-VD with system-level mode switch, while task-level mode switch is a generalization of system-level mode switch by definition. To handle this issue, we develop another scheduling algorithm, called EDF-AD-E, and its MC schedulability analysis that strictly dominates EDF-VD. EDF-AD-E identifies a subset of tasks triggering the scheduling anomaly and isolates them from other tasks.

To address Q3, we propose the speedup factor for task dropping. Although the conventional speedup factor for the MC scheduling problem is effective to evaluate MC scheduling algorithms under system-level mode switch, it cannot be used to evaluate the quality of task dropping under task-level mode switch. We apply the speedup factor for a different MC scheduling problem, called the *task dropping problem*, extending the MC scheduling problem into the optimization problem of task dropping under task-level mode switch. We derive that the speedup factor of MC-ADAPT for the task dropping problem is $\frac{1+\sqrt{5}}{2}$ (≈ 1.618). This implies that the MC-ADAPT can behave the same as the optimal scheduling framework with optimal task dropping if the processor is speeded up by a factor of 1.618. To the

best of our knowledge, this is the first work that quantifies task dropping performance of MC scheduling algorithm with the processor speedup factor [36]. In addition, we evaluate MC-ADAPT via simulation in terms of schedulability and resource utilization.

To address Q4, we adopt a semi-partitioned approach (allowing the migration of tasks in the limited cases), which is a variant of partitioned scheduling approaches. The existing partitioned MC scheduling approaches have focused on partition algorithms, whose performance is evaluated by the number of required processors. We will focus on the performance of LO-tasks, which is orthogonal to the existing work. A challenge is how to reduce the dropping of LO-tasks under semi-partitioned multiprocessor platforms. We propose to migrate LO-tasks that the scheduler decides to drop at mode-switch if there is an available processor.

In summary, we makes the following contributions in this chapter:

- We present the uniprocessor U-MC-ADAPT framework supporting online adaptive task dropping under task-level mode switch (Chapters 5.2, 5.3, and 5.4).

  - We propose new scheduling algorithms, called EDF-AD and EDF-AD-E, that drop a minimal set of LO-tasks based on run-time schedulability analysis (Chapters 5.2 and 5.3).

  - We propose the speedup factor for the task dropping problem and derive that the speedup factor of EDF-AD-E for the task dropping problem is 1.618 (Chapters 5.4).

- Based on U-MC-ADAPT, we propose a partitioned multiprocessor scheduling framework for MC-ADAPT (Chapter 5.5).

  - We develop the MC-ADAPT migration algorithm which moves the to-be-dropped LO-tasks into another processor at runtime mode switch instant in order to reduce the dropping of LO-tasks.

- Our simulation shows the effectiveness of our framework in terms of schedulability and

resource utilization under uniprocessor and multiprocessor platforms (Chapter 5.6).

5.2. The U-MC-ADAPT Framework on Uniprocessor Platforms

In this section, we present the U-MC-ADAPT framework that supports adaptive task dropping under task-level mode switch.

5.2.1. The Overview of the U-MC-ADAPT Framework

We introduce the U-MC-ADAPT scheduling framework that seeks to drop as few LO-tasks as possible under the MC-schedulability. The key features of U-MC-ADAPT include task-level mode switch and adaptive LO-task dropping. To enable such features and leverage them for achieving our goal raises several issues to address. We first need to design a new scheduling algorithm that supports task-level mode switch effectively. It is desirable to generalize or dominate the similar ones based on system-level mode switch. We then need to develop a method of task dropping that finds a minimal set of LO-tasks to drop while securing the additional resources requested by a mode-transiting task. This requires to analyze run-time variation on the resource demand of HI-tasks under task-level mode switch. When a task exhibits HI-behavior, the amount of additional resource demand for all HI-tasks to meet their deadlines can vary depending on a different runtime system state, i.e., a different combination of tasks in HI-mode, LO-mode, active state, and dropped state. This requires to calculate such resource demand precisely based on a runtime system state and determine which LO-tasks to be dropped so as to guarantee the required resources as well as minimize the number of dropped tasks at runtime in an efficient manner.

To develop the U-MC-ADAPT framework, we design a scheduling algorithm (Chapter 5.2.2) and its online analysis (Chapter 5.2.3) and its offline analysis (Chapter 5.2.4) building upon the principle of EDF-VD (Chapter 2.4.3). By identifying and analyzing the schedulability loss from EDF-VD, we enhance the scheduling algorithm to become a generalization of EDF-VD (Chapter 5.3).

5.2.2. The EDF-AD Scheduling Algorithm

To minimize the dropping of LO-tasks, we propose the EDF-AD (EDF-Adaptive task Dropping) algorithm, which consists of a resource-efficient MC scheduling algorithm and task dropping algorithm to choose a minimal set of LO-tasks for additional resource requested by a mode-switching task.

Runtime Scheduling Policy. To guarantee the schedulability of HI-tasks after mode switch, we apply VDs to HI-tasks in their LO-mode. EDF-AD adopts the same VD assignment[2] as EDF-VD, but changes the priorities of tasks according to task level mode switch. EDF-AD schedules the job with the earliest effective deadline and operates under the following rules:

- Schedule LO-tasks with their real deadlines.
- For each HI-task τ_i, schedule the task with its VD in its LO-mode ($M_i = \mathsf{LO}$) and with its real deadline otherwise.
- At the mode switch of a HI-task τ_i, set $M_i := \mathsf{HI}$ and drop LO-tasks[3] by the EDF-AD task dropping algorithm.

If the task mode of a HI-task is changed to HI at the mode switch, the relative deadline of the task is postponed from its VD to its real deadline.

Task Dropping Algorithm. To drop as few LO-tasks as possible at mode switch, we need to know how many resources are required to satisfy MC-schedulability. To do it, we develop an *online schedulability test* and drop LO-tasks by the test.

To construct such a test, we introduce *system state* that captures the dynamic system

[2]The VD coefficient is derived from the schedulability analysis for the initial system state (all HI-tasks are in LO-mode and all LO-tasks are active), which is identical for both EDF-VD and EDF-AD.

[3]If a task is dropped by the scheduler, then the currently-released job of the task is immediately stopped (not guaranteed to meet its deadline) and no further job of the task is released. If a task is active (not dropped), all jobs released by the task meet their deadlines.

behavior at mode switch, including the task mode (execution state) of each task.

Definition 5.1 (System state)**.** *For a given task set τ, a system state S is defined as four tuple of disjoint sets: $S = (\tau_{H1}, \tau_{H2}, \tau_{L1}, \tau_{L2})$ where*

- *τ_{H1} : the LO-mode HI-task set ($\tau_{H1} = \{\tau_i \in \tau_H | M_i = \mathsf{LO}\}$),*
- *τ_{H2}: the HI-mode HI-task set (including the mode switching task) ($\tau_{H2} = \tau_H \setminus \tau_{H1}$),*
- *τ_{L1}: the active LO-task set, and*
- *τ_{L2}: the dropped LO-task set (including the dropping LO-tasks at mode switch) ($\tau_{L2} = \tau_L \setminus \tau_{L1}$).*

The initial system state is $S_0 = (\tau_H, \emptyset, \tau_L, \emptyset)$.

We present the EDF-AD task dropping algorithm as follows:

- Before system start, sort LO-tasks in decreasing order of their task utilization.
- Drop the LO-task with the highest utilization among the active LO-task set (τ_{L1}) until the dropped LO-task set (τ_{L2}) satisfies the online schedulability test (Eq. (5.1)). This algorithm minimizes the number of the dropping tasks during the entire running time, which is optimal with respect to the online schedulability test.

We present the online schedulability test to determine which LO-tasks should be dropped at mode-switch:

$$U_{L2}^{L} \geq \frac{U_{L1}^{L} + U_{H1}^{L}/x + U_{H2}^{H} + U_{L}^{L} - 1}{1 - x}. \tag{5.1}$$

Its correctness is presented in Section 5.2.3.

The runtime complexity of U-MC-ADAPT is $O(n)$, a linear complexity. The task dropping algorithm takes $O(n)$: identifying resource deficiency takes $O(n)$ and selecting the drop candidates of LO-tasks (sorting LO-tasks is done offline) takes $O(n)$. The required memory

space for the system state is at most n bits to store criticality-modes of HI-tasks.

5.2.3. Online Schedulability Analysis

We analyze online schedulability at a specific mode switch, which means whether a given task set is schedulable by EDF-AD when the system state at mode switch is given. Let S_k be the system state after k-th mode switch ($k \geq 1$). To find the collective resource demand on a given interval, we compute the resource demand of each task depending on its task mode (execution state).

We consider online schedulability on two different kinds of system states: the initial system state (S_0) and the system state (S_k) that is switched from any feasible system state (S_{k-1}). Since S_0 is the same as system LO-mode in EDF-VD, we can reuse the result of EDF-VD for online schedulability on S_0.

Lemma 5.1. *A task set τ is schedulable by EDF-AD on S_0 if*

$$U_L^L + \frac{U_H^L}{x} \leq 1. \tag{5.2}$$

Proof. It is immediate from Lemma 2.4. □

Next, consider online schedulability on S_k.

Theorem 5.1. *Consider a task set τ. Assume that the task set is schedulable with S_{k-1}. Let S_k be the system state transited from S_{k-1}. Then, τ is schedulable by EDF-AD on S_k if*

$$U_{L1}^L + \frac{U_{H1}^L}{x} + xU_{L2}^L + U_{H2}^H \leq 1. \tag{5.3}$$

■

We can derive the online schedulability test (Eq. (5.1)) from Theorem 5.1. Eq. (5.3) is

rewritten to

$$
\begin{aligned}
& (U_L^L - U_{L2}^L) + \frac{U_{H1}^L}{x} + xU_{L2}^L + U_{H2}^H \leq 1 && (\because U_L^L = U_{L1}^L + U_{L2}^L) \\
\Leftrightarrow\ & U_L^L + \frac{U_{H1}^L}{x} + U_{H2}^H - 1 \leq U_{L2}^L - xU_{L2}^L,
\end{aligned}
$$

which is Eq. (5.1).

