MC−FLEX: Flexible Mixed-Criticality Real-Time Scheduling by Task-Level Mode Switch

Jaewoo Lee, *Member, IEEE* and Jinkyu Lee, *Senior Member, IEEE*

Abstract—Mixed-criticality (MC) scheduling becomes popular in real-time systems as it supports different criticality levels in a resource-efficient manner. Although it has been well established (i) how to guarantee MC schedulability *offline*, existing studies have paid less attention to achieve (ii) how to minimize deadline misses of low-criticality tasks *at runtime*; in addition, it has not matured yet how to address (ii) without compromising (i). In this paper, we propose MC−FLEX, which employs a task-level mode transition mechanism (as opposed to system-level one). MC−FLEX not only determines time instants at which each high-criticality task enters and exits the critical mode in a task level, but also selects time instants and target low-criticality task(s) to be dropped and resumed for each task-level mode switch of individual high-criticality tasks, yielding the achievement of both (i) and (ii). Via simulation results, we demonstrate that the proposed framework reduces the job deadline miss ratio of low-criticality tasks at runtime (by over 54.8% compared to the existing work), without compromising offline MC schedulability.

Index Terms—Mixed-criticality systems, real-time scheduling, task-level mode switch, EDF-VD

1 Introduction

Recent complex embedded systems are inherently *Mixed-Criticality (MC) systems*, where different components with different criticality levels are integrated under a shared platform. A typical example is an automotive system that contains both high-criticality components (e.g., the engine) and low-criticality ones (e.g., rear lights). To support such different criticality components, the automotive industry defined ISO 26262, an international standard for functional safety; for example, the engine belongs to ASIL (Automotive Safety Integrity Level) D (i.e., the highest criticality), while the rear lights belong to ASIL A (i.e., the lowest criticality).

MC studies for real-time systems have typically targeted two levels of criticality: high and low. Each MC task has two different Worst-Case Execution Time (WCET) estimates depending on its confidence level; a MC task may execute for up to its high- and low-confidence WCET, respectively. The system requirements are (i) all high-criticality tasks always satisfy their deadline requirements and (ii) all low-criticality tasks meet their deadline requirements as long as no high-criticality task executes for more than its low-confidence WCET. Since Vestal's seminal work [1], earlier MC studies [2], [3], [4], [5] have succeeded in providing *offline* guarantees on (i) and (ii), by handling the system-level mode switch from the situation where there does not exist any high-criticality task with more than its low-criticality WCET to the situation where there exists such a task (called *system-level mode switch*).

However, in order to apply MC systems into diverse industry domains, we need to guarantee the timely execution of low-criticality tasks as much as possible even under existence of a high-criticality task executing for more than its low-confidence WCET. For example, an autonomous driving vehicle may enter the critical mode when a collision is anticipated. Once the vehicle transits the critical mode, subsystems thereof may need additional computing resources due to extra tasks such as the advanced emergency braking system and the collision avoidance system; to supply additional computing resources to the extra tasks, the vehicle may decide to turn off low-criticality subsystems such as a navigation system. After the vehicle safely escapes from the critical situation, the vehicle should recover its system mode to the normal driving mode and turn on low-criticality subsystems for full functionality of the vehicle. In diverse industry domains such as an autonomous driving vehicle, it is important to provide *offline* guarantees on (i) and (ii) while satisfying the following requirement *at runtime*: (iii) minimizing job deadline misses of low-criticality tasks. So far, this matter has been partially addressed in several attempts [6], [7], [8], [9], [10], [11], [12]; however, few studies have systematically addressed (iii) *at runtime* without compromising *offline* guarantees on (i) and (ii). Note that (iii) is more favorable to many practical systems than reducing execution of each job. For example, in a system where degraded services are not allowed, a completion of a job and a drop of the next job yield one complete result out of two, while partial execution of the two jobs cannot yield any single meaningful result.

In this paper, we propose MC−FLEX, which employs a task-level mode transition mechanism (as opposed to system-level one). MC−FLEX carefully determines time instants at which each high-criticality task enters and exits the critical mode in a task level manner (called *switch-forward*

- *Jaewoo Lee is with the Department of Industrial Security, Chung-Ang University, Seoul 156-756, Republic of Korea. E-mail: jaewoolee@cau.ac.kr.*
- *Jinkyu Lee is with the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Suwon 16419, Republic of Korea. E-mail: jinkyu.lee@skku.ac.kr.*

Manuscript received 7 June 2020; revised 17 Feb. 2021; accepted 4 Sept. 2021. Date of publication 14 Sept. 2021; date of current version 11 July 2022. (Corresponding author: Jinkyu Lee.)
Recommended for acceptance by B. Childers.
Digital Object Identifier no. 10.1109/TC.2021.3111743

and *switch-backward*, respectively), such that the mechanism not only minimizes the duration for the critical mode addressing (iii) *at runtime*, but also preserves *offline* guarantees on (i) and (ii). In particular, we re-design EDF-VD [3], one of the most popular MC scheduling algorithms and develop the offline and online schedulability analysis for the re-designed algorithm, so as to address (iii) without downgrading offline guarantee for (i) and (ii) achieved by the vanilla EDF-VD. Although there have been a few studies for (iii) [13], [14], [15], [16], [17], [18], they have only partially addressed (iii) in that none of them has succeeded in employing a mechanism that supports task-level resuming of each low-criticality task. Compared to those studies, MC−FLEX carefully designs task-level switch-forward/backward mechanisms with offline schedulability guarantees; in addition, MC−FLEX allows task-level dropping/resuming of low-criticality tasks, which can minimize their job deadline misses. To this end, MC−FLEX addresses the following two challenges.

Q1. When a high-criticality task performs a switch-forward and switch-backward, how can we determine a set of low-criticality task(s) to be dropped and resumed, respectively, so as to achieve (i), (ii) and (iii) during each mode change?

Q2. How can we derive offline schedulability analysis that gives offline guarantees on (i) and (ii), which accords with the answer of Q1?

We first address Q1 and Q2 with two assumptions (to be detailed in Section 4): one regarding the inter-arrival time of two mode switches, and the other regarding the rate of low- and high-confidence WCET. As to Q1, we derive a condition that depends on which tasks belong to the following four categories between two consecutive mode switches: (a) high-criticality tasks in the high-criticality mode, (b) high-criticality tasks in the low-criticality mode, (c) low-criticality tasks to be executed, and (d) low-criticality tasks not to be executed. We prove that all high-criticality tasks meet their deadline requirements between two consecutive mode switches as long as the condition is satisfied. We then propose two policies of determining a set of low-criticality tasks whose state changes between (c) and (d), upon a mode switch of a high-criticality task between (a) and (b). The policies aim at minimizing the number of job deadline misses of low-criticality tasks, without compromising the condition. For Q2, we analyze the condition, and derive the worst-case requirement that holds under any sequences of mode switches, yielding offline schedulability analysis.

Once Q1 and Q2 are successfully addressed under the two assumptions, the next step is to relax the two assumptions. To this end, we propose two techniques for the advanced version of MC−FLEX. First, we develop a notion of "virtual criticality mode", which separates the time instant of a high-criticality task's switch-backward and that of a set of low-criticality tasks' resuming, yielding the relaxation of the first assumption. Second, we find that the schedulability analysis can be improved if some high-criticality tasks always execute in the high-criticality mode. To utilize the finding, we add a run-time mechanism that disallows such high-criticality tasks to become the low-criticality mode, which not only relaxes the second assumption, but also improves schedulability. We detail the two techniques in Section 5.

We evaluate the performance of MC−FLEX framework in terms of offline schedulability and deadline miss ratio at runtime, via simulation based on synthetic workloads. In comparison with existing studies, simulation results show that MC−FLEX exhibits the equivalent offline schedulability and reduce the job deadline miss ratio of low-criticality tasks by over 54.8%.

This paper makes the following contributions:

- We develop MC−FLEX framework on MC systems to guarantee MC schedulability offline and minimize job deadline misses of low-criticality tasks at runtime, which is the first mode switch mechanism that supports not only task-level switch-forward/backward of high-criticality tasks, but also task-level dropping/resuming of low-criticality tasks.
- We present a run-time mechanism of MC−FLEX that determines a set of low-criticality tasks to be resumed/dropped at each switch-forward/backward of high-criticality tasks, and derive a schedulability test that offers timing guarantees at each mode switch.
- Based on the run-time mechanism and its schedulability test, we derive offline schedulability test for MC−FLEX that offers timing guarantees under any mode switch.
- To improve the basic version of MC−FLEX, we propose/apply two advanced techniques, yielding relaxation of the two assumptions.
- Via simulations, we demonstrate that MC−FLEX significantly reduces the job deadline miss ratio of low-criticality tasks without degrading offline schedulability performance.

The rest of the paper is structured as follows. Section 2 presents the system model, and Section 3 explains motivation and background. Section 4 proposes the basic version of MC−FLEX, which is improved in Section 5. Section 6 evaluates MC−FLEX, and Section 7 explains related work. Section 8 discusses remaining issues, and Section 9 concludes the paper.

2 System Model

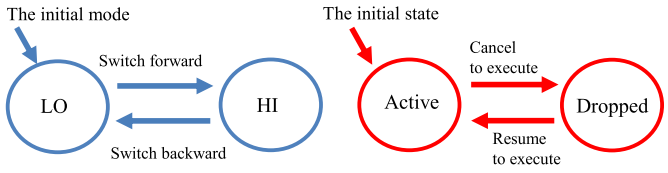
We consider a *dual-criticality* uniprocessor system with two distinct criticality levels: HI (high) and LO (low).

Task Model. We consider an implicit-deadline sporadic task system (denoted τ) of n MC tasks. Each MC task τ_i generates an infinite sequence of jobs $\{J_i^1, J_i^2, \ldots\}$ and is characterized by $(T_i, C_i^L, C_i^H, \chi_i)$, where

- $T_i \in \mathbb{R}$ is the minimum inter-job separation time (or *period*),
- $C_i^L \in \mathbb{R}$ is a LO-confidence WCET (L-WCET),
- $C_i^H \in \mathbb{R}$ is a HI-confidence WCET (H-WCET), and
- $\chi_i \in \{\mathsf{HI}, \mathsf{LO}\}$ is a task criticality level.

We can categorize individual tasks τ_i by their criticality levels χ_i. For notational convenience, let τ_{HC} denote a set of tasks with HI-criticality levels (or a HC task set), i.e., $\tau_{HC} \stackrel{\text{def}}{=} \{\tau_i \in \tau | \chi_i = \mathsf{HI}\}$. Likewise, τ_{LC} denotes a set of tasks with LO-criticality levels (or a LC task set), i.e., $\tau_{LC} \stackrel{\text{def}}{=} \{\tau_i \in \tau | \chi_i = \mathsf{LO}\}$.

