A Hybrid Scheduling Mechanism for Multi-Programming in Mixed-Criticality Systems

Mohammed Bawatna, Behnaz Ranjbar and Akash Kumar

August 29, 2022

A Hybrid Scheduling Mechanism for Multi-programming in Mixed-Criticality Systems

Mohammad Bawatna, Behnaz Ranjbar, and Akash Kumar
Chair for Processor Design, CFAED, Technische Universität Dresden, Dresden, Germany
{mohammed.bawatna, behnaz.ranjbar, akash.kumar}@tu-dresden.de

Abstract—**In the last decade, the rapid evolution of the Commercial-Off-The-Shelf (COTS) platforms led safety-critical systems towards integrating tasks and applications with different criticality levels in a shared hardware platform, i.e., Mixed-Criticality Systems (MCS)s. Therefore, several scheduling algorithms and approaches have been proposed upon a commonly used model, i.e., Vestal's model. However, consolidating software functions onto shared processors cannot be implemented directly in real-life applications and industrial systems while complying with certification requirements. The existing scheduling approaches do not provide a simple solution for eliminating the interference effect among the tasks with different criticality levels on the shared processing resources. Moreover, the system mode switch guarantees the timing constraints of the high-criticality tasks throw the termination of the low-criticality tasks. In this paper, we developed a new scheduling algorithm that addresses these challenges based on the round-robin technique, which improves the overall schedulability. We compared the proposed algorithm against existing scheduling algorithms in both academia and industry using extensive experiments to evaluate it. Our results show improvements in the schedulability from 0.8% to 14.0% and from 2.7% to 10.7% compared to the conventional Earliest Deadline First with Virtual Deadline (EDF-VD) and Fixed Priority Preemptive (FPP) scheduling approaches, respectively.**

Index Terms—**Mixed-Criticality; Real-time; Round Robin Scheduling; Earliest Deadline First Scheduling; Fixed Priority Scheduling;**

I. Introduction

SAFETY-CRITICAL systems play an essential role in many industrial domains such as automotive systems, consumer electronics, avionics, and defense systems. In these domains, embedded software must pass a strict certification process according to different criticality levels [1]. Integrating these applications of different levels of criticality onto a shared hardware platform while ensuring timeliness is the main goal for all the research and publications in the field of Mixed-Criticality Systems (MCS)s. Missing deadline for tasks in MCS has a different impact that may cause catastrophic consequences. The key mechanism to achieving this goal is by designing an efficient scheduling algorithm. The scheduling problem of MC systems has been intensively studied in recent years (see Section II for a brief review). Most existing solutions for scheduling are based on Vestal's model [2] that we will summarise below. Moreover, they disregard the fact that not all task's deadlines are equally critical, and these approaches vary in complexity and accuracy as in [3]. As a result, these scheduling approaches did not address the need for multi-programming tasks isolation on the shared system resources.

This work is supported by a grant from Software Campus through the German Federal Ministry of Education and Research, under the project SARA: Safety-Aware Relocation of functions in a multi-core computer Architecture.

Vestal's model. This model [2] assumes that the system has a static set of sporadic tasks and several execution modes. Each task is characterized by its deadline, period, criticality level, and a set of estimated Worst-Case Execution Times (WCET)s. As the mode of execution gets higher, the WCET is more pessimistic. If any task overruns its execution time during the run-time, the system must switch to a higher mode. As a result, all tasks with low criticality level will be suspended.

MCSs challenges. In most MC systems, the timing correctness of High-Criticality (HC) tasks is guaranteed at the expense of Low-Criticality (LC) tasks though completely discarding the LC tasks and dedicating the system resources for meeting the timing constraints of HC tasks. Moreover, the limited resources in most MCSs, such as the processing and memory spaces in the integrated architecture become a bottleneck in the architectural design for real-life applications. The second challenge for the multi-programming in the MCS environment is the system mode switch, which is triggered when a deadline of HC task is missed. The system mode switch guarantees the timing constraints for the HC tasks throw isolation from the LC tasks. However, it causes the termination of the LC tasks. Task isolation is an essential requirement in multi-programming to avoid missing deadlines caused by data movement that can block a shared resource or consume too high processor execution time by LC task(s). Besides, the failure of an LC task must be isolated from HC tasks, such that they cannot delay the activities of each other as in [4]. Moreover, the increasing computational demand of the tasks can delay each other due to contention on caches, memory, and processing unit(s) resources. Eliminating the interference effect is a challenge due to the uncertainty concerning the state of the resources.

Motivation and contributions. In safety-critical domains, tasks with different levels of criticality require careful design and analysis, where the failures in an LC task affect one or more HC tasks. As a result, there is a need for techniques that guarantee the timing constraints and comply with certification requirements that are derived from the industrial standards. The existing scheduling approaches do not provide a simple solution for the multi-programming challenges that share the same system resources. Therefore, this paper aims to propose an efficient scheduling mechanism based on the round-robin technique to solve the challenges mentioned above while keeping low complexity. The contributions achieved by this work are stated as follows:

1) In this paper, we propose a scheduling mechanism that eliminates the interference effect among tasks with different safety criticality levels based on the round-robin

technique. The task set is partitioned into two subsets: HC and LC, each of which is assigned a part of the total processor utilization (U). Based on the type of the tasks at each of these two partitions, a scheduling algorithm is then applied independently.

2) We improve the schedulability of LC tasks in High critical mode (HI mode) by controlling the time slot of the overall LC tasks during the run-time phase based on statistical analysis of the response times. In this approach, the LC tasks are still guaranteed some service in the HI mode.
3) We propose the task drop algorithm that minimizes the effect of system mode switches on the LC tasks and allows the maximum number of LC tasks to continue running without being degraded.