Now, we present the proof strategy for Theorem 5.1 and present auxiliary lemmas for the proof. We prove it by contradiction. Suppose that a deadline is missed. Let I denote a *minimal*[4] instance of jobs released by τ on which a deadline is missed by EDF-AD. Without loss of generality, we assume that the first job in I is released at time 0 and the deadline miss occurs at time t_1[5]. For task τ_i and any time t, let $\mathsf{DEM_i}(t)$ be an upper bound of the demand[6] of task τ_i over time interval $[0, t)$ in I. Let $\mathsf{DEM}(t)$ be the sum of $\mathsf{DEM_i}(t)$ of all the tasks in τ. Since a deadline is missed at t_1, we have $\mathsf{DEM}(t_1) > t_1$. We will show that our calculation of $\mathsf{DEM}(t_1)$ is no greater than t_1, which leads to a contradiction.

To find $\mathsf{DEM}(t_1)$, we consider individual task demand over $[0, t_1)$. The following lemma bounds the demand for a HI-task in LO-mode and the demand of a LO-task in the active state.

Lemma 5.2. *Consider any time t. (a) For a HI-task in LO-mode ($\tau_i \in \tau_{H1}$), $\mathsf{DEM_i}(t) = (u_i^L/x)t$, and (b) for a LO-task in the active state ($\tau_i \in \tau_{L1}$), $\mathsf{DEM_i}(t) = u_i^L \cdot t$.*

Proof. **(a)** The task demand over $[0, t)$ is smaller than or equal to $C_i^L \cdot t/(xT_i)$. Thus, $\mathsf{DEM_i}(t) = (u_i^L/x)t$.

(b) The task demand over $[0, t)$ is smaller than or equal to $C_i^L \cdot t/T_i$. Thus, $\mathsf{DEM_i}(t_1) = u_i^L \cdot t_1$. □

[4]Since I is minimal, EDF-AD can schedule any proper subset of I.

[5]All jobs in I are necessary to construct the deadline miss. Otherwise, the unnecessary job can be removed from I, which contradicts the minimality of I.

[6]The demand of a task for a time interval indicates the worst-case resource demand to meet deadlines of jobs released by the tasks for the time interval [14].

For the demand for a HI-task in HI-mode and a LO-task in the dropped state, we utilize a characteristic of the jobs that are included in $\mathsf{DEM}(t_1)$.

Lemma 5.3 (from [4])**.** *Consider the minimal instance I. All jobs that execute in $[0, t_1)$ have deadline $\leq t_1$.*

Based on Lemma 5.3, we bound the demand of a HI-task in HI-mode. For a HI-task τ_i, let J_i^* be the mode-switching job of τ_i in I and a_i^* be the release time of J_i^*.

Lemma 5.4. *If HI-task τ_i is mode switched ($\tau_i \in \tau_{H2}$), then*

$$\mathsf{DEM_i}(t_1) = \begin{cases} (u_i^L/x)t_1 & \text{if } t_1 < a_i^* + xT_i, \\ u_i^L \cdot a_i^* + u_i^H(t_1 - a_i^*) & \text{otherwise.} \end{cases}$$

Proof. At mode switch of τ_i, the deadline of J_i^* is changed from $a_i^* + xT_i$ to $a_i^* + T_i$ and the execution requirement is changed from C_i^L to C_i^H. By Lemma 5.3, the changed demand is not considered when $t_1 < a_i^* + T_i$. The task demand over $[0, t_1)$ is different depending on whether time t_1 is before the VD of J_i^* or not.

Case $(t_1 < a_i^* + xT_i)$**.** Since $t_1 < a_i^* + xT_i \leq a_i^* + T_i$, the task demand is the same as Lemma 5.2a.

Case $(t_1 \geq a_i^* + xT_i)$**.** We calculate the task demand of jobs before time a_i^* and the task demand after time a_i^*. Since jobs before time a_i^* execute for LO-WCET (C_i^L), the task demand of jobs before a_i^* is $C_i^L \cdot a_i^*/T_i$. Since job J_i^* and its successive jobs execute HI-WCET (C_i^H), the task demand of jobs after time a_i^* is $C_i^H(t_1 - a_i^*)/T_i$. Then, $\mathsf{DEM_i}(t_1) = C_i^L \cdot a_i^*/T_i + C_i^H(t_1 - a_i^*)/T_i$. □

The following lemma bounds the demand of a LO-task in the dropped state.

Lemma 5.5. *Let τ_q be the last (k-th) mode-switched task in I on S_k. If LO-task τ_i is*

dropped ($\tau_i \in \tau_{L2}$), then

$$\mathsf{DEM_i}(t_1) = \begin{cases} u_i^L \cdot t_1 & \text{if } t_1 < a_q^* + xT_q, \\ u_i^L(a_q^* + xT_q) & \text{otherwise.} \end{cases}$$

Proof. Since τ_q is the last mode-switched task, τ_i is dropped before or at the mode switch of J_q^*. The mode switch of J_q^* happens before or at its VD ($a_q^* + xT_q$). The task demand of τ_i over $[0, t_1)$ is different depending on whether time t_1 is before the VD of J_q^* or not.

Case ($t_1 < a_q^* + xT_q$)**.** In the worst case, the mode switch happens at the VD of J_q^*. Then, the upper bound of the demand is the same as Lemma 5.2b.

Case ($t_1 \geq a_q^* + xT_q$)**.** No job of the task executes after the mode switch of J_q^*. To execute before the mode switch, jobs must have a deadline no greater than the VD of J_q^*. Then, $\mathsf{DEM_i}(t_1) = C_i^L(a_q^* + xT_q)/T_i$. □

We consider the task demand on S_k based on the task demand on S_{k-1}. We know that S_{k-1} is a feasible system state and S_k is the transited state from S_{k-1} by the mode switch of HI-task τ_q. Let $\mathsf{DEM_i^k}(t)$ be the demand of task τ_i over $[0, t)$ on the system state S_k. We compute $\mathsf{DEM_i^k}(t_1)$ based on $\mathsf{DEM_i^{k-1}}(t_1)$ for task τ_i depending on whether time t_1 is before the VD of J_q^* or not (Lemmas 5.6 and 5.7).

Lemma 5.6. *If $t_1 < a_q^* + xT_q$, then $\mathsf{DEM_i^k}(t_1) = \mathsf{DEM_i^{k-1}}(t_1)$.*

Proof. Task τ_q belongs to τ_{H2} when the system state is S_k and belongs to τ_{H1} when the system state is S_{k-1}. Since $\tau_q \in \tau_{H2}$ on S_k, we have $\mathsf{DEM_i^k}(t_1) = (u_i^L/x)t_1$ by Lemma 5.4. Since $\tau_q \in \tau_{H1}$ on S_{k-1}, we have $\mathsf{DEM_i^{k-1}}(t_1) = (u_i^L/x)t_1$ by Lemma 5.2a.

Consider task τ_i that is dropped by τ_q. The task belongs to τ_{L2} when the system state is S_k and belongs to τ_{L1} when the system state is S_{k-1}. Since $\tau_i \in \tau_{L2}$ on S_k, we have $\mathsf{DEM_i^k}(t_1) = u_i^L \cdot t_1$ by Lemma 5.5. Since $\tau_q \in \tau_{L1}$ on S_{k-1}, we have $\mathsf{DEM_i^{k-1}}(t_1) = u_i^L \cdot t_1$ by Lemma 5.2b. Consider task $\tau_i \in \tau$ that is not t_q and not dropped by τ_q. Since its task

mode (execution state) is not changed from S_{k-1} to S_k, we have $\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t) = \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(t)$ for any t. □

Lemma 5.7. *If $t_1 \geq a_q^* + xT_q$, then*

$$\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) \leq \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*) \begin{cases} u_i^L/x & \text{if } \tau_i \in \tau_{H1}, \\ u_i^L & \text{if } \tau_i \in \tau_{L1}, \\ u_i^H & \text{if } \tau_i \in \tau_{H2}, \\ x \cdot u_i^L & \text{if } \tau_i \in \tau_{L2}. \end{cases}$$

Proof. For $\tau_i \in \tau_{H1}$, we have $\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) = \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*)u_i^L/x$ by Lemma 5.2a. For $\tau_i \in \tau_{L1}$, we have $\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) = \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*)u_i^L$ by Lemma 5.2b.

Consider $\tau_i \in \tau_{H2}$. If $a_i^* \leq a_q^*$, we have $\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) = \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t - a_q^*)u_i^H$ by Lemma 5.4. Otherwise, we have

$$\begin{aligned} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) =& \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (a_i^* - a_q^*)u_i^L + (t_1 - a_i^*)u_i^H \\ & \qquad \text{(by Lemma 5.4)} \\ \leq& \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*)u_i^H. \end{aligned}$$

Consider $\tau_i \in \tau_{L2}$. By Lemma 5.3, the deadline of J_q^* is no greater than t_1: $a_q^* + T_q \leq t_1$. By Lemma 5.5, we have

$$\begin{aligned} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) =& (a_q^* + xT_q)u_i^L \\ \leq& \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + x(t_1 - a_q^*)u_i^L \qquad (\because T_q \leq t_1 - a_q^*). \end{aligned}$$

□

Based on the relation between the demand on S_k and the demand on S_{k-1}, we now prove

Theorem 5.1.