Utilization. The utilization of a task τ_i is defined as $u_i^L \stackrel{\text{def}}{=} C_i^L/T_i$ for LO-confidence utilization and $u_i^H \stackrel{\text{def}}{=} C_i^H/T_i$



(a) The task mode of a HC task (b) The execution state of an LC task

Fig. 1. Task behavioral model.

for HI-confidence utilization, respectively. For notational convenience, we define the collective utilization of a task set as follows:

$$U_{LC}^{L} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{LC}} u_i^L, \quad U_{HC}^{L} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{HC}} u_i^L, \quad U_{HC}^{H} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{HC}} u_i^H.$$

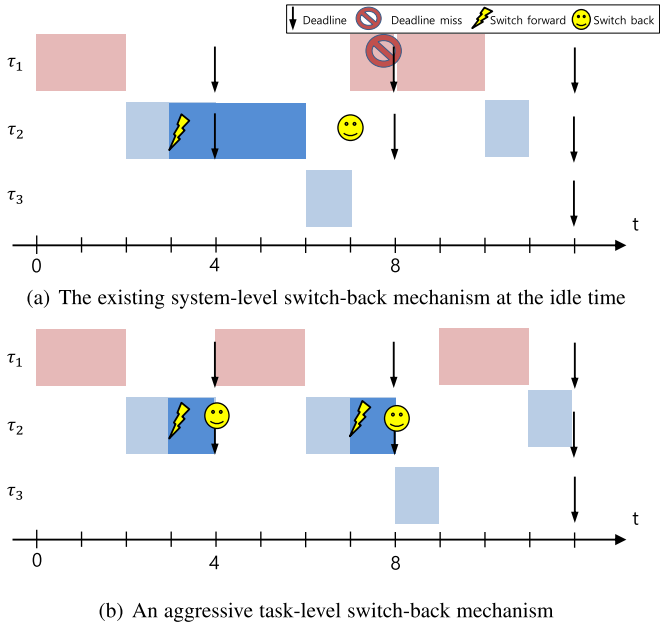
Task Behavior Model. We assume some degree of uncertainty on the execution time of different jobs for a task. We consider task-level criticality mode (*task mode*). Each HC task τ_i has its own task mode (denoted as M_i) that indicates its behavior. A task τ_i is said to be in LO mode ($M_i = \text{LO}$)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

As the name indicates, an individual task changes its task mode independently. Each HC task starts in LO mode, and change its own task mode to HI mode (called *switch-forward*) when its actual execution time exceeds C_i^L (see Fig. 1a). For a HC task in HI mode, its task mode can be switched to LO mode (called *switch-backward* or *switch-back*) when the execution time of the task is observed as no more than C_i^L (see Fig. 1b). Each scheduling algorithm needs to determine the time instance of switch-back, and Section 4.1 will discuss how MC−FLEX does. We define *mode switch* for a HC task as the transition of the task-level criticality mode including both switch-forward and switch-backward.

In addition to HC tasks, we consider the execution state of a LC task (see Fig. 1b); each LC task is in either an *active* state or a *dropped* state. Initially, all LC tasks are active. When the scheduler needs more computing resource (on switch-forward of at least one HC task), some active LC tasks are allowed to be dropped in order to support HC tasks with their additional resource requests. When a LC task is dropped, the current job of the task is suspended and no subsequent job of the task is processed. When the scheduler has available resource (on switch-back of at least a HC task), some LC tasks can be resumed to execute (i.e., jobs of the LC tasks newly-released after the resuming decision are processed) as some HC tasks require less resources. Each scheduling algorithm needs to determine which LC task is dropped/resumed, and Section 4.1 will detail this for MC−FLEX.

System Goal. Besides the schedulability of MC systems, the performance of LC tasks is also important [4], [19]. Our system goal is to drop as few jobs of LC tasks as possible, without compromising *MC schedulability* that consists of the following two conditions:

- *Condition MC-A*: HC tasks are always schedulable.
- *Condition MC-B*: LC tasks are schedulable if no HC task is in HI mode.



(a) The existing system-level switch-back mechanism at the idle time

(b) An aggressive task-level switch-back mechanism

Fig. 2. Comparison of two switch-back mechanisms for Example 1.

3 Motivation and Background

In this section, we first motivate the necessity of a task-level mode switch mechanism. To this end, we present an example of how a task-level mode switch mechanism can improve the existing system-level one in terms of minimizing drops of low-criticality tasks. We then explain the EDF-VD scheduling algorithm, which is a base prioritization policy of MC−FLEX to be explained in Sections 4 and 5.

3.1 Motivation

As discussed in Section 1, we need a mode transition mechanism that provides *offline* guarantees on (i) and (ii) while satisfying (iii) *at runtime*. However, most existing studies have not fully addressed such a mechanism as they have employed a system-wide switch-forward/backward mechanism. That is, in existing studies, a single job execution of a HC task for longer than its L-WCET incurs a system-level switch-forward, and the system incurs a system-level switch-back only if it is guaranteed that all HC tasks do not have their jobs whose execution are longer than their L-WCET. Both yields unnecessary drops of LC tasks; the former starts dropping earlier than it should do, and the latter keeps dropping even when it is not necessary. We now explain how the existing system-level mode switch mechanism incurs unnecessary job drops of LC tasks.

Example 1. Consider task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ where τ_1 is $(T_i = 4, C_i^L = 2, C_i^H = 2, \chi_i = \text{LO})$, τ_2 is $(4, 1, 2, \text{HI})$, and τ_3 is $(12, 1, 2, \text{HI})$. Suppose that the task set is scheduled by EDF (that gives a higher priority to a job with an earlier deadline); for tie-breaking, a task with a smaller index has a higher priority. As shown in both Figs. 2a and 2b, suppose that the first job and the second job of τ_2 executed for more than its L-WCET and the other jobs of HC tasks (τ_2 and τ_3) completed their execution within L-WCET. The scheduler performed switch-forward for τ_2 (from LO mode to HI mode) at time 3 and dropped LC

task τ_1 immediately (or allowed τ_1 to execute only when no job of τ_2 and τ_3 is waited in the scheduler).

(a) Consider the existing switch-back mechanism at the idle time [3], [14], [16] (see Fig. 2a). switch-back (of HC tasks) is performed at the idle time (no HC job is waited in the scheduler), which is time 7. At switch-back, we can resume to execute all LC tasks since all HC tasks are LO mode. Therefore, the second job of τ_1 cannot start its execution until time 7, yielding its deadline miss. The deadline miss ratio of the LC task's jobs is 33.3%

(b) Consider an aggressive switch-back mechanism (see Fig. 2b). In this simple example, τ_2 can be switched back in the absolute deadline of its first job, which is time 4. At switch-back, we can resume to execute all LC tasks since all HC tasks are LO mode. The second switch-forward and switch-back are triggered at time 7 and 8, respectively. Then, no job of τ_1 missed its deadline, which means 0% deadline miss ratio.

As seen in Example 1, a task-level switch mechanism described in Example 1(b) seems to yield an efficient mode change in terms of dropping LC tasks' jobs. However, in order to utilize a task-level switch mechanism for general cases, we need a more systematic scheduling mechanism that addresses the following issues.

- **I1**. When do we perform a task-level switch-forward or switch-backward for each HC task?
- **I2**. Which LC tasks are dropped for each switch-forward and resumed for each switch-back? How can we make conditions such that these selections do not compromise timing guarantees of HC tasks?
- **I3**. How can we derive offline schedulability analysis from the derived conditions in I2?

The next subsection explains EDF-VD that is a basis for MC−FLEX, and Section 4 proposes MC−FLEX that addresses I1–I3.

3.2 Recapitulation of EDF-VD Scheduling

To develop MC−FLEX scheduling framework, we utilize the EDF-VD [3] scheduling mechanism, whose algorithm and offline schedulability analysis are simple yet optimal in the speedup factor and competitive in empirical evaluation. Due to its simplicity, EDF-VD has been extended into various directions (e.g., the constrained-deadline task model [5], [20] and imprecise computation model [21]). Based on the classical EDF dynamic priority scheduling algorithm, EDF-VD introduced the concept of *virtual deadline*, which is the modified scheduling deadline in the EDF priority-driven scheduling policy.

Capturing the characteristics of MC tasks that HC tasks are subject to different WCET requirements in different modes, EDF-VD assigns different absolute deadlines to jobs of each HC task in different modes (i.e., the absolute virtual deadline in LO mode and the absolute real deadline in HI mode). Since the WCET requirement of HC task τ_i is usually increased (C_i^L to C_i^H) at each mode switch,[1] we need to complete the job execution earlier in LO mode, which will provide a time duration for the HC task to finish the remaining WCET requirement $(C_i^H - C_i^L)$ if a mode switch happens. For this reason, we assign the virtual relative[2] deadline of the task by $V_i := xT_i$ where x is the virtual-deadline coefficient ($x \in \mathbb{R}$ s.t. $0 < x \leq 1$). Even if the HC task mode switches at its virtual deadline (i.e., a job's release time plus the virtual relative deadline), it still has time duration $(T_i - V_i)$ for its remaining execution. We note that the virtual-deadline coefficient is a controllable scheduling parameter for EDF-VD, which can be derived later in an offline schedulability test.

Scheduling Policy. EDF-VD employs a system-level mode switch mechanism (i.e., when a single HC task switches to HI mode, all the other HC tasks switch to HI mode simultaneously). EDF-VD schedules the job of the earliest effective deadline (i.e., the release time plus the virtual relative deadline if a HC task in HI mode, the release time plus the period in LO mode).

Offline Schedulability Analysis. We present the offline schedulability condition of EDF-VD in the next lemma and the assignment of the virtual-deadline coefficient.

Lemma 1 (from [3]). *A task set τ is schedulable by EDF-VD if the following two inequalities hold:*

$$U_{LC}^L + \frac{U_{HC}^L}{x} \leq 1, \tag{1}$$

$$xU_{LC}^L + U_{HC}^H \leq 1. \tag{2}$$

We can derive the virtual-deadline coefficient from Eq. (2) [3]: $x = \min\big(1, (1 - U_{HC}^H)/U_{LC}^L\big)$.

4 MC−FLEX Scheduling: Basic Version

We now develop the MC−FLEX scheduling framework that employs a task-level mode switch mechanism, based on EDF-VD explained in Section 3.2. To this end, this section presents a basic version of the MC−FLEX scheduling algorithm and its schedulability analysis, under two assumptions to be presented. Then, Section 5 will show how to relax the assumptions.

4.1 Scheduling Algorithm

We develop the MC−FLEX scheduling algorithm extending EDF-VD as follows. On the one hand, the prioritization policy for MC−FLEX exactly follows that for EDF-VD. This means, we assign different deadlines to jobs of each HC task depending on its task mode: the *virtual deadline* in LO mode and the real deadline in HI mode. For HC task τ_i, the relative virtual deadline of the task (V_i) is assigned by $V_i = xT_i$, where x is the virtual-deadline coefficient ($x \in \mathbb{R}$ s.t. $0 < x \leq 1$); whenever a job of a HC task τ_i is released at t_a in LO mode, its virtual deadline $(t_a + V_i)$ is used for prioritization instead of its real deadline $(t_a + T_i)$. For MC−FLEX, x is set to $\min(1, (1 - U_{HC}^H)/U_{LC}^L)$, which is the same as that of EDF-VD; we will explain how to set x later in Section 4.2.