Structure of the paper. The rest of this paper is structured as follows. In Section II, the related work is discussed. Second, in Section III, we explain the task model, assumptions used, and a brief review of the algorithms that we used for the comparison, which are Earliest Deadline First with Virtual Deadline (EDF-VD) and Fixed Priority Preemptive (FPP). Next, in Section IV, we will introduce our proposed algorithm. After that, in Section V, we present the simulation results and discuss the analysis. Future work and conclusions are offered in Sections VI and VII.

II. Related Works

In the last decade, most of the MC scheduling algorithms in the real-time community are based on a model proposed by Vestal [2], which was briefly discussed in the introduction. In this section, we will summarize the approaches that solve the MC challenges on a single-core processor. Vestal discussed in his model two system modes: Low critical (LO) and HI, on a single-core processor. In this model, Vestal showed that the Deadline Monotonic (DM) policy is not optimal for MCSs. This model was extended to multiple system modes as in [3]. In this paper, we discuss the MC preemptive scheduling techniques on a single-core processor, such as FPP scheduling [5] and EDF-VD [6]. In [7] and [5], the authors studied the response times analysis for scheduling tasks with two levels of criticality, LC and HC under the fixed-priorities. In their work, if the system is in LO mode, LC WCETs are used to determine the priorities. If the system mode changes from LO to HI mode, the LC tasks are dropped and HC tasks use another priority assignment. Moreover, they showed in their study a method for FP assignment of periodic tasks with more than two criticality levels. In [8], the authors presented advanced analysis for the FP scheduling with various levels of execution. In [9], the execution demands of MC systems under EDF-VD scheduling systems was studied for the case of two criticality levels. The authors proposed the demand bound functions for the LO and the HI modes on a single-core processor system. In [10], a similar technique called ECDF for the case of two criticality levels was presented.

Although tasks isolation, which is one of the main subjects in this paper, is an essential requirement in industrial MCSs, most research focuses on schedulability and shared resource management among the tasks. In contrast, these publications did not discuss the data and fault isolation. Tasks isolation in the field of MCS was discussed via two approaches: physical and virtual isolation. In the first approach, the isolation is performed by allocating unique hardware resources to tasks with different criticality levels. However, due to the lack of a hardware platform mainly designed for MCS, we could find very little research work as in [11] and [12] that discussed the ability to implement this approach using TrustZone technologies. In the second approach, which is well-known by the MCS community, virtual separation is performed by partitioned hardware using logical isolation (virtual machines (VMs)), which allows running multiple software components on the same hardware platform as in [13]. However, VMs requires complicated resource management, which affects the performance and predictability of the real-time system. On the other hand, limited memory availability and unpredictability issues are handled by existing memory reservation techniques as in [14] and [15] . In section IV, we present a scheduling mechanism that preserves the isolation among tasks on the shared processing resources based on round-robin technique and still guaranteed some service for the LC tasks in the HI mode. The second challenge we addressed in this work is to improve the schedulability of LC tasks in HI mode, which was discussed in [16] by suggesting three approaches to keep some LC tasks during the HI mode. The three approaches are: to change the priority of LC tasks, to extend the periods of LC tasks when the system mode changes to HI mode, and to reduce the execution budget of LC tasks when the system mode switches. The first two approaches are not suitable for most of real-life applications which prefer an on-time result with a degraded quality rather than a delayed result with a perfect quality. In this work, we consider the third approach which improves the schedulability of LC tasks by controlling the ration between the HC and LC time slots as we will present in section IV.

III. Concepts, Models and Assumptions

In this section, we introduce the notations and models and present the preliminary knowledge of the scheduling algorithms that we used compared with priority-driven and deadline-driven algorithms. The selected algorithms were the FPP and the EDF-VD. Further terminology will be introduced as it gets necessary along with this paper.

A set of related jobs which provide a function is called a task. Tasks are classified into periodic, sporadic, and aperiodic tasks. In periodic tasks, the jobs are always released at exact points in time, where inter-arrival time T_i between any two consecutive jobs is constant. Table I shows some simple notation in the glossary below. In sporadic tasks, the jobs are released periodically. However, each job arrives at an unpredictable time, but at least T_i time units from the last job of the same task. Aperiodic tasks are those tasks that have no period of repetition.

MCS Task Models. MCSs consist of two or more different criticality levels. The tasks in MCSs are classified as HC tasks and LC tasks. The classic dual-criticality system model consists of a task set Γ that comprises a set of n tasks $\{\tau 1, ..., \tau n\}$ that can be scheduled on a processing unit(s). The dual-criticality system model is typically adopted in most of the research works in MCSs, because it simplifies the problem to more general cases.

TABLE I: The notation used in the models and assumptions

Ti	Inter-arrival time. If task i is a periodic task, then Ti is the bound on the time between successive arrivals jobs of task i.
Ci	The execution time required by task i on each release.
Di	The deadline of task i, where $D_i \leq T_i$.
Ji	The release jitter time, the time task i can spend waiting after arrival.
WCETi	The worst case execution time required by task i on each release.
Ii	Interference on task i, which is the time that higher priority tasks can prevent i from executing.
Bi	The blocking time on task i which is the longest critical section of lower priority tasks.
Ri	The worst-case response time for a task I. where $R_i \leq D_i$ for schedulable task.
hp(i)	The set of tasks of higher priority than task i.
lp(i)	The set of tasks of lower priority than task i.