Proof of Theorem 5.1. We summarize the proof strategy stated before. Let $\mathsf{DEM}^{\mathsf{k}}(t)$ be the sum of $\mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t)$ of all the tasks in τ. Since S_{k-1} is a feasible system state, we have $\mathsf{DEM}^{\mathsf{k-1}}(t) \leq t$ for any t. By proof by contradiction, we assume a deadline misses in I. Then, we have $\mathsf{DEM}^{\mathsf{k}}(t_1) > t_1$. To lead to a contradiction, we only need to show that $\mathsf{DEM}^{\mathsf{k}}(t_1) \leq t_1$. Let τ_q be the last mode-switched task in I. $\mathsf{DEM}^{\mathsf{k}}(t_1)$ is different depending on whether t_1 is before the VD of J_q^* or not.

Case 1 ($t_1 < a_q^* + xT_q$)**.** We calculate $\mathsf{DEM}^{\mathsf{k}}(t_1)$:

$$\begin{aligned}
\mathsf{DEM}^{\mathsf{k}}(t_1) &= \sum_{\tau_i \in \tau} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) \\
&= \sum_{\tau_i \in \tau} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(t_1) && \textit{(by Lemma 5.6)} \\
&= \mathsf{DEM}^{\mathsf{k-1}}(t_1),
\end{aligned}$$

which is smaller than or equal to 1 by the assumption on $\mathsf{DEM}^{\mathsf{k-1}}$.

Case 2 ($t_1 \geq a_q^* + xT_q$)**.** We calculate $\mathsf{DEM}^{\mathsf{k}}(t_1)$:

$$\begin{aligned}
\mathsf{DEM}^{\mathsf{k}}(t_1) &= \sum_{\tau_i \in \tau} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k}}(t_1) \\
&\leq \sum_{\tau_i \in \tau} \mathsf{DEM}_{\mathsf{i}}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*)(U_{L1}^L + \frac{U_{H1}^L}{x} \\
&\quad + xU_{L2}^L + U_{H2}^H) && \textit{(by Lemma 5.7)} \\
&\leq \mathsf{DEM}^{\mathsf{k-1}}(a_q^*) + (t_1 - a_q^*) \quad \textit{(Eq. (5.3) with } S_k\textit{)} \\
&\leq a_q^* + (t_1 - a_q^*) \quad \textit{(by the assumption on } \mathsf{DEM}^{\mathsf{k-1}}\textit{)} \\
&= t_1.
\end{aligned}$$

From Cases 1 and 2, we showed that $\mathsf{DEM}^{\mathsf{k}}(t_1) \leq t_1$. □

Task	χ_i	u_i^L	u_i^H
τ_1	HI	0.10	0.35
τ_2	HI	0.20	0.30
τ_3	LO	0.18	N/A
τ_4	LO	0.12	N/A
τ_5	LO	0.10	N/A

Table 5: The Parameters of an Example Task Set

5.2.4. Offline Schedulability Analysis

We looked at online schedulability analysis at a mode switch. However, we do not yet know whether a task set is schedulable by EDF-AD under any sequences of mode switches, which is offline schedulability. To know it, we need to check whether EDF-AD can schedule the task set on any system state satisfying MC-schedulability (Condition A and B in Chapter 2.2), based on the online schedulability analysis (Lemma 5.1 and Theorem 5.1).

Theorem 5.2. *A task set τ is MC-schedulable by EDF-AD if*

$$U_L^L + \frac{U_H^L}{x} \leq 1, \tag{5.4}$$

$$xU_L^L + \sum_{\tau_i \in \tau_H} \max(\frac{u_i^L}{x}, u_i^H) \leq 1. \tag{5.5}$$

Proof. To show that τ is MC-schedulable, we need to satisfy both Conditions A and B. Since Eq. (5.4) holds, by Lemma 5.1, τ is schedulable on S_0, which satisfies Condition B. For Condition A, by Theorem 5.1, we show that Eq. (5.3) holds with any $\tau_{H2} \neq \emptyset$: since each HI-task is either LO-mode or HI-mode, and all LO-tasks may be dropped in the worst case,

$$xU_L^L + \sum_{\tau_i \in \tau_H} \max(\frac{u_i^L}{x}, u_i^H) \leq 1,$$

which holds from Eq. (5.5). □

We show an example task set schedulable by EDF-AD.

Example 5.1. *Consider the example task set in Table 5. According to the VD assignment, we have* $x = 0.3/(1-0.4) = 0.5$. *We show that Eqs.* (5.4) *and* (5.5) *hold:* $0.4 + 0.3/0.5 = 1$ *and* $0.5 * 0.4 + \max(0.1/0.5, 0.35) + \max(0.2/0.5, 0.30) = 0.2 + 0.35 + 0.4 = 0.95 \leq 1$. *Then, the task set is MC-schedulable by Theorem 5.2.*

5.3. An Enhanced U-MC-ADAPT Framework on Uniprocessor Platforms

A straightforward extension of EDF-VD, which is EDF-AD, yields a counter-intuitive result that it does not dominate EDF-VD in schedulability. In Chapter 5.3.1, we find the characteristics of the subset of tasks that cause the schedulability loss. In Chapter 5.3.2, we present a new scheduling algorithm that isolates them from the other tasks.

5.3.1. The Schedulability Loss of EDF-AD

Let's look at an example schedulable by EDF-VD but not by EDF-AD.

Example 5.2. *We modify the task set in Table 5 by changing* u_1^H *of* τ_1 *to* 0.45. *Since* U_L^L *and* U_H^L *are not changed,* x *will not be changed. The task set is schedulable by EDF-VD because Lemmas 2.4 and 2.5 hold:* $0.4 + 0.3/0.5 = 1$ *and* $0.5 \cdot 0.4 + 0.75 = 0.95 \leq 1$. *However, the task set is not schedulable by EDF-AD because Eq.* (5.5) *in Theorem 5.2 does not hold:* $0.5 \cdot 0.4 + \max(0.1/0.5, 0.45) + \max(0.2/0.5, 0.30) = 1.05 > 1$.

We investigate which difference between EDF-VD and EDF-AD causes the schedulability loss. When checking Condition B in MC-schedulability, both EDF-VD and EDF-AD consider the initial system state S_0. Thus, the loss is related to check Condition A. We define *the critical task mode* as the combination of task modes for HI-tasks where the collective resource demand of HI-tasks is maximized. While the critical task mode for EDF-VD is system HI-mode (among system HI-mode and system LO-mode), the one for EDF-AD is not system HI-mode (all HI-tasks are in HI-mode). Since a HI-task executes with its VD in LO-mode, there may exist a HI-tasks whose resource utilization in LO-mode (C_i^L/V_i) is higher than the one in HI-mode (C_i^H/T_i). Thus, the critical task mode is the combination of the task mode of each HI-task where its resource utilization is maximized. However, EDF-VD adopting

the system-level mode switch (from system LO-mode to system HI-mode) is irrelevant to the critical task mode of EDF-AD.

We investigate why some HI-task has higher resource utilization in LO-mode. All HI-tasks execute with their VD ($V_i = xT_i$) in their LO-mode and the VD coefficient is derived from the collective utilization of the task set such that $U_H^L/x \leq U_H^H$. However, the VD assignment may not be the best choice for an individual task, specially for the HI-task that has relatively small difference between HI-WCET and LO-WCET. Then, the task has $u_i^L/x > u_i^H$. From the understanding of the schedulability loss, we will present a resolution in the next subsection.

5.3.2. The EDF-AD-E Scheduling Algorithm

Since a fully-independent task-level mode switch may produce the schedulability loss, we apply a limited mode switch not to sacrifice schedulability. We propose another scheduling algorithm, called *EDF-AD-E*.

Scheduling Algorithm. We formally define the subset of HI-tasks that produces the schedulability loss.

Definition 5.2. *HI-mode-preferred tasks (τ_F) are defined as a set of HI-tasks s.t. $C_i^L/V_i > C_i^H/T_i$:* $\tau_F = \{\tau_i \in \tau_H \mid u_i^L/x > u_i^H\}$

The schedulability loss may happen when HI-mode-preferred tasks remain in LO-mode and the other tasks have mode-switched. In addition, since a HI-mode-preferred task has resource utilization in HI-mode lower than the one in LO-mode, it is better for the task to execute in HI-mode from system start. We now present the EDF-AD-E (Enhanced) algorithm as follows:

- The VD of each HI-task τ_i is assigned by $V_i = xT_i$ where $x = \min(1, (1 - U_H^H)/U_L^L)$.
- For HI-mode-preferred tasks, execute them in HI-mode from system start.
- All the other runtime scheduling policies (including the task dropping algorithm) are

the same as EDF-AD.

We cannot use the VD coefficient in EDF-AD because the offline schedulability of EDF-AD-E is different from EDF-AD. We compute the VD coefficient from EDF-AD-E offline schedulability (Theorem 5.4). The initial system state of EDF-AD-E is different from EDF-AD: $S_0 = (\tau_H \setminus \tau_F, \tau_F, \tau_L, \emptyset)$.

Online Schedulability Analysis. Since the EDF-AD-E scheduling algorithm is modified from EDF-AD, we need to check whether online schedulability analysis of EDF-AD is also applicable to EDF-AD-E. Since S_0 is different from EDF-AD, we re-derive online schedulability on S_0.