On the other hand, the mode switch mechanism for MC−FLEX is totally different from that for EDF-VD, as it

1. In classic MC systems, only switch-forward is considered.

2. From now on, if the term "deadline" means "absolute deadline", we omit "absolute"; on the other hand, if the term "deadline" means "relative deadline", we specify "relative".

operates in a task-level manner. As to switch-forward, whenever a HC task in LO mode executes for more than its L-WCET, the HC task itself changes its task mode from LO to HI while other HC tasks keep their task mode unchanged. For switch-backward, whenever a HC task in HI mode reaches its job deadline, the HC task itself changes its task mode from HI to LO while other HC tasks keep their task mode unchanged.

From now on, we detail the MC−FLEX scheduling algorithm, under the following two assumptions: one regarding runtime inter-arrival time between switch-back and switch-forward, and the other regarding the ratio of L- and H-WCET.

A1. When a switch-back occurs at t, no switch-forward happens until $(t + \max_{\tau_i \in \tau_{HC}} V_i)$.

A2. Each HC task τ_i satisfies $C_i^L / x \leq C_i^H$.

The assumptions are applied due to the following reasons. A1 enables to simplify tricky cases that make it difficult develop a tight schedulability condition (e.g., a switch-forward occurs right after a switch-back). A2 allows not to consider the case where a LO-mode bandwidth of a HC task is larger than its HI-mode bandwidth. Note that the advanced version of MC−FLEX to be presented in Section 5 will relax the above assumptions.

We first present the runtime scheduling policy of the MC−FLEX scheduling algorithm.

Runtime Scheduling Policy. MC−FLEX schedules the job of the earliest effective deadline (which is the same as EDF-VD) with the following instructions.

P1. Initially, all LC tasks are active and all HC tasks are in LO mode.

P2. When a job of HC task τ_i arrives at time t_a, schedule the task with its virtual deadline $(t_a + V_i)$ in LO mode and with its real deadline $(t_a + T_i)$ is in HI mode.

P3. When a job of LC task τ_i arrives at time t_a, schedule the task with its real deadline $(t_a + T_i)$.

P4. When a job of HC task τ_i in LO mode executes for more than its C_i^L (i.e., a switch-forward for τ_i occurs), set $M_i :=$ HI and drop active LC task(s) selected by the MC−FLEX task dropping algorithm to be presented.

P5. When a job of HC task τ_i in HI mode reaches its deadline (i.e., a switch-backward for τ_i occurs), set $M_i :=$ LO and resume dropped LC task(s) selected by the MC−FLEX task resuming algorithm to be presented.

P6. When no job of any task waits in the ready queue (when the system is idle), the system changes its state to the initial state (all LC tasks are active and all HC tasks are in LO mode).

Before presenting task dropping/resuming algorithms, we introduce *system state* that captures the dynamic system behavior at mode switch, including the task mode for each HC task and the execution state for each LC task.

Definition 1 (System State). *S_j indicates the system state after the jth mode switch, and is defined as a four-tuples of disjoint sets: $S_j = (\tau_{H1}, \tau_{H2}, \tau_{L1}, \tau_{L2})$ where*

- *τ_{H1}: a set of HC tasks whose task mode is* LO,
- *τ_{H2}: a set of HC tasks whose task mode is* HI,
- *τ_{L1}: a set of LC tasks whose execution state is "active," and*
- *τ_{L2}: a set of LC tasks whose execution state is "dropped."*

By definition, $\tau_{HC} = \tau_{H1} \cup \tau_{H2}$ and $\tau_{H1} \cap \tau_{H2} = \emptyset$ hold, and $\tau_{LC} = \tau_{L1} \cup \tau_{L2}$ and $\tau_{L1} \cap \tau_{L2} = \emptyset$ hold. Note that the initial system state is $S_0 = (\tau_{HC}, \emptyset, \tau_{LC}, \emptyset)$.

Similarly to U_{LC}^L, U_{HC}^L and U_{HC}^H, we define collective utilization of each of the above disjoint task sets

$$U_x^y \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_x} u_i^y,$$

where $x \in \{L1, L2, H1, H2\}$ and $y \in \{L, H\}$.

Considering that individual tasks that belong to τ_{H1}, τ_{H2}, τ_{L1} and τ_{L2} contribute to different resource demand (i.e., execution requirement for a unit of period), we may express a MC-schedulability condition at a time instant using the following inequality:

$$\mathcal{F}(U_{L1}^L, U_{L2}^L, U_{H1}^L, U_{H2}^H) \leq 1. \tag{3}$$

We will derive the function $\mathcal{F}(\cdot)$ in Section 4.2.

Using the function, we present the MC−FLEX task dropping/resuming algorithm as follows:

- *Task dropping algorithm*: repeat to drop an LC task from the active LC task set (i.e., τ_{L1}) until Eq. (3) becomes satisfied.[3]
- *Task resuming algorithm*: repeat to resume an LC task in the dropped LC task set (i.e., τ_{L2}) as long as Eq. (3) is still satisfied.[4]

Then, whenever P4 or P5 happens at runtime, we can choose the LC task to be dropped or resumed using the following two criteria.

C1. Choose the LC task with the highest utilization[5] (i.e., C_i^L / T_i) among the active LC task set (i.e., τ_{L1}) for the task dropping algorithm, and the LC task with the lowest utilization among the dropped LC task set (i.e., τ_{L2}).

C2. Choose the LC task with the highest value of task utilization multiplying task period[6] (i.e., $C_i^L / T_i \cdot T_i = C_i^L$) among the active LC task set (i.e., τ_{L1}) for the task dropping algorithm, and the LC task with the lowest value of task utilization multiplying task period (i.e., C_i^L) among the dropped LC task set (i.e., τ_{L2}).

C1 and C2 are heuristic approaches in minimizing the deadline miss ratio of LC tasks' jobs, which are designed as follows. C1 aims at achieving the minimum number of LC tasks whose execution state is "droppped" in a greedy manner, while C2 aims at achieving the minimum number of jobs whose invoking LC tasks' execution state is "dropped" in a greedy manner. Section 6 will show the performance of

3. A feasible system can execute all HC tasks of any task mode by dropping all LC tasks in the worst case.

4. A feasible system can resume to execute all LC tasks when all HC task are switched back.

5. As a task with a higher utilization is dropped, the total number of dropped tasks decreases.

6. As a task with a larger period is dropped, the total number of dropped jobs decreases.

TABLE 1
Task Parameters for Example 2

	T_i	C_i^L	C_i^H	χ_i
τ_1	3	1	1	LO
τ_2	12	1	1	LO
τ_3	4	1	2	HI
τ_4	9	1	2	HI

MC−FLEX with C1 and that with C2, in terms of the deadline miss ratio of LC tasks' jobs.

We explain the MC−FLEX scheduling algorithm using the following example.

Example 2. Consider task set $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ as shown in Table 1. We can compute x by Eq. (8) to be presented in Section 4.2: $x = \min(1, (1 - U_{HC}^H)/U_{LC}^L) = \min\big(1, (1 - 26/36)/(5/12)\big) = 2/3$. We show that assumption A2 holds for τ_3 and τ_4: $1/x \leq 2$. We assign the relative virtual deadline for τ_3 and τ_4: $V_3 = 4/x = 8/3$ and $V_4 = 9/x = 6$.

As shown in Fig. 3, suppose that the first job of τ_3 executed for more than its L-WCET and the other jobs of HC tasks (τ_3 and τ_4) completed their execution within L-WCET. At time 1, the scheduler switches τ_3 forward and checks the online schedulability by Eq. (4) to be presented in Section 4.2: $1/3 + 1/12 + (1/9)/(2/3) + 2/4 > 1$, which means unschedulable without dropping LC tasks. Then, the scheduler drops τ_1 and check Eq. (4): $(2/3) \cdot (1/3) + 1/12 + (1/9)/(2/3) + 2/4 \leq 1$, which means schedulable now. At time 4, the scheduler switches τ_3 back and is able to resume the dropped LC task since Eq. (4) still holds even with the resume of τ_1: $1/3 + 1/12 + (1/4 + 1/9)/(2/3) \leq 1$.

4.2 Schedulability Analysis

In this subsection, we first analyze online schedulability at a specific mode switch, which means whether a given task set is schedulable by MC−FLEX for a given mode switch situation. Then, we can analyze offline schedulability by finding the worst case of mode switches, which means whether a given task set is schedulable by MC−FLEX with any sequence of mode switches.

Online Schedulability Analysis. To describe various system scenarios for MC systems, we define system scenario Z_k as below.

Definition 2 (System Scenario). *For a given task set τ and a series of system states $\{S_0, S_1, \ldots, S_k\}$, system scenario $Z_k = \{S_0, S_1, \ldots, S_k\}$ is defined that the system starts with S_0, changes its state to S_1 after a mode switch, and reaches its state to S_k after k mode switches.*

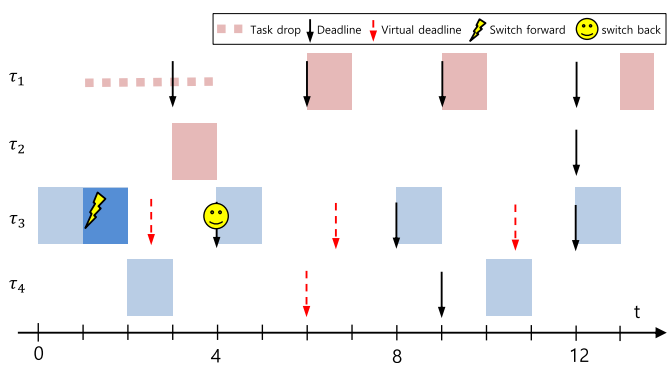


Fig. 3. Schedules by MC − FLEX for Example 2.

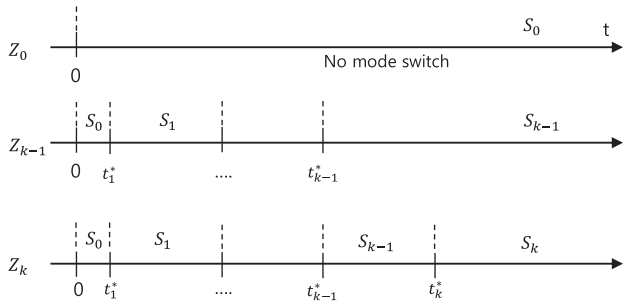


Fig. 4. Illustration of Z_0, Z_{k-1}, and Z_k: note that Z_k is the same as Z_{k-1} except the addition of the kth mode switch.