In the dual-criticality system model, the system switches between LO and HI modes. There is a mode for each criticality level in the more general MCS model. Initially, HC and LC tasks are scheduled in a shared processing unit(s) in LO mode. In this mode, HC tasks have a lower execution demand. The demand bound function (DBF) of each task is calculated to determine the maximum required execution time of a task τi. If a HC task overruns its lower execution demand, a system switches its mode from LO to HI, where the HC tasks have higher execution demand. Therefore, LC tasks must be dropped to accommodate the HI mode workload. The utilization (U) for the task set is computed as in Eq. (1), where M and X are the system mode and tasks criticalities, respectively as follows: $M \in \{LO, HI\}$ and $X \in \{LC, HC\}$.

$$U_M^X = \sum_{m_i = M} (\frac{C_i^X}{T_i}) \tag{1}$$

Tasks can be either independent or dependent. Each task τi, is characterized, as shown in Table I, by a $WCET_i$ where $WCET_i = \{WCET_i^{LO}\ ,\ WCET_i^{HI}\ \}$, a minimal inter-arrival time T_i, a criticality level X_i, and a deadline D_i. In this work we consider implicit deadlines where $D_i = T_i$. In the dual-criticality system model, each LC task only has a $WCET^{LO}$, and each HC task has a $WCET_i^{LO}$ and $WCET_i^{HI}$, where $WCET_i^{LO} \leq WCET_i^{HI} \leq R_i \leq D_i \leq T_i$.

In MCSs, there are two task models: the task demand model and the task resource model. The operations in the MCS models are summarized as follows: first, the MCS starts in LO mode, where all tasks execute at the $WCET^{LO}$ to guarantee their deadlines. If an LC task overruns its $WCET_i$ in the LO mode, it must be dropped. Second, if any HC task overruns its $WCET_i$, then the MCS switches its mode from LO to HI. LC tasks are dropped in HI mode, and HC tasks must meet their deadlines. When the MCS is in HI mode, the system mode switch from HI to LO can be scheduled in the idle state.

Schedulability Test. A schedulabililty test is concerned with verifying that no task misses its deadline under a specified scheduling algorithm. A MCS is considered schedulable if the worst-case response time R_i is not greater than the deadline D_i ($R_i \leq D_i \leq T_i$) for each τ_i in Γ. In this section, we recapitulate existing response-time analyses for FPP and EDV-VD scheduling algorithms. We use hp(i) and lp(i) to denote respectively the set of indices of tasks with priorities higher than, and lower than that of task τ_i.

A. FPP Schedulability analysis

Fixed-priority preemptive scheduling [5] ensures that the highest priority task in the ready queue will execute first by a system clock interrupt. However, the lower-priority tasks could wait an indefinite amount of time. Therefore, the exact worst-case response time is computed by Eq. (2).

$$R_i = C_i + \sum_{j \in hp(i)} (\frac{R_i}{T_j}) * C_j \tag{2}$$

The first iteration starts with $R_i = C_i$. If the task is non-schedulable, this equation ends with $R_i > D_i$. For periodic tasks with deadlines equal to periods, the utilization bound of FPP is at most 70% for large task sets.

B. EDF-VD Schedulability analysis

The EDF schedules the tasks according to their deadlines, and it is an optimal scheduling algorithm for the independent tasks on a single processor. For periodic tasks with deadlines equal to periods, the utilization bound of EDF is 100% when $U_M^X \leq 1$ as in Eq. (1). The EDF-VD [6] adapts EDF to MCS, where the processor's capacity is limited by shortening the HC deadlines (virtual deadlines) while running in the LO mode. The virtual deadlines are computed as $D_i^v = xT_i$ where x is the scaling factor with x $\in (0, 1)$ for all HC tasks. The tasks are schedulable by the EDF-VD technique if two conditions are valid. Eq. (3) and Eq. (4) show these conditions as follows:

$$U_{LO}^{HC} + U_{LO}^{LC} \leq 1 \tag{3}$$

$$U_{HI}^{HC} \leq 1 \tag{4}$$

For periodic tasks with deadlines equal to periods, the utilization bound of EDF-VD is at most 75% as proposed by Baruah et al. in [3]. The upper and lower bounds of the scaling factor can be computed as shown in Eq. (5) and Eq. (6).

$$\frac{U_{LO}^{HC}}{1 - U_{LO}^{LC}} \leq x \tag{5}$$

$$x \leq \frac{1 - U_{HI}^{HC}}{U_{LO}^{LC}} \tag{6}$$

IV. Proposed Scheduling Algorithm

This section introduces our scheduling mechanism, which facilitates a computational design of MCSs, and isolates the interference between the HC and LC tasks on the shared processing resource. This approach is not only suitable for different types of multi-programming with different characteristics of tasks, but also allows the maximum number of LC tasks to continue running during the HI mode.

To illustrate the proposed mechanism and the equations from Eq. (7) to Eq. (10), we provide an example, as shown

TABLE II: Illustrative example for the proposed mechanism.

#	HC Tasks	LC Tasks	U_{LO}^{HC}	U_{LO}^{LC}	D_{max}^{HC}	D_{max}^{LC}	$U_{LO}^{HC} x D_{max}^{HC}$	$U_{LO}^{LC} x D_{max}^{LC}$	A_{LO}^{HC}	B_{LO}^{LC}
1	$\tau 1$	$\tau 2$, $\tau 3$	0.453	0.4	7	5	3.171	2	0.946	1
2	$\tau 2$	$\tau 1$, $\tau 3$	0.686	0.167	7	6	4.802	1	0.833	1
3	$\tau 3$	$\tau 1$, $\tau 2$	0.567	0.286	6	7	3.402	2	0.882	1
4	$\tau 2$, $\tau 3$	$\tau 1$	0.4	0.453	5	7	2	3.171	1	0.946
5	$\tau 1$, $\tau 3$	$\tau 2$	0.167	0.686	6	7	1	4.802	1	0.833
6	$\tau 1$, $\tau 2$	$\tau 3$	0.286	0.567	7	6	2	3.402	1	0.882

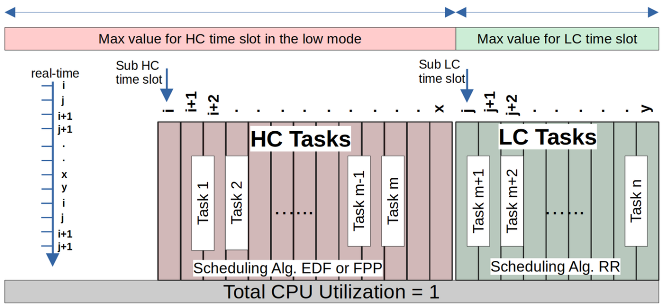

Fig. 1: The maximum values of the HC and LC time slots in the proposed mechanism.