Lemma 5.8. *A task set τ is schedulable by EDF-AD-E on S_0 if*

$$U_L^L + \sum_{\tau_i \in \tau_H} \min(\frac{u_i^L}{x}, u_i^H) \leq 1. \tag{5.6}$$

Proof. On S_0, we have $\tau_{H1} = \tau_H \setminus \tau_F$, $\tau_{H2} = \tau_F$ and $\tau_{L1} = \tau_L$. Since the demand of task $\tau_i \in \tau_F$ over $[0, t)$ is no greater than $C_i^H \cdot t/T_i$, we have

$$\tau_i \in \tau_F, \mathsf{DEM_i}(t) = u_i^H \cdot t. \tag{5.7}$$

We show that the demand over $[0, t)$ is no greater than t:

$$
\begin{aligned}
& \mathsf{DEM}(t_1) \\
= & \sum_{\tau_i \in \tau_H \setminus \tau_F} \mathsf{DEM_i}(t) + \sum_{\tau_i \in \tau_L} \mathsf{DEM_i}(t) + \sum_{\tau_i \in \tau_F} \mathsf{DEM_i}(t) \\
= & \Big(\sum_{\tau_i \in \tau_H \setminus \tau_F} u_i^L / x + \sum_{\tau_i \in \tau_{L1}} u_i^L \Big) t + \sum_{\tau_i \in \tau_F} \mathsf{DEM_i}(t) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{(by Lemma 5.2)} \\
= & \Big(\sum_{\tau_i \in \tau_H \setminus \tau_F} u_i^L / x + U_L^L \Big) t + \sum_{\tau_i \in \tau_F} u_i^H \cdot t \qquad\qquad \textit{(by Eq. (5.7))} \\
= & \Big(U_L^L + \sum_{\tau_i \in \tau_H \setminus \tau_F} u_i^L / x + \sum_{\tau_i \in \tau_F} u_i^H \Big) t \\
\leq & \ 1 \cdot t \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{(by assumption)}
\end{aligned}
$$

□

Online schedulability on S_k is the same as EDF-AD.

Theorem 5.3. *Consider a task set τ. Assume that the task set is schedulable with S_{k-1}. Let S_k be the system state that is transited from S_{k-1}. Then, the task set is schedulable by EDF-AD-E on S_k if Eq. (5.3) holds.*

Proof. The proof is the same as Theorem 5.1. □

Offline Schedulability Analysis. The following theorem derives the offline schedulability of EDF-AD-E.

Theorem 5.4. *A task set τ is MC-schedulable by EDF-AD-E if*

$$U_L^L + \sum_{\tau_i \in \tau_H} \min\left(\frac{u_i^L}{x}, u_i^H\right) \leq 1, \tag{5.8}$$

$$x U_L^L + U_H^H \leq 1. \tag{5.9}$$

Proof. To show that τ is MC-schedulable, we need to satisfy both Conditions A and B in MC-schedulability. Since Eq. (5.8) holds, by Lemma 5.8, τ is schedulable on S_0, which satisfies Condition B. For Condition A, by Theorem 5.3, we show that Eq. (5.3) holds with any $\tau_{H2} \neq \emptyset$: since each HI-task except HI-mode-preferred tasks is LO-mode or HI-mode, and all LO-tasks may be dropped in the worst case,

$$\begin{aligned} & xU_L^L + \sum_{\tau_i \in \tau_H \setminus \tau_F} \max(\frac{u_i^L}{x}, u_i^H) + \sum_{\tau_i \in \tau_F} u_i^H \leq 1 \\ \Leftrightarrow\ & xU_L^L + \sum_{\tau_i \in \tau_H \setminus \tau_F} u_i^H + U_F^H \leq 1 \qquad \textit{(by Def. 5.2)} \\ \Leftrightarrow\ & xU_L^L + U_H^H - U_F^H + U_F^H \leq 1, \end{aligned}$$

which holds from Eq. (5.9). □

Properties. EDF-AD-E strictly dominates EDF-VD in terms of MC-schedulability (Lemma 5.9 and Example 5.3).

Lemma 5.9. *If any task set is MC-schedulable by EDF-VD, the task set is also MC-schedulable by EDF-AD-E.*

Proof. Since the task set is MC-schedulable by EDF-VD, by Lemmas 2.4 and 2.5, Eqs. (2.1) and (2.2) hold. If Eqs. (5.8) and (5.9) holds, by Theorem 5.4, the task set is also MC-schedulable by EDF-AD-E. Eq. (5.8) holds: $U_L^L + \sum_{\tau_i \in \tau_H} \min(\frac{u_i^L}{x}, u_i^H) \leq U_L^L + \frac{U_H^L}{x} \leq 1$ from Eq. (2.1). Eq. (5.9) holds from Eq. (2.2). □

Example 5.3. *We modify the task set in Table 5 by changing* u_1^H *of* τ_1 *to* 0.55. *We will schedule the task set by EDF-VD. Since* U_L^L *and* U_H^L *are not changed,* x *will not be changed. The task set is not schedulable by EDF-VD because Eq.* (2.2) *in Lemma 2.5 does not hold:* $0.5 * 0.4 + 0.85 = 1.05 > 1$. *We will schedule the task set by EDF-AD-E. By the VD assignment, we have* $x = (1 - 0.85)/0.4 = 0.375$. *Task* τ_2 *is a HI-mode-preferred task because* $u_i^L/x = 0.2/0.375 = 0.53 > u_i^H = 0.30$. *We show that Eqs.* (5.8) *and* (5.9)

hold: $0.4 + 0.1/0.375 + 0.3 = 0.96 \leq 1$ *and* $0.375 * 0.4 + 0.85 = 1$. *Then, the task set is MC-schedulable by Theorem 5.4,*

5.4. The Speedup Factor

In this section, we quantify the effectiveness of EDF-AD-E based on the metric of processor speedup factor [36]. The speedup factor ($\alpha \in \mathbb{R}$ s.t. $\alpha \geq 1$) is a reliable performance metric for comparing the worst-case behavior of different algorithms for solving the same problem. The smaller speedup factor of an algorithm indicates that the behavior of the algorithm is closer to that of the optimal algorithm. Previously, the speedup factor for the MC scheduling problem is effective to evaluate MC scheduling algorithms (e.g., [4]). However, it only evaluates MC-schedulability, and cannot evaluate the quality of task dropping. So, we propose the speedup factor for the task dropping problem which extends the existing MC scheduling problem with the runtime performance of LO-tasks. First, we define the task dropping problem.

Problem 5.1 (The task dropping problem)**.** *For a given feasible MC task set τ, a subset of HI-tasks ($\tau_R \in \tau$), and scheduling algorithm $\mathbb{A}$, the task dropping problem is: if tasks in τ_R mode-switch on runtime, how many LO-tasks are required to be dropped for scheduling τ by scheduling algorithm $\mathbb{A}$?*

Now, we define the speedup factor of scheduling algorithm $\mathbb{A}$ for the task dropping problem

Definition 5.3. *Let OPT be the optimal clairvoyant scheduling algorithm with optimal task dropping*[7]*. The speedup factor of $\mathbb{A}$ for task drop is defined as the smallest real number α (≥ 1) such that the number of LO-tasks required to be dropped to schedule any given task set τ under any given mode switch sequence (specified by $\tau_R \in \tau$) by OPT on a speed-1 processor is the same as the one to schedule τ under the mode switch sequence by $\mathbb{A}$ on a speed-α processor.*

[7] In MC systems, a clairvoyant scheme is the one that knows the time instant of mode switch before runtime scheduling.

The speedup factor for the MC scheduling problem evaluates scheduling algorithms in terms of MC-schedulability. Similarly, the speedup factor for the task dropping problem evaluates scheduling algorithms in terms of the number of the required task dropping under any possible scheduling scenarios. Although the existing work cannot provide any performance guarantee on LO-tasks via the speedup factor, we propose the first work to evaluate how many LO-tasks can be scheduled after mode switch via the speedup factor.

Next, we evaluate EDF-AD-E via the proposed metric.

Theorem 5.5. *EDF-AD-E has a speedup factor of* $\frac{1+\sqrt{5}}{2}$ *for task drop.*

To prove Theorem 5.5, we present an auxiliary lemma.

Lemma 5.10. *Consider a task set* τ *and a subset of HI-tasks* $\tau_R \in \tau$*. Consider a scheduling scenario that tasks in* τ_R *mode-switches on runtime. EDF-AD-E can schedule* τ *under the scenario by dropping a subset of LO-tasks* $\tau_G \in \tau$ *if*

$$U_L^L + \frac{U_H^L}{x} \leq 1, \tag{5.10}$$

$$U_{L1}^L + \frac{U_{H1}^L}{x} + xU_{L2}^L + U_{H2}^H \leq 1 \tag{5.11}$$

where $\tau_{H2} = \tau_R$ *and* $\tau_{L2} = \tau_G$*.*

Proof. We need to show that τ is schedulable on S_0 and any legitimate S_k considering τ_R and τ_G. For schedulability on S_0, by Lemma 5.8, we need to satisfy Eq. (5.6):

$$U_L^L + \sum_{\tau_i \in \tau_H} \min(\frac{u_i^L}{x}, u_i^H) \leq 1,$$

which holds by $\sum_{\tau_i \in \tau_H} \min(\frac{u_i^L}{x}, u_i^H) \leq U_H^L$ and Eq. (5.10).