In other words, Z_k and Z_{k-1} are identical until t_k^* where t_k^* denotes the time instant when the system state is changed from S_{k-1} to S_k under Z_k. At t_k^*, while the system state under Z_k is changed into S_k, that under Z_{k-1} remains the same as shown in Fig. 4. Note that by definition, $Z_0 = \{S_0\}$ and $Z_k = Z_{k-1} \cup S_k$ hold.

To analyze system scenario Z_k, we need to consider online schedulability on two different kinds of system states: the initial state (S_0) and the transitive state (S_k) that is switched from the previous state (S_{k-1}) with $k > 0$, which are described in Fig. 4.

First, we consider schedulability on S_0. We know that S_0 is identical to system-level LO mode in EDF-VD [3].

Lemma 2 (From [3]). *Consider a task set τ. Under system state S_0 (defined in Definition 1), the task is schedulable by* MC−FLEX *if*

$$U_{LC}^L + \frac{U_{HC}^L}{x} \leq 1.$$

Proof. From the sustainability property of preemptive uniprocessor EDF, we can schedule LC tasks with their original period and HC tasks with their reduced period (due to the relative virtual deadline). Then, the schedulability condition in Lemma 2 can be derived from the utilization bound result of EDF. Its detail can be found in Theorem 1 of Baruah *et al.* [3]. □

Next, we consider schedulability on S_k with $k > 0$.

Theorem 1. *Consider a task set τ and its system scenario Z_k. Suppose that the task set is MC-schedulable by* MC−FLEX *on Z_{k-1} with $k > 0$. Then, τ is MC-schedulable by* MC−FLEX *on Z_k if*

$$U_{L1}^L + \frac{U_{H1}^L}{x} + xU_{L2}^L + U_{H2}^H \leq 1. \tag{4}$$

To prove the above theorem, we consider system scenario Z_k (defined in Definition 2). We know that Z_k is the same as Z_{k-1} except addition of the last (i.e., kth) mode switch in Z_k. Assume that the last mode switch happens at time t_k^* (called mode switch time instant) and the system state is changed from S_{k-1} to S_k at t_k^*. Then, we know that S_{k-1} of Z_k is identical to S_{k-1} until t_k^* of Z_{k-1}.

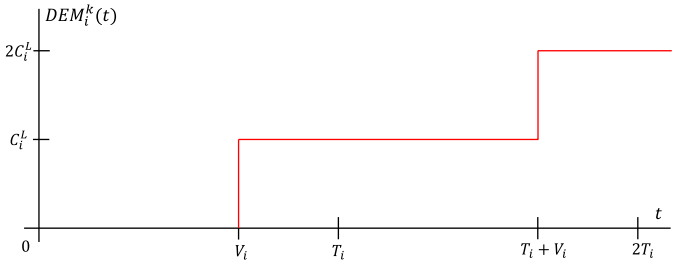


Fig. 5. Illustration of the demand of task τ_i over time interval $[0, t)$ under system scenario Z_k, i.e., $\mathsf{DEM}_i^k(t)$.

To check the schedulability on Z_k, we need to bound the resource demand (i.e., execution requirement) on the system. Since the collective resource demand can be derived from the resource demand of each task, we define the demand of a task as follows:

Definition 3 (Resource Demand). $\mathsf{DEM}_i^k(t)$ *is defined as the demand of task* τ_i *over time interval* $[0, t)$ *under system scenario* Z_k*, which is the amount of necessary execution of jobs of* τ_i *within* $[0, t)$ *in order to avoid any job deadline miss of* τ_i *under the* MC−FLEX *scheduling algorithm. Until the first mode switch,* $\mathsf{DEM}_i^k(t)$ *is similar to the demand of* τ_i *in* $[0, t)$ *under single-criticality systems; for example, if* $\tau_i \in \tau_{HC}$ *invokes its jobs in a strictly periodic manner from 0,* $\mathsf{DEM}_i^0(t)$ *is calculated by* $\left\lfloor \frac{t+T_i-V_i}{T_i} \right\rfloor \cdot C_i^L$*, as shown in Fig. 5 (which is similar to [22]). After some mode switches, it is challenging to develop how to calculate* $\mathsf{DEM}_i^k(t)$*, which will be explained in Eq. (5) as well as Lemmas 4 and 5.*

The amount of the resource demand of a task (shown in Definition 3) is changed by each mode switch: when a HC task switches forward, the demand of the task is changed from L-WCET (C_i^L) to H-WCET (C_i^H); after switch-back, the demand of the task is changed from H-WCET (C_i^L) to L-WCET. To calculate the amount, we need to define the time instant when the demand is changed as follows.

Definition 4 (The demand change time instant). *Let* t'_k *indicate the start time when the resource demands is changed due to the* k*th mode switch.*

In scenario Z_k, consider the kth (last) mode switch occurs at t_k^* as shown in Fig. 6, and let J_k^* denote the job that incurs the kth mode switch from S_{k-1} to S_k at t_k^*. For switch-forward, we set the release time of J_k^* to t'_k, because the WCET requirement of the switch-forwarding job is changed from L-WCET to H-WCET and the job is released before the mode switch. For switch-back, we set the deadline of J_k^* to t'_k because the WCET requirement of jobs following the switch-back job is changed from H-WCET to L-WCET and the following jobs are released after the deadline of the job. Note that for switch-back, t'_k is the same as t_k^* by P5 of the runtime scheduling policy as shown in Fig. 6b.

With the demand change time instant, we can compute the demand before t'_k. For task τ_i, and system scenarios Z_{k-1} and Z_k, we have

$$\forall t \le t'_k, \quad \mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t). \tag{5}$$

This is because of the definition of Z_{k-1} and Z_k. Until t_k^*, all the situations in Z_{k-1} are the same as those in Z_k, and $t'_k \le t_k^*$

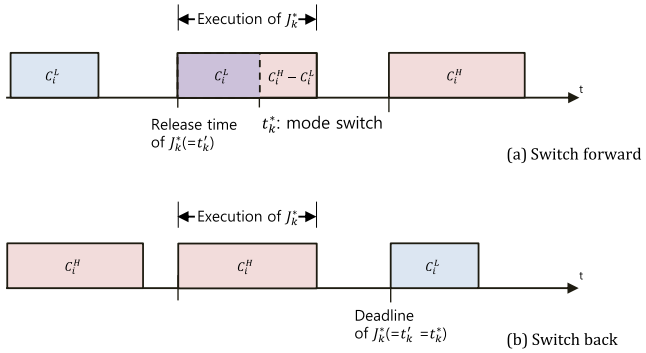


Fig. 6. Demand change time instant in Definition 4.

holds for both switch-forward and backward. For initialization, we have $\mathsf{DEM}_i^0(0) = 0$ for any task τ_i and the initial system scenario Z_0.

To tightly upper-bound $\sum_{\tau_i \in \tau} \mathsf{DEM}_i^k(t)$, we need to maximize the interval in which each HC task exhibits LO mode as much as possible. Targeting J_k^* that incurs switch-forward at t_k^*, we can tightly upper-bound $\sum_{\tau_i \in \tau} \mathsf{DEM}_i^k(t)$ in an interval between t'_k (i.e., the virtual deadline of J_k^*, which is also the demand change time instant) and t_k^*, as long as there exists no task $\tau_i \in \tau_{H1}$ that incurs its switch-forward in the interval. The next lemma shows a property that every LO-mode HC task ($\tau_i \in \tau_{H1}$) does not change its mode in $[t'_k, t_k^*)$. Then, the lemma enables to tightly upper-bound the resource demand of a LO-mode HC task at a mode switch instant without the history of previous mode switches of the task.

Lemma 3. *For system scenario* Z_k*, assume that the* k*th (last) mode switch happens at* t_k^* *and the mode switch type is switch-forward. Consider a HC task with* LO *mode* ($\tau_i \in \tau_{H1}$) *at* t_k^**, which does not invoke* J_k^* *(i.e., job incurring the last mode switch). Then, the task stays in* LO *mode during* $[t'_k, t_k^*)$*.*

Proof. From assumption A1, we know that τ_i executes in LO mode from the system initialization or switches back to LO mode at least $\max_{\tau_i \in \tau_{HC}} V_i$ earlier than t_k^* (i.e., the last switch-forward time instant of Z_k).

Let τ_j denote the task that invokes J_k^*. Since the switch-forward of J_k^* happens before its virtual deadline ($t_k^* - t'_k \le V_j$) and $V_j \le \max_{\tau_i \in \tau_{HC}} V_i$ holds, we conclude that τ_i executes in LO mode during $[t'_k, t_k^*)$. □

Using the above lemma, we derive the following relationship between $\mathsf{DEM}_i^k(\cdot)$ and $\mathsf{DEM}_i^{k-1}(\cdot)$.

Lemma 4. *For system scenario* Z_k *and time* $t > t'_k$*, the following condition holds for a HC task:*

$$\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + (t - t'_k) \cdot u_i^L / x,$$
$$\mathsf{DEM}_i^k(t) \le \mathsf{DEM}_i^{k-1}(t'_k) + (t - t'_k) \cdot u_i^H,$$

Proof. We can compute the demand within time $[0, t'_k]$ by Eq. (5) using the previous system scenario Z_{k-1}. For the demand in time $[t'_k, t)$, which is the changed demand associated with a new mode switch in Z_k, we divide cases depending on which task mode the task belongs to.

Case (a) Consider a task $\tau_i \in \tau_{H1}$, which can be divided into two subcases according to whether J_k^* is an instance of τ_i or not.

Case (a1) (J_k^* is an instance of τ_i)**.** Since the task is in LO mode in S_k, we know that J_k^* incurs a switch-back. After the mode change instant ($t_k^* = t'_k$), the jobs of τ_i execute its L-WCET (C_i^L) until its virtual deadline. Then, we have

$$\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + \frac{u_i^L}{x}(t - t'_k).$$

Case (a2) (J_k^* is not an instance of τ_i)**.** First, consider that J_k^* incurs a switch-back. Since τ_i is in LO mode after $t_k^* = t'_k$, we have

$$\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + \frac{u_i^L}{x}(t - t'_k).$$

Second, consider that J_k^* incurs a switch-forward. Since τ_i remains in LO mode in $[t'_k, t_k^*)$ (by Lemma 3) and τ_i is LO mode in $[t_k^*, t)$ by the case, we have

$$\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + \frac{u_i^L}{x}(t - t'_k).$$

Case (b) Consider a task $\tau_i \in \tau_{H2}$, which can be also divided into two subcases according to whether J_k^* is an instance of τ_i or not.