in Table II. For simplicity, we consider the overhead by the preemption, and the operating system is negligible.

lets consider three tasks $\tau 1$, $\tau 2$, and $\tau 3$ which have implicit deadlines. Let the deadline of $\tau 1$ be $D_1 = 5$, and its $WCET1_{LO} = 2$. Let the deadline of $\tau 2$ be $D_2 = 6$, and its execution time be $WCET2_{LO} = 1$. Let the deadline of $\tau 3$ be $D_3 = 7$, and its execution time be $WCET3_{LO} = 2$. Based on Eq. (1), the utilization of the three tasks is 0.853. As shown in Table II, there are six possible cases to divide these three tasks into HC and LC.

The required time slot for running the HC tasks, as shown in Eq. (7), depends on four variables which are: the utilization of the HC tasks (U_{LO}^{HC}), the maximum period in the HC task set (D_{max}^{HC}), the LO window adjustment value for the HC tasks (A^{HC}), and the HI window adjusting value for the HC tasks (α), where α is between 1 and α_{THOLD}.

$$TimeSlot_{HC} = U_{LO}^{HC} * D_{max}^{HC} * A^{HC} * \alpha \tag{7}$$

If the period for each task equals its deadline, then the value of A^{HC} is calculated as in Eq. (8), where $\sum C_{HC}$ is the summation of the execution times for the HC tasks, and β_{HC} is a variable greater than zero which depends on the number of HC tasks per set, $\sum C_{HC}$, the average context switch time, the minimum period in the HC task set, and the selected scheduling algorithm for running the HC tasks.

$$A^{HC} = \frac{\beta_{HC} * \sum C_{HC}}{U_{LO}^{HC} * D_{max}^{HC}} \tag{8}$$

The increase in α value will provide the HC tasks with more processing resources by slightly increasing the HC time slot as shown in Figure 1. In section V, we will show by the simulation results the effect of varying the variables α and β during the run-time phase.

The required time slot for running the LC tasks, as shown in Eq. (9), depends on three variables which are: the utilization of the LC tasks (U_{LO}^{LC}), the maximum period in the LC task(s) set (D_{max}^{LC}), and the LO window adjusting value for the LC tasks (B^{LC}).

$$TimeSlot_{LC} = U_{LO}^{LC} * D_{max}^{LC} * B^{LC} \tag{9}$$

If the period for each task equals its deadline, then the value of B^{LC} is calculated as in Eq. (10), where $\sum C_{LC}$ is the summation of the execution times for the LC tasks, and β_{LC} is a variable greater than zero that depends on the number of LC tasks per set, $\sum C_{LC}$, the context switch time, the minimum period in the LC task set, and the selected scheduling algorithm to run the LC tasks. In this above illustrative example, the values of $\beta_{HC} = \beta_{LC} = 1$.

$$B^{LC} = \frac{\beta_{LC} * \sum C_{LC}}{U_{LO}^{LC} * D_{max}^{LC}} \tag{10}$$

On the LC tasks side, the required time slot for running the LC tasks is upper limited by $\beta_{LC} = 1$. Therefore, the increases in the LC tasks execution times on the shared processing units will not affect the overall performance, which is a major challenge in the conventional algorithms. Figure 1 shows the maximum values of the HC and LC time slots in the LO mode. The values of β_{HC} and β_{LC} divide the overall time slots into sub-time lots according to the request rates of the HC tasks as shown in the same figure. During the run-time phase, the trade off between improving the quality of LC tasks in the HI mode and provide more resources for the HC tasks is controlled by the LC time slot through statistical analysis of the response times of the LC tasks. In this algorithm, we used the values β_{HC} and β_{LC} to further divide the HC and LC time slot into slices based on the periodicity of the HC tasks.

A. *Pre-runtime Processing*

As shown in Figure 2, the algorithm starts by partitioning the tasks according to their criticalities into two groups: HC and LC.

In this work, we consider two types of tasks: independent and dependent tasks. For the independent HC tasks, we selected the EDF-VD which is deadline-driven scheduling algorithm. For the dependent HC tasks, we selected the FPP which is a priority-driven scheduling algorithm. The selection was based on the most widely known in the academic and industrial fields. During the pre-runtime pahse, if an LC task has an input to an HC task, then it is added to the HC task list. After determining the HC and LC tasks sets, a utilization test based on Eq. (1) is performed. The processor time is divided into two parts, where a round-robin mechanism is used to switch between the two subgroups. The time slots for each portion are calculated based on the utilization of each group, as shown in Eq. (7) and Eq. (9).

A scheduling algorithm is then applied to each of these two partitions independently, and the tasks in each division can

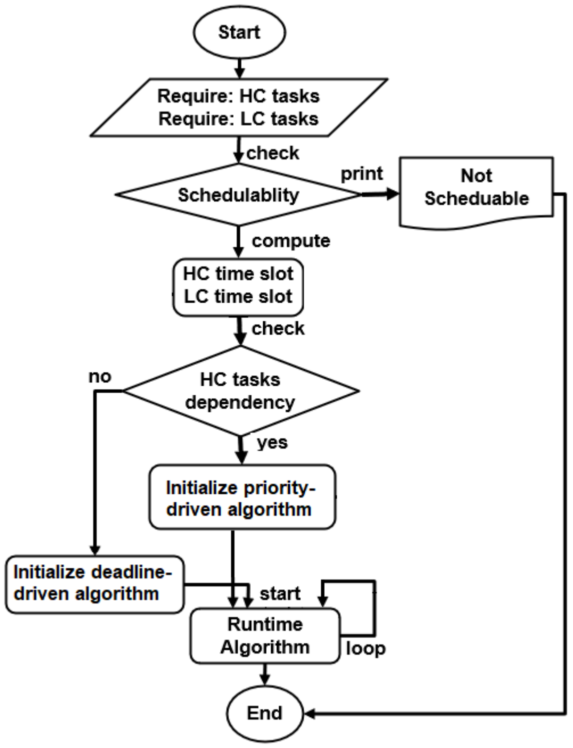

Fig. 2: Flowchart for the pre-runtime processing.