Consider any legitimate S_k considering τ_R and τ_G. Since each task in τ_R is either HI-mode or LO-mode, τ_{H2} in S_k is any subset of τ_R. In S_k, we set $\tau_{L2} := \tau_G$ because any LO-task in τ_G may be dropped. To show that the task set is schedulable with S_k, by Theorem 5.3, we

need to satisfy Eq. (5.3):

$$\begin{aligned} U_{L1}^L + xU_{L2}^L + \sum_{\tau_i \in \tau_{H1} \setminus \tau_F} \frac{u_i^L}{x} + \sum_{\tau_i \in \tau_{H2} \setminus \tau_F} \max(\frac{u_i^L}{x}, u_i^H) \\ + \sum_{\tau_i \in \tau_F} u_i^H \leq 1 \\ \Leftrightarrow \; U_{L1}^L + xU_{L2}^L + \sum_{\tau_i \in \tau_{H1}} \min(\frac{u_i^L}{x}, u_i^H) + U_{H2}^H \leq 1, \end{aligned}$$

which holds by $\sum_{\tau_i \in \tau_{H1}} \min(\frac{u_i^L}{x}, u_i^H) \leq U_{H1}^H$ and Eq. (5.11). □

Proof of Theorem 5.5. Consider a task set τ. Consider a scheduling scenario that tasks in a set of HI-tasks $\tau_R \in \tau$ mode-switch on runtime. We will prove that if τ is schedulable under the scenario by the optimal clairvoyant scheduling algorithm with dropping a set of LO-tasks $\tau_G \in \tau$ on a speed-1 processor, τ is also schedulable under the scenario by EDF-AD-E with dropping τ_G on a speed-$\frac{1+\sqrt{5}}{2}$ processor.

Let $\tau_{H1} := \tau_H \setminus \tau_R$, $\tau_{H2} := \tau_R$, $\tau_{L1} := \tau_L \setminus \tau_G$, and $\tau_{L2} := \tau_G$. Let b denote an upper bound on the utilization of τ on the initial state and the minimum utilization of τ on the worst-case task modes of HI-tasks:

$$\max(U_L^L + U_H^L, U_{L1}^L + U_{H1}^L + U_{H2}^H) \leq b. \tag{5.12}$$

To show that the task set is schedulable, by Lemma 5.10, it is required that Eqs. (5.10) and (5.11) hold. Suppose that there exists α s.t. $1/x \leq \alpha$ and $1 + x \leq \alpha$ for some $x \in \mathbb{R}$ s.t. $0 < x \leq 1$. We show that Eq. (5.10) holds:

$$\begin{aligned} U_L^L + \frac{U_H^L}{x} &\leq U_L^L + \alpha U_H^L \\ &\leq \alpha(U_L^L + U_H^L), \end{aligned}$$

which is smaller than or equal to 1 if $U_L^L + U_H^L \leq 1/\alpha$.

We show that Eq. (5.11) holds. We divide cases depending on whether $U^H_{H2} - U^L_{H2} \leq U^L_{L2}$ or not. When $U^H_{H2} - U^L_{H2} \leq U^L_{L2}$, we show that Eq. (5.11) holds:

$$\begin{aligned} & U^L_{L1} + \frac{U^L_{H1}}{x} + xU^L_{L2} + U^H_{H2} \\ & \leq U^L_{L1} + \alpha U^L_{H1} + (1+x)U^L_{L2} + U^L_{H2} \\ & \leq \alpha(U^L_L + U^L_H), \end{aligned}$$

which is smaller than or equal to 1 if $U^L_L + U^L_H \leq 1/\alpha$. When $U^L_{L2} < U^H_{H2} - U^L_{H2}$, we show that Eq. (5.11) holds:

$$\begin{aligned} & U^L_{L1} + \frac{U^L_{H1}}{x} + xU^L_{L2} + U^H_{H2} \\ & \leq U^L_{L1} + \alpha U^L_{H1} + U^L_{H2} + (1+x)(U^H_{H2} - U^L_{H2}) \\ & \leq \alpha(U^L_{L1} + U^L_{H1} + U^H_{H2}), \end{aligned}$$

which is smaller than or equal to 1 if $U^L_{L1} + U^L_{H1} + U^H_{H2} \leq 1/\alpha$. In sum, Eqs. (5.10) and (5.11) hold if $b \leq 1/\alpha$.

We need to find the range of α. To do it, we show the existence of x satisfying both of $1/\alpha \leq x$ and $x \leq \alpha - 1$:

$$1/\alpha \leq \alpha - 1 \;\Leftrightarrow\; \alpha^2 - \alpha - 1 \geq 0,$$

which is always true if $\alpha \geq \frac{1+\sqrt{5}}{2}$. □

5.5. The Extension with Partitioned Multiprocessor Scheduling

To extend the uniprocessor U-MC-ADAPT scheduling framework into partitioned multiprocessor platforms, we propose an MC-ADAPT scheduling framework. In the framework, we present a partition algorithm reflecting the characteristics of U-MC-ADAPT (Chapter 5.5.1). Then, we present a new semi-partitioned scheduling approach (allowing the migration of

LO-tasks) which further reduces the dropping of LO-tasks at runtime (Chapter 5.5.2).

5.5.1. The MC-ADAPT Partition Algorithm

To schedule tasks on partitioned multiprocessor platforms, we need partition algorithms that partition tasks into processors. In existing work, many partition algorithms have been proposed for multiprocessor MC systems [11, 28, 46]. Baruah et al. [11] proposed a First-Fit (FF) partition algorithm based on the EDF-VD scheduling algorithm [4]. The Worst-Fit (WF) partition algorithms are adopted for individual VD tuning [28] based on the EY scheduling algorithm [25], and for load balance across the processors based on its own scheduling algorithm [46]. Since our EDF-AD-E is an extension of EDF-VD, we follow the similar approaches to Baruah et al. [11]. This is our partition algorithm based on the FF partition policy.

The Partition Algorithm for MC-ADAPT. We partition tasks into m processors as follows:

- Sort tasks in a decreasing order of criticality-dependent task utilization (u_i^H if $\chi_i = \mathsf{HI}$ and u_i^L otherwise).
- Allocate each task to processors by the FF partition policy based on the offline schedulability of EDF-AD-E (Theorem 5.4 in Chapter 5.3.2).
- If every processor is MC-schedulable, then partitioning is successful. Otherwise, the system is not MC-schedulable under the MC-ADAPT framework with m processors.

After partitioning, we have a separate workload for each processor. Then, we can apply the U-MC-ADAPT scheduling framework for each processor.

5.5.2. The MC-ADAPT Migration Algorithm

By adopting semi-partitioned approaches (allowing task migration to another processor), we can further reduce the dropping of LO-tasks in partitioned multiprocessor platforms. Our

goal is to reduce the number of task dropping via task migration under the MC-ADAPT scheduling framework. When the scheduler of a processor decides on some LO-tasks to drop, we can consider to migrate the LO-task to another processor. A challenge is to develop a migration decision algorithm which does not incur deadline misses of HI-tasks. We need to analyze runtime slack of each component which can schedule the migrating LO-task, which is almost the same as the online schedulability of the U-MC-ADAPT (Chapter 5.3.2). In addition, we need to check whether the original workload of a processor and the migrating LO-task are schedulable even if there is further mode-switches, which is similar to the U-MC-ADAPT offline schedulability analysis.

The Offline Analysis for Migration Allowance. Consider the workload of a processor after applying the partition algorithm. Since we only migrate LO-tasks, the set of HI-tasks is unchanged during runtime scheduling. We need to analyze how many LO-tasks of other processors are allowed to migrate into the processor. To do this, we regard the LO-task set as a flexible set which can be added with new migrating LO-tasks from other processors. The following theorem presents how many LO-tasks (from other processors) can be migrated into a processor under EDF-AD-E by utilizing the offline schedulability analysis of EDF-AD-E (Theorem 5.4 in Chapter 5.3.2).

Theorem 5.6. *Consider a task set τ on a processer. Assume that the LO-task set $\tau_{L'} \notin \tau$ is migrated into the processor at runtime. Then, the revised task set ($\tau \cup \tau_{L'}$) is MC-schedulable under EDF-AD-E if*

$$U_{L'}^{L} \leq \frac{(1-U_H^H)(1-U_F^H)}{1+U_H^L-U_F^L-U_H^H} - U_L^L. \tag{5.13}$$

.

Proof. Let $\tau_{L\#}$ be $\tau_L \cup \tau_{L'}$. Then, the revised task set $\tau \cup \tau_{L'}$ is equivalent to $\tau_H \cup \tau_{L\#}$. To be MC-schedulable under EDF-AD-E, by Theorem 5.4, we need to satisfy Eq. (5.8) and

(5.9) with the revised task set $(\tau_H \cup \tau_{L'})$, which is equivalent to:

$$\begin{aligned}
&\frac{U_H^L - U_F^L}{1 - U_F^H - U_{L\#}^L} \leq \frac{1 - U_H^H}{U_{L\#}^L} \\
\Leftrightarrow\ & U_{L\#}^L(U_H^L - U_F^L) \leq (1 - U_H^H)(1 - U_F^H - U_{L\#}^L) \\
\Leftrightarrow\ & U_{L\#}^L(1 + U_H^L - U_F^L - U_H^H) \leq (1 - U_H^H)(1 - U_F^H) \\
\Leftrightarrow\ & U_{L\#}^L \leq \frac{(1 - U_H^H)(1 - U_F^H)}{1 + U_H^L - U_F^L - U_H^H},
\end{aligned}$$

which is Eq. (5.13) because $U_{L\#}^L = U_L^L + U_{L'}^L$. □

Since we consider the runtime migration, we need a different assignment of the VD coefficient than EDF-AD-E that does not consider the migration. The VD coefficient by EDF-AD-E makes the worst-case utilization after mode switch to be 1. Then, the migration of LO-tasks into the processor makes the system unschedulable. To handle the migration, we present a new assignment of the VD coefficient for the runtime migration based on Theorem 5.6, which is $x := \frac{1-U_H^H}{U_L^L + U_{L*}^L}$ where U_{L*}^L is the largest value of $U_{L'}^L$ satisfying Eq. (5.13).