Case (b1) (J_k^* is an instance of τ_i)**.** Since the task is in HI mode in S_k, we know that J_k^* is the switch-forward job. Consider time t such that $t \le t_k^*$. Before the last mode switch time instant (t_k^*), the task τ_i needs to finish its execution for L-WCET within in its relative virtual deadline (V_i). By assumption A2, we have $C_i^L/x \le C_i^H$, which is equivalent to $C_i^L/(x \cdot T_i) \le C_i^H/T_i$, yielding $u_i^L/x \le u_i^H$. Then, we have

$$\begin{aligned}\mathsf{DEM}_i^k(t) &\le \mathsf{DEM}_i^{k-1}(t'_k) + u_i^L/x(t - t'_k)\\ &\le \mathsf{DEM}_i^{k-1}(t'_k) + u_i^H(t - t'_k). \qquad \text{(by A2)}\end{aligned}$$

Consider the other time t (i.e., $t > t_k^*$). Due to the last switch-forward, the task may execute up to its H-WCET (C_i^H) within its period (T_i) after the last mode switch time instant (t_k^*). Then, we have

$$\mathsf{DEM}_i^k(t) \le \mathsf{DEM}_i^{k-1}(t'_k) + u_i^H(t - t'_k).$$

Case (b2) (J_k^* is not an instance of τ_i)**.** First, consider that J_k^* incurs a switch-back. Since τ_i is in HI mode in $[t_k^* = t'_k, t)$, we have

$$\mathsf{DEM}_i^k(t) \le \mathsf{DEM}_i^{k-1}(t'_k) + u_i^H(t - t'_k).$$

Second, consider that J_k^* incurs a switch-forward. By Lemma 3, τ_i is in HI mode in $[t'_k, t_k^*)$; by the case, τ_i is also in HI mode in $[t_k^*, t)$. Therefore,

$$\mathsf{DEM}_i^k(t) \le \mathsf{DEM}_i^{k-1}(t'_k) + u_i^H(t - t'_k).$$

□

Lemma 5. *For system scenario Z_k and time $t > t'_k$, the following condition holds for a LC task.*

$$\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + (t - t'_k) \cdot \begin{cases} u_i^L, & \tau_i \in \tau_{L1},\\ x \cdot u_i^L, & \tau_i \in \tau_{L2}.\end{cases}$$

Proof. We can compute the demand within time $[0, t'_k]$ by Eq. (5). For the demand in time $[t'_k, t)$, we divide cases depending on which execution state the task belongs to.

Case (a). Consider a task $\tau_i \in \tau_{L1}$. Since the task is active at and after t'_k, we have $\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + u_i^L(t - t'_k)$.

Case (b). Consider a task $\tau_i \in \tau_{L2}$. Since τ_i's execution state is "dropped" after t_k^*, its execution state can be active in $[t'_k, t_k^*)$. This means, τ_i's execution state is changed into "dropped" due to a HC task's switch-forward at t_k^*. Then, for τ_i to contribute to the demand in $[t'_k, t]$, its job deadline should be earlier than the HC task's virtual deadline (which is no later than t). Therefore, τ_i's job deadline should be no later than $t'_k + x \cdot (t - t'_k)$. Then, τ_i can contribute to the demand for up to $u_i^L \cdot x \cdot (t - t'_k)$, and therefore we have $\mathsf{DEM}_i^k(t) = \mathsf{DEM}_i^{k-1}(t'_k) + u_i^L \cdot x \cdot (t - t'_k)$. □

Based on these lemmas, we will prove Theorem 1.

Proof of Theorem 1 The base case (i.e., MC-schedulablity under Z_0) holds by Lemma 2. Now, we prove whether τ is MC-schedulable under Z_k, assuming τ is MC-schedulable under Z_{k-1} (i.e., $\mathsf{DEM}_i^{k-1}(t) \le t$ holds for all $t > 0$) and Eq. (4) holds under Z_k. We will divide cases depending on t.

Case 1 ($t \le t'_k$).

$$\sum_{\tau_i \in \tau} \mathsf{DEM}_i^k(t) = \sum_{\tau_i \in \tau} \mathsf{DEM}_i^{k-1}(t), \qquad (by\ Eq.\ (5))$$

which is less than or equal to t by the assumption of MC-schedulability under Z_{k-1}.

Case 2 ($t > t'_k$).

$$\begin{aligned}\sum_{\tau_i \in \tau} \mathsf{DEM}_i^k(t) &\le \sum_{\tau_i \in \tau} \mathsf{DEM}_i^{k-1}(t'_k) + (U_{L1}^L + \frac{U_{H1}^L}{x} + xU_{L2}^L\\ &\quad + U_{H2}^H)(t - t'_k) \qquad (by\ Lemmas\ 4,\ 5)\\ &\le t'_k + \left(U_{L1}^L + \frac{U_{H1}^L}{x} + xU_{L2}^L + U_{H2}^H\right)(t - t'_k),\\ &\quad (by\ the\ supposition\ of\ Theorem\ 1, i.e.,\\ &\quad \text{MC-schedulability under } Z_{k-1})\end{aligned}$$

which is less than or equal to t if Eq. (4) holds. □

By Theorem 1, τ is MC-schedulable by MC−FLEX if MC−FLEX employs the task dropping/resuming algorithm associated with Eq. (4); in other words, $\mathcal{F}(\cdot)$ in Eq. (3) is set to the left-hand-side of Eq. (4).

For each mode switch, the complexity of the runtime scheduling algorithm is $O(n)$ because it depends on the task dropping/resuming algorithm, where Eq. (4) requires the task parameters of each task in τ_{H1}, τ_{H2}, τ_{L1}, and τ_{L2}: $|\tau_{H1}| + |\tau_{H2}| + |\tau_{L1}| + |\tau_{L2}| = n$. If switch-forward happens k times on runtime, the MC−FLEX task dropping/resuming algorithm is invoked at most $2 \cdot k$ times (the system invokes the switch-back mechanism at most k times.)

Offline Schedulability Analysis. Although we derived online schedulability analysis at each specific mode switch situation, we need to develop offline schedulability analysis, which indicates whether a task set is MC-schedulable (defined in Section 2) by MC−FLEX under any sequences of mode switches. To this end, we need to find the worst-case conditions of all possible mode switch sequences when the

scheduler drops/resumes LC tasks by the online schedulability test (i.e., Eq. (4)), recorded in the following theorem.

Theorem 2. *A task set τ is MC schedulable by* MC−FLEX *that employs the task dropping/resuming algorithm associated with Eq. (4), if the following two conditions hold:*

$$U_{LC}^{L} + \frac{U_{HC}^{L}}{x} \leq 1, \tag{6}$$

$$xU_{LC}^{L} + U_{HC}^{H} \leq 1. \tag{7}$$

Proof. For MC schedulability, we need to show that both conditions MC-A and MC-B in Section 2 are satisfied. From Eq. (6), τ is schedulable on S_0 by Lemma 2, which corresponds condition MC-B. For condition MC-A, we show that Eq. (4) holds with any $\tau_{H2} \neq \emptyset$ by Theorem 1. By assumption A2, the situation where all HC tasks are in LO mode yields the largest value for HC tasks to contribute to the left-hand-side of Eq. (4). In this case, the smallest value for LC tasks to contribute to the left-hand-side of Eq. (4) occurs when all LC tasks' execution mode is "dropped". Considering that the dropping/resuming algorithm can adjust the execution mode of LC tasks based on the task mode of HC tasks, we need $xU_{LC}^{L} + U_{HC}^{H} \leq 1$, which is Eq. (7). □

Similar to EDF-VD, we derive the virtual-deadline coefficient of MC−FLEX from Eq. (7) and the range of x $(0 \leq x \leq 1)$

$$x = \min\big(1, (1 - U_{HC}^{H})/U_{LC}^{L}\big). \tag{8}$$

We show how Theorem 2 works in the following example.

Example 3. Consider task set $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ in Example 2. From Eq. (6), we have $1/3 + 1/12 + (1/4 + 1/9)/x \leq 1$, i.e., $13/21 \leq x$. From Eq. (7), we have $x \cdot (1/3 + 1/12) + 2/4 + 2/9 \leq 1$, i.e., $x \leq 2/3$. By Theorem 2, the task set is schedulable when $13/21 \leq x \leq 2/3$, approximately, $0.619 \leq x \leq 0.667$.

For a given task set, the complexity of offline schedulability test (i.e., Theorem 2) is $O(n)$ because Eqs. (6) and (7) require investigating the task parameters of all tasks in τ_{HC} and τ_{LC}: $|\tau_{HC}| + |\tau_{LC}| = n$.

5 MC−FLEX Scheduling: Advanced Version

In this section, we present the advanced version of MC−FLEX by relaxing the two assumptions (i.e., A1 and A2) of the basic version presented in Section 4. First, we introduce a concept of virtual criticality mode to relax A1. We then present how to handle fixed-mode high-criticality tasks separately, yielding relaxation of A2 as well as improvement of the schedulability of MC−FLEX.

5.1 Virtual Criticality Mode

This subsection aims at relaxing the assumption of A1 in MC−FLEX: the inter-arrival time between a switch-back and the next mode switch should be upper-bounded by $(t + \max_{\tau_i \in \tau_{HC}} x \cdot T_i)$. Relaxing the assumption may invalidate the online and offline schedulability analysis (i.e., Theorems 1 and 2), because the relaxation compromises Lemma 3 and consequently Lemmas 4 and 5. Without the lemmas, the online and offline schedulability analysis will be much more pessimistic as the runtime utilization of τ_{H1} cannot be bounded. To resolve the problem, we introduce a concept of virtual mode.

Definition 5 (Virtual Mode). *For task τ_i, we define the virtual mode M_i', which is used for the online schedulability test (i.e., Eq. (4)). At the switch-forward of τ_i, we set $M_i' :=$ HI, which is the same as the task mode for the switch-forward (i.e., $M_i :=$ HI). However, while we set $M_i :=$ LO at the switch-backward of τ_i, we set $M_i' :=$ LO after waiting for the longest relative virtual deadline ($\max_{\tau_i \in \tau_{HC}} V_i$) from the switch-back of τ_i.*

Using the virtual mode (that delays the mode transition of switch-back), we bound the runtime utilization of τ_{H1}, which improves the offline schedulability and reduces the online job deadline miss ratio.

Under the virtual mode, we separate the switch of criticality mode and the resuming of LC tasks. We present runtime scheduling policy of the advanced MC−FLEX, which is modified from P5 in the runtime scheduling policy in Section 4.1:

P5′. When a job of HC task τ_i in HI mode reaches its deadline (i.e., a switch-backward for τ_i occurs) at t, set $M_i :=$ LO. At the virtual switch-back of the task at $(t + \max_{\tau_i \in \tau_{HC}} V_i)$, set $M_i' =$ LO, and resume dropped LC task(s) selected by the MC−FLEX task resuming algorithm presented in Section 4.

For task dropping/resuming algorithms, we need to consider the virtual mode of HC tasks in Eq. (4).

Based on P5′, we replace the proof of Lemma 3, which does not rely on the assumption of A1 in the basic MC−FLEX.

Proof of Lemma 3 By P5′, we know that τ_i stays in LO mode from the system start or switches back to LO mode at least $(\max_{\tau_i \in \tau_{HC}} V_i)$ earlier than t_k^* (i.e., the last switch-forward time instant of Z_k).