run if they have not used up their assigned time slot. For the independent tasks, the summation of both HC and LC tasks utilization should not exceed 0.75, as was proved in [3]. For the dependent tasks, the summation of both HC and LC tasks utilization should not exceed 0.7, as was proved in [17]. The proposed technique does not require modifying the selected scheduling algorithm within a partition.

B. Runtime Dispatching

During the runtime phase, as shown in Algorithm 1, if a HC task missed its deadline in line 3, then the first step to providing the HC tasks with more processing resources is by increment α in line 5, which will slightly increase the HC time slot.

If the value of α reaches its threshold, then Algorithm 1 will call the task drop procedure (Algorithm 2) in line 8, which will drop (at least) one LC task depending on the HC tasks execution demands. After calling the task drop, the required time for the HC tasks to continue running without missing deadlines is first computer using Eq. (1) in line 2 in Algorithm 2. Then the LC tasks will be sorted in line 4, based on their importance. If all the LC tasks have the same importance, then the selection will be based on the execution times of the LC tasks.

If the execution demands for the HC tasks was increased for a short period of time, the LC tasks will be reactivated, as shown in the lines from 14 to 23 in Algorithm 1, based on the average response times of the HC and LC task sets.

Algorithm 1 Runtime

```
Input: $U_{LM}$ and $U_{HM}$ (HC tasks), $U_{LM}$ (LC tasks)
 1: while (true) do
 2:   for each $\tau_i$ in HC tasks list
 3:     if $R_i > D_i$ then – Deadline is missed
 4:       if $\alpha_{HC} < \alpha_{Threshold}$ then
 5:         increment $\alpha_{HC}$
 6:         Update HC timeslot
 7:       else
 8:         Call TaskDrop function
 9:         Update HC timeslot
10:         Update LC timeslot
11:         Update LC Tasks
12:       end if
13:   end for
14:   for each $\tau_i$ in HC tasks list
15:     if avg$(D_i - R_i) > Thres_{HC}$ then
16:       if $\alpha_{HC} > 1$ then
17:         decrement $\alpha_{HC}$
18:       else
19:         re-activate a LC task
20:         calculate $\alpha_{HC}$
21:       end if
22:     end if
23:   end for
24: end while
```

Algorithm 2 Task drop

```
Input: $U_{LM}$ and $U_{HM}$ (HC tasks), $U_{LM}$ (LC tasks)
Input: $\tau_i$ (missed deadline)
Output: Updated $U_{LM}$ and $U_{HM}$ (HC tasks), $U_{LM}$ (LC tasks)
 1: procedure TASKDROP()
 2:     Calculate $U_s$ of $Task_i$ – Required space
 3:     Check LC portion
 4:     Sort $list$ by importance
 5:     for each $Task_j$ in $list$
 6:       Calculate $U_j$ of $Task_j$
 7:       Drop $Task_j$
 8:       if $U_j \geq U_s$ then
 9:       Update $U_{LM}$ and $U_{HM}$ (HC tasks), $U_{LM}$ (LC tasks)
10:       end if
11:     end for
12:     return $U_{LM}$ and $U_{HM}$ (HC tasks), $U_{LM}$ (LC tasks)
13: end procedure
```

V. Experimental Evaluation

This section evaluates the benefits and drawbacks of the proposed approach introduced in Section IV by performing extensive simulation and comparing the results with other state-of-the-art scheduling algorithms from the literature in [5] and [6]. For comparing our proposed mechanism with the priority-driven (i.e., FPP) in [5] and deadline-driven (i.e., EDF-VD) in [6] algorithms. As mentioned in section II, the EDF-VD is an adaption of EDF to MC systems and its proved to be the most optimal scheduling algorithm for the independent MC tasks, while the FPP scheduling algorithm is the widely used to schedule the dependent tasks in the real-life applications and industry-based scenarios.

We generated 10000 task sets for each utilization value. The WCETs ranges in the generated test data were in [30:500]ms. The UUniFast algorithm [18] was used to generate valid utilization values for each task in a set, and a log-uniform distribution approach proposed in [19] was used to create a random arrival jitter for the tasks in [0:100] ms. Each data-point of the curves shown below was obtained by randomly generating 10000 task sets and testing each for schedulability according to the corresponding algorithms and their upper

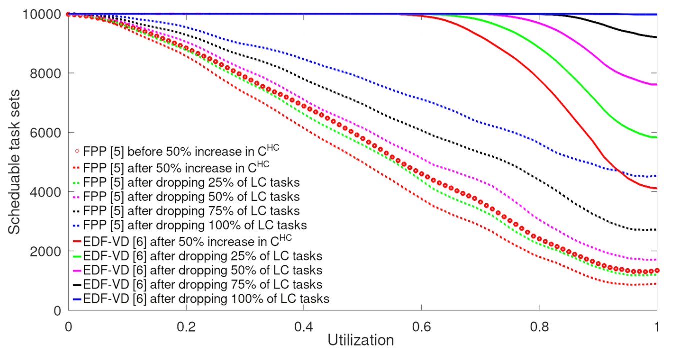

Fig. 3: The number of schedulable task sets versus utilization for ten tasks per set is compared with the conventional EDF-VD, and FPP scheduling algorithms on the same randomly generated task sets after dropping LC tasks.