The MC-ADAPT Online Migration Algorithm. We present the migration algorithm as shown in Algorithm 5.1. The algorithm is activated when mode-switch in a processor Φ_{MS} happens and the scheduler of Φ_{MS} decides to drop an LO-task τ_j. Instead of dropping the LO-task, we seek another processor Φ_k that can execute the LO-task without any deadline miss (Line 1–8). In Line 3, we check whether Φ_k can schedule its original workload and the migrated LO-tasks according to Theorem 5.6. If such a Φ_k exists, we can migrate τ_j on Φ_{MS} into Φ_k (Line 4–6). Since τ_j may execute partly, we need to consider the executed time of τ_j on Φ_{MS} at the future mode-switch on Φ_{MS}. Thus, we do not remove τ_j from the task set of Φ_{MS} and just treat τ_j as the dropped LO-task in the online schedulability test, although we remove τ_j from the run-queue of Φ_{MS} after the migration. If such a Φ_k does not exist, the algorithm declares that the migration of τ_j fails and the scheduler drops τ_j, which is the original decision of the uniprocessor EDF-AD-E scheduling algorithm (Line 9).

Algorithm 5.1 The MC-ADAPT online migration algorithm

Input: $\{W_i\}$ (the workload of processors $\{\Phi_i\}$), Φ_{MS} (the processor that mode switch happens at), τ_j (the LO-task that the scheduler of Φ_{MS} decides to drop at mode-switch of a HI-task)

Output: $\{W'_i\}$ (the workload of processors $\{\Phi_i\}$ after migration)

for $k := 1$ to m **do**
    $W'_k := W_k \cup \tau_j$
    if Eq. (5.13) with W'_k holds **then**
        $\forall \Phi_i$ s.t. $\Phi_i \neq \Phi_k$, $W'_i := W_i$
        return $\{W'_i\}$ ▷ τ_j can migrate into Φ_k. τ_j does not need to be dropped.
    end if
end for
return $\{W_i\}$ ▷ τ_j cannot migrate. τ_j needs to be dropped.

5.6. Evaluation

We evaluate MC-ADAPT on uniprocessor platforms (Chapter 5.6.2) and multiprocessor platforms (Chapter 5.6.3). In the previous section, we looked at the speedup factor of the EDF-AD-E for the task dropping problem. In this section, we evaluate the effectiveness of EDF-AD-E in comparison with the existing approaches, via simulation with synthetic workloads. In addition, we compare EDF-AD and EDF-AD-E with the existing approaches in terms of MC-schedulability.

5.6.1. Experiment Setup

We generate random task sets according to the workload-generation algorithm [40]. Let U^b be the upper bound of both LO-criticality and HI-criticality utilizations. A random task is generated as follows (all task parameters are randomly drawn in uniform distribution): for a task τ_i,

- U_i (task utilization) is a real number drawn from the range $[0.02, 0.2]$.
- T_i (task period) is an integer drawn from the range $[20, 300]$.
- R_i (the ratio of u_i^H/u_i^L) is a real number drawn from the range $[1, 4]$.
- P_i (the probability that the task is a HI-task) is a real number from the range [0,1].

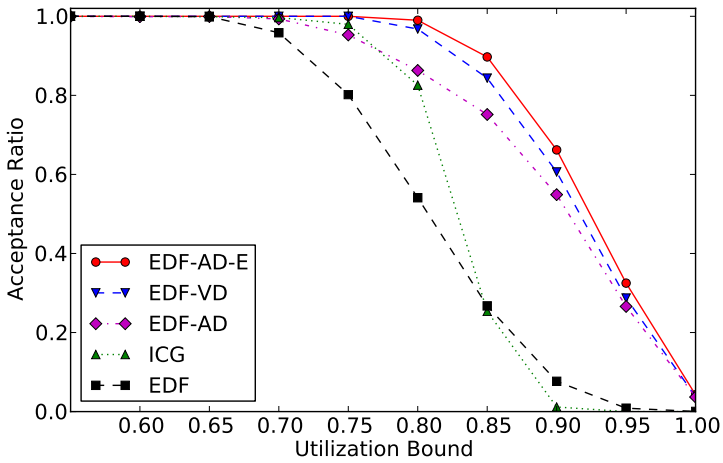


Figure 9: MC-schedulability varying utilization bound

If $P_i < P^{\mathsf{HI}}$ (default value of P^{HI} is 0.5), set $\chi_i := LO$ and $C_i^L := \lfloor U_i \cdot T_i \rfloor$. Otherwise, set $\chi_i := HI$, $C_i^H := \lfloor U_i \cdot T_i \rfloor$, and $C_i^L := \lfloor U_i \cdot T_i / R_i \rfloor$.

Repeat generating a task in the task set until $\max(U_H^L + U_L^L,\ U_H^H) > U^b$. Then, discard the task added last.

5.6.2. Evaluation on Uniprocessor Platforms

Before presenting simulation results on multiprocessor platforms, we first evaluate U-MC-ADAPT on uniprocessor platforms in terms of MC-schedulability and the Deadline Miss Ratio (DMR)[8] of LO-tasks.

MC-schedulability. We compare the MC-schedulability of EDF-AD and EDF-AD-E with the existing MC scheduling algorithms, which are regular EDF, EDF-VD [4], and ICG [33][9]. We mathematically compute the schedulability of the randomly-generated systems via the schedulability test of each scheduling algorithms.

Fig. 9 shows the acceptance ratio (the ratio of schedulable task sets) over varying utilization

[8]DMR is the ratio of the number of the unfinished jobs over the total number of jobs released in a given time interval. We assume that an LO-task in the dropped state releases its job but does not execute.

[9]The MC-schedulability of the ICG is maximum (because interference to LO-tasks are all constrained) when the interference constraint graph is fully connected from HI-tasks to LO-tasks.

bound U^b from 0.55 to 1.0 in increments of 0.05. Each data point is based on 5,000 systems. The result shows that EDF-AD-E dominates other approaches. Although EDF-AD has a higher acceptance ratio than regular EDF, we confirmed that EDF-AD has the schedulability anomaly and has lower MC-schedulability than EDF-VD for all utilization ranges and ICG for some utilization ranges.

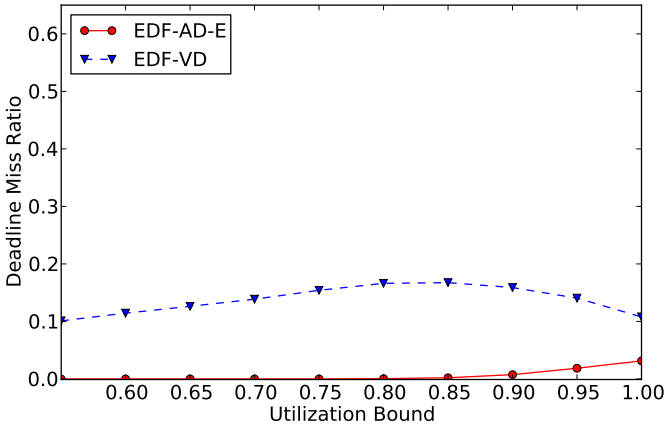
The Deadline Miss Ratio. We compare EDF-AD-E with EDF-VD [4] in terms of the DMR of LO-tasks. For a given randomly-generated system schedulable by EDF-VD, we simulate the behavior of tasks with a given probability of mode switch for any HI-task, denoted as P^{MS} (default value of P^{MS} is 0.4), for 10,000 time units[10].

Fig. 10 shows the average DMR with varying utilization bound U^b for different probabilities of mode switch: $P^{\mathsf{MS}} = 0.1$, $P^{\mathsf{MS}} = 0.4$ and $P^{\mathsf{MS}} = 0.7$. For each utilization bound, we generate 5,000 systems. The result shows that EDF-AD-E significantly outperforms EDF-VD because the resource-efficient scheduling of EDF-AD-E minimizes the additional resource request at mode switch and the EDF-AD-E task dropping algorithm selects the minimal set of LO-tasks for the resource request.

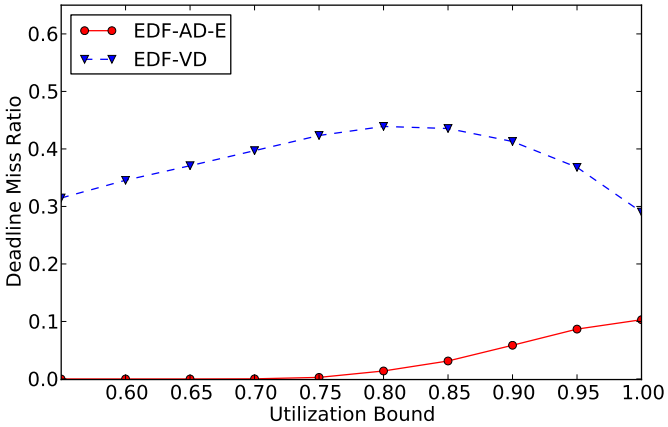
Fig. 11 shows the DMR with fixing $U^b := 0.8$ and varying P^{HI} from 0.5 to 0.95 in increments of 0.5. For each data point, we generate 5,000 systems. In the simulation result, EDF-VD shows higher DMR for higher P^{HI} while EDF-AD-E shows little variance for different P^{HI}. For the simulation where P^{HI} is high (meaning a large number of HI-tasks), at every mode switch of HI-tasks, EDF-VD drops all LO-tasks while EDF-AD-E drops only the minimal set of LO-tasks.

Fig. 12 shows DMR varying simulation durations ($U^b = 0.85$ and $P^{MS} = 0.4$). It shows that the simulation duration does not affect the simulation results. Due to the return protocol to LO-mode, the DMR of EDF-VD and EDF-AD-E are converged in a large simulation duration.