Let τ_j denote the task that invokes J_k^*. Since the switch-forward of J_k^* happens before its virtual deadline $(t_k^* - t_k' \leq V_j)$ and $V_j \leq \max_{\tau_i \in \tau_{HC}} V_i$ holds, we conclude that τ_i executes in LO mode during $[t_k', t_k^*)$. □

We explain the virtual mode in the following example.

Example 4. Consider task set $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ in Example 2. Since $x = 2/3$, we have $V_3 = 8/3, V_4 = 6$. As shown in Fig. 7, suppose that the first job of τ_3 executed for more than its L-WCET and the other jobs of HC tasks (τ_3 and τ_4) completed their execution within L-WCET. At time 1, the scheduler switches τ_3 forward and drops τ_1 by Eq. (4). The scheduler switches τ_3 back (from HI mode to LO mode) at time $t^{SB} = 4$ (real switch-back). A virtual switch-back of τ_3 is invoked at time 10 since $t^{SB} + \max_{\tau_i \in \tau_{HC}} V_i = 4 + 6 = 10$. Then, the scheduler resumes τ_1 at time 10 (the next job of τ_1 released at time 12 will be serviced) by Eq. (4): $1/3 + 1/12 + (1/4 + 1/9)/(2/3) \leq 1$.

5.2 Fixed-Mode High-Criticality Tasks

In this subsection, we present how to relax the assumption A2 and how to improve offline schedulability. As investigated in Lee *et al.* [12], a scheduling algorithm with a naive

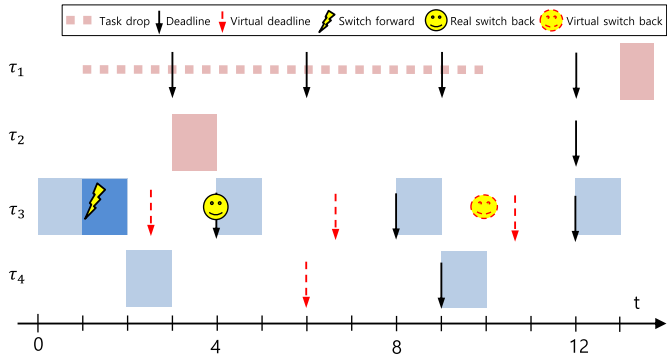


Fig. 7. Schedules with a virtual switch-back for Example 4.

task-level mode switch may have schedulability performance lower than approaches with the system-level mode switch because they have different worst-case mode switch patterns. This necessitates a modification of a naive task-level mode switch policy such that some HC tasks initiate its state at HI mode and never be mode-switched, which outperforms the existing system-level mode switch approaches. This policy relaxes the assumption of A2 in the basic MC−FLEX (Section 4). We apply such a task-level mode switch idea from Lee *et al.* [12], to MC−FLEX.

We define the fixed-mode tasks for HC tasks that execute only in HI mode.

Definition 6. *Fixed-mode tasks (τ_F) are a set of HC tasks s.t. $C_i^L/V_i > C_i^H/T_i$: $\tau_F = \{\tau_i \in \tau_{HC} \mid u_i^L/x > u_i^H\}$. At runtime, fixed-mode tasks are executed only in* HI *mode and not switched back to* LO *mode.*

Next, we present the revised runtime scheduling policy (i.e., P1 in Section 4.1), where the basic MC−FLEX runtime scheduling policy is modified with the fixed-mode task.

P1′. Initially, all LC task are active. All fixed-mode HC tasks (τ_F) are in HI mode and other HC tasks ($\tau_H \setminus \tau_F$) are in LO mode.

Then, the advanced version of MC−FLEX has the runtime scheduling policy consisting of P1′, P2, P3, P4, P5′, and P6, which does not necessitate any assumption such as A1 and A2 in Section 4. When it comes to the online and offline schedulability tests of the advanced version, the former remains the same while the latter should be changed. That is, dues to the fixed-mode task, S_0 and the worst case mode switch pattern of the advanced version is different from that of the basic version, which derives the following theorem.

Theorem 3. *A task set τ is MC-schedulable by the advanced* MC−FLEX *that employs the task dropping/resuming algorithm assicated with Eq. (4), if the following two conditions hold:*

$$U_{LC}^L + \sum_{\tau_i \in \tau_{HC} \setminus \tau_F} \frac{u_i^L}{x} + \sum_{\tau_i \in \tau_F} u_i^H \leq 1, \tag{9}$$

$$xU_{LC}^L + U_{HC}^H \leq 1. \tag{10}$$

Proof. The proof of Theorem 3 is the same as that of Theorem 2 except the fact that some HC tasks (i.e., τ_F) always stay in HI mode. In this case, Eq. (6) should be modified to accommodate tasks in τ_F; by moving the contribution of tasks τ_F from τ_{H1} to τ_{H2}, Eq. (9) holds. □

From the modified offline schedulability, we confirm that the virtual-deadline coefficient assignment is not changed between the basic and advanced versions of MC−FLEX. We show how to apply Theorem 3 in the following example.

Example 5. Consider task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ where τ_1 is $(T_i = 3, C_i^L = 1, C_i^H = 1, \chi_i = \text{LO})$, τ_2 is $(8, 1, 4, \text{HI})$, and τ_3 is $(12, 3, 4, \text{HI})$. We can compute $x = (1 - 5/6)/(2/6) = 1/2$. We know that τ_3 is a fixed-mode task since $(3/12)/(1/2) > 4/12$. Since Eqs. (9) and (10) hold (i.e., $1/3 + (1/8)/(1/2) + 4/12 \leq 1$ and the choice of x), the task set is MC schedulable by Theorem 3.

6 Evaluation

In this section, we evaluate performance of MC−FLEX (the advanced version in Section 5) compared to the existing approaches. To show that our approach does not sacrifice offline schedulability of the base algorithm, we compare MC−FLEX with EDF-VD [3] (i.e., the base algorithm of MC−FLEX), MC-ADAPT [12] (i.e., our previous work based on EDF-VD with task-level mode switch), and FMC [16] (i.e., an existing task-level mode switch algorithm based on EDF-VD) in terms of MC-schedulability, based on synthetic workloads.[7] Next, we evaluate the runtime performance (i.e., job deadline miss ratio) of MC−FLEX in comparison with EDF-VD, MC-ADAPT, and FMC, via runtime simulation with synthetic workloads; we consider the following two different task dropping/resuming algorithms for MC−FLEX:

- MC−FLEX−C1: task dropping/resuming algorithm to minimize the number of dropped LC tasks (i.e., C1 in Section 4.1).
- MC−FLEX−C2: task dropping/resuming algorithm to minimize the number of dropped jobs of LC tasks (i.e., C2 in Section 4.1).

6.1 Simulation Setup

In this subsection, we describe simulation setup regarding task set generation and runtime simulation.[8]

Task Set Generation. We generate random task sets according to the workload-generation algorithm [3], [11], [12], [16]. Let U^b be the upper bound of both collective utilization of MC tasks in LO mode (i.e., $U_{LC}^L + U_{HC}^L$) and collective utilization of MC tasks in HI mode (i.e., U_{HC}^H). A random task is generated as follows (note that 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 chosen in the range $[20, 150]$.
- R_i (the ratio of u_i^H/u_i^L) is a real number drawn from the range [1,4].

7. We exclude FMC-MST [17] and Boudjadar *et al.* [18] from comparison because the former is based on multi-criticality systems and the latter applies fixed-priority scheduling.

8. Our simulation code is available at https://github.com/icpslab/mcflex.

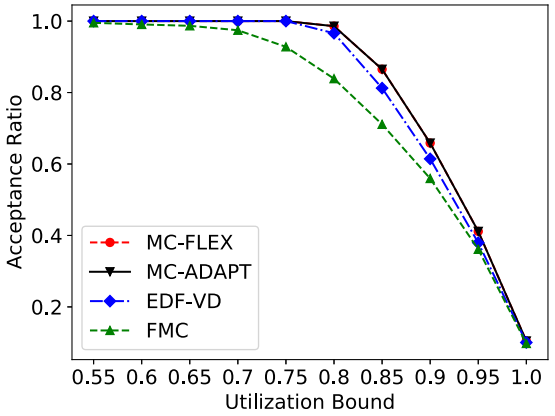


Fig. 8. Acceptance ratio of different scheduling algorithms under varying utilization bound (MC-FLEX and MC-ADAPT are identical).

- P^{HC} (the probability of being a HC task) is set to 0.25, 0.5 and 0.75; note that P^{HC} is set to 0.5 unless specified. We uniformly generate a real number in $[0.0, 1.0]$. If the number is not larger than P^{HC}, we set $\chi_i :=$ HI, $C_i^H := \lfloor u_i \cdot T_i \rfloor$, and $C_i^L := \lfloor u_i \cdot T_i \cdot R_i \rfloor$. Otherwise (i.e., the number is larger than P^{HC}), we set $\chi_i :=$ LO and $C_i^L := \lfloor u_i \cdot T_i \rfloor$.

Each task set is constructed by repeating generation of a task until $\max(U_{LC}^L + U_{HC}^L,\ U_{HC}^H) > U^b$ holds and discarding the task added last.

Runtime Simulation. After evaluating the offline schedulability of MC−FLEX by simply applying Theorem 3, we will evaluate the performance of MC−FLEX in terms of the deadline miss ratio (DMR)[9] for jobs of LC tasks, via runtime simulation (Figs. 9, 10, 11, 12, and 13). For a randomly-generated synthetic workload, which is schedulable by MC-ADAPT, we simulate the behavior of tasks with a given probability of switch-forward for any HC task's job, denoted as P^{SF}; note that P^{SF} is set to 0.2 unless specified. We simulate each synthetic workload with each scheduling algorithm (either EDF-VD, MC-ADAPT, FMC, MC−FLEX−C1 or MC − FLEX− C2) for 32,000 time units, to be justified later with Fig. 13.

In simulation for runtime performance, we consider two kinds of runtime simulation:

- The existing deterministic runtime environment (DRE): jobs of dropped LC tasks never execute. The system only allows to execute jobs which can complete their execution. This setting is widely used for evaluating runtime performance of MC systems (e.g., [12], [16]).
- A new best-effort runtime environment (BRE): jobs of the dropped LC tasks may execute only when no job of reserved tasks (i.e., all HC tasks and LC tasks with the active state) waits for execution. The system executes jobs in the best effort manner. Jobs executed in the best-effort manner will be prioritized according to EDF.

6.2 Simulation Results

We evaluate the performance of MC−FLEX in terms of offline schedulability (Fig. 8), the runtime performance of MC−FLEX via DRE runtime simulation (Fig. 9), and the runtime performance of MC−FLEX via BRE runtime simulation (Figs. 10, 11, 12, and 13).