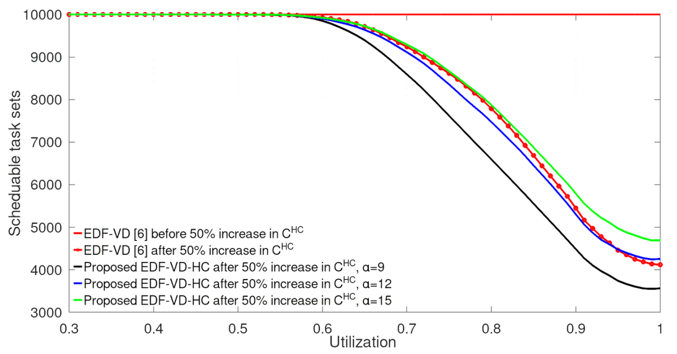

Fig. 4: The number of schedulable task sets versus utilization, without dropping LC tasks, when $\beta_{HC} = \beta_{LC} = 0.1$ and $\alpha = 9$, 12, and 15.

bounds in section III. The default task set size was 10, and the task deadlines were implicit $D_i = T_i$. Tasks priorities range in [0:9], where the first five values are for HC tasks. The priorities of each task set were randomly assigned to the tasks to represent the industry-based scenarios. Tasks were placed in queues according to their arrival times. In this comparison, the impact of the number of HC tasks and a varying increase in the tasks execution demand is investigated.

Figure 3 shows the baselines for the comparison with our proposed algorithm. In this figure, the number of schedulable task sets versus the utilization is simulated using the conventional EDF-VD (solid lines) and FPP (dashes lines) algorithms. The percentage of HC tasks in each task set was 60%, and we increased the execution demands for the HC tasks to 50%. The maximum value to increase the execution demands for the HC tasks was 66.6%. The task set is considered to be schedulable in the conventional approaches if there is no missed deadline for all of the HC tasks. The red curves in Figure 3 (solid and dot lines) are fully implemented based on the EDF-VD and the FPP scheduling algorithms, and we refer to [6] and [5] for the proof of it. As the value of the utilization increase, the total number of scheduable task sets decreases due to the preemption cost, which is higher in the case of the FPP scheduling algorithm. The generated task sets, before increasing the execution demands for the HC tasks, were fully scheduable by the EDF-VD algorithm for all the utilization values. We didn't plot it in this graph for the sake of brevity. However, after increasing the execution demands for the HC tasks with 50%, the number of scheduable task sets decreases when the utilization values increase. This is due to the increase in the ratio of C_i/T_i for the HC tasks. The FPP algorithm (red dots line) shows a decrease in the total number of scheduable task sets when the value of the utilization increase, even before increasing the HC tasks execution demands. This is due to the algorithm's preemption cost. As shown in Figure 3, for both EDF-VD and FPP algorithms, the number of schedulable task sets will increase as the percentage of LC tasks drops. The percentage of tasks dropped was increased from 0% up to 100%, which means that all the four LC tasks in the set are dropped.

A. Deadline-driven scheduling algorithm at HC portion

This section evaluates the proposed mechanism when selecting the EDF-VD algorithm to schedule the HC tasks and the round-robin RR algorithm to schedule the LC tasks. As mentioned before, the proposed mechanism does not require changing the selected algorithm that schedule the tasks in each portion. Since all the tasks in the HC portion are HC tasks, the tasks are scheduled according to EDF. Figure 4 shows the simulation results when increasing the value of α from 9 to 15, which increases the number of schedulable task sets by increasing the overall HC time slot of the HC tasks. Compared with the conventional EDF-VD, the proposed mechanism shows better results when the percentage of HC tasks per set is below 100%, i.e., 60%. If the percentage of the HC tasks increase towards the pure HC tasks, then the task set is not considered as a MC task set. To show the effect of increasing α on the total number of scheduable task sets, we calculated the HC and LC time slots, based on Eq. (7) and Eq. (9), before increasing the execution demands for the HC task. After increasing the HC tasks execusion demands, which means increasing the utilization ratios C_i/T_i by 50#, the increase in α value provides the HC portion with more processing resources through increasing the HC time slot. In Figure 4, when α at the utilization value of 0.9 equal 9, the proposed mechanism shows -17.8% in the number of schedulable task sets compared with the conventional approach. After increasing α to 15, the proposed approach shows 5.3% better results than the conventional approach. If the HC tasks per set increase, i.e., more than 90%, the proposed mechanism will compare to the chosen scheduling algorithm.

If the value of α reaches its threshold, then LC task(s) should be dropped based on the increase in the execution demands of the HC tasks. After dropping LC tasks, i.e., 25%, the value of α will be reset, i.e., starts from 1. As shown in Figure 5, if the percentage of dropping LC tasks increases from 25% to 75%, the proposed mechanism shows slightly better results than the conventional approach with 4.2%, 2.3%, and 0.8% when the utilization value equal 0.9, respectively.

In Table III, we listed the average response times R_{avg} and the waiting times in the queues Q_{avg} for both the conventional EDF-VD scheduling algorithm and the proposed mechanism for the simulation results shown in Figure 5. As the percentage of dropping the LC tasks increase, both the R_{avg} and Q_{avg} decrease. The challenge of having the same scheduling algo-

TABLE III: The average response times (R_{avg}) and the waiting times at the queues (Q_{avg}) for the simulation results presented in Figure 4 are listed in this table.