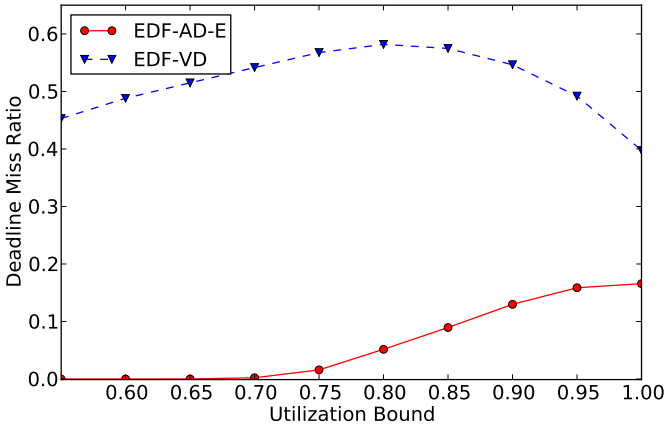
[10] According to EDF-VD [4], on idle tick, the system is switched back to the initial state (all HI-tasks are in LO-mode and all LO-tasks are active).



(a) $P^{\mathsf{MS}} = 0.1$



(b) $P^{\mathsf{MS}} = 0.4$



(c) $P^{\mathsf{MS}} = 0.7$

Figure 10: The DMR for different P^{MS}

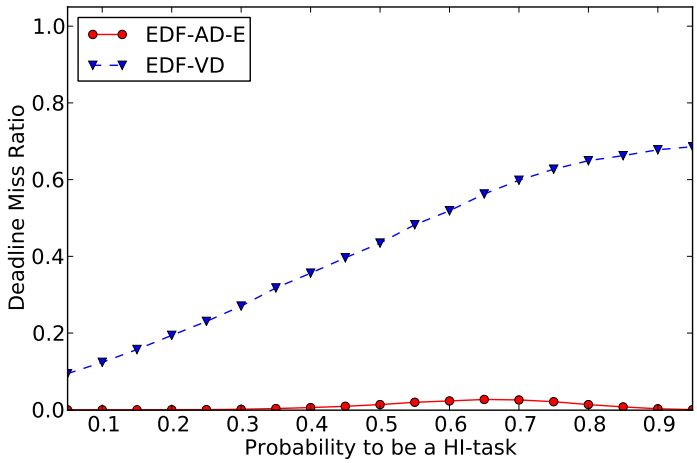


Figure 11: The DMR for different P^{HI}

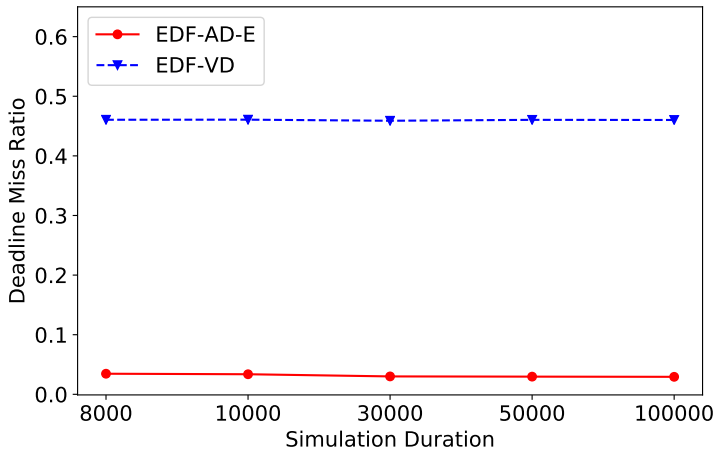


Figure 12: The DMR for Different Simulation Duration

5.6.3. Evaluation on Multiprocessor Platforms

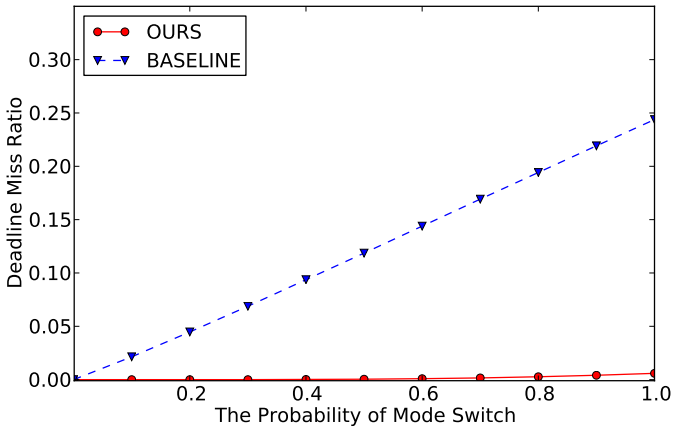
To evaluate the effectiveness of our migration algorithm on multiprocessor platforms, we compare the DMR of BASELINE (the MC-ADAPT scheduling algorithm without the migration algorithm) and OURS (the MC-ADAPT scheduling algorithm with the migration algorithm) for randomly-generated systems that are schedulable by MC-ADAPT. We conduct the simulation on four processors ($m = 4$) and eight processors ($m = 8$). We measure the DMR of each approach in different loads: light load ($U^b/m = 0.65$), medium load ($U^b/m = 0.75$), and heavy load ($U^b/m = 0.85$). For each load, we vary the probability of mode switch (P^{MS}) from 0.0 to 1.0 in increments of 0.1. For each data point, we measure the DMRs of 5,000 random systems and take their average value. For each task system that is generated by the random task set generator, we simulate the system for 10,000 time units[11].

Figure 13 is the simulation result on two processors. In the light load (Figure 5.13(a)), OURS significantly outperforms BASELINE because the migration algorithm can utilize the large remaining resources on each processor. As the probability of mode-switch increases, the performance gap increases because OURS can effectively migrate LO-tasks that are dropped in BASELINE. As load (utilization bound) increases (Figure 5.13(b) and 5.13(c)), the performance gap between OURS and BASELINE decreases because the remaining processor utilization for each processor decreases. Similar trends are also observed in the simulation result on eight processors (Figure 14).

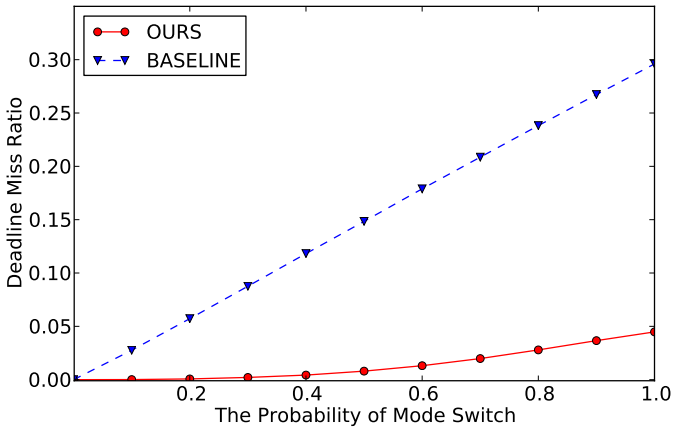
5.7. Summary

We present the uniprocessor U-MC-ADAPT framework that makes online adaptive task dropping utilizing the dynamic system state under task-level mode switch. We develop the EDF-AD-E scheduling algorithm that decides the dropping of LO-tasks via runtime

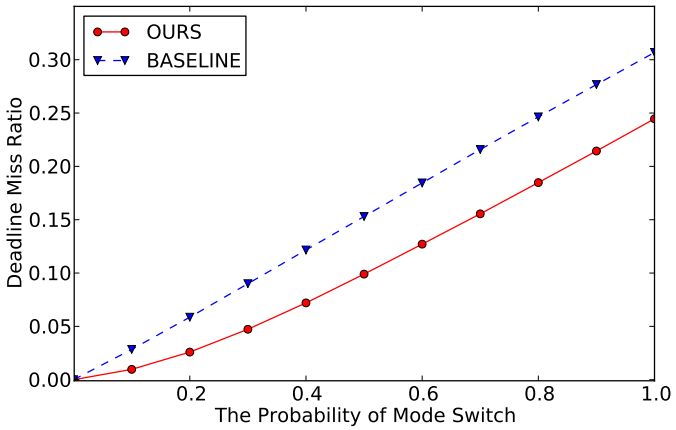
[11] On idle tick, the system is switched back to the initial state (all HI-tasks are in LO-mode and all LO-tasks are active) and the LO-task set for a processor is fixed to the union of the original LO-tasks of the processor and the migrated LO-tasks.