Acceptance Test for Offline Schedulability. Fig. 8 shows the acceptance ratio (the ratio of schedulable task sets over total task sets) varying utilization bound U^b from 0.55 to 1.0 in step of 0.05. The number of task sets for each data point is 5,000 task sets and the number of total task sets are 50,000 (which is also applicable to Figs. 9, 10, 11, 12, and 13). In the figure, FMC shows the lower acceptance ratio than EDF-VD because FMC applies task-level mode switch without applying the fixed-mode HC tasks.[10] MC-ADAPT and MC−FLEX dominate EDF-VD by applying their fixed-mode HC task strategy. Since we aim to improve runtime performance of LC tasks without sacrifice in schedulability, we confirm that MC−FLEX has acceptance ratio identical to MC-ADAPT.

DRE Runtime Simulation for Deadline Miss Ratio. Fig. 9 shows the average DMR of LC tasks' jobs under varying utilization bound U^b for different probabilities of switch-forward: $P^{SF} = 0.05$, $P^{SF} = 0.2$, and $P^{SF} = 0.5$, according to Gu *et al.* [11]. The result shows that task-mode approaches (FMC, MC-ADAPT, MC−FLEX−C1, and MC−FLEX−C2) significantly outperform the system-mode approach (EDF-VD) in all cases. FMC and MC-ADAPT show identical DMR performance because we conjecture their online schedulability test is equivalent. Both MC−FLEX approaches outperform MC-ADAPT and FMC in most cases. In Fig. 9c, MC−FLEX−C1 has higher DMR than MC-ADAPT when $U^b = 0.95$. This is because MC-ADAPT employs a task dropping strategy similar to C2, while MC−FLEX−C1 employs C1 as a task dropping/resuming strategy. MC−FLEX−C2 exhibits slightly better performance than MC−FLEX−C1 regardless of P^{SF}. As P^{SF} increases, the performance gap increases.

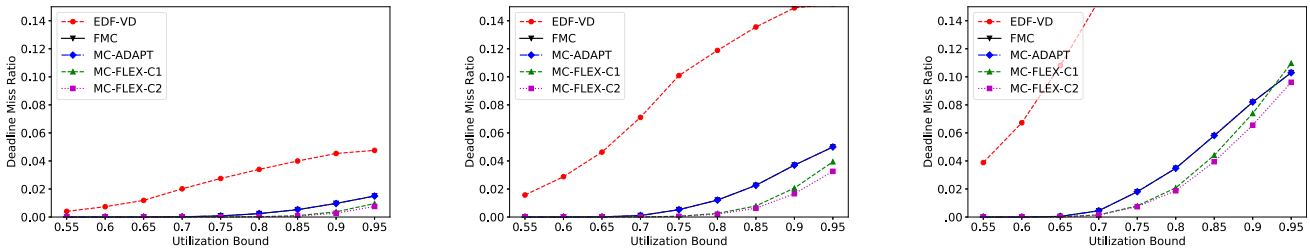
In lower utilization bound ($U^b < 0.65$), all algorithms except EDF-VD have no deadline miss because their algorithms pass the online schedulability test (i.e., Eq. (4)) without dropping any LC tasks. In higher utilization bound ($U^b \geq 0.65$), MC−FLEX−C2 reduces the DMR of MC−FLEX−C1 by up to 28.0%.

BRE Runtime Simulation for Deadline Miss Ratio. Fig. 10 shows the average DMR of LC tasks' jobs under varying utilization bound U^b for different probabilities of switch-forward. The result shows similar trends with Fig. 9. Due to the best-effort job executions, the performance gap between MC−FLEX and MC-ADAPT is larger compared to Fig. 9. In BRE, MC−FLEX−C1 dominates MC-ADAPT and FMC even for $U^b = 0.95$ and $P^{SF} = 0.5$. This implies that, whether the task-level resuming strategy in MC−FLEX is applied or not has a greater impact on the DMR than which dropping/resuming approaches are selected (between C1 and C2). In higher utilization bound ($U^b \geq 0.65$), MC−FLEX−C2 reduces the DMR of MC-ADAPT by at least 54.8%. In Fig. 10a, MC−FLEX−C2 reduces the DMR of MC−FLEX−C1 by up to 77.4%

Fig. 11 shows the average DMR of LC tasks' jobs under varying utilization bound U^b for different probabilities of being a HC task: $P^{HC} = 0.25$, $P^{HC} = 0.5$, and $P^{HC} = 0.75$. As P^{HC} is closer to either 0.0 or 1.0, the DMR of all algorithms

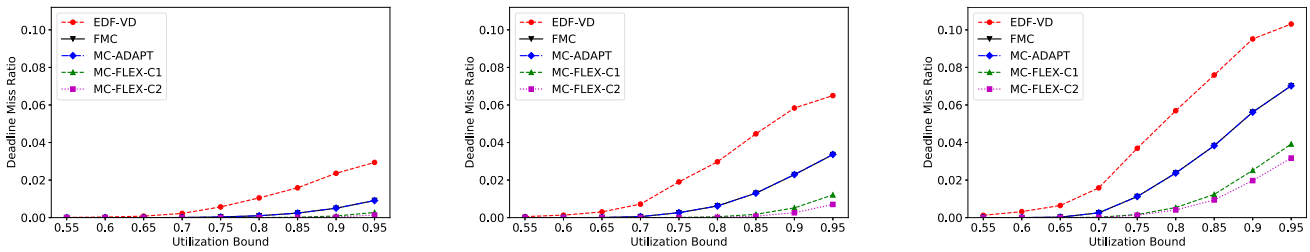
9. DMR is the ratio of the number of the unfinished jobs over the total number of jobs released in a given time interval.

10. The schedulability anomaly is explained in the beginning of Section 5.2.



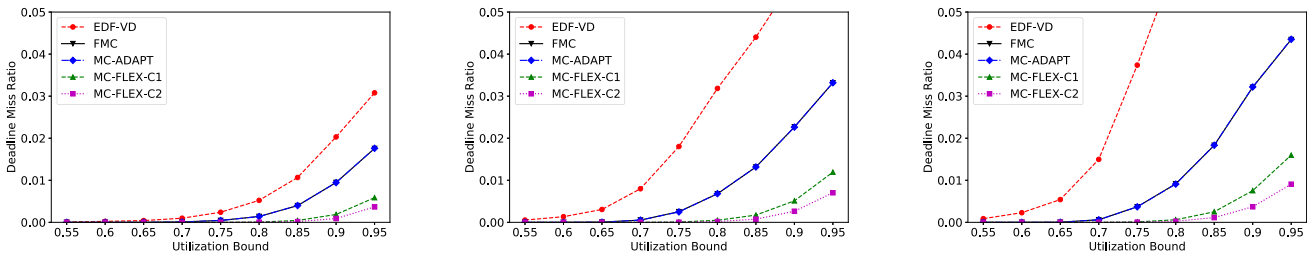
(a) $P^{SF} = 0.05$ (b) $P^{SF} = 0.2$ (c) $P^{SF} = 0.5$

Fig. 9. (DRE runtime) Deadline miss ratio of LC tasks' jobs with different P^{SF} (FMC and MC-ADAPT are identical).



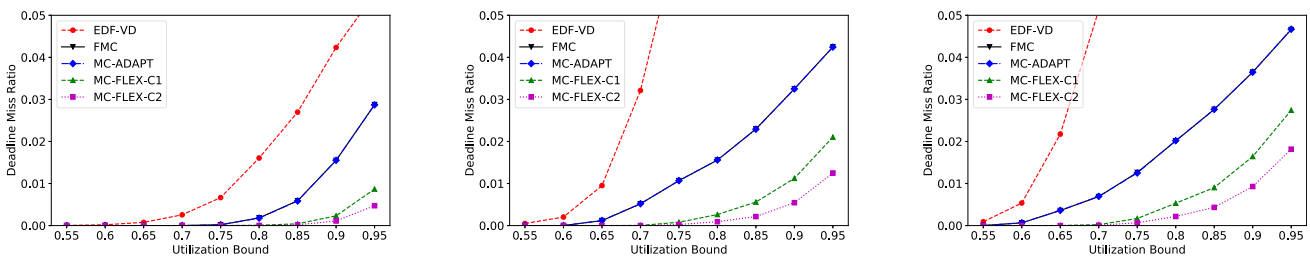
(a) $P^{SF} = 0.05$ (b) $P^{SF} = 0.2$ (c) $P^{SF} = 0.5$

Fig. 10. (BRE runtime) Deadline miss ratio of LC tasks' jobs with different P^{SF} (FMC and MC-ADAPT are identical).



(a) $P^{HC} = 0.25$ (b) $P^{HC} = 0.5$ (c) $P^{HC} = 0.75$

Fig. 11. (BRE runtime) Deadline miss ratio of LC tasks' jobs with different P^{HC} (FMC and MC-ADAPT are identical).



(a) The range of $R_i = [1, 2]$ (b) The range of $R_i = [2, 3]$ (c) The range of $R_i = [3, 4]$

Fig. 12. (BRE runtime) Deadline miss ratio of LC tasks' jobs with different range of R_i (FMC and MC-ADAPT are identical).

decreases because the task set is closer to a non-MC task set. In higher utilization bound ($U^b \geq 0.65$), MC−FLEX−C2 reduces the DMR of MC-ADAPT by at least 78.8%. In Fig. 11a, MC−FLEX−C2 reduces the DMR of MC−FLEX−C1 by up to 85.3%. For $U^b = 0.65$ in Figs. 11b and 11c, MC− $FLEX - C1$ shows slightly better performance than MC−FLEX−C2, which means that MC−FLEX−C2 is not always better task dropping/resuming strategy compared to MC−FLEX−C1.

Fig. 12 shows the average DMR of LC tasks' jobs under varying utilization bound U^b for different range of R_i: [1,2], [2,3], and [3,4]. As R_i is closer to 1.0, the DMR of all algorithms decreases because HC tasks are similar to normal non-MC tasks due to a smaller gap between L-WCET and H-WCET, which requires less LC task dropping. In higher utilization bound ($U^b \geq 0.65$), MC−FLEX−C2 reduces the DMR of MC-ADAPT by at least 61.1%. In Fig. 12b, MC−FLEX−C2 reduces the DMR of MC−FLEX−C1 by up to 68.6%.

BRE Simulation Duration Test. Fig. 13 shows the average DMR of LC tasks' jobs under different simulation duration: 4,000, 8,000, 16,000, 32,000, 64,000, and 128,000. In this experiment, we use the following parameters: $U^b = 0.90$, $P^{SF} = 0.3$, and $P^{HC} = 0.5$. In the results, the DMR of all approaches is slightly higher in smaller simulation duration (e.g., 4,000, 8,000). In smaller duration, the performance is much affected by the worst case release pattern (critical instant). The DMR of all algorithms is converged in a large simulation duration (over 32,000 time units).