	Conventional				Proposed			
Drop%	R^{HC}_{avg}	R^{LC}_{avg}	Q^{HC}_{avg}	Q^{LC}_{avg}	R^{HC}_{avg}	R^{LC}_{avg}	Q^{HC}_{avg}	Q^{LC}_{avg}
0%	1559.8	233.17	1149.3	178.63	1626.4	537.04	1216.3	482.41
25%	1374.4	105.51	988.05	75.594	1415.6	312.29	1030.4	282.85
50%	1157.8	34.713	799.10	21.941	1209.2	152.47	852.44	139.74
75%	941.69	6.80	616.27	0	1003.7	44.040	679.75	40.366
100%	734.52	0	448.42	0	792.63	0	507.62	0

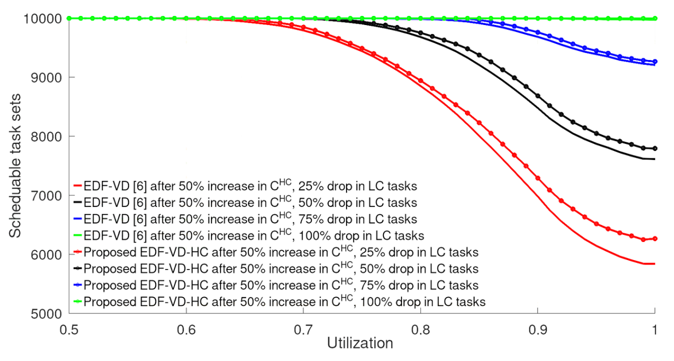

Fig. 5: Comparison between the conventional EDF-VD and the proposed mechanism after dropping LC tasks.

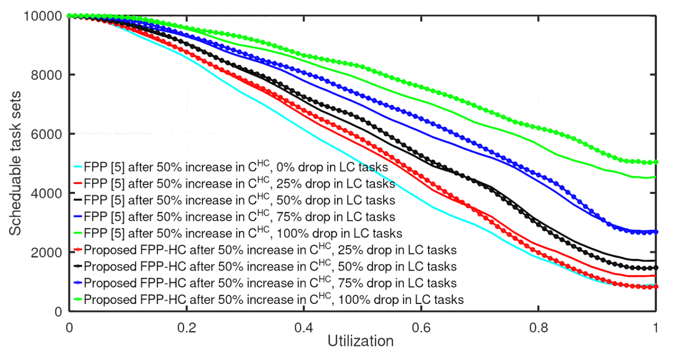

Fig. 7: Comparison between the conventional FPP and the proposed algorithm after dropping LC tasks.

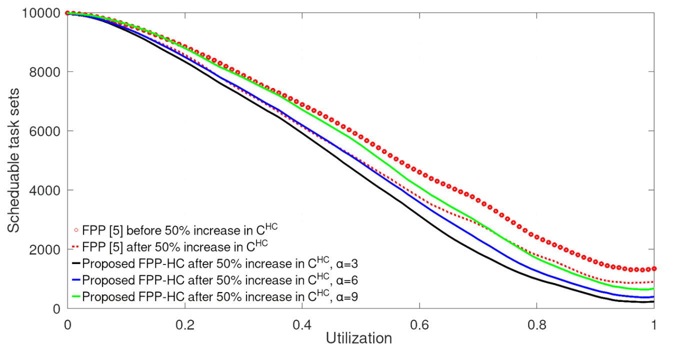

Fig. 6: The number of schedulable task sets versus utilization, without dropping LC tasks, when $\beta_{HC} = \beta_{LC} = 0.15$ and α = 3, 6, and 9.

rithm for both HC and LC tasks in the conventional approaches allows the LC tasks to take more of the processor utilization when they exceed their execution demands, which directly affects the HC tasks. In the proposed approach, we solved this challenge by fixing the upper bound of the LC time slot and dynamically changing the HC time slot as shown in Figure 1.

B. *Priority-driven scheduling algorithm at HC portion*

We evaluated the FPP algorithm (FPP-HC) to schedule the HC tasks and the round-robin algorithm RR to schedule the LC tasks. Figure 6 shows the simulation results when increasing the value of α from 3 to 9, which increases the number of schedulable task sets by increasing the overall HC time slot. In Figure 6, when α at the utilization value of 0.5 equal 3, the proposed mechanism shows -8.9% in the number of schedulable task sets compared with the conventional approach. After increasing α to 9, the proposed approach shows 10.7% better results than the conventional approach.

Figure 7 compares the conventional FPP scheduling algorithm and the proposed approach by running the FPP scheduling algorithm on the HC portion with increasing the percentage of dropping in LC tasks when α equal 9. As shown in Figure 7, if the percentage of dropping LC tasks increases from 25% to 100%, the proposed mechanism shows better results than the conventional approach with 3.8%, 2.7%, 7.1%, and 6.5% when the utilization value equal 0.6, respectively.

VI. Issues and Future Works

The proposed approach was evaluated by extensive simulation on synthetic data with medium percentages of HC tasks per task set. We showed that the proposed mechanism has better results than the conventional approaches to guarantee the achievement of the deadlines for the HC tasks. However, increasing the percentage of HC tasks to its upper limit, i.e., 100% per task set is not considered as a MC challenge. The values of the LC time slot kept below its maximum limit in all the conducted experiments. Decreasing the LC time slot without dropping one or more of the LC tasks will increase the average response times for all the LC tasks, and this decrease can be achieved by making the variable β_{LC} less than the variable β_{HC}. In this work, we controlled both β_{HC} and β_{LC} values between zero and one to guarantee the deadlines for the periodic HC tasks and at the same time to provide the LC tasks with the best average of response times. Table IV shows that the values of α and β should be determined during the run time, especially when handling situations where the number of tasks per set is not fixed. Moreover, the values of α and β depends on the utilization of the task set, the number of tasks per set, and the average execution times of the tasks. We postpone a discussion regarding how our proposed mechanism generalize to more than ten tasks per set and with different non preemptive scheduling approaches to an extended paper which is currently under preparation. The total number of context switches, which added by the proposed mechanism, depends on the execution demands for the tasks in the generated data set as shown in Table IV.