(a) Light load ($U^b/m = 0.65$)

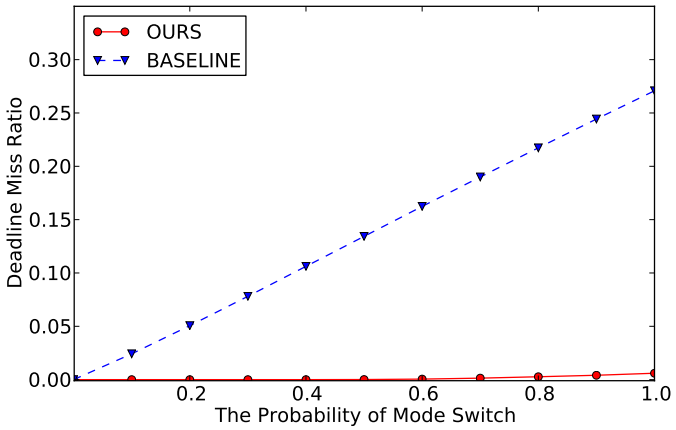


(b) Medium load ($U^b/m = 0.75$)

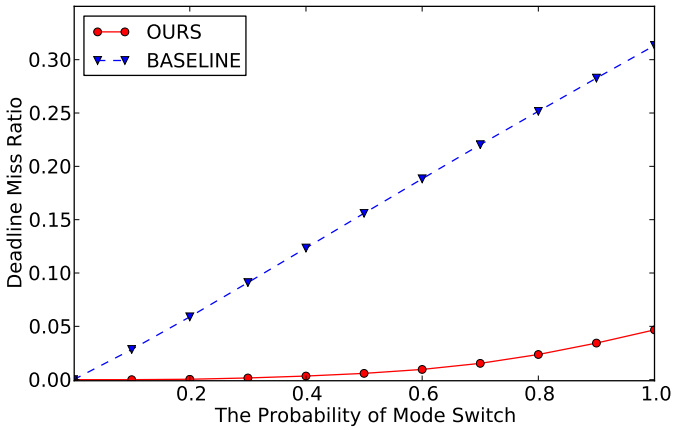


(c) Heavy load ($U^b/m = 0.85$)

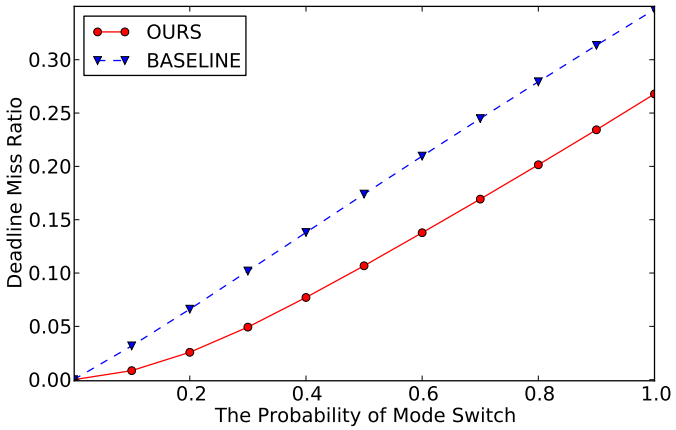
Figure 13: The DMR varying P^{MS} on four processors ($m = 4$)



(a) Light load ($U^b/m = 0.65$)



(b) Medium load ($U^b/m = 0.75$)



(c) Heavy load ($U^b/m = 0.85$)

Figure 14: The DMR varying P^{MS} on eight processors ($m = 8$)

analysis, which captures the dynamic system state efficiently without tracking the previous history. To evaluate the quality of task dropping, we propose the speedup factor for the task dropping problem while the speedup factor for the MC scheduling problem only evaluates MC scheduling algorithms in terms of the worst-case schedulability. We derive that the speedup factor of EDF-AD-E for the task dropping problem is 1.618.

Based on U-MC-ADAPT, we propose the partitioned multiprocessor scheduling framework for MC-ADAPT. To further reduce the dropping of LO-tasks with multiprocessor platforms, we propose the MC-ADAPT migration algorithm which migrates the LO-tasks that should be dropped.

CHAPTER 6 : Conclusion and Future Work

This dissertation studied resource-efficient scheduling techniques on multiprocessor MC systems. First, we want to know whether the optimality result in uniprocessor MC scheduling is applicable to the multiprocessor domain. Extending the optimal fluid scheduling model from the regular (non-MC) domain, we presented the MC-Fluid scheduling framework, which has the same speedup optimality as uniprocessor EDF-VD. Second, we considered how to relax the unrealistic assumption of MC-Fluid (i.e., the use of a fractional processor). Here, we presented the MC-DP-Fair scheduling algorithm and also showed that MC-DP-Fair is also speedup optimal. Third, we considered the drawback of global multiprocessor scheduling approaches (e.g., preemption/migration and global run-queue overheads) and the limitation of conventional MC scheduling algorithms (i.e., all LO-tasks are dropped when exceeding a certain threshold limit). Overcoming these limitations, we presented the MC-ADAPT scheduling framework to drop as few LO-tasks as possible with partitioned scheduling approaches. Addressing the limitation of the conventional speedup factor for the MC scheduling problem which evaluates only the worst-case schedulability, we applied the speedup factor for the task dropping problem. We derived that the speedup factor of MC-ADAPT for the task dropping problem is 1.618. Based on the uniprocessor U-MC-ADAPT, we proposed a partitioned multiprocessor scheduling framework for MC-ADAPT with a runtime migration technique which reduces the dropping of LO-tasks via task migration.

As future work, we plan to extend our frameworks into general multi-criticality systems. We also consider to extend our framework into component-based systems for large-scale open MC systems. More specifically, we plan to extend the MC-Fluid framework with more than two execution rates per task, which further increases resource efficiency. In order to deploy MC-Discrete on hardware platforms which only supports discrete-time schedule, we plan to extend MC-Discrete based on the Boundary Fair scheduling algorithm [50], which is a non-MC solution for the same problem. For a better speedup factor for the task dropping

problem, we plan to extend MC-ADAPT by developing a new speedup derivation technique for MC-ADAPT or proposing a new scheduling algorithm with adaptive task dropping.

Bibliography

[1] James H. Anderson, Sanjoy K. Baruah, and Bjorn B. Brandenburg. Multicore operating-system support for mixed criticality. In *Workshop on Mixed Criticality*, 2009.

[2] N. C. Audsley. On priority asignment in fixed priority scheduling. *Information Processsing Letters*, 79(1):39–44, May 2001.

[3] AUTOSAR. AUTomotive Open System ARchitecture. `www.autosar.org`.

[4] 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*, pages 145–154, July 2012.

[5] S. Baruah, V. Bonifaci, G. D'Angelo, Haohan Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *Computers, IEEE Transactions on*, 61(8):1140–1152, Aug 2012.

[6] S. Baruah, A. Eswaran, and Z. Guo. MC-Fluid: Simplified and Optimally Quantified. In *Real-Time Systems Symposium, 2015 IEEE*, pages 327–337, Dec 2015.

[7] S. Baruah, Haohan Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 13–22, April 2010.

[8] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Real-Time Systems (ECRTS), 2008 21st Euromicro Conference on*, pages 147–155, July 2008.

[9] 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*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM.

[10] Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *Algorithms - European Symposium on Algorithms (ESA) 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 555–566. Springer Berlin Heidelberg, 2011.

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

[12] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne Van Der Ster, and Leen Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proceedings of the 19th European Conference on Algorithms*, ESA'11, pages 555–566, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] S.K. Baruah, A. Burns, and R.I. Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43, Nov 2011.

[14] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, Dec 1990.

[15] Andrea Bastoni, Bjorn B. Brandenburg, and James H. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Proceedings of the Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2010)*, pages 33–44, July 2010.

[16] Iain Bate, Alan Burns, and Robert I. Davis. A Bailout Protocol for Mixed Criticality Systems. In *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems*, ECRTS '15, pages 259–268, Washington, DC, USA, 2015. IEEE Computer Society.

[17] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[18] Alan Burns and Sanjoy Baruah. Towards a more practical model for mixed criticality systems. In *Proceedings of the First Workshop of Mixed Criticality Systems (WMC 2013)*, pages 1–6, Dec 2013.

[19] Alan Burns and Robert Davis. Mixed criticality systems – a review. `http://www-users.cs.york.ac.uk/burns/review.pdf`, Jan 2017. the ninth edition.

[20] Hyeonjoong Cho, B. Ravindran, and E.D. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 101–110, Dec 2006.

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

[22] Sudarshan K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[23] Francois Dorin, Pascal Richard, Michael Richard, and Joel Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems*, 46(3):305–331, 2010.

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

[25] P. Ekberg and Wang Yi. Bounding and shaping the demand of mixed-criticality sporadic tasks. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 135–144, July 2012.

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

[27] Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed Criticality Systems with Weakly-hard Constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 237–246, New York, NY, USA, 2015. ACM.

[28] 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)*, pages 1–6, March 2014.

[29] X. Gu and A. Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 47–56, Nov 2016.

[30] Xiaozhe Gu, A. Easwaran, Kieu-My Phan, and Insik Shin. Resource efficient isolation mechanisms in mixed-criticality scheduling. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 13–24, July 2015.

[31] Nan Guan, P. Ekberg, M. Stigge, and Wang Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 13–23, Nov 2011.

[32] Philip Holman and James H. Anderson. Adapting pfair scheduling for symmetric multiprocessors. *J. Embedded Comput.*, 1(4):543–564, December 2005.

[33] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele. Interference Constraint Graph - A new specification for mixed-criticality systems. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2013.

[34] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. Maximizing the execution rate of low-criticality tasks in mixed criticality systems. In *Proceedings of the First Workshop of Mixed Criticality Systems (WMC 2013)*, Dec 2013.

[35] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[36] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of ACM*, 47(4):617–643, July 2000.

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

[38] 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*, pages 3–13, July 2010.

[39] Haohan Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 183–192, Nov 2010.

[40] Haohan Li and S. Baruah. Global mixed-criticality scheduling on multiprocessors. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 166–175, July 2012.

[41] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–46, Nov 2016.

[42] Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[43] R.M. Pathan. Schedulability analysis of mixed-criticality systems on multiprocessors. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 309–320, July 2012.

[44] P.J. Prisaznuk. Integrated modular avionics. In *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, pages 39–45 vol.1, May 1992.

[45] Saravanan Ramanathan and Arvind Easwaran. MC-Fluid: rate assignment strategies. In *International Workshop of Mixed-Criticality Systems (WMC) in conjuction of RTSS*, December 2015.

[46] J. Ren and Linh Thi Xuan Phan. Mixed-criticality scheduling on multiprocessors using task grouping. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 25–34, July 2015.

[47] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 155–165, July 2012.

[48] H. Su and D. Zhu. An Elastic Mixed-Criticality task model and its scheduling algorithm. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 147–152, March 2013.

[49] 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*, pages 239–243, Dec 2007.

[50] Dakai 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*, pages 142–151, Dec 2003.