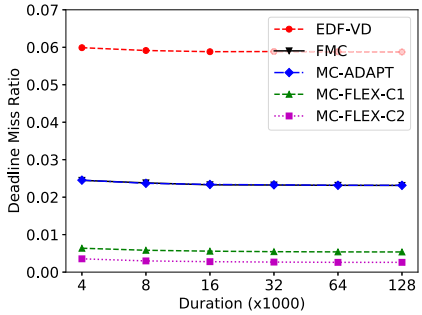


Fig. 13. (BRE runtime) The deadline miss ratio of LC tasks' jobs with different simulation duration (FMC and MC-ADAPT are identical).

7 Related Work

Since Vestal introduced a concept of MC real-time systems [1], a number of studies have addressed MC real-time scheduling (see [23] for a survey). Baruah *et al.* [2] introduced runtime criticality mode to identify the time instant when a task executes for more than low-confidence WCET. Most of early MC studies [2], [3], [5], [24] employed the system-level mode switch mechanism: once a single high-criticality task violates its low-confidence WCET, all the other high-criticality tasks will simultaneously enter the high-criticality mode. For static certification, they took the pessimistic strategy of dropping all low-criticality tasks immediately.

Recent MC studies considered graceful degradation to provide better runtime performance, which is not considered in static verification domain. There is a method to delay dropping low-criticality tasks by adjusting the threshold of mode switch in offline computation [25] or runtime computation [7]. Degraded service is provided to low-criticality tasks after system-level mode switch: stretching their periods [4], [8], [9], [26], lowering their priorities [4], skipping their jobs [6], reducing their execution times [7], [21], [27], or providing more resource supply with workload shaper [28], [29]. Some studies [10], [11], [12], [15], [16], [18], [30], [31] relaxed the system-level mode switch assumption and considered task-level mode switch: one task's violation of low-confidence WCET does not affect other tasks. It enables low-criticality tasks to be penalized selectively in the event of individual task mode switch.

To provide flexibility in MC systems, an efficient mechanism to recover degraded service of low-criticality tasks is necessary when the system becomes stable (not high-criticality mode). Buruah *et al.* [3] proposed a simple solution that the system can be transited to low-criticality mode and resume to execute low-criticality tasks when the system is idle. Bate *et al.* [13] presented a scheduling protocol for returning to the low-criticality mode so as to resume the execution of low-criticality tasks. Guo *et al.* [14] bounds the time duration from high-criticality mode to the system idle time. Boudjadar *et al.* [18] proposed switch-back at the absolute deadline of the job which triggers system-level mode switch under fixed-priority scheduling.

For further flexibility, this paper proposes a mechanism to resume each low-criticality task independently. Although some existing studies [3], [13], [14], [16], [17], [18] searched a time instant to resume all low-criticality tasks at once, this paper proposes an algorithm to identify a time instant to resume a subset of low-criticality tasks when resource is available. To this end, this paper develops an efficient switch-back mechanism through task-level mode switch.[11] With the task-level resuming algorithm and the task-level switch-back mechanism, this paper minimizes the job deadline miss ratio of low-criticality tasks without scarifying MC schedulability.

There has been MC work on providing pre-defined QoS guarantee on LC tasks [14], [21]. They have investigated trade off between schedulability and guarantee on low-criticality tasks, which can be complements to this paper that yields no schedulability loss.

8 Discussion

Extension to Multi-Criticality Systems. While this paper considers dual-criticality systems only, we can generalize our MC−FLEX scheduling framework toward multiple levels of criticality with the consideration of multiple WCET estimates in the MC task model. For example, we may follow how FMC [16] that considers dual-criticality levels is extended to multiple criticality levels in FMC-MST [17]. In future work, we will extend online and offline schedulability analysis (i.e., Theorems 1 and 3) according to the multi-criticality environment.

The Runtime Overheads of MC−FLEX . The runtime scheduling algorithm of MC−FLEX consists of the EDF-VD scheduling algorithm (as a prioritization policy) *and* C1 and C2 (as task dropping/resuming algorithms). The task dropping/resuming algorithms are invoked every mode switch time instant with additional computation resource to determine which LC task is dropped/resumed. The required computation resource is acceptable because the complexity of online schedulability test in the task dropping/resuming algorithm is $O(n)$, which traverses the status of all MC tasks. In the future, we will investigate the possibility of enabling the runtime scheduling algorithm to exhibit less runtime overhead without compromising schedulability performance.

9 Conclusion

We present the MC−FLEX framework that employs a task-level mode transition mechanism (supporting both switch-forward and switch-backward) and minimizes the job deadline misses of low-criticality tasks at runtime. To reduce deadline misses, we present a runtime mechanism that determines which low-criticality tasks are dropped/resumed at each switch-forward/backward. Simulation results show that our proposed framework reduces the job deadline miss ratio of low-criticality tasks at runtime (by over 54.8% compared to the existing work), without any loss of offline schedulability.

As future work, we have a plan to apply our framework on real-world testbed such as automotive systems (e.g.,[32], [33]) or locomotive systems. During a real-world case study, we can extend our framework considering practical issues such as adaptive variable-rate tasks in engine components [34].

Acknowledgments

This research was supported in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2020-

11. The existing work with switch-back through task-level mode switch [16], [17], [18] does not support to resume each low-criticality tasks independently.

0-01655) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). This research was also supported in part by the National Research Foundation of Korea (NRF) funded by the MSIT under Grants 2021R1A2B5B02001758 and 2018R1C1B5083050. This work was also supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry & Energy (MOTIE) of the Republic of Korea under Grant 20199710100060.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. Real Time Syst. Symp.*, 2007, pp. 239–243.
[2] S. Baruah, A. Burns, and R. Davis, "Response-Time analysis for mixed criticality systems," in *Proc. Real Time Syst. Symp.*, 2011, pp. 34–43.
[3] S. Baruah *et al.*, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. Euromicro Conf. Real-Time Syst.*, 2012, pp. 145–154.
[4] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Proc. Workshop Mixed Criticality Syst.*, 2013, pp. 1–6.
[5] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. Euromicro Conf. Real-Time Syst.*, 2012, pp. 135–144.
[6] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proc. Real-Time Netw. Syst.*, 2015, pp. 237–246.
[7] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *Proc. Real Time Syst. Symp.*, 2016, pp. 47–56.
[8] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality systems," in *Proc. Workshop Mixed Criticality Syst.*, 2013, pp. 1–6.
[9] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2014, pp. 1–10.
[10] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph - A new specification for mixed-criticality systems," in *Proc. Emerg. Technol. Factory Automat.*, 2013, pp. 1–8.
[11] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *Proc. Euromicro Conf. Real-Time Syst.*, 2015, pp. 13–24.
[12] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 163:1–163:21, Sep. 2017.
[13] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *Proc. Euromicro Conf. Real-Time Syst.*, 2015, pp. 259–268.
[14] Z. Guo, K. Yang, S. Vaidhun, S. Arefin, S. K. Das, and H. Xiong, "Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 373–383.
[15] A. V. Papadopoulos, E. Bini, S. Baruah, and A. Burns, "AdaptMC: A control-theoretic approach for achieving resilience in mixed-criticality systems," in *Proc. 30th Euromicro Conf. Real-Time Syst.*, 2018, pp. 14:1–14:22.
[16] G. Chen *et al.*, "Utilization-based scheduling of flexible mixed-criticality real-time tasks," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 543–558, Apr. 2018.
[17] G. Chen, N. Guan, B. Hu, and W. Yi, "EDF-VD scheduling of flexible mixed-criticality system with multiple-shot transitions," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2393–2403, Nov. 2018.
[18] J. Boudjadar, S. Ramanathan, A. Easwaran, and U. Nyman, "Combining task-level and system-level scheduling modes for mixed criticality systems," in *Proc. IEEE/ACM 23rd Int. Symp. Distrib. Simul. Real Time Appl.*, 2019, pp. 1–10.
[19] A. Burns, R. I. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1478–1491, Oct. 2018.
[20] A. Easwaran, "Demand-Based scheduling of mixed-criticality sporadic tasks on one processor," in *Proc. Real Time Syst. Symp.*, 2013, pp. 78–87.
[21] D. Liu, *et al.*, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," in *Proc. Real Time Syst. Symp.*, 2016, pp. 35–46.
[22] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. Real-Time Syst. Symp.*, 1990, pp. 182–190.
[23] A. Burns and R. Davis, "Mixed criticality systems – A review," 2019. [Online]. Available: http://www-users.cs.york.ac.uk/burns/review.pdf
[24] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *Proc. Real Time Syst. Symp.*, 2011, pp. 13–23.
[25] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP," in *Proc. Euromicro Conf. Real-Time Syst.*, 2012, pp. 155–165.
[26] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proc. Des., Automat., Test Eur.*, 2013, pp. 147–152.
[27] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *Proc. 28th Euromicro Conf. Real-Time Syst.*, 2016, pp. 131–138.
[28] S. Tobuschat, M. Neukirchner, L. Ecco, and R. Ernst, "Workload-aware shaping of shared resource accesses in mixed-criticality systems," in *Proc. Int. Conf. Hardware/Softw. Codes. Syst. Synth.*, 2014, pp. 1–10.
[29] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive workload management in mixed-criticality systems," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 1, pp. 1–27, Oct. 2016.
[30] J. Ren and L. T. X. Phan, "Mixed-Criticality scheduling on multiprocessors using task grouping," in *Proc. Euromicro Conf. Real-Time Syst.*, 2015, pp. 25–34.
[31] H. Choi, H. Kim, and Q. Zhu, "Job-class-level fixed priority scheduling of weakly-hard real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2019, pp. 241–253.
[32] M. O'Kelly *et al.*, "F1/10: An open-source autonomous cyber-physical platform," 2019, *arXiv:1901.08567v1*. [Online]. Available: http://arxiv.org/abs/1901.08567
[33] S. Kato, *et al.*, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. ACM/IEEE 9th Int. Conf. Cyber-Phys. Syst.*, 2018, pp. 287–296.
[34] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proc. 19th IEEE Real-Time Syst. Symp.*, 1998, pp. 286–295.

Jaewoo Lee (Member, IEEE) received the BS and MS degrees in computer science and engineering from Seoul National University, Republic of Korea, in 2006 and 2008, respectively, and the PhD degree in computer and information science from the University of Pennsylvania, USA, in 2017. He is currently an assistant professor with Chung-Ang University, Republic of Korea, where he joined in 2018. From 2017 to 2018, he was a postdoctoral research fellow with Seoul National University, Republic of Korea . His research interests include cyber-physical systems, realtime embedded systems, and information system security.

Jinkyu Lee (Senior Member, IEEE) received the BS, MS, and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology, Republic of Korea, in 2004, 2006, and 2011, respectively. He is currently an associate professor with Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea, where he joined in 2014. From 2011 to 2014, he was a visiting scholar or research fellow with the Department of Electrical Engineering and Computer Science, University of Michigan, USA. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in realtime embedded systems, mobile systems, and cyber-physical systems. He was the recipient of Best Student Paper Award from 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011 and Best Paper Award from 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.