TABLE IV: The comparison between conventional FPP and the proposed algorithm at HC and LC portions using the FPP scheduling algorithm at the HC portion and round-robin RR scheduling algorithm at the LC portion while dropping LC tasks. The number of schedulable HC tasks per utilization (Sch_{HC}), the average response times (R_{avg}), the waiting times at the queues (Q_{avg}), and the average context switch per utilization (CX_{Alg}) for α equal 12, and $\beta_{HC} = \beta_{LC} = 0.05$. CX_{Alg} determines the additional overhead that is added by the presented mechanism. U is the utilization value, #tasks is the number of tasks per set, and C_{range} is the minimum and maximum values of the execution times for the generated tasks in the data set. In the conventional approach, Sch_{HC} determines the number of task sets without a deadline missing for all the HC tasks.

			Conventional					Proposed					
#tasks	U	C_{range}	R_{avg}^{HC}	R_{avg}^{LC}	Q_{avg}^{HC}	Q_{avg}^{LC}	Sch_{HC}	R_{avg}^{HC}	R_{avg}^{LC}	Q_{avg}^{HC}	Q_{avg}^{LC}	CX_{Alg}	Sch_{HC}
10	0.5	100-400	1587	477	1200	426	7562	2272	531	1886	480	377	8975
20	0.5	100-400	2994	936	2606	886	6240	4014	994	3625	944	375	6629
30	0.5	100-400	4408	1393	4020	1344	5176	4737	1450	4348	1399	375	4356
10	0.7	100-400	1572	490	1186	437	4408	2291	537	1903	485	379	4261
20	0.7	100-400	2983	947	2595	896	2746	3968	998	3582	947	374	2749
30	0.7	100-400	4401	1400	4013	1350	1687	4731	1449	4342	1398	374	1090
10	0.5	500-1000	4741	1447	3579	1292	9299	7444	1583	6284	1427	1127	9177
20	0.5	500-1000	8967	2844	7801	2692	9153	11332	3025	10166	2872	1125	9030
30	0.5	500-1000	13239	4184	12071	4035	9054	14139	4381	12971	4231	1124	8245
10	0.7	500-1000	4755	1421	3591	1267	5081	7461	1585	6299	1428	1129	3752
20	0.7	500-1000	9009	2808	7841	2657	4158	11386	3036	10219	2883	1126	3592
30	0.7	500-1000	13235	4207	12066	4057	3324	14115	4422	12949	4270	1124	2170

Presently, the proposed mechanism dynamically update the values of α and β to guarantee the minimum misses of deadlines for the HC tasks on a single processor and the best average response times for the LC tasks. This process increment and decrements α, together with dropping and reactivating the LC tasks. An implementation of the proposed mechanism on a real platform is in progress. It is worth to mention that the cost of preemption for the selected algorithm to schedule the HC tasks affects the values of α and β. Our results shows that the the EDF-HC is better than the FPP-HC due to the preemption cost for for the same utilization's value.

VII. Conclusion

This paper proposed a new approach for scheduling the Mixed-Criticality (MC) tasks on a single processor based on the round-robin technique. Initially, tasks are divided into two partitions: HC and LC, where each partition is assigned a time slot on a portion of the total processor utilization. In each partition, tasks start to run for their optimistic WCETs; if one or more HC tasks in the HC partition run over its optimistic $WCET_{LC}$, then an increase in the HC time slots will provide the HC portion with the required resources. Therefore, the proposed mechanism considered more than two levels of the criticality system. In each time slot, tasks are scheduled by a selected algorithm based on the dependencies of the tasks without modifying the chosen algorithm. We evaluated the EDF-VD on the HC task as a deadline-driven algorithm and the FPP as a priority-driven algorithm. In both cases, we scheduled the LC tasks using the round-robin technique. To evaluate the proposed approach, we conducted a large set of experiments, and we showed that the proposed approach has better results in the total number of schedulable tasks per utilization value. As the execution demands of HC tasks increase, the proposed mechanism can provide better results by decreasing the LC time slot and increasing the HC time slot to its upper bound, where the total processor utilization is given to the HC tasks.

References

[1] S. Baruah, "Mixed-criticality scheduling theory: Scope, promise, and limitations," *IEEE Design & Test*, vol. 35, no. 2, pp. 31–37, 2018.

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

[3] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–97, 2022.

[4] Y. Li, M. Danish, and R. West, "Quest-v: A virtualized multikernel for high-confidence systems," 2011.

[5] S. Baruah, A. Burns, and R. Davis, "An extended fixed priority scheme for mixed criticality systems," in *ReTiMiCS, RTCSA*, L. George and G. Lipari, Eds., 2013, pp. 18–24.

[6] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 145–154.

[7] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2011, pp. 34–43.

[8] Y. Chen, K. G. Shin, and H. Xiong, "Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems," *Information Processing Letters*, vol. 116, no. 8, pp. 508–512, 2016.

[9] P. Ekberg and W. Yi, "Outstanding paper award: Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 135–144.

[10] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2013, pp. 78–87.

[11] P. Dong, A. Burns, Z. Jiang, and X. Liao, "Tzdks: A new trustzone-based dual-criticality system with balanced performance," in *Proc. of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 59–64.

[12] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral, "Ltzvisor: Trustzone is the key," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2017.

[13] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *2010 Second International Conference on Computer and Network Technology*, 2010, pp. 222–226.

[14] F. Farshchi, P. K. Valsan, R. Mancuso, and H. Yun, "Deterministic Memory Abstraction and Supporting Multicore System Architecture," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Altmeyer, Ed., vol. 106. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 1:1–1:25.

[15] M. Chisholm, N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting mode changes while providing hardware isolation in mixed-criticality multicore systems," in *Proc. of International Conference on Real-Time Networks and Systems (RTNS)*. New York, NY, USA: ACM, 2017, p. 58–67.

[16] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Proc. WMC, RTSS*, 2013, pp. 1–6.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, p. 46–61, 1973.

[18] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," vol. 30, no. 1–2, p. 129–154, 2005.

[19] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. of International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2010, pp. 6–